



# Python for Data Science

Enthought, Inc.  
[www.enthought.com](http://www.enthought.com)

(c) 2001-2015, Enthought, Inc.

All Rights Reserved.

All trademarks and registered trademarks are the property of their respective owners.

Enthought, Inc.  
515 Congress Avenue  
Suite 2100  
Austin, TX 78701

[www.enthought.com](http://www.enthought.com)

Q3-2015  
letter  
489b71a

# Python for Data Science

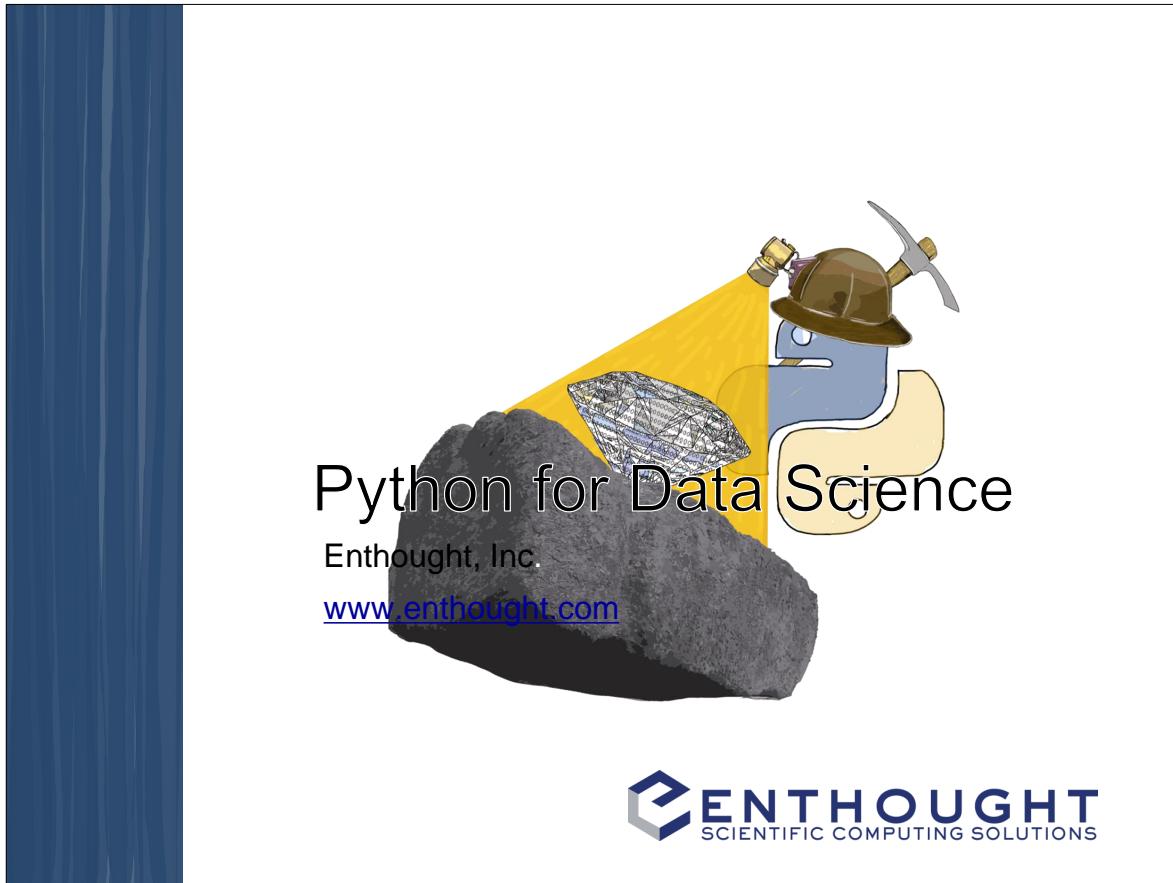
Enthought, Inc.

[www.enthought.com](http://www.enthought.com)

<b>Introduction</b> .....	3
Python for Data Science .....	5
Python data analytics platform and environment .....	10
<b>Language Introduction, data types</b> .....	34
Numerical types .....	36
Booleans .....	39
Strings .....	40
Lists (Indexing and Slicing) .....	45
Assignment .....	50
Mutable vs. Immutable .....	52
Tuple .....	53
Dictionaries .....	55
Sets .....	58
<b>Language Introduction, control statements</b> .....	61
If Statements .....	63
While Loops .....	65
For Loops .....	66
List Comprehensions .....	67
Looping Patterns .....	68
<b>Language Introduction, organizing code</b> .....	70
Functions .....	72
Function calling conventions .....	76
Modules .....	77
<b>Core libraries for Data Analytics</b> .....	83
<b>NumPy</b> .....	92
Introducing NumPy Arrays .....	95
Multi-Dimensional Arrays .....	98
Slicing/Indexing Arrays .....	99
Fancy Indexing .....	103
Creating arrays .....	107
Array Creation Functions .....	109
Array Calculation Methods .....	112
The array data structure .....	117
Advanced NumPy overview .....	124

<b>SciPy overview</b>	128
<b>Matplotlib basics</b>	132
<b>Data analysis with pandas</b>	142
Pandas	143
Storing data in Pandas	144
1D pandas.Series	146
2D pandas.DataFrame	148
Indexing	151
Re-indexing	157
Dealing with dates and times	160
Dealing with missing data	163
Computations and statistics	167
Data aggregation	172
Data summarization	180
<b>Python tools for Data Analytics</b>	183
<b>Data access</b>	187
Importing files	189
Reading files	190
Writing files	192
Pandas IO	193
Connecting to Databases	198
DB-API 2.0	200
SQL Alchemy	205
SQL queries from Pandas	211
<b>Visual exploration</b>	214
Plotting distributions of individual variables	218
Plotting joint distributions	223
Small multiples, a.k.a. Trellis plot	225
Pair grids	230
<b>Statistics</b>	233
Benford's law and fraud detection	236
Hypothesis testing	240
Multiple measurements	241
Type I and Type II error	242
<b>Advanced data analytics with Machine Learning</b>	246
Introduction	247
sklearn	251
<b>Preprocessing</b>	257
Outliers	260
Normalization	263
Missing values	265
Dimensionality reduction	266

<b>Regression problems</b> .....	272
Definition .....	273
Linear models .....	274
Evaluating regression results .....	281
Support Vector Regressor .....	282
Decision trees .....	285
Overfitting .....	290
Regularization .....	291
Bias/variance trade-off .....	295
<b>Model selection</b> .....	298
Definition .....	299
Cross-validation .....	301
Pipeline .....	311
GridSearchCV .....	313
<b>Classification</b> .....	317
Linear models .....	319
Multiple classes .....	322
Maximum margin classifiers .....	324
Support Vector Machines .....	325
Decision trees .....	327
Instance-based classifiers .....	328
Evaluating classifiers .....	330
<b>Parallel sklearn</b> .....	335
<b>Natural Language Processing</b> .....	338
Python packages for NLP .....	339
Text modeling workflow .....	341
Preparing text .....	342
Tokenizers .....	344
Stemmers .....	347
Extracting text features .....	348
Text classification .....	355
Naive Bayes classifiers .....	357
Topic modeling .....	364
Latent Dirichlet Allocation .....	366
<b>Appendix</b> .....	373
Language reference .....	374
NumPy reference .....	379
matplotlib reference .....	387



## Schedule

	Day 1	Day 2	Day 3	Day 4	Day 5
Morning	Intro and Platform	Intro Python	Pandas	Visual exploration	Regression
<b>Break</b>					
Morning	Intro Python	Data processing tools	Pandas	Visual exploration	Cross-validation
<b>Lunch</b>					
Afternoon	Intro Python	NumPy	Pandas	Statistics	Classification
<b>Break</b>					
Afternoon	Intro Python	NumPy, matplotlib	Data access	Advanced data analytics	NLP (time permitting)

# What Is Data Science?

## ONE LINER

Data science is a set of practices and fundamental principles that guide the extraction of useful information and knowledge from (usually) large volumes of data in order to improve business decision-making.

## DATA SCIENCE HIGHLIGHTS

- Not a single discipline
- Workflow involves many distinct phases: Preparing data, Exploring, Analyzing, Modeling, Packaging for Deployment, Decision Making
- Data analysis involves a collection of diverse tools: Visualization, Statistics, Machine Learning, Natural Language Processing, etc.
- Significant effort spent wrangling & managing data

3

# What Is Python?

## ONE LINER

Python is an interpreted programming language that is easy to learn, easy to use, and comprehensive in terms of data science tools (data munging, stats, machine learning, NLP, etc.) and programming styles (from exploratory analysis to repeatable science to software engineering for production deployment).

## PYTHON HIGHLIGHTS

- Interpreted and interactive
- Automatic garbage collection
- Dynamic typing
- Object-oriented
- Free
- Portable / Cross-Platform
- Easy to Learn and Use
- “Batteries Included”

4

# Why Python for Data Science?

- High-level language, allows rapid prototyping to explore multiple approaches
- Libraries and tools exist to support you during all phases of your workflow
- Easy, Matlab-like visualization tools
- Active, growing scientific community

5

# Who is using Python for Data Science?

## WALL STREET

Some of the largest investment banks and hedge funds rely on Python for their core trading and risk mgmt / fraud detection systems.

## TRAVEL INDUSTRY

Travel companies use Python for data mining and predictive analytics

- Travel pricing insights
- Recommendation systems
- Predicting travel delays

## PETROLEUM INDUSTRY

Geophysics and exploration tools:

- ConocoPhillips
- Shell

## ASTRA ZENECA

AstraZeneca consolidated some of their disparate drug discovery tools into a suite called PyDrone with great success.

## SOCIAL MEDIA

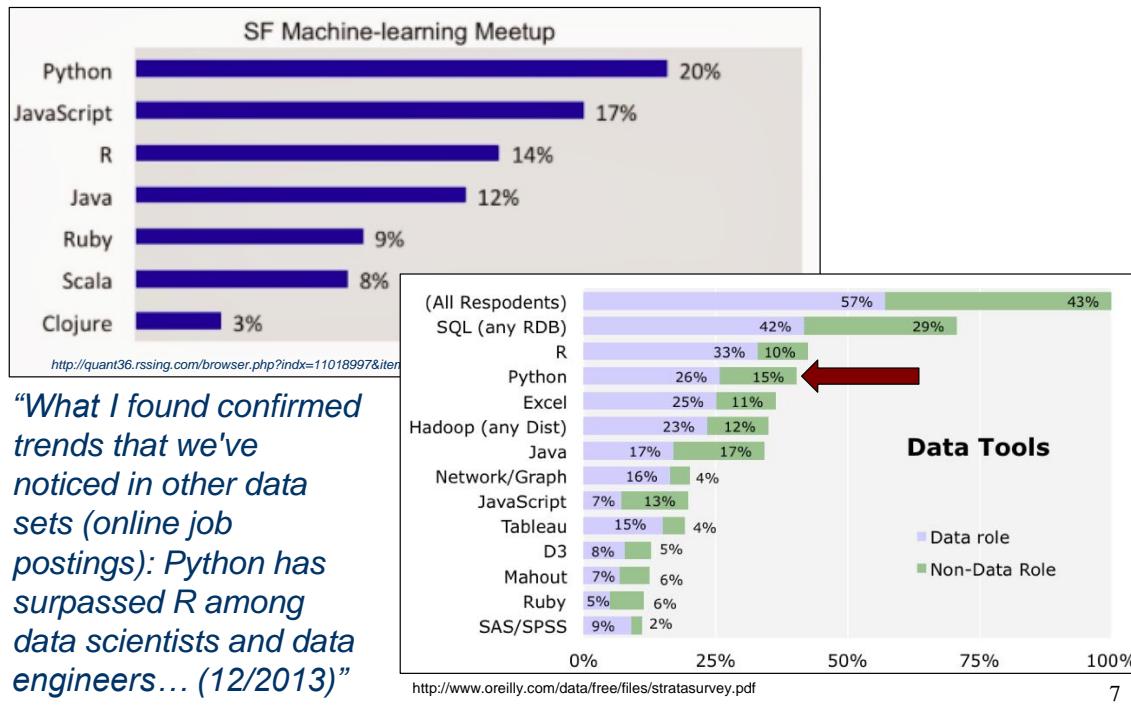
Many people and companies use Python to analyze social media data from Google, LinkedIn, Facebook, Twitter, etc. (for sentiment analysis, customer segmentation, prediction).

## MANY MORE

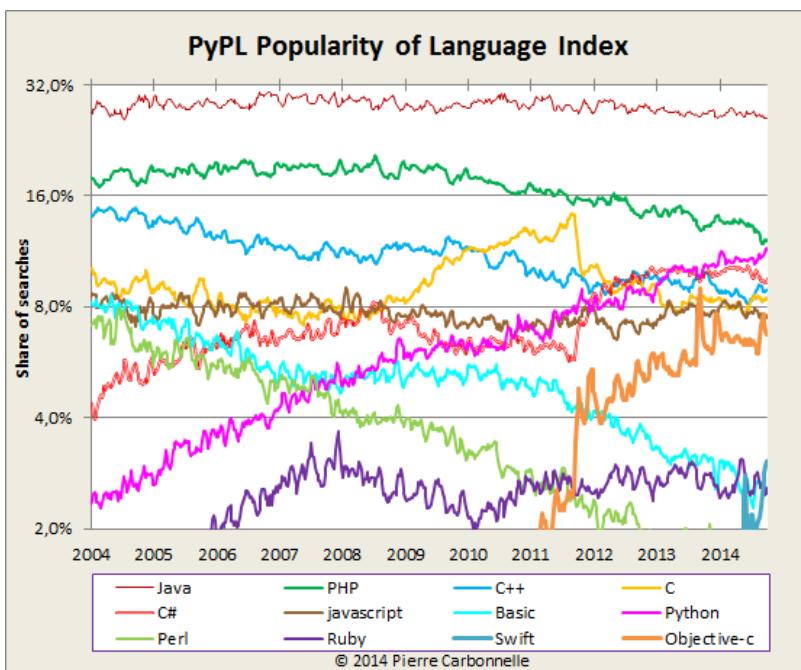
- Gov: National Labs, SEC, ...
- PayPal
- Uber
- ...

6

## Interest in Python for Data Science

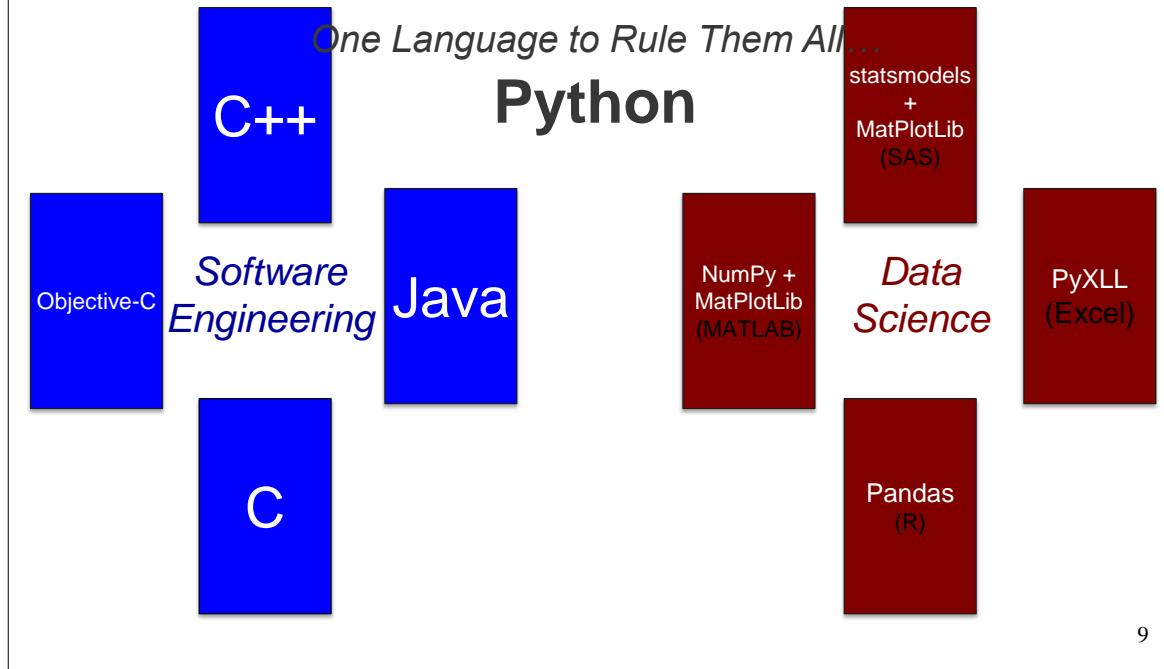


## PYPL PopularitY of Programming Language index



\* <https://sites.google.com/site/pydatalog/pypl/PyPL-PopularitY-of-Programming-Language> (CC by 3.0 license)

# Computing Paradigms



9

Python data analytics platform  
and environment

10

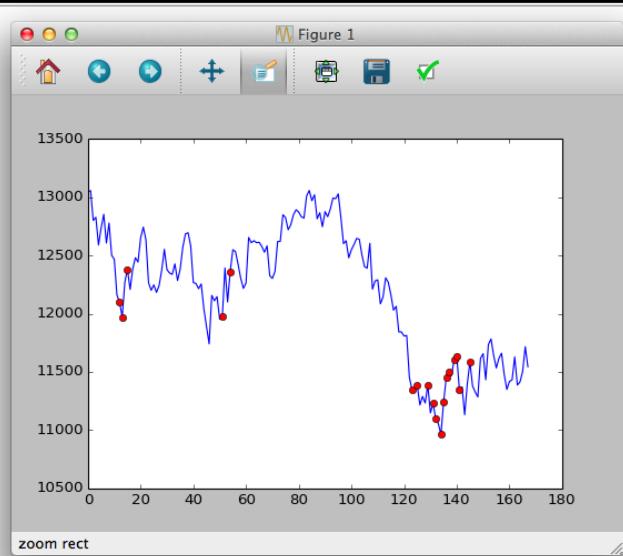
# IPython

An enhanced  
interactive Python shell

11

## IPython Prompt: Rapid Exploration

```
In [3]: dow = loadtxt('dow.csv', delimiter=',')
In [4]: high_volume_mask = dow[:, 4] > 5.5e9
In [5]: high_volume_days = sum(high_volume_mask)
In [6]: high_vol_index = where(high_volume_mask)[0]
In [7]: figure()
Out[7]: <matplotlib.figure.Figure at 0xa044330>
In [8]: plot(dow[:,5], 'b-')
Out[8]: [<matplotlib.lines.Line2D at 0xa069cb0>]
In [9]: plot(high_vol_index, dow[high_vol_index, 5], 'ro')
Out[9]: [<matplotlib.lines.Line2D at 0xbbe1e03d>]
In [10]:
```



12

# Starting IPython

IPython can be started:

- w/ the pylab icon
- by typing `ipython --pylab=qt` in any terminal/command prompt



```
Last login: Sun Aug 4 20:03:00 on ttys005
Admins-MacBook-Pro-2:~ jrocher: ipython --pylab
Enthought Python Distribution - www.enthought.com

Python 2.7.3 | 64-bit | (default, Jun 14 2013, 18:17:36)
Type "copyright", "credits" or "license" for more information.

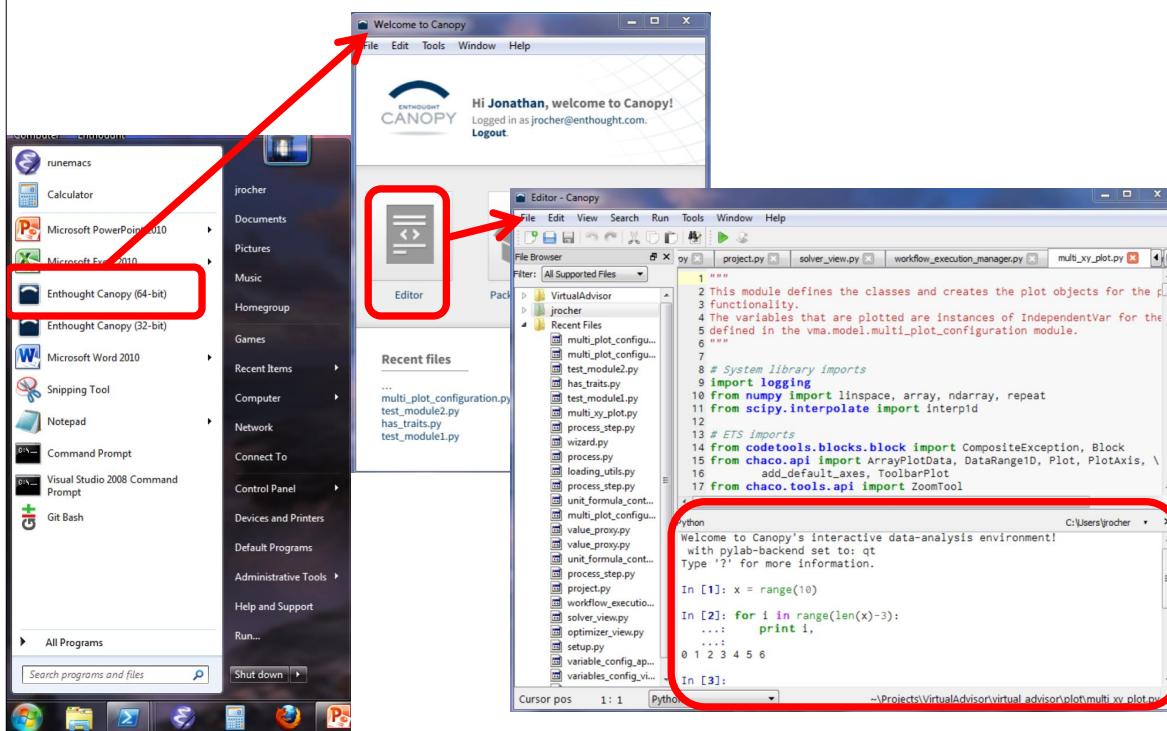
IPython 0.13.1 -- An enhanced Interactive Python.
?           --> Introduction and overview of IPython's features.
%quickref --> Quick reference.
help        --> Python's own help system.
object?     --> Details about 'object', use 'object??' for extra details.

Welcome to pylab, a matplotlib-based Python environment [backend: MacOSX]
For more information, type 'help(pylab)'.

In [1]:
```

13

# Starting IPython in Canopy



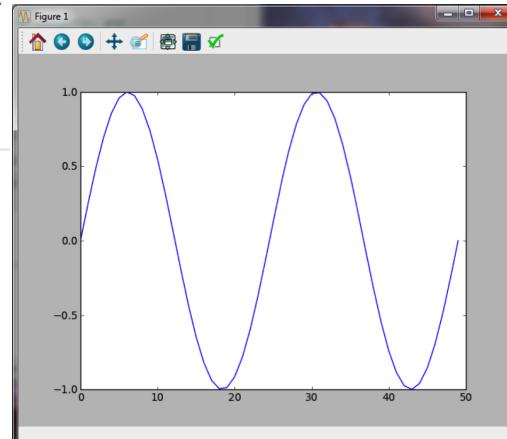
# PyLab: Interactive Python Environment

Python

```
In [4]: x = linspace(0, 4*pi)
In [5]: plot(sin(x))
Out[5]: [<matplotlib.lines.Line2D at 0x84909e8>]
In [6]: print sin(3*pi/4)
0.707106781187
In [7]: sin(3*pi/4) == sqrt(2)/2
Out[7]: True
In [8]:
```

The PyLab mode in IPython handles some gory details behind the scenes. It allows both the Python command interpreter (above) and the GUI plot window (right) to coexist. This involves a bit of multi-threaded magic.

PyLab also imports some handy functions into the command interpreter for user convenience.



15

# IPython

## STANDARD PYTHON

```
In [1]: a=1
In [2]: a
Out[2]: 1
```

## AVAILABLE VARIABLES

```
In [3]: b = [1,2,3]
# List available variables.
In [4]: %whos
Variable      Type      Data/Info
-----
a            int        1
b            list      n=3
```

## RESET

```
# Remove user defined variables.
In [5]: %reset
In [6]: %whos
Interactive namespace is empty.
```



“%reset” also removes the names imported by PyLab, such as the plot command.

```
In [7]: plot
NameError: name 'plot' is not defined
```

```
# Reload pylab.
In [8]: %pylab
Welcome to pylab, ...
```

16

# Directory Navigation in IPython

```
# Change directory (note Unix style forward slashes!)
In [9]: cd c:/python_class/Demos/speed_of_light
c:\python_class\Datas\speed_of_light
```



Tab completion helps you find and type directory and file names.

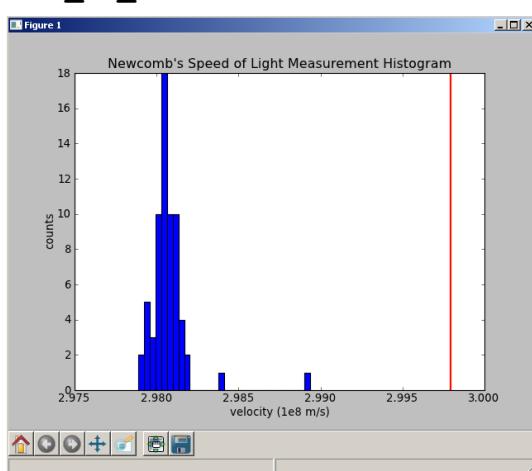
```
# List directory contents (Unix style, not "dir").
In [10]: ls
Volume in drive C has no label.
Volume Serial Number is 5417-593D
Directory of c:\python_class\Datas\speed_of_light
09/01/2008 02:53 PM <DIR> .
09/01/2008 02:53 PM <DIR> ..
09/01/2008 02:48 PM 1,188 exercise_speed_of_light.txt
09/01/2008 02:48 PM 2,682,023 measurement_description.pdf
09/01/2008 02:48 PM 187,087 newcomb_experiment.pdf
09/01/2008 02:48 PM 1,312 speed_of_light.dat
09/01/2008 02:48 PM 1,436 speed_of_light.py
09/01/2008 02:48 PM 1,232 speed_of_light2.py
6 File(s) 2,874,278 bytes
2 Dir(s) 11,997,437,952 bytes free
```

17

# Running Scripts in IPython

```
# tab completion
In [11]: %run speed_of_li
speed_of_light.dat  speed_of_light.py
```

```
# execute a python file
In [11]: %run speed_of_light.py
```



18

# Function Info

## HELP USING ?

```
# Follow a command with '?' to print its documentation.
```

```
In [19]: len?
```

```
Type:      builtin_function_or_method
```

```
String form: <built-in function len>
```

```
Namespace:  Python builtin
```

```
Docstring:
```

```
len(object) -> integer
```

```
Return the number of items of a sequence or mapping.
```

19

# Function Info

## SHOW SOURCE CODE USING ??

```
# Follow a command with '??' to print its source code.
```

```
In [43]: squeeze??
```

```
def squeeze(a):
```

```
    """Remove single-dimensional entries from the shape of a.
```

```
Examples
```

```
-----
```

```
>>> x = array([[1,1,1],[2,2,2],[3,3,3]])
```

```
>>> x.shape
```

```
(1, 3, 3)
```

```
>>> squeeze(x).shape
```

```
(3, 3)
```

```
"""
```

```
try:
```

```
    squeeze = a.squeeze
```

```
except AttributeError:
```

```
    return _wrapit(a, 'squeeze')
```

```
return squeeze()
```



?? can't show the source  
code for "extension" functions  
that are implemented in C.

20

# IPython History

## HISTORY COMMAND

```
# list previous commands. Use
# 'magic' % because 'hist' is
# histogram function in pylab
In [3]: %hist
a=1
a
```

## INPUT HISTORY

```
# list string from prompt[2]
In [4]: _i2
Out[4]: 'a\n'
```

## OUTPUT HISTORY

```
# grab previous result
In [5]: _
Out[5]: 'a\n'

# grab result from prompt[2]
In [6]: _2
Out[6]: 1
```



The up and down arrows scroll through your ipython input history.

21

# Reading Simple Tracebacks

## ERROR ADDING AN INTEGER TO A STRING

```
In [9]: 1 + "hello"
-----
TypeError      Traceback (most recent call last)
C:\...\ipython-input...> in <module>()
----> 1 1 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Location and code where error occurred.

The “type” of error that occurred.

Short message about why it occurred.

## ERROR TRYING TO ADD A NON-EXISTENT VARIABLE

```
# Again we fail when adding two variables, but note that the
# traceback tells us we have a completely different problem.
# In this case, our variable doesn't exist, so the operation fails.
In [10]: undefined_var + 1
...
NameError: name 'undefined_var' is not defined
```

22

# IPython Notebook: Repeatable science

One .ipynb file with:

- code
- results
- inline figures
- formatted text (including equations)
- titles
- and more...

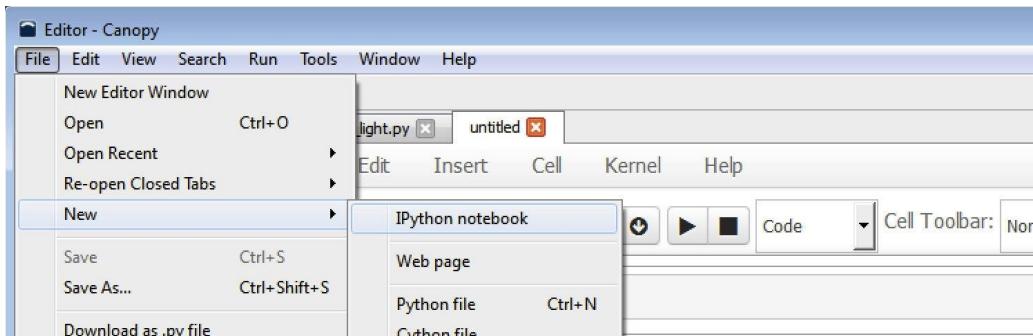
```
In [1]: x = linspace(0, 4*pi, 100)
In [2]: plot(sin(x))
Out[2]: [

```

23

## Creating a notebook in Canopy

Menu: File > New > IPython notebook

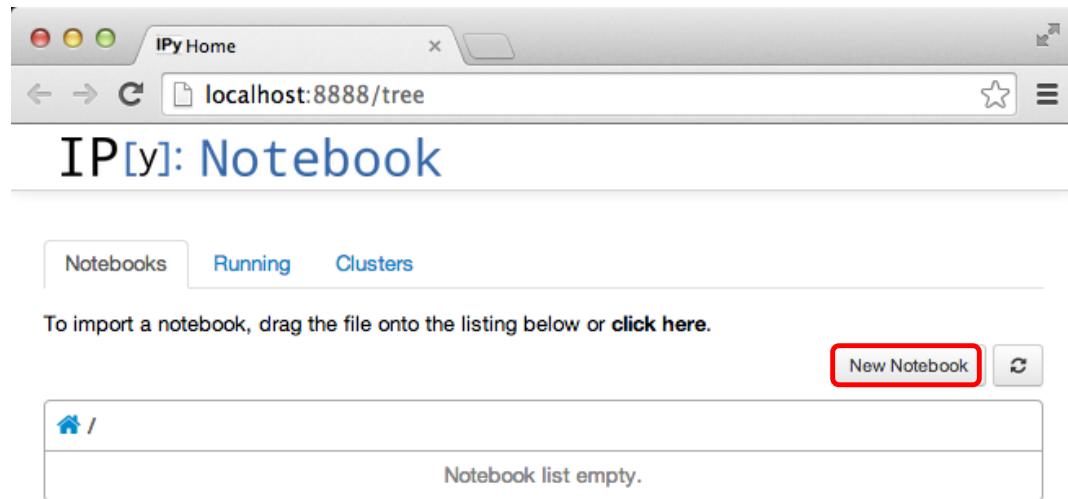


24

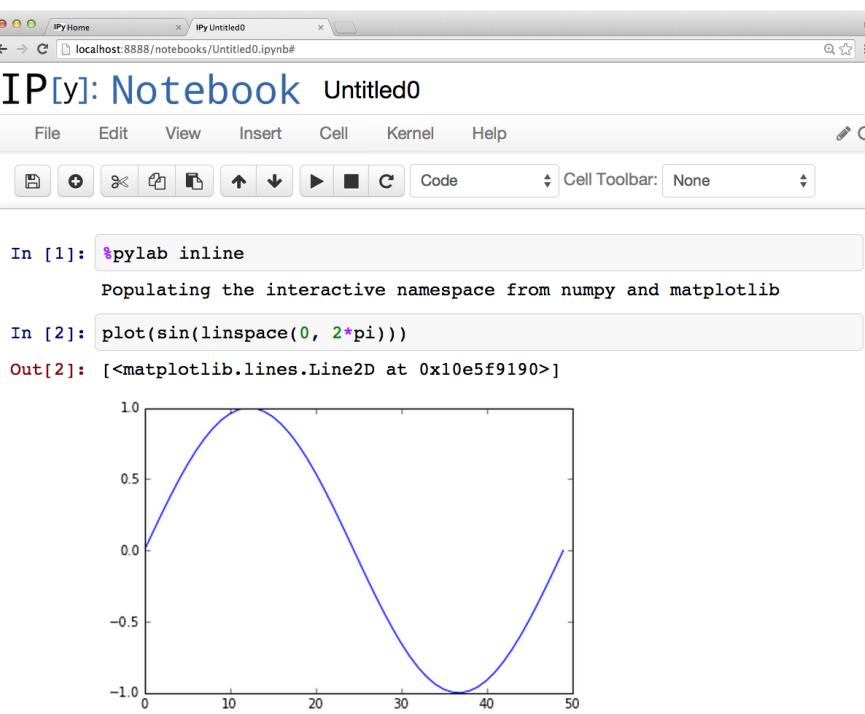
# Creating a notebook from terminal

From a terminal/command prompt, start a notebook server that is viewed in your default web browser:

```
$ ipython notebook
```



# Pylab mode in the notebook



# IPython notebook features

1. Cells can group multiple commands. Execute cells with SHIFT-ENTER.
2. Make a cell a “Markdown” cell to create titles, control the font, ...
3. Cells can be deleted, moved around, inserted AND executed in random order.
4. Insert images, webpages, LaTeX formulas, YouTube videos, ... using `IPython.display` objects or functions:

```
In [2]: from IPython.display import Latex
Latex("$\int_a^b f(x) dx$")
```

```
Out[2]:  $\int_a^b f(x) dx$ 
```

27

# Script Editor: Rapid Prototyping

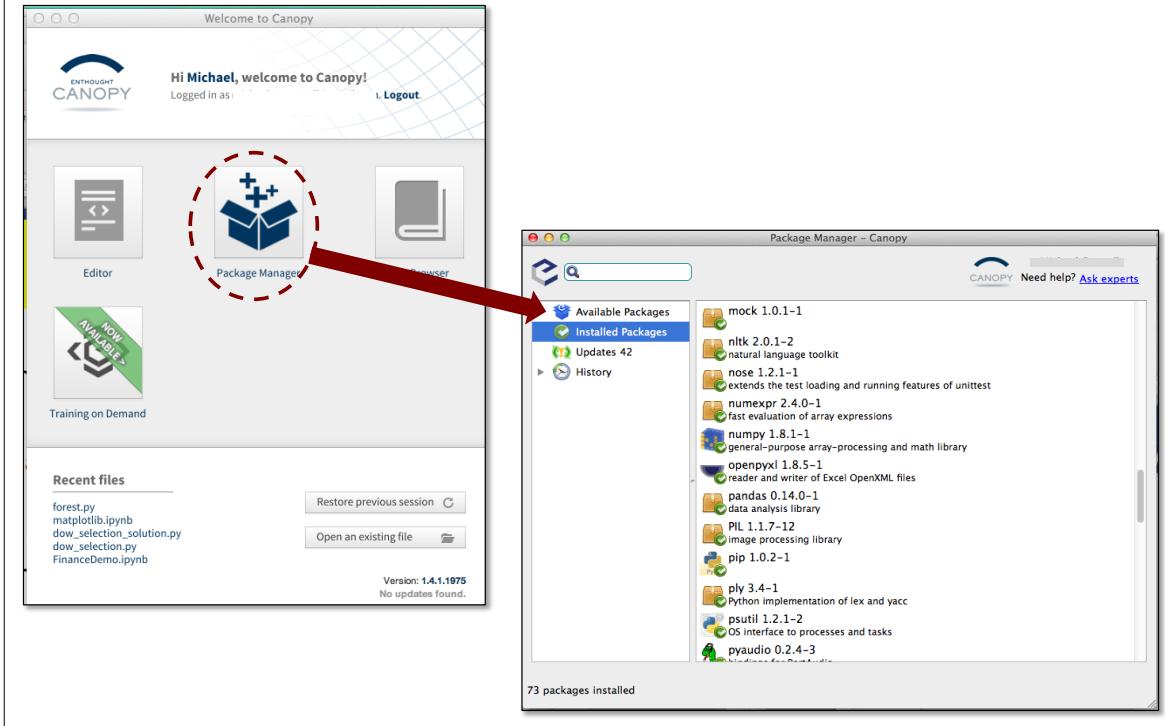
The screenshot shows the Enthought Script Editor interface. On the left, a code editor displays a Python script for simulating a forest. The script imports numpy, random, and various traits-based classes. It defines a `Forest` class with methods for generating lightning strikes, sapling growth, and fires, as well as for advancing time and burning trees. On the right, there are two plots: one showing the fractional area covered by trees (a green heatmap) and another showing the fractional area with fires (a red heatmap). Below the plots are two sliders labeled "trees" and "fires", each with a value of 0.0025 and 0.0001398 respectively, and buttons for "Stop", "trees", "Advance 1 Day".

```

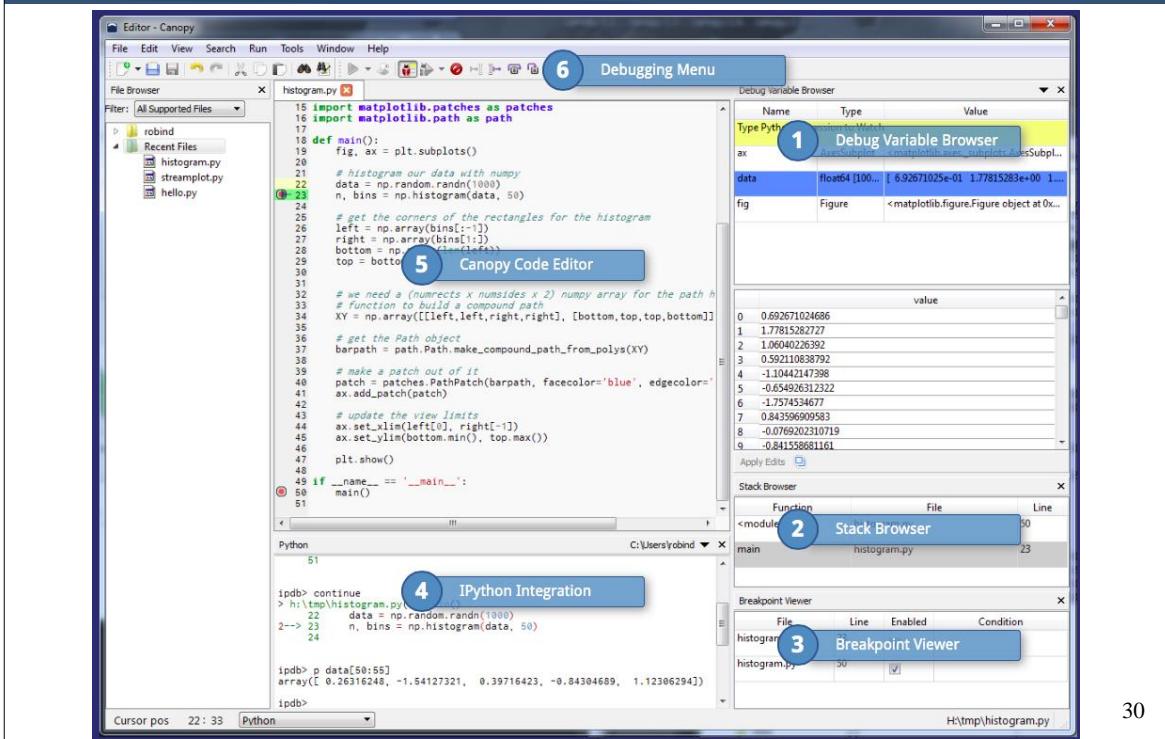
1 # Source at https://github.com/timdiller/complexity/blob/master/forest.py
2 import numpy as np
3 from numpy.random import uniform
4 from scipy.ndimage.measurements import label
5
6 from chaco.api import ArrayPlotData, Plot, VPlotContainer
7 from enable.api import ComponentEditor
8 from pyface.timer.api import Timer
9 from traits.api import (HasTraits, Array, Bool, Button, DelegatesTo, Enum,
10                         Instance, Int, Property, Range,
11                         String)
12 from traitsui.api import ButtonEditor, HGroup, Item, VGroup, View
13
14 history_length = 3000
15
16
17 def randbool(nx, ny, p):
18     return uniform(size=(nx, ny)) <= p
19
20
21 class Forest(HasTraits):
22     p_lightning = Range(0., 0.0005, 5.e-6)
23     p_sapling = Range(0., 0.005, 0.0025)
24     forest_trees = Array(dtype=bool)
25     forest_fires = Array(dtype=bool)
26     size_x = Int(150)
27     size_y = Int(150)
28
29     def _forest_trees_default(self):
30         return np.zeros((self.size_x, self.size_y))
31
32     def _forest_fires_default(self):
33         return np.zeros((self.size_x, self.size_y))
34
35     def advance_one_day(self):
36         self.grow_trees()
37         self.start_fires()
38         self.burn_trees()
39
40     def burn_trees(self):
41         fires = np.zeros((self.size_x + 2, self.size_y + 2), dtype=bool)
42         fires[1:-1, 1:-1] = self.forest_fires
43         north = fires[:-2, 1:-1]

```

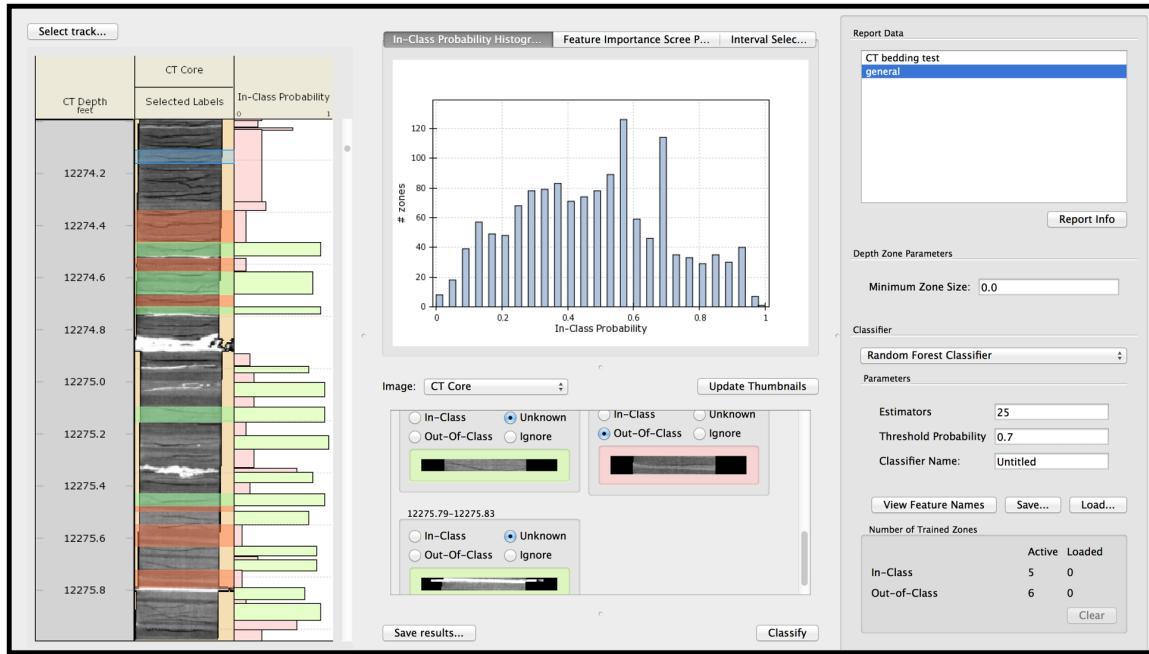
# Canopy Package Manager



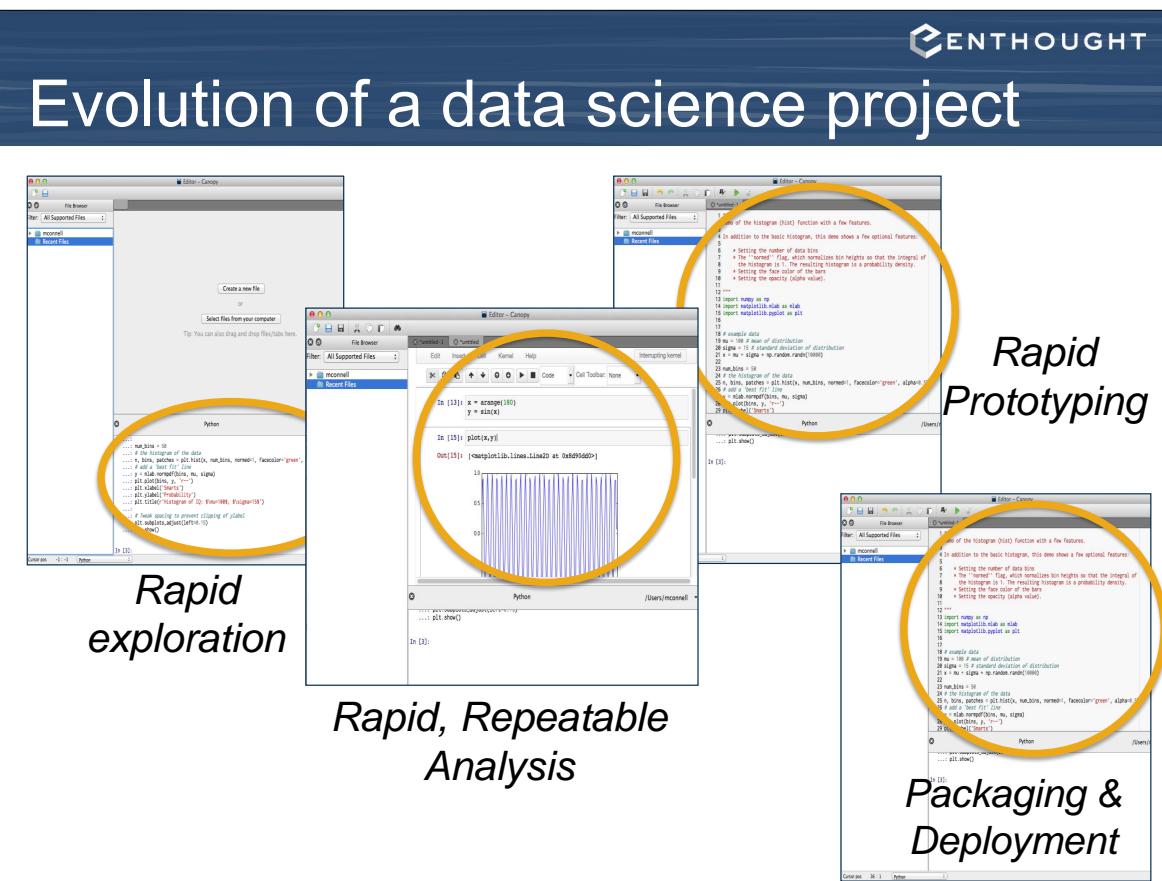
# Canopy Interactive Debugger



# Tools for Packaging and Deployment



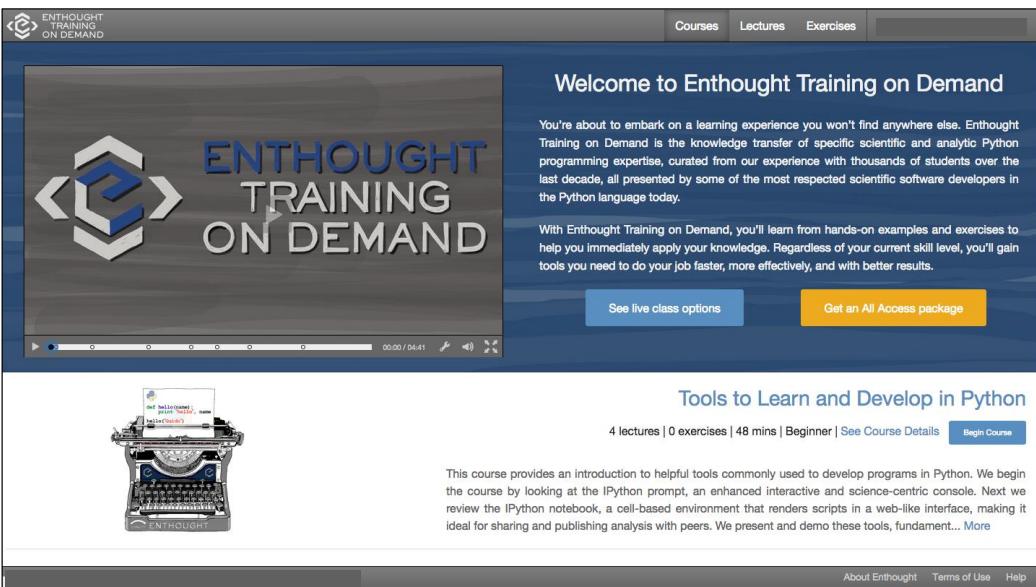
31



# Enthought Training on Demand

## training.enthought.com

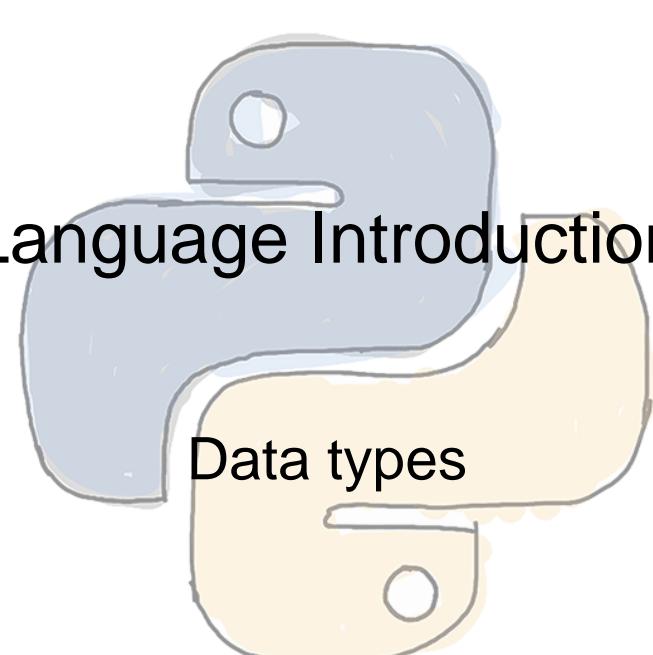
30-day subscription – during and after course



The screenshot shows the Enthought Training on Demand website. At the top, there's a navigation bar with 'Courses', 'Lectures', and 'Exercises'. Below the header, a video player displays the 'Enthought Training On Demand' logo. To the right, a 'Welcome to Enthought Training on Demand' section includes a brief description of the service, a call-to-action button for 'See live class options', and another for 'Get an All Access package'. Below this, a course card for 'Tools to Learn and Develop in Python' is shown, featuring an image of a typewriter, a brief description, and a 'Begin Course' button.

33

## Language Introduction



Data types

34

# Outline

- Data types:
  - Numerical types: int, long, float, complex
  - Booleans
  - Strings
  - Lists and tuples
  - Dictionaries and sets
  - Things to know about efficiency

35

# Interactive Calculator

```
# adding two values
>>> 1 + 1
2
# setting a variable
>>> a = 1
>>> a
1
# checking a variable's type
>>> type(a)
<type 'int'>
# an arbitrarily long integer
>>> a = 12345678901234567890
>>> a
12345678901234567890L
>>> type(a)
<type 'long'>
# Remove 'a' from the 'namespace'
>>> del a
>>> a
NameError: name 'a' is not
defined
```

```
# real numbers
>>> b = 1.4 + 2.3
>>> b
3.6999999999999997
# "prettier" version.
>>> print b
3.7
>>> type(b)
<type 'float'>
# complex numbers
>>> c = 2+1.5j
>>> c
(2+1.5j)
```

The four numeric types in Python on 64-bit architectures are:



integer 8 byte (4 byte on Windows)  
 long integer Any precision  
 float 8 byte, like C's double  
 complex 16 byte

The NumPy library, which we will see later, supports a larger number of numeric types.<sup>36</sup>

# More Interactive Calculation

## ARITHMETIC OPERATIONS

```
>>> 1+2-(3*4/6)**5+(7%5)
-27
```

## SIMPLE MATH FUNCTIONS

```
>>> abs(-3)
3
>>> max(0, min(10, -1, 4, 3))
0
>>> round(2.718281828)
3.0
```

## OVERWRITING FUNCTIONS (!)

```
# don't do this
>>> max = 100

# ...some time later...
>>> x = max(4, 5)
TypeError: 'int' object is not
callable
```

## TYPE CONVERSION

```
>>> int(2.718281828)
2
>>> float(2)
2.0
>>> 1+2.0
3.0
```

## IN-PLACE OPERATIONS

```
>>> b = 2.5
>>> b += 0.5      # b = b + 0.5
>>> b
3.0
# Also -=, *=, /=, etc.
```

37

# Give it a try!

Compute

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

for

$$a \leftarrow -1.0$$

$$b \leftarrow 3.0$$

$$c \leftarrow 5.0$$

38

# Logical expressions, bool data type

## COMPARISON OPERATORS

```
# <, >, <=, >=, ==, !=
>>> 1 >= 2
False
>>> 1 + 1 == 2
True
>>> 2**3 != 3**2
True
# Chained comparisons
>>> 1 < 10 < 100
True
```

## bool DATA TYPE

```
>>> q = 1 > 0
>>> q
True
>>> type(q)
<type 'bool'>
```

## and OPERATOR

```
>>> 1 > 0 and 5 == 5
True
# If first operand is false,
# the second is not evaluated.
>>> 1 < 0 and max(0,1,2) > 1
False
```

## or OPERATOR

```
>>> a = 50
>>> a < 10 or a > 90
False
# If first operand is true,
# the second is not evaluated.
>>> a = 0
>>> a < 10 or a > 90
True
```

## not OPERATOR

```
>>> not 10 <= a <= 90
True
```

39

# Strings

## CREATING STRINGS

```
# using double quotes
>>> s = "hello world"
>>> print s
hello world
# single quotes also work
>>> s = 'hello world'
>>> print s
hello world
```

## STRING OPERATIONS

```
# concatenating two strings
>>> "hello " + "world"
'hello world'

# repeating a string
>>> "hello " * 3
'hello hello hello '
```

## STRING LENGTH

```
>>> s = "12345"
>>> len(s)
5
```

## SPLIT/JOIN STRINGS

```
# split space-delimited words
>>> s = "hello world"
>>> wrd_lst = s.split()
>>> print wrd_lst
['hello', 'world']

# join words back together
# with a space in between
>>> space = ' '
>>> space.join(wrd_lst)
'hello world'
```

40

# Multi-line Strings

## TRIPLE QUOTES

```
# Strings in triple quotes retain line breaks
>>> a = """hello
... world"""
>>> print a
hello
world
```

## NEW LINE CHARACTER

```
# Including a newline character
>>> a = "hello\nworld"
>>> print a
hello
world
```

41

# A few string methods and functions

## REPLACING TEXT

```
>>> s = "hello world"
>>> s.replace('world','Mars')
'hello Mars'
```

## CONVERT TO UPPER CASE

```
>>> s.upper()
'HELLO WORLD'
```

## REMOVE WHITESPACE

```
>>> s = "\t hello \n"
>>> s.strip()
'hello'
```

## NUMBERS TO STRINGS

```
>>> str(1.1 + 2.2)
'3.3'
>>> repr(1.1 + 2.2)
'3.3000000000000003'
>>> str(1)
'1'
```

## STRINGS TO NUMBERS

```
>>> int('23')
23
>>> int('FF', 16)
255
>>> float('23')
23.0
```

42

# String Formatting

The `format()` method replaces any ***replacement fields*** in the string with the values given as arguments.

**Replacement field format:** {*Optional <name>* :*Optional <format\_spec>*}

```
# If 'name' is an integer, it refers to the argument position.
>>> '{0} is greater than {1}'.format(100, 50)
'100 is greater than 50'

# If 'name' is text, it refers to a keyword argument.
>>> '{last}, {first}'.format(first='Ellen', last='Ripley')
'Ripley, Ellen'
```

43

# String Formatting – Format spec

The optional format specification is used to control how the values are displayed. (See Appendix for details.)

```
# Fixed point format (and a named keyword argument).
>>> print '[{x:5.0f}] [{x:5.1f}] [{x:5.2f}]'.format(x=12.3456)
[ 12] [ 12.3] [12.35]

# Alignment (and using a numbered positional argument).
>>> print '[{0:<10s}] [{0:>10s}] [{0:^10s}]'.format('PYTHON')
[PYTHON      ] [      PYTHON] [    PYTHON    ]

# Alignment with fill character.
>>> template = '[{0:*<10s}] [{0:*>10s}] [{0:.*^10s}]'
>>> print template.format('PYTHON')
[PYTHON*****] [*****PYTHON] [**PYTHON**]
```

44

# List objects

## LIST CREATION WITH BRACKETS

```
>>> a = [10,11,12,13,14]
>>> print a
[10, 11, 12, 13, 14]
```

## CONCATENATING LIST

```
# simply use the + operator
>>> [10, 11] + [12, 13]
[10, 11, 12, 13]
```

## REPEATING ELEMENTS IN LISTS

```
# the multiply operator
# does the trick
>>> [10, 11] * 3
[10, 11, 10, 11, 10, 11]
```

## range(start, stop, step)

```
# the range function is helpful
# for creating a sequence
>>> range(5)
[0, 1, 2, 3, 4]

>>> range(2,7)
[2, 3, 4, 5, 6]

>>> range(2,7,2)
[2, 4, 6]
```

45

# Indexing

## RETRIEVING AN ELEMENT

```
# list
# indices: 0  1  2  3  4
>>> a = [10,11,12,13,14]
>>> a[0]
10
```

## SETTING AN ELEMENT

```
>>> a[1] = 21
>>> print a
[10, 21, 12, 13, 14]
```

## OUT OF BOUNDS

```
>>> a[10]
Traceback (innermost last):
File "<interactive input>", line 1, in ?
IndexError: list index out of range
```

## NEGATIVE INDICES

```
# negative indices count
# backward from the end of
# the list
#
# indices: -5 -4 -3 -2 -1
>>> a = [10,11,12,13,14]

>>> a[-1]
14
>>> a[-2]
13
```



The first element in an array has index=0 as in C. **Take note Matlab and Fortran programmers!**

46

# More on list objects

## LIST CONTAINING MULTIPLE TYPES

```
# list containing integer,
# string, and another list
>>> a = [10,'eleven',[12,13]]
>>> a[1]
'eleven'
>>> a[2]
[12, 13]

# use multiple indices to
# retrieve elements from
# nested lists
>>> a[2][0]
12
```

## LENGTH OF A LIST

```
>>> len(a)
3
```

## DELETING OBJECT FROM LIST

```
# use the del keyword
>>> del a[2]
>>> a
[10,'eleven']
```

## DOES THE LIST CONTAIN x ?

```
# use in or not in
>>> a = [10,11,12,13,14]
>>> 13 in a
True
>>> 13 not in a
False
```

47

# Slicing

## `var[lower:upper:step]`

Extracts a portion of a sequence by specifying a lower and upper bound.

The lower-bound element is included, but the upper-bound element *is not included*.

Mathematically: [lower, upper). The step value specifies the stride between elements.

## SLICING LISTS

```
# indices:
#      -5 -4 -3 -2 -1
#       0  1  2  3  4
>>> a = [10,11,12,13,14]
# [10,11,12,13,14]
>>> a[1:3]
[11, 12]

# negative indices work also
>>> a[1:-2]
[11, 12]
>>> a[-4:3]
[11, 12]
```

## OMMITTING INDICES

```
# omitted boundaries are
# assumed to be the beginning
# (or end) of the list

# grab first three elements
>>> a[:3]
[10, 11, 12]
# grab last two elements
>>> a[-2:]
[13, 14]
# every other element
>>> a[::-2]
[10, 12, 14]
```

48

## Lists in action

```
>>> a = [10,21,23,11,24]

# add an element to the list
>>> a.append(11)
>>> print a
[10,21,23,11,24,11]
# how many 11s are there?
>>> a.count(11)
2
# extend with another list
>>> a.extend([5,4])
>>> print a
[10,21,23,11,24,11,5,4]
# where does 11 first occur?
>>> a.index(11)
3
# insert 100 at index 2?
>>> a.insert(2, 100)
>>> print a
[10,21,100,23,11,24,11,5,4]
```

# pop the item at index=3  
 >>> a.pop(3)  
 23  
 # remove the first 11  
 >>> a.remove(11)  
 >>> print a  
 [10,21,100,24,11,5,4]  
 # sort the list (in-place)  
 # Note: use sorted(a) to  
 # return a new list.  
 >>> a.sort()  
 >>> print a  
 [4,5,10,11,21,24,100]  
 # reverse the list  
 >>> a.reverse()  
 >>> print a  
 [100,24,21,11,10,5,4]

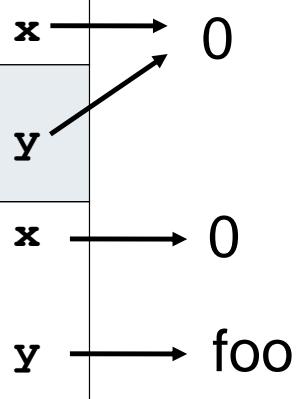
49

## Assignment of “simple” object

Assignment creates object references.

```
>>> x = 0
# This causes x and y to point
# to the same value.
>>> y = x

# Re-assigning y to a new value
# decouples the two variables.
>>> y = "foo"
>>> print x
0
```



50

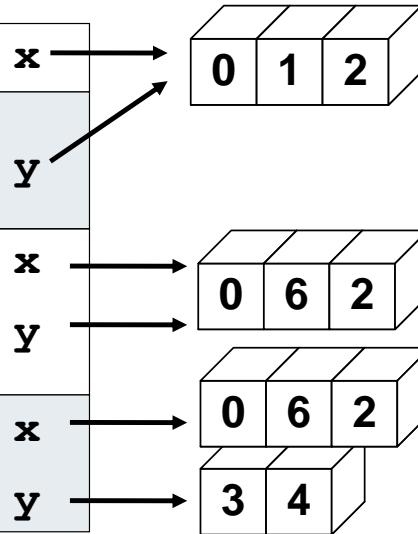
# Assignment of Container object

Assignment creates object references.

```
>>> x = [0, 1, 2]
# This causes x and y to point
# to the same list.
>>> y = x

# A change to y also changes x.
>>> y[1] = 6
>>> print x
[0, 6, 2]

# Re-assigning y to a new list
# decouples the two variables.
>>> y = [3, 4]
```



51

# Mutable vs. Immutable

## MUTABLE OBJECTS

```
# Mutable objects, such as
# lists, can be changed
# in place.

# insert new values into list
>>> a = [10,11,12,13,14]
>>> a[1:3] = [5,6]
>>> print a
[10, 5, 6, 13, 14]
```

## IMMUTABLE OBJECTS

```
# Immutable objects, such as
# integers and strings,
# cannot be changed in place.

# try inserting values into
# a string
>>> s = 'abcde'
>>> s[1:3] = 'xy'
Traceback (innermost last):
File "<interactive input>", line 1, in ?
TypeError: object doesn't support
slice assignment

# here's how to do it
>>> s = s[:1] + 'xy' + s[3:]
>>> print s
'axyde'
```

52

# Tuple – Immutable Sequence

## TUPLE CREATION

```
>>> a = (10,11,12,13,14)
>>> print a
(10, 11, 12, 13, 14)
```

## PARENTHESES ARE OPTIONAL

```
>>> a = 10,11,12,13,14
>>> print a
(10, 11, 12, 13, 14)
```

## LENGTH-1 TUPLE

```
>>> (10,)
(10,)

>>> (10)
10
```



(10) is not a tuple,  
but an integer  
with parentheses.

## TUPLES ARE IMMUTABLE

```
# create a list
>>> a = range(10,15)
[10, 11, 12, 13, 14]

# cast the list to a tuple
>>> b = tuple(a)
>>> print b
(10, 11, 12, 13, 14)

# try inserting a value
>>> b[3] = 23
TypeError: 'tuple' object doesn't
support item assignment
```

53

# Tuple (un)packing

## (UN)PACKING TUPLES

```
# Creating a tuple without ()
>>> d = 1, 2, 3
>>> d
(1, 2, 3)

# Multiple assignments from a
# tuple
>>> a, b, c = d
>>> print b
2

# Multiple assignments
>>> a, b, c = 1, 2, 3
>>> print b
2
```

## WHY IS IT USEFUL?

We will see later on that this feature is very common in Python code, e.g.:

```
# Returning multiple values
def monomials(x):
    return 1, x, x**2
a0, a1, a2 = monomials(3)

# Iterating over tuples
friends = [('Andrew', 28),
            ('Mario', 51),
            ('Karl', 30)]
for name, age in friends:
    txt = "{} is {} years old"
    print txt.format(name, age)
```

# Dictionaries

Dictionaries store *key/value* pairs. Indexing a dictionary by a *key* returns the *value* associated with it. The *key* must be immutable.

## DICTIONARY EXAMPLE

```
# Create an empty dictionary using curly brackets.
>>> record = {}

# Each indexed assignment creates a new key/value pair.
>>> record['first'] = 'Jmes'
>>> record['last'] = 'Maxwell'
>>> record['born'] = 1831
>>> print record
{'first': 'Jmes', 'born': 1831, 'last': 'Maxwell'}
# Create another dictionary with initial entries.
>>> new_record = {'first': 'James', 'middle':'Clerk'}
# Now update the first dictionary with values from the new one.
>>> record.update(new_record)
>>> print record
{'first': 'James', 'middle': 'Clerk', 'last':'Maxwell',
'born': 1831}
```

55

# Accessing and deleting keys and values

## ACCESS USING INDEX NOTATION

```
>>> print record['first']
James
```

## ACCESS WITH get(key, default)

The `get()` method returns the value associated with a key; the optional second argument is the return value if the key is not in the dictionary.

```
>>> record.get('born', 0)
1831
>>> record.get('home', 'TBD')
'TBD'
>>> record['home']
KeyError: ...
```

## REMOVE AN ENTRY WITH DEL

```
>>> del record['middle']
>>> record
{'born': 1831, 'first':
'James', 'last': 'Maxwell'}
```

## REMOVE WITH pop(key, default)

`pop()` removes the key from the dictionary and returns the value; the optional second argument is the return value if the key is not in the dictionary.

```
>>> record.pop('born', 0)
1831
>>> record
{'first': 'James', 'last':
'Maxwell'}
>>> record.pop('born', 0)
0
```

56

# Dictionaries in action

```
# dict of animals:count pairs
>>> barn = {'cows': 1,
...             'dogs': 5,
...             'cats': 3}

# test for chickens
>>> 'chickens' in barn
False

# get a list of all keys
>>> barn.keys()
['cats', 'dogs', 'cows']

# get a list of all values
>>> barn.values()
[3, 5, 1]

# return key/value tuples
>>> barn.items()
[('cats', 3), ('dogs', 5),
 ('cows', 1)]

# How many cats?
>>> barn['cats']
3

# Change the number of cats.
>>> barn['cats'] = 10
>>> barn['cats']
10

# Add some sheep.
>>> barn['sheep'] = 5
>>> barn['sheep']
5
```

57

# Set objects

## DEFINITION

A set is an *unordered collection of unique, immutable objects*.

## CONSTRUCTION

```
# an empty set
>>> s = set()

# convert a sequence to set
>>> t = set([1,2,3,1])
# note removal of duplicates
>>> t
set([1, 2, 3])
```

## ADD/REMOVE ELEMENTS

```
>>> t.add(5)
>>> t
set([1, 2, 3, 5])
>>> t.update([5,6,7])
>>> t
set([1, 2, 3, 5, 6, 7])
```

## REMOVE ELEMENTS

```
>>> t.remove(1)
set([2, 3, 5, 6, 7])
```

## SET OPERATIONS

```
>>> a = set([1,2,3,4])
>>> b = set([3,4,5,6])
>>> a.union(b)
set([1, 2, 3, 4, 5, 6])
>>> a.intersection(b)
set([3, 4])
>>> a.difference(b)
set([1, 2])
>>> a.symmetric_difference(b)
set([1, 2, 5, 6])
```






58

# Selecting a data type

Selecting the appropriate data type is important

	insert	remove	find	ordered
list	linear	linear	linear	✓
set	constant	constant	constant	✗
dict	constant	constant	constant	✗

Typical usages for each data type:

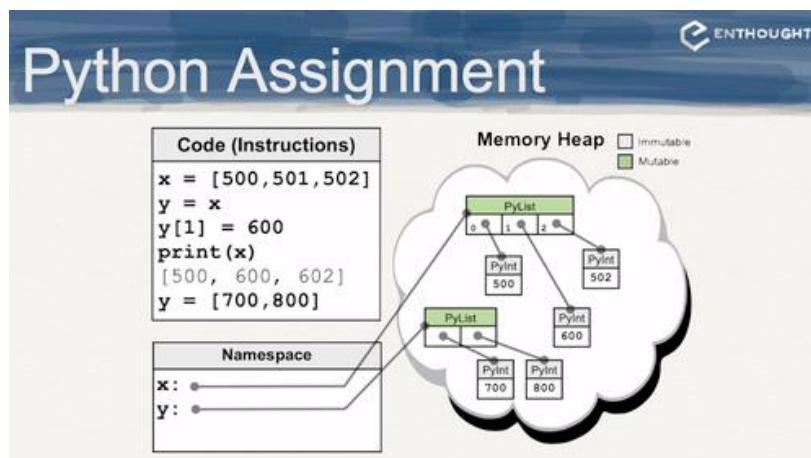
- Lists: Represent ordered collections of items, stacks, and queues [1]
- Sets: Represent collections of unique, unordered items
- Dicts: Represent registries, caches, mappings in general

[1] Also see the `deque` module for an efficient queue implementation

59

# Further learning online

Additional material and exercises available online:



# Language Introduction

## Control statements

61

## Outline

- If statements
- While loops
- For loops
  - List comprehensions
  - Looping patterns

62

# If statements

`if/elif/else` provides conditional execution of code blocks.

## IF STATEMENT FORMAT

```
if <condition>:  
    <statement 1>  
    <statement 2>  
elif <condition>:  
    <statements>  
else:  
    <statements>
```

## IF EXAMPLE

```
# a simple if statement  
>>> x = 10  
>>> if x > 0:  
...     print 'Hey!'  
...     print 'x > 0'  
... elif x == 0:  
...     print 'x is 0'  
... else:  
...     print 'x is negative'  
... < hit return >  
Hey!  
x > 0
```

63

# Test Values

- zero, `None`, "", and empty objects are treated as `False`.
- All other objects are treated as `True`.

## EMPTY OBJECTS

```
# empty objects test as false  
>>> x = []  
>>> if x:  
...     print 1  
... else:  
...     print 0  
... < hit return >  
0
```

It often pays to be explicit. If you are testing for an empty list, then test for:

```
if len(x) == 0:  
    ...
```



This is clearer to future readers of your code. It also can avoid bugs where `x==None` may be passed in and unexpectedly go down this path.

64

# While loops

**while** loops iterate until a condition is met

```
while <condition>:
    <statements>
```

## WHILE LOOP

```
# the condition tested is
# whether lst is empty
>>> lst = range(3)
>>> while lst:
...     print lst
...     lst = lst[1:]
... < hit return >
[0, 1, 2]
[1, 2]
[2]
```

## BREAKING OUT OF A LOOP

```
# breaking from an infinite
# loop
>>> i = 0
>>> while True:
...     if i < 3:
...         print i,
...     else:
...         break
...     i = i + 1
... < hit return >
0 1 2
```

65

# For loops

**for** loops iterate over a sequence of objects

```
for <loop_var> in <sequence>:
    <statements>
```

## TYPICAL SCENARIO

```
>>> for item in range(5):
...     print item,
... < hit return >
0 1 2 3 4

# For a large range, xrange()
# is faster and more memory
# efficient.
>>> for item in xrange(10**6):
...     print item,
... < hit return >
0 1 2 3 4 5 6 7 8 9 10 11 ...
```

## LOOPING OVER A STRING

```
>>> for item in 'abcde':
...     print item,
... < hit return >
a b c d e
```

## LOOPING OVER A LIST

```
>>> animals=['dogs', 'cats',
...             'bears']
>>> accum = ''
>>> for animal in animals:
...     accum += animal + ' '
... < hit return >
>>> print accum
dogs cats bears
```

66

# List Comprehension

## LIST TRANSFORM WITH LOOP

```
# element by element transform of
# a list by applying an
# expression to each element
>>> a = [10,21,23,11,24]
>>> results=[]
>>> for val in a:
...     results.append(val+1)
>>> results
[11, 22, 24, 12, 25]
```

## FILTER-TRANSFORM WITH LOOP

```
# transform only elements that
# meet a criteria
>>> a = [10,21,23,11,24]
>>> results=[]
>>> for val in a:
...     if val>15:
...         results.append(val+1)
>>> results
[22, 24, 25]
```

## LIST COMPREHENSION

```
# list comprehensions provide
# a concise syntax for this sort
# of element by element
# transformation
>>> a = [10,21,23,11,24]
>>> [val+1 for val in a]
[11, 22, 24, 12, 25]
```

## LIST COMPREHENSION WITH FILTER

```
>>> a = [10,21,23,11,24]
>>> [val+1 for val in a if val>15]
[22, 24, 25]
```

Consider using a list comprehension whenever you need to transform one sequence to another.

67

# Looping Patterns

## MULTIPLE LOOP VARIABLES

```
# Looping through a sequence of
# tuples allows multiple
# variables to be assigned.
>>> pairs = [(0,'a'),(1,'b'),
...             (2,'c')]
>>> for index, value in pairs:
...     print index, value
0 a
1 b
2 c
```

## ENUMERATE

```
# enumerate -> index, item.
>>> y = ['a', 'b', 'c']
>>> for index, value in enumerate(y):
...     print index, value
0 a
1 b
2 c
```

## ZIP

```
# zip 2 or more sequences
# into a list of tuples
>>> x = [0, 1, 2]
>>> y = ['a', 'b', 'c']
>>> zip(x,y)
[(0,'a'), (1,'b'), (2,'c')]
>>> for index, value in zip(x,y):
...     print index, value
0 a
1 b
2 c
```

## REVERSED

```
>>> z = [(0,'a'),(1,'b'),(2,'c')]
for index, value in reversed(z):
...     print index, value
2 c
1 b
0 a
```

68

# Looping over a dictionary

```
>>> d = {'a':1, 'b':2, 'c':3}
```

## DEFAULT LOOPING (KEYS)

```
>>> for key in d:  
...     print key, d[key]  
a 1  
c 3  
b 2
```

## LOOPING OVER KEYS (EXPLICIT)

```
>>> for key in d.keys():  
...     print key, d[key]  
a 1  
c 3  
b 2
```

## LOOPING OVER VALUES

```
>>> for val in d.values():  
...     print val  
1  
3  
2
```

## LOOPING OVER ITEMS

```
>>> for key, val in d.items():  
...     print key, val  
a 1  
c 3  
b 2
```

69

# Language Introduction

Organizing code

70

# Outline

- Functions
- Modules
- Packages

71

# Functions

Functions are reusable snippets of code.

- Definition
- Positional and keyword arguments

72

# Anatomy of a function

The keyword **def** indicates the start of a function.

Function arguments are listed, separated by commas. They are passed by *assignment*.

Indentation is used to indicate the contents of the function. It is *not* optional, but a part of the syntax.

```
def add(arg0, arg1):
    """Add two numbers"""
    a = arg0 + arg1
    return a
```

An optional **return** statement specifies the value returned from the function. If **return** is omitted, the function returns the special value **None**.

A colon (:) terminates the function signature.

An optional **docstring** documents the function in a standard way for tools like ipython.

73

# Our new function in action

```
# We'll create our function
# on the fly in the
# interpreter.
>>> def add(x,y):
...     a = x + y
...     return a
```

```
# Test it out with numbers.
>>> val_1 = 2
>>> val_2 = 3
>>> add(val_1, val_2)
5
```

```
# How about strings?
>>> val_1 = 'foo'
>>> val_2 = 'bar'
>>> add(val_1, val_2)
'foobar'
```

```
# Functions can be assigned
# to variables.
>>> func = add
>>> func(val_1, val_2)
'foobar'
```

```
# How about numbers and strings?
>>> add('abc',1)
Traceback (innermost last):
File "<interactive input>", line 1, in ?
File "<interactive input>", line 2, in add
TypeError: cannot add type "int" to string
```

74

# Give it a try!

Create a function called `count_letter` that takes as input a string `txt` and a character `char`, and returns the number of times `char` appears in `txt`, ignoring case.

For example:

```
>>> count_letter("Php, Perl, or Python?", "p")
4
```

# Function Calling Conventions

## POSITIONAL ARGUMENTS

```
# The "standard" calling
# convention we know and love.
>>> def add(x, y):
...     return x + y

>>> add(2, 3)
5
```

## KEYWORD ARGUMENTS

```
# specify argument names
>>> add(x=2, y=3)
5
# or even a mixture if you are
# careful with order
>>> add(2, y=3)
5
```

## DEFAULT VALUES

```
# Arguments can be
# assigned default values.
>>> def quad(x,a=1,b=1,c=0):
...     return a*x**2 + b*x + c

# Use defaults for a, b and c.
>>> quad(2.0)
6.0

# Set b=3. Defaults for a & c.
>>> quad(2.0, b=3)
10.0

# Keyword arguments can be
# passed in out of order.
>>> quad(2.0, c=1, a=3, b=2)
17.0
```

# Modules and packages

Modules and packages are Python's "libraries", i.e. a collection of constants, functions, and classes.

77

# Importing a module

## BASIC IMPORTS

```
# The most basic import
>>> import numpy
>>> numpy.pi
3.141592653589793
# Use an 'alias'
>>> import numpy as np
>>> np.pi
3.141592653589793
```

## IMPORTING SPECIFIC SYMBOLS

```
# Select specific names to
# bring into the local
# namespace.
>>> from numpy import add, pi
>>> pi
3.141592653589793
>>> add(2, 3)
5
```

## IMPORTING \*EVERYTHING\*

```
# Pull *everything* into the
# local namespace.
>>> from numpy import *
>>> pi
3.141592653589793
>>> add(3, 4.5)
7.5
```

## MODULES ARE .PY FILES

Modules are just .py files.

```
# my_tools.py
def greetings():
    return "Hello everyone"
```

```
>>> import my_tools
>>> my_tools.greetings()
'Hello everyone'
```

78

# Packages

## PACKAGES

Often a library will contain several modules. These are organized as a hierarchical directory structure, and imported using "dotted module names". The first and the intermediate names (if any) are called "packages".

Example:

```
>>> from email.utils import parseaddr
>>> from email import utils
>>> utils.parseaddr('John Doe <jdoe@company.com>')
('John Doe', 'jdoe@company.com')
```

`utils` is a *module* in the package `email` .

## PACKAGES ARE DIRECTORIES

```
foo/
    __init__.py
    bar.py (defines func)
    baz.py (defines zap)
```

The file `__init__.py` indicates that `foo` is a package. It often is an empty file.

79

# Give it a try!

1. Import functions `join` and `expanduser` from module `os.path`  
 Execute  
`join(expanduser('~'), 'myfile.txt')`
2. Import module `pandas` with alias `pd`
3. Import everything from module `numpy`

80

# Standard Modules

Python has a large library of standard modules ("batteries included"):

- re** - regular expressions
- copy** – shallow and deep copy operations
- datetime** - time and date objects
- math, cmath** - real and complex math
- decimal, fractions** - arbitrary precision decimal and rational number objects
- os, os.path, shutil** - filesystem operations
- sqlite3** - internal SQLite database
- gzip, bz2, zipfile, tarfile** – compression and archiving formats
- csv, netrc** – file format handling
- xml** – various modules for handling XML
- htmllib** – an HTML parser
- httplib, ftplib, poplib, socket, etc.** – modules for standard internet protocols

- cmd** – support for command interpreters
- pdb** – Python interactive debugger
- profile, cProfile, timeit** – Python profilers
- collections, heapq, bisect** – standard CS algorithms and data structures
- mmap** – memory-mapped files
- threading, Queue** – threading support
- multiprocessing** – process based ‘threading’
- subprocess** – executing external commands
- pickle, cPickle** – object serialization
- struct** – interpret bytes as packed binary data
- urllib2** – open and read from URLs

and many more... To see the content of one:

```
>>> dir(module_name)
```

81

# Further learning online

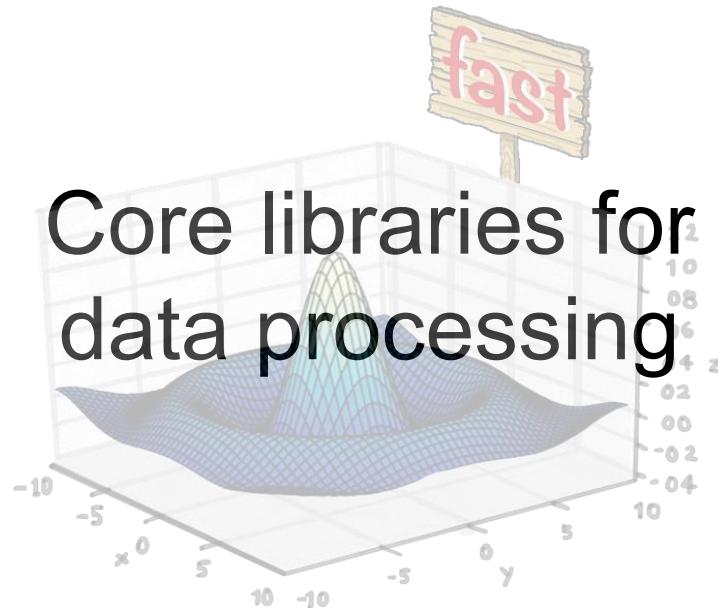
Additional material and exercises available online:

The collage consists of five separate screenshots arranged in a grid-like layout:

- Exceptions**: A screenshot showing a code snippet: 

```
If, at first, you don't succeed...
```

. The code tries to access a non-existent key in a dictionary and handles the `KeyError`.
- Decorators**: A screenshot showing a diagram where a `decorator` is applied to a `Function`.
- The with statement**: A screenshot showing a diagram with three boxes labeled `Enter`, `Calculate`, and `Exit`, each with a circular arrow icon below it.
- Generators**: A screenshot showing a circular diagram with a red arrow pointing clockwise and the word `YIELD` written vertically along the arrow.
- Intro to OOP: Overview**: A screenshot showing a diagram illustrating Object-Oriented Programming concepts. It shows a dashed line connecting a `Class` box to an `Object A` circle, and another dashed line connecting `Object A` to an `Object B` rectangle.



83

## Outline

- Overview of the Python data analytics ecosystem
- NumPy
- SciPy
- matplotlib

84

# Data analytics ecosystem: an overview

85

## Data analytics ecosystem

High performance computing

Interfacing  
with other  
languages

Scientific  
algorithms

Data storage and  
management

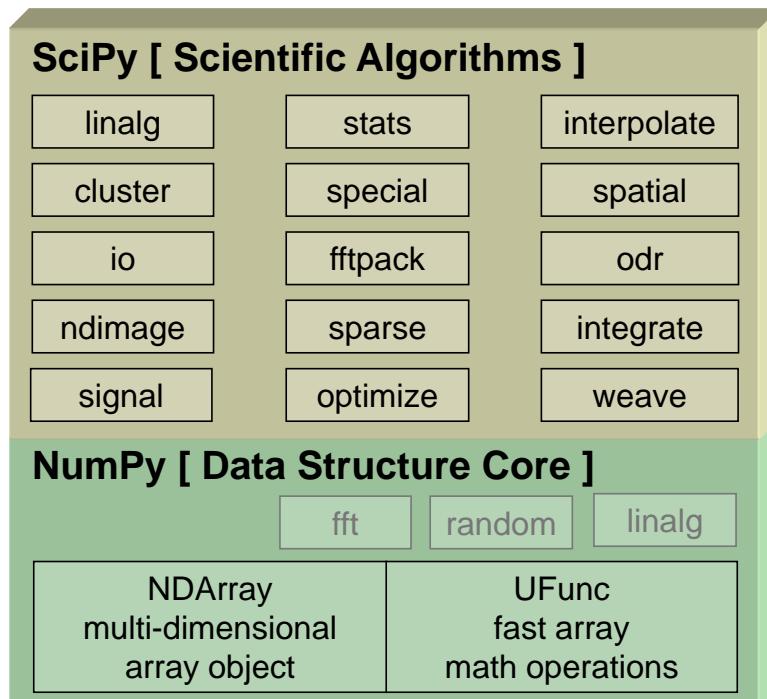


Visualization  
and GUI  
building

Array data  
structures

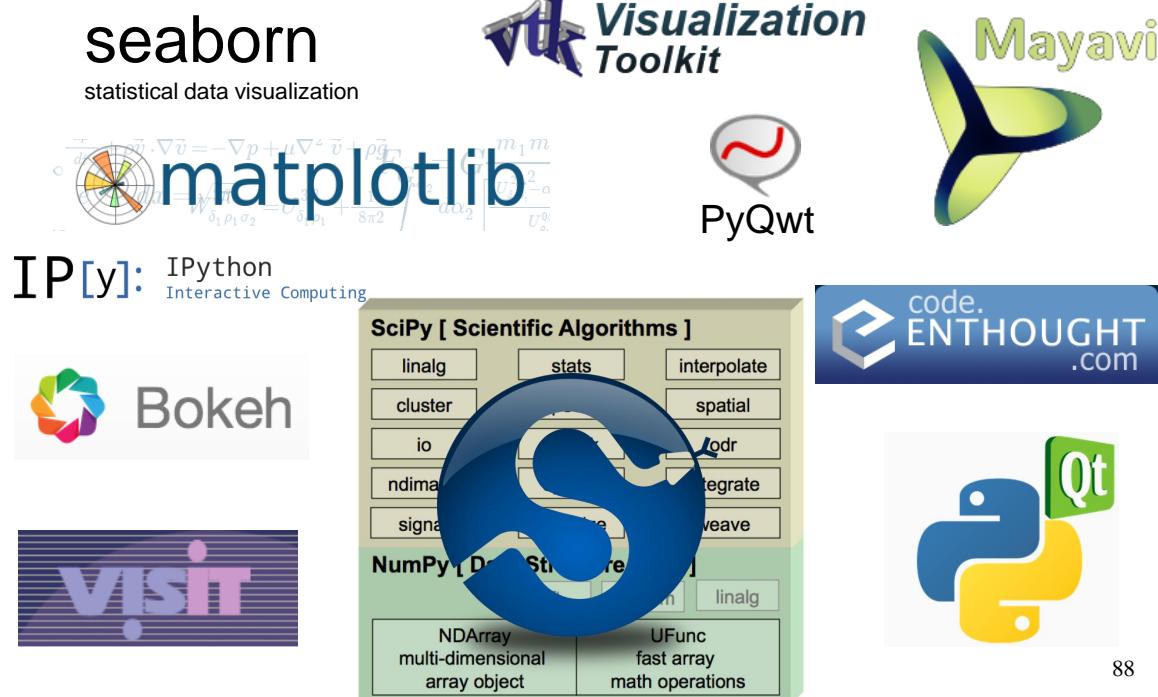
86

# Core libraries: NumPy and SciPy



87

# Visualization



88

# Algorithms

The collage includes logos for:

- SfePy
- pyTrilinos
- PyMC
- statsmodels
- OpenCV
- FENICS PROJECT
- scikit-image
- pandas
- NetworkX
- scikit-learn

A central box displays the SciPy [Scientific Algorithms] interface, showing modules like linalg, stats, interpolate, cluster, io, ndimage, signal, etc., and NumPy [Data Structure] features like NDArray, UFunc, and math operations.

89

The collage includes logos for:

- HPC
- MPI4Py
- IP[y]: IPython Interactive Computing
- PyCUDA
- SCOOOP
- Cython
- Disco massive data - minimal code
- StarCluster
- llvmPy
- SciPy [Scientific Algorithms]
- NumPy [Data Structure]
- theano
- numexpr
- NUMBA
- pypy

A central box displays the SciPy [Scientific Algorithms] interface, showing modules like linalg, stats, interpolate, cluster, io, ndimage, signal, etc., and NumPy [Data Structure] features like NDArray, UFunc, and math operations.

90

# Before we start...

- Version check for the packages needed in this section:

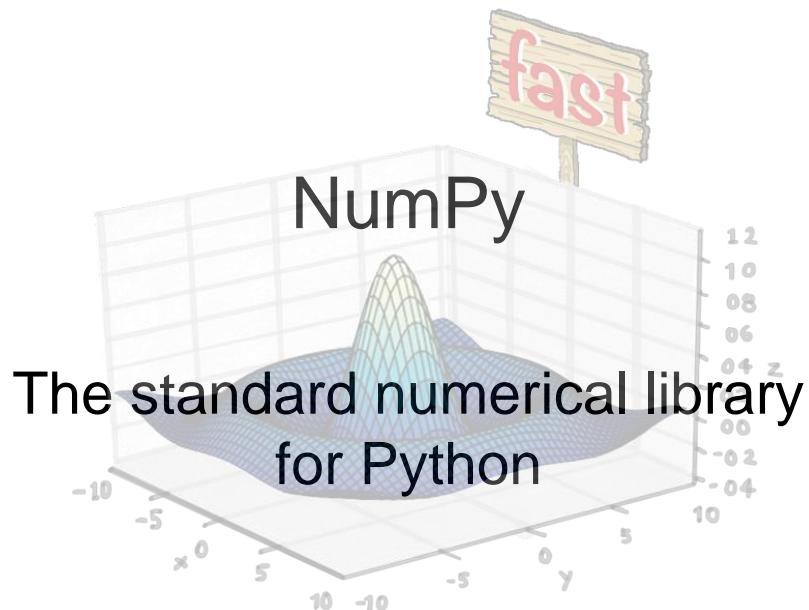
Package name	Version
numpy	> 1.8.1
scipy	> 0.15.1
pandas	> 0.15.2
matplotlib	> 1.4.3

Example:

```
>>> import numpy  
>>> numpy.__version__  
'1.8.1'
```

- Use Canopy's Package Manager to install / update these packages (Tools > Package Manager)

91



92

# NumPy arrays

- Defining arrays
- Indexing and slicing
- Creating arrays
- Array calculations
- The array data structure
- Advanced NumPy, overview

93

# Getting Started

## IMPORT NUMPY

```
In [1]: from numpy import *
In [2]: __version__
Out[2]: 1.8.1
        or
In [1]: from numpy import \
        array, ...
```

Often at the command line, it is handy to import everything from NumPy into the command shell.

However, if you are writing scripts, it is easier for others to read and debug in the future if you use explicit imports.

## USING IPYTHON -PYLAB

```
C:\> ipython --pylab
In [1]: array([1,2,3])
Out[1]: array([1, 2, 3])
```

IPython has a ‘pylab’ mode where it imports all of NumPy and Matplotlib, into the namespace for you as a convenience. It also enables threading for showing plots.



While IPython is used for all the demos, ‘>>>’ is used on future slides instead of ‘In [1]’ to save space.

94

# Introducing NumPy Arrays

## SIMPLE ARRAY CREATION

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

## CHECKING THE TYPE

```
>>> type(a)
numpy.ndarray
```

## NUMERIC 'TYPE' OF ELEMENTS

```
>>> a.dtype
dtype('int32')
```

## NUMBER OF DIMENSIONS

```
>>> a.ndim
1
```

## ARRAY SHAPE

```
# Shape returns a tuple
# listing the length of the
# array along each dimension.
>>> a.shape
(4,)
```

## BYTES PER ELEMENT

```
>>> a.itemsize
4
```

## BYTES OF MEMORY USED

```
# Return the number of bytes
# used by the data portion of
# the array.
>>> a.nbytes
16
```

95

# Array Operations

## SIMPLE ARRAY MATH

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
>>> a * b
array([ 2,  6, 12, 20])
>>> a ** b
array([ 1,  8,  81, 1024])
```



NumPy defines these constants:  
 $\pi = 3.14159265359$   
 $e = 2.71828182846$

## MATH FUNCTIONS

```
# create array from 0 to 10
>>> x = arange(11.)

# multiply entire array by
# scalar value
>>> c = (2*pi)/10.
>>> c
0.62831853071795862
>>> c*x
array([ 0., 0.628,..., 6.283])
```

## # in-place operations

```
>>> x *= c
>>> x
array([ 0., 0.628,..., 6.283])
```

```
# apply functions to array
>>> y = sin(x)
```

96

# Setting Array Elements

## ARRAY INDEXING

```
>>> a[0]
0
>>> a[0] = 10
>>> a
array([10, 1, 2, 3])
```

## BEWARE OF TYPE COERCION

```
>>> a.dtype
dtype('int32')

# assigning a float into
# an int32 array truncates
# the decimal part
>>> a[0] = 10.6
>>> a
array([10, 1, 2, 3])

# fill has the same behavior
>>> a.fill(-4.8)
>>> a
array([-4, -4, -4, -4])
```

97

# Multi-Dimensional Arrays

## MULTI-DIMENSIONAL ARRAYS

```
>>> a = array([[ 0,  1,  2,  3],
              [10,11,12,13]])
>>> a
array([[ 0,  1,  2,  3],
       [10,11,12,13]])
```

## SHAPE = (ROWS,COLUMNS)

```
>>> a.shape
(2, 4)
```

## ELEMENT COUNT

```
>>> a.size
8
```

## NUMBER OF DIMENSIONS

```
>>> a.ndim
2
```

## GET/SET ELEMENTS

```
>>> a[1,3]
13
    ↑
    |
    column
    ↓
    row

>>> a[1,3] = -1
>>> a
array([[ 0,  1,  2,  3],
       [10,11,12, -1]])
```

## ADDRESS SECOND (ONETH) ROW USING SINGLE INDEX

```
>>> a[1]
array([10, 11, 12, -1])
```

98

# Slicing

**var[lower:upper:step]**

Extracts a portion of a sequence by specifying a lower and upper bound.

The lower-bound element is included, but the upper-bound element is **not** included.

Mathematically: [lower, upper). The step value specifies the stride between elements.

## SLICING ARRAYS

```
# indices: 0 1 2 3 4
>>> a = array([10,11,12,13,14])
# [10,11,12,13,14]
>>> a[1:3]
array([11, 12])

# negative indices work also
>>> a[1:-2]
array([11, 12])
>>> a[-4:3]
array([11, 12])
```

## OMMITTING INDICES

```
# omitted boundaries are
# assumed to be the beginning
# (or end) of the list

# grab first three elements
>>> a[:3]
array([10, 11, 12])
# grab last two elements
>>> a[-2:]
array([13, 14])
# every other element
>>> a[::-2]
array([10, 12, 14])
```

99

# Array Slicing

## SLICING WORKS MUCH LIKE STANDARD PYTHON SLICING

```
>>> a[0,3:5]
array([3, 4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])
```

## STRIDES ARE ALSO POSSIBLE

```
>>> a[2::2,:,:2]
array([[20, 22, 24],
       [40, 42, 44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

## Give it a try!

Create the array below with the command

```
a = arange(25).reshape(5, 5)
```

and extract the slices as indicated.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

## Slices Are References

Slices are references to memory in the original array.

Changing values in a slice also changes the original array.

```
>>> a = array((0,1,2,3,4))
          2

# create a slice containing only the
# last element of a
>>> b = a[2:4]
>>> b
array([2, 3])
>>> b[0] = 10

# changing b changed a!
>>> a
array([ 0,  1, 10,  3,  4])
```

# Fancy Indexing

## INDEXING BY POSITION

```
>>> a = arange(0,80,10)

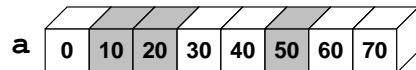
# fancy indexing
>>> indices = [1, 2, -3]
>>> y = a[indices]
>>> print(y)
[10 20 50]
```

## INDEXING WITH BOOLEANS

```
# manual creation of masks
>>> mask = array([0,1,1,0,0,1,0,0],
...                 dtype=bool)

# conditional creation of masks
>>> mask2 = a < 30

# fancy indexing
>>> y = a[mask]
>>> print(y)
[10 20 50]
```



103

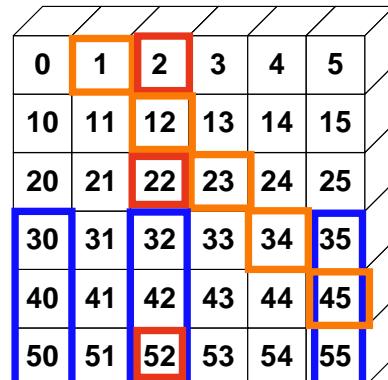
# Fancy Indexing in 2-D

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
...                  dtype=bool)

>>> a[mask,2]
array([2,22,52])
```



Unlike slicing, fancy indexing creates copies instead of a view into original array.

104

# Give it a try!

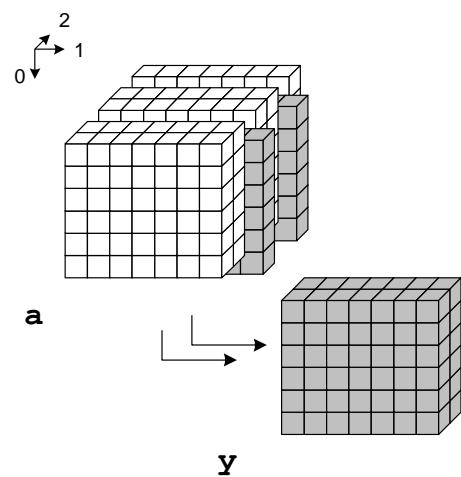
1. Create the array below with  
 $a = \text{arange}(25).reshape(5, 5)$   
 and extract the elements indicated in blue.
2. Extract all the numbers divisible by 3 using a boolean mask.

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

## 3D Example

### MULTIDIMENSIONAL

```
# retrieve two slices from a
# 3D cube via indexing
>>> y = a[:, :, [2, -2]]
```



# Creating arrays

107

## Array Constructor Examples

### FLOATING POINT ARRAYS

```
# Default to double precision
>>> a = array([0,1.0,2,3])
>>> a.dtype
dtype('float64')
>>> a nbytes
32
```

### UNSIGNED INTEGER BYTE

```
>>> a = array([0,1,2,3],
...             dtype=uint8)
>>> a.dtype
dtype('uint8')
>>> a nbytes
4
```

### REDUCING PRECISION

```
>>> a = array([0,1.,2,3],
...             dtype=float32)
>>> a.dtype
dtype('float32')
>>> a nbytes
16
```

108

# Array Creation Functions

## ARANGE

```
arange(start, stop=None, step=1,
       dtype=None)
```

Nearly identical to Python's `range()`. Creates an array of values in the range [start,stop) with the specified step value. Allows non-integer values for start, stop, and step. Default `dtype` is derived from the start, stop, and step values.

```
>>> arange(4)
array([0, 1, 2, 3])
>>> arange(0, 2*pi, pi/4)
array([ 0.000,  0.785,  1.571,
       2.356,  3.142,  3.927,  4.712,
       5.497])

# Be careful...
>>> arange(1.5, 2.1, 0.3)
array([ 1.5,  1.8,  2.1])
```

## ONES, ZEROS

```
ones(shape, dtype=float64)
zeros(shape, dtype=float64)
```

`shape` is a number or sequence specifying the dimensions of the array. If `dtype` is not specified, it defaults to `float64`.

```
>>> ones((2,3), dtype=float32)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]],

       dtype=float32)
>>> zeros(3)
array([ 0.,  0.,  0.])
```

109

# Array Creation Functions (cont.)

## IDENTITY

```
# Generate an n by n identity
# array. The default dtype is
# float64.
>>> a = identity(4)
>>> a
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
>>> a.dtype
dtype('float64')
>>> identity(4, dtype=int)
array([[ 1,  0,  0,  0],
       [ 0,  1,  0,  0],
       [ 0,  0,  1,  0],
       [ 0,  0,  0,  1]])
```

## EMPTY AND FILL

```
# empty(shape, dtype=float64,
#       order='C')
>>> a = empty(2)
>>> a
array([1.78021120e-306,
       6.95357225e-308])

# fill array with 5.0
>>> a.fill(5.0)
array([5.,  5.])

# alternative approach
# (slightly slower)
>>> a[:] = 4.0
array([4.,  4.])
```

110

# Array Creation Functions (cont.)

## LINSPACE

```
# Generate N evenly spaced
# elements between (and
# including) start and
# stop values.
>>> linspace(0,1,5)
array([0., 0.25, 0.5, 0.75, 1.0])
```

## LOGSPACE

```
# Generate N evenly spaced
# elements on a log scale
# between base**start and
# base**stop (default base=10).
>>> logspace(0,1,5)
array([ 1., 1.77, 3.16, 5.62,
       10.])
```

## ARRAYS FROM/TO TXT FILES

### Data.txt

```
-- BEGINNING OF THE FILE
% Day, Month, Year, Skip, Avg Power
01, 01, 2000, x876, 13 % crazy day!
% we don't have Jan 03rd
04, 01, 2000, xfed, 55
```

```
# loadtxt() automatically generates
# an array from the txt file
arr = loadtxt('Data.txt', skiprows=1,
              dtype=int, delimiter=",",
              usecols = (0,1,2,4),
              comments = "%")

# Save an array into a txt file
savetxt('filename', arr)
```

111

# Array calculation methods

112

# Array Calculation Methods

## SUM FUNCTION

```
>>> a = array([[1,2,3],
   [4,5,6]])

# sum() defaults to adding up
# all the values in an array.
>>> sum(a)
21

# supply the keyword axis to
# sum along the 0th axis
>>> sum(a, axis=0)
array([5, 7, 9])

# supply the keyword axis to
# sum along the last axis
>>> sum(a, axis=-1)
array([ 6, 15])
```

## SUM ARRAY METHOD

```
# a.sum() defaults to adding
# up all values in an array.
>>> a.sum()
21

# supply an axis argument to
# sum along a specific axis
>>> a.sum(axis=0)
array([5, 7, 9])
```

## PRODUCT

```
# product along columns
>>> a.prod(axis=0)
array([ 4, 10, 18])

# as a function
>>> prod(a, axis=0)
array([ 4, 10, 18])
```

113

# Min/Max

## MIN

```
>>> a = array([2.,3.,0.,1.])
>>> a.min(axis=0)
0.0

# Use NumPy's amin() instead
# of Python's built-in min()
# for speedy operations on
# multi-dimensional arrays.
>>> amin(a, axis=0)
0.0
```

## ARGMIN

```
# Find index of minimum value.
>>> a.argmax(axis=0)
2

# as a function
>>> argmin(a, axis=0)
2
```

## MAX

```
>>> a = array([2.,3.,0.,1.])
>>> a.max(axis=0)
3.0
```

## # as a function

```
>>> amax(a, axis=0)
3.0
```

## ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1

# as a function
>>> argmax(a, axis=0)
1
```

114

# Statistics Array Methods

## MEAN

```
>>> a = array([[1,2,3],
   [4,5,6]])

# mean value of each column
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])

# average can also calculate
# a weighted average
>>> average(a, weights=[1,2],
 ...           axis=0)
array([ 3.,  4.,  5.])
```

## STANDARD DEV./VARIANCE

```
# Standard Deviation
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])

# variance
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

115

# Give it a try!

Create the array below with

```
arange(-12, 13).reshape(5, 5) ** 2
```

and compute:

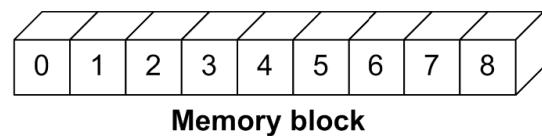
1. The maximum of each row (one max per row)
2. The mean of each column (one mean per column)
3. The position of the overall minimum (requires 2-3 steps)

144	121	100	81	64
49	36	25	16	9
4	1	0	1	4
9	16	25	36	49
64	81	100	121	144

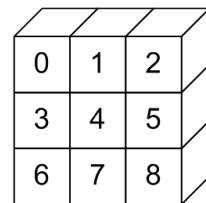
# The array data structure

117

## Array Data Structure

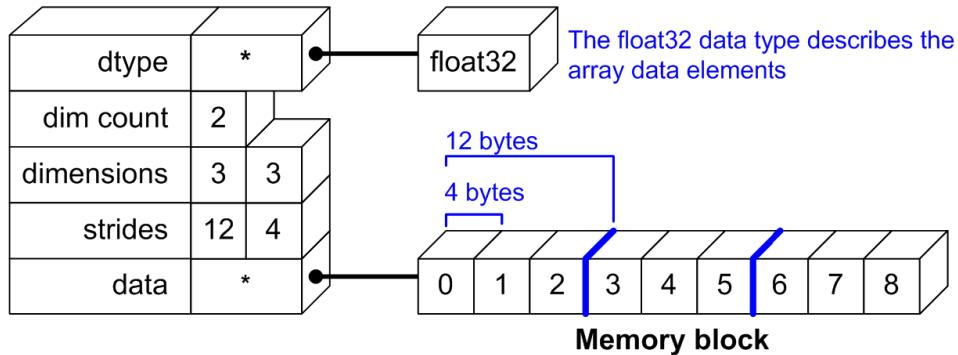


Python View:



118

# Array Data Structure

**NDArray Data Structure****Python View:**

0	1	2
3	4	5
6	7	8

119

# Operations on the array structure

Operations that only affect the array structure, not the data, can be executed without copying memory.

120

# Transpose

## TRANSPOSE

```
>>> a = array([[0,1,2],
...             [3,4,5]])
>>> a.shape
(2,3)
# Transpose swaps the order
# of axes.
>>> a.T
array([[0, 3],
       [1, 4],
       [2, 5]])
>>> a.T.shape
(3,2)
```

## TRANSPOSE RETURNS VIEWS

```
# Transpose does not move
# values around in memory. It
# only changes the order of
# "strides" in the array
>>> a.strides
(12, 4)

>>> a.T.strides
(4, 12)
```

121

# Reshaping Arrays

## RESHAPE

```
>>> a = array([[0,1,2],
...             [3,4,5]])

# Return a new array with a
# different shape (a view
# where possible)
>>> a.reshape(3,2)
array([[0, 1],
       [2, 3],
       [4, 5]])

# Reshape cannot change the
# number of elements in an
# array
>>> a.reshape(4,2)
ValueError: total size of new
array must be unchanged
```

## SHAPE

```
>>> a = arange(6)
>>> a
array([0, 1, 2, 3, 4, 5])
>>> a.shape
(6,)

# Reshape array in-place to
# 2x3
>>> a.shape = (2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
```

122

# Flattening Arrays

## FLATTEN (SAFE)

`a.flatten()` converts a multi-dimensional array into a 1-D array. The new array is a *copy* of the original data.

```
# Create a 2D array
>>> a = array([[0,1],
   [2,3]])

# Flatten out elements to 1D
>>> b = a.flatten()
>>> b
array([0,1,2,3])

# Changing b does not change a
>>> b[0] = 10
>>> b
array([10,1,2,3])
>>> a
array([[0, 1],
       [2, 3]])
```

no change

## RAVEL (EFFICIENT)

`a.ravel()` is the same as `a.flatten()`, but returns a *reference* (or *view*) of the array if possible (i.e., the memory is contiguous). Otherwise the new array copies the data.

```
# Flatten out elements to 1-D
>>> b = a.ravel()
>>> b
array([0,1,2,3])

# Changing b does change a
>>> b[0] = 10
>>> b
array([10,1,2,3])
>>> a
array([[10, 1],
       [2, 3]])
```

changed!

123

# Advanced NumPy overview

NumPy is the low-level core of most Python Data Science libraries.

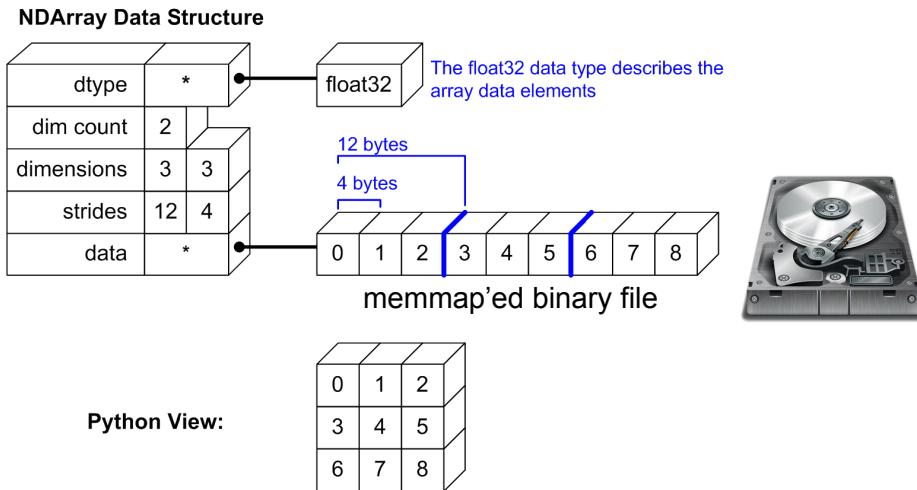
There are a couple of advanced NumPy topics that it's worth being aware of:

- memmap'ed arrays
- structured arrays

124

# memmap'ed arrays

The array data can come from any buffer-like storage, including memmap'ed files. Users can use the array object transparently, and the OS creates memory pages as needed.



125

# Structured arrays

Structured arrays allow interpreting the array elements as fields of multiple types. Combined with memmaps, it increases the opportunities for creating disk-backed arrays from binary files.

Elements of an array can be any fixed-size data structure!

<b>name</b>	<b>char[10]</b>
<b>age</b>	<b>int</b>
<b>weight</b>	<b>double</b>

Brad	Jane	John	Fred
33	25	47	54
135.0	105.0	225.0	140.0
Henry	George	Brian	Amy
29	61	32	27
154.0	202.0	137.0	187.0
Ron	Susan	Jennifer	Jill
19	33	18	54
188.0	135.0	88.0	145.0

## EXAMPLE

```
>>> from numpy import dtype, empty
# structured data format
>>> fmt = dtype([('name', 'S10'),
   ... ('age', int),
   ... ('weight', float)
   ... ])
>>> a = empty((3,4), dtype=fmt)
```

126

# Further learning online

Additional material and exercises available online:

**Array Broadcasting**

```
a = array ([0,10,20,30])
b = array ([0,1,2])
y = a[:, newaxis] + b
```

**Array Fields and Nested dtypes**

**Memory Maps**

```
>>> image = memmap('some_file.dat',
                   dtype='uint16',
                   mode='r+',
                   shape=(5,5),
                   offset=header_size)
```

**Reading Binary Files**

```
Name      Time   Value
chak[12].int64 float
MSFT     10    6.20
GOOG    12    -1.08
MSFT_profit 18    8.40
INTC    25    -0.20
...       ...    ...
disk
MSFT_profit 10    6.20  GOOG_profit 12    -1.08  ...
```



127

# SciPy overview



128

# SciPy content and the scikits

## CURRENT PACKAGES

- Special Functions (scipy.special)
- Signal Processing (scipy.signal)
- Image Processing (scipy.ndimage)
- Fourier Transforms (scipy.fftpack)
- Optimization (scipy.optimize)
- Numerical Integration (scipy.integrate)
- Linear Algebra (scipy.linalg)
- Input/Output (scipy.io)
- Statistics (scipy.stats)
- Clustering Algorithms (scipy.cluster)
- Sparse Matrices (scipy.sparse)
- Interpolation (scipy.interpolate)
- ... and more.

## SCIKITS

SciPy's "all in one" philosophy has been replaced by "scikits" projects:

- scikits.learn (sklearn), for Machine Learning functionality
- scikits.image, for image processing features
- statsmodel, for classical statistics (models, tests, etc)

129

# Further learning online

Additional material and exercises available online:

The grid contains six modules:

- Curve Fitting: Introduction**: Shows a scatter plot of data points and a fitted curve.
- Minimization: Part 1**: Shows a graph of a parabolic function with a red dot at its peak labeled "Maximum Possible Height".
- Interpolation**: Shows a plot of raw data points and their corresponding interpolated values.
- ODE Integrate**: Shows a diagram of a falling object with forces labeled: Drag Force  $D = f(v)$ , Gravitational Force  $mg$ , and air resistance  $\frac{Dx}{dt}$ .
- Integration: Symbolic**: Shows the formula  $F(x) = \int_0^x f(\theta) d\theta$ .
- SymPy**: Shows the SymPy logo.



# Helpful Sites

## SCIPY DOCUMENTATION PAGE

<http://docs.scipy.org/doc>

SciPy.org » Numpy and Scipy Documentation »



### Numpy and Scipy Documentation

Welcome! This is the documentation for Numpy and Scipy .

**For contributors:**

- Write, review and proof the documentation
- Numpy developer guide

**Latest: (development versions)**

- Numpy Reference Guide  
[HTML+zip], [HTML-help (CHM)], [PDF]
- Numpy User Guide (DRAFT)  
[PDF]
- Scipy Reference Guide  
[HTML+zip], [CHM], [PDF]

**Releases:**

- Numpy 1.6 Reference Guide, [HTML+zip], [CHM], [PDF]
- Numpy 1.6 User Guide (DRAFT), [PDF]
- Scipy 0.9.0 Reference Guide, [HTML+zip], [PDF]
- Numpy 1.5 Reference Guide, [HTML+zip], [CHM], [PDF]

Numpy 4.0 Help Guide (DRAFT) None

**See also:**

- SciPy.org  
all things NumPy/SciPy (bug reports, downloads, conferences, etc.)
- Additional documentation  
additional tutorials and other documentation resources
- Cookbook  
user-contributed examples and recipes for common tasks
- Ask Scipy  
Q & A forum
- Mailing Lists  
main discussion channels

## NUMPY EXAMPLES

[http://wiki.scipy.org/Numpy\\_Example\\_List\\_With\\_Doc](http://wiki.scipy.org/Numpy_Example_List_With_Doc)

wiki
SciPy
Documentation
Mailing Lists
Download
Installing SciPy
Topical Software
Cookbook
Developer Zone
RecentChanges
FindPage
...

### Numpy Example List Wiki

This is an auto-generated version of [Numpy Examples](#).  
Contents

1. ...
2. [ ]
3. T
4. abs()
5. absolute()
6. accumulate
7. add()

#### apply\_along\_axis()

`numpy.apply_along_axis(func1d, axis, arr, *args)`

Execute `func1d(arr[i],*args)` where `func1d` takes 1-D arrays and `arr` is an N-d array. `i` varies so as to apply the function along the given axis for each 1-d subarray in `arr`.

#### Example:

```
>>> from numpy import *
>>> def myfunc(a):
...     return (a[0]+a[-1])/2
...
>>> b = array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> apply_along_axis(myfunc, 0, b)
array([4, 5, 6])
>>> apply_along_axis(myfunc, 1, b)
# apply myfunc to each row
array([2, 5, 8])
```

131

# Matplotlib Basics



132

# Matplotlib “State Machine”

## State information for Figure

columns: 2

rows: 2

active subplot: 2

hold: True

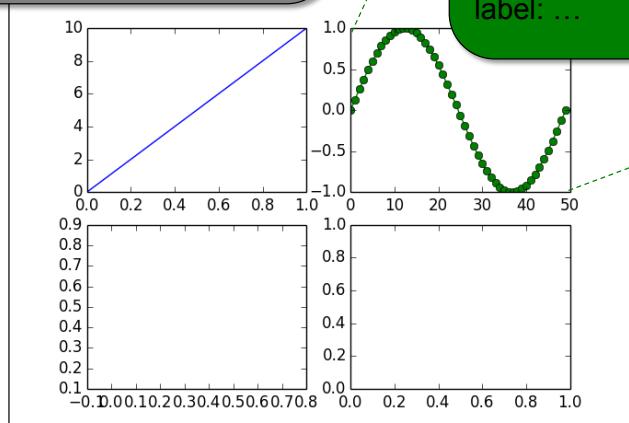
## State information for subplot 2

color: green

x-axis ticks: ...

y-axis ticks: ...

label: ...



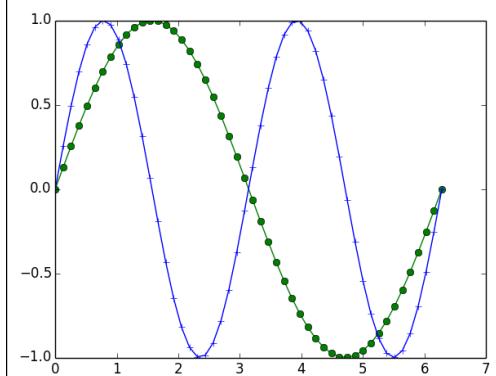
133

# Line Plots

## LINE PLOTS

```
>>> x = linspace(0,2*pi,50)
>>> y1 = sin(x)
>>> y2 = sin(2*x)

>>> figure() # Create figure
>>> hold(False)
>>> plot(y1)
>>> plot(x, y1)
>>> # red dot-dash circle
>>> plot(x, y1, 'r-o')
>>> # red marker only circle
>>> plot(x, y1, 'ro')
>>> plot(x, y1, 'g-o',
        x, y2, 'b-+')
>>> clf() # Clear plot
```



Symbol	Color
b	Blue
g	Green
r	Red
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

Symbol	Marker
.	Point
o	Circle
< > ^ v	Triangles
8	Octagon
s	Square
*	Star
+	Plus
	...and more

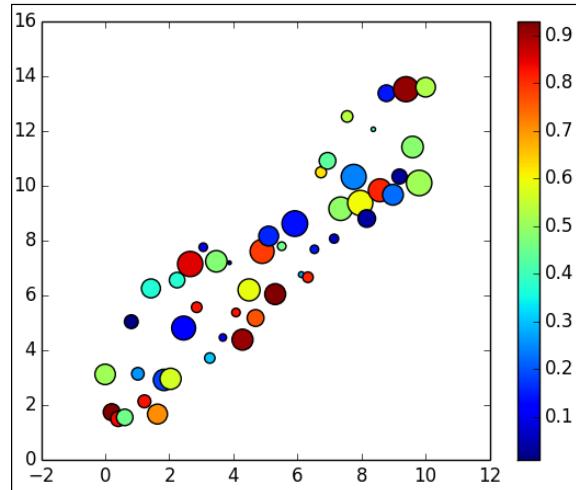
134

# Scatter Plots

## SCATTER PLOTS

```
>>> N = 50 # no. of points
>>> x = linspace(0,10,N)
>>> e = rand(N)*5.0 # noise
>>> y1 = x + e

>>> figure() # Create figure
>>> areas = rand(N)*300
>>> scatter(x, y1, s=areas)
>>> hold(False) # overwrite
>>> colors = rand(N)
>>> scatter(x, y1, c=colors)
>>> colorbar()
```



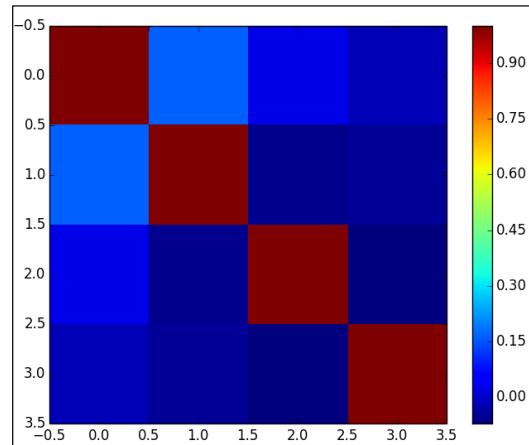
135

# Image “Plots”

## IMAGE PLOTS

```
>>> # Create some data
>>> e1 = rand(100)
>>> e2 = rand(100)*2
>>> e3 = rand(100)*10
>>> e4 = rand(100)*100
>>> corrmatrix = \
...     corrcoef([e1, e2, e3, e4])

>>> # Plot corr matrix as image
>>> imshow(corrmatrix,
... interpolation='nearest')
>>> colorbar()
```



136

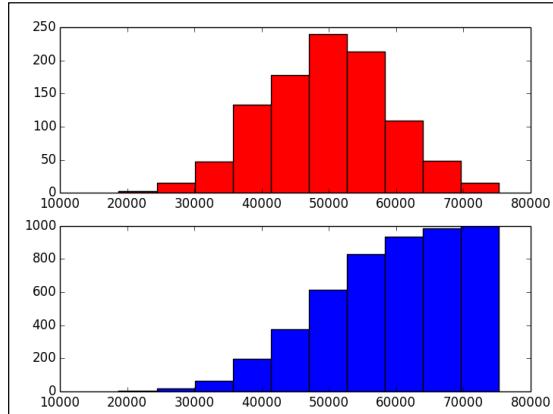
# Histogram Plots

## HISTOGRAM PLOTS

```
>>> # Create array of data
>>> data = randint(10000,
...      size=(10,1000))
>>> # Approx norm distribution
>>> x = sum(data, axis=0)

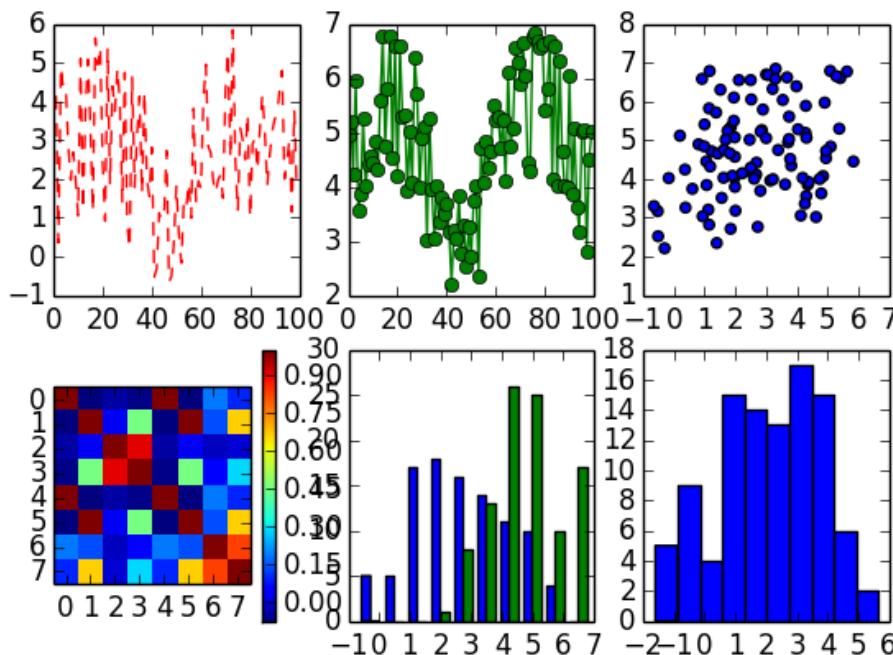
>>> # Set up for stacked plots
>>> subplot(2,1,1)
>>> hist(x, color='r')
>>> # Plot cumulative dist
>>> subplot(2,1,2)
>>> hist(x, cumulative=True)

>>> # For multiple histograms:
>>> hist([d1, d2, ...])
```



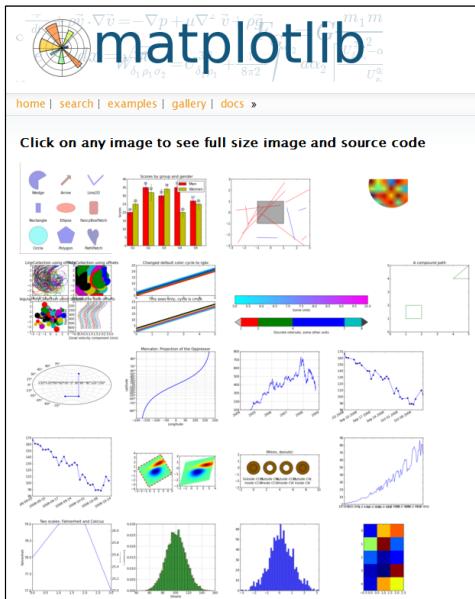
137

# MPL Exercise: Desired Output



138

# Additional Matplotlib Resources



- Simple examples with increasing difficulty  
<http://matplotlib.org/examples/index.html>
- Gallery (huge)  
<http://matplotlib.org/gallery.html>
- See appendix for reference materials

139

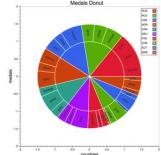
# Further learning online

Additional material and exercises available online:

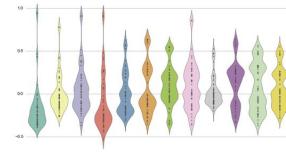
The screenshot shows a video thumbnail titled 'Matplotlib Basics'. The thumbnail features a large donut chart with several colored segments and handwritten annotations like 'dt', 'e', and 'dx'. To the right of the chart is a grayscale image of a human brain. Below the brain image is a histogram labeled 'MRI density' with values ranging from -1.0 to 1.0. At the bottom right is a line graph with four traces labeled PG9, PG7, PG5, and PG3, each showing a different signal over time from 0 to 9.

# Other visualization libraries

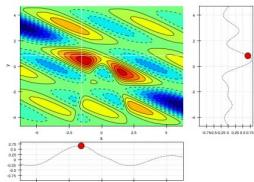
**Seaborn:** Better looking, high-level plots based on matplotlib



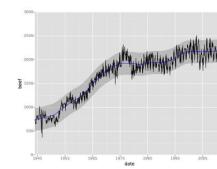
**bokeh:** Library for D3-like visualization in web browsers (HTML file, or inline in IPython notebooks)



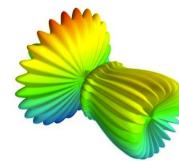
**yhat/ggplot:** ggplot implementation for Python



**chaco:** For interactive, custom plots embedded in user applications



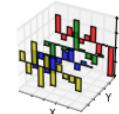
**mayavi:** 3D visualization, based on VTK



141

# Data Analysis with pandas

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



142

# Pandas

Pandas is a library that makes the analysis of complex, tabular datasets *easy*.

## FEATURES

- Defines **tabular data types**: database-like tables, with labelled rows and columns;
- **Data consolidation and data integration**: remove duplicates, clean data, manage missing values; automatically align tables by index;
- **Summarization**: create “pivot” tables;
- **In-memory, SQL-like operations**: join, aggregate (group by);
- Very flexible **import/export** of data;
- **Date and time** handling built-in, including timezones;
- **Easy visualization** based on Matplotlib.

143

# Storing data in Pandas

144

# New Data Structures

PANDA = PANel DAta = multi-dimensional data in stats & econometrics.

Introduces 3 size-mutable, labeled data-structures:

- A `Series` is a 1D data-structure.
- A `DataFrame` is a 2D data-structure that can be viewed as a dictionary of `Series`.
- A `Panel` is a 3D data-structure that can be viewed as a dictionary of `DataFrames`.

	Rk	Fst N	Lst N	DoB	Gend	Grade	offset
UT	TAMU						
1	4	John	Doe	1981	M	4.18	-1.18
2	2	Jill	Ford	1975	F	5.26	1.26
3	5	Chris	Jones	NaN	M	3.91	0.91
4	6	Dave	Smith	1965	M	1.23	NaN
5	1	Ellen	Frank	1973	F	5.52	0.52
6	3	Frank	Hart	1976	M	4.71	-0.71

145

# Series

## Definition

Conceptually, `pandas.Series` are indexed arrays:

- NumPy arrays map a range of integers to values
- Series map arbitrary sets of labels to values
- Series may also be seen as a specialized, ordered dictionary where values all have the same type and are stored efficiently

```
>>> from pandas import *
>>> s = Series({'a':0,'b':1,'c':2,'d':3})
# Dict-like access can be label-based
>>> s['b']
1
```

The labels are accessed via the `s.index` attribute and the values by the `s.values` attribute (NumPy array).

146

# Creating Series

## FROM LIST AND DICT

```
# Data and corresponding indices can
# be stored in lists.
>>> index = ['a', 'b', 'c', 'd']
>>> Series(range(4), index=index,
           name='first series')
a    0
b    1
c    2
d    3
Name: first series
# data + indices in a dict
>>> d = {'a':0,'b':1,'c':2,'d':3}
>>> s = Series(d, name='first series')
>>> s.index
Index([a, b, c, d], dtype=object)
>>> s.values, type(s.values)
array([0, 1, 2, 3], dtype=int64)
numpy.ndarray
>>> s.dtype
dtype('int64')
```

## FROM A NUMPY ARRAY

```
>>> from numpy.random import randn
>>> Series(randn(4), index=index)
a   -1.062984
b   -0.961625
c   -0.720323
d    0.336753
```

## ACCESS OR ADD ELEMENTS

```
# Request existing values
>>> s['b']
1
# Modify an existing value
>>> s['b'] = 3
# Add new elements
>>> s['e'] = 5
>>> s
a    0
b    3
c    2
d    3
e    5
```

147

# Dataframes

## DEFINITION

A DataFrame object can be viewed as a dictionary of Series sharing a common index:

- Dataframes have both row (`index`) and column (`columns`) indices
- Each column may have a different type
- Adding a column is ‘cheap’

```
>>> s1 = Series({'a': 1, 'b': 2, 'c': 3})
>>> s2 = Series({'a': True, 'b': False, 'c': True})
>>> df = DataFrame({'col1': s1, 'col2': s2})
# Dict-like access is column-based
>>> df['col1']
a    1
b    2
c    3
Name: col1, dtype: int64
```

148

# Creating DataFrames

## FROM A DICT OF SERIES

```
# DF from a dict of series: keys are
# column names.
>>> s2=Series([-0.9, -1.7, 1.1],
             index=index[1:])
>>> d = {'A':s, 'B':s2}
>>> df = DataFrame(d)
      A      B
a  0   NaN
b  1  -0.9
c  2  -1.7
d  3   1.1
>>> df.index, df.columns
Index([a, b, c, d], dtype=object)
Index([A, B], dtype=object)
>>> df.shape, df.dtypes
(4, 2)
A          int64
B          float64
>>> df.values
array([[ 0. ,  -nan],
       [ 1. , -0.9],
       [ 2. , -1.7],
       [ 3. ,  1.1]])
```

## FROM A NUMPY ARRAY

```
>>> DataFrame(randn(4,4), index=index,
             columns=['A','B','C','D'])
           A         B         C         D
a  0.28164 -0.36826  0.04011  1.25030
b -0.71049 -1.23956 -0.08504 -0.08336
c -1.29446  0.70709  1.39642  0.49035
d  0.74632 -0.03512 -0.69237  0.81488
```

## ACCESS OR ADD COLUMNS

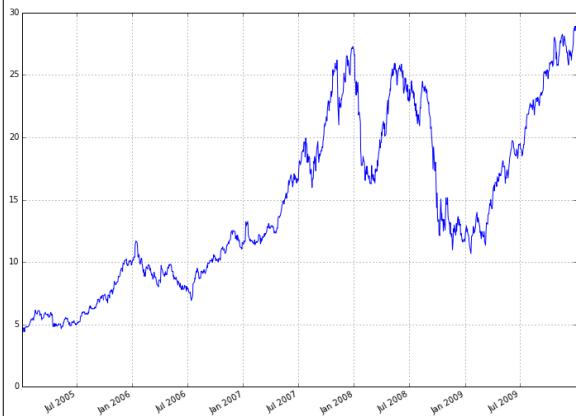
```
# Columns accessed like a dict...
>>> col1 = df['A']
# Create a new column
>>> df['Flag'] = df['B'] > 0
>>> df
      A      B     Flag
a  0   NaN  False
b  1  -0.9  False
c  2  -1.7  False
d  3   1.1   True
```

149

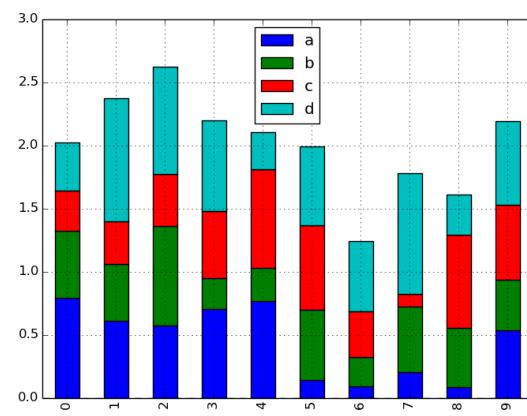
# Easy-to-use visualization

Pandas provides easy-to-use visualization methods.

```
>>> ts = df['AAPL']
>>> ts.plot()
```



```
>>> df2 = DataFrame(rand(10,4),
                  columns=list('abcd'))
>>> df2.plot(kind='bar',
             stacked=True)
```



See the `pandas_plotting/ demo`.

150

# Accessing values in Pandas: Indexing

151

## Pandas and indexing

Series and DataFrames have powerful indexing capabilities:

- Values are accessible as NumPy arrays
- More interestingly: label-based indexing
- Indices allow automatic alignment: especially interesting with timeseries, and for NaN (missing data) handling (more on this later)

Essentially:

- Series[label] -> scalar
- Dataframe[label] -> column

```
>>> s = Series({'a': 0, 'b': 1,
   ...: 'c': 2})
>>> s['a']
0
>>> df = DataFrame({'A': s, 'B': -s})      # BUT if you do slicing
>>> df['A']
a    0
b    1
c    2
      ...:               A    B
      ...:      a    0    0
      ...:      b    1   -1
```

152

# Pandas and indexing (Cont.)

## LABEL-BASED VS POSITION-BASED INDEXING

Indexing operator [] has an ambiguity:

- Series[integer\_value]: position or label?
- DataFrame[integer\_value]: position or column name ?

API is a bit messy here, greatly improved in versions >= 0.11.0:

- .loc attribute: purely “label”
- .iloc attribute: purely index-based, aka position (integer value)

```
>>> s = Series({'a': 0, 'b': 1, 'c': 2})
>>> s.iloc[1]
1
>>> s.iloc['a']
TypeError: the label [a] is not a proper indexer for this index ...
>>> s.loc['a']
0
>>> s.loc[0]
KeyError: 'the label [0] is not in the [index]'
```

153

# Indexing into Series

## ACCESSING 1 ELEMENT

```
>>> index = ['a', 'b', 'c', 'd']
>>> s = Series(range(4), index=index)
# Access elements based on position
>>> s.iloc[2]
2
# Access elements based on label
>>> s.loc['c']
2
# Indexing into a Series is equivalent
>>> s['c']
2
```

Form: s.loc[label\_lower:label\_upper:step]

```
>>> s.loc['a':'c']
a    0
b    1
c    2 # upper limit included!
```



## FANCY-INDEXING

```
# Custom selection of elements
>>> s[[True, False, True, True]]
a    0
c    2
d    3
```

# Masks can be created by comparing  
# values in the Series or another one

```
>>> s>1
a    False
b    False
c    True
d    True
>>> s[s>1]
c    2
d    3
```

## SLICING ELEMENTS OUT

Form: s.iloc[pos\_lower:pos\_upper:step]

```
>>> s.iloc[:2]
a    0
b    1
# Every other element
>>> s.iloc[::2]
a    0
c    2
```

154

# Indexing into DataFrames

## ACCESS ELEMENTS

```
>>> df
      A      B
a  0    NaN
b  1   -0.9
c  2   -1.7
d  3   1.1
# 1 (or more) column accessed like a
# dict...
>>> df['A']
a  0
b  1
c  2
d  3
... or like an object
>>> series2 = df.B
# Access all columns for 1 index
>>> df.loc['c']
A          2
B         -1.7
Name: c, dtype: float64
# or 1 element of the table
>>> df.loc['c','B']
-1.7
```

## SLICING ELEMENTS OUT

Form: `s.loc[row lower:row upper:step,  
 col lower:col upper:step]`

```
>>> sub_df = df.loc["c":, "A":"B"]

# Incomplete slicing assumes all
# elements in other dimensions.
>>> df.loc["c"]:
```

A	B
2	-1.7
3	1.1

## MIXED INDEXING

Mixed indexing using `.ix`:

```
>>> sub_df = df.ix[2, "B"]
-1.7
```

155

# Give it a try!

Create a simple DataFrame:

```
import pandas as pd
import numpy as np
data = np.arange(12).reshape(4, 3)
df = pd.DataFrame(data,
                   index=['one', 'two', 'three', 'four'],
                   columns=['X', 'Y', 'Z'])
```

1. Get column 'Y'
2. Get row 'three' (by name)
3. Get the second and fourth row (by index)
4. Get the columns 'Y' and 'Z' of rows 'two' and 'three'
5. Plot the data as a box plot (`kind='box'`)

# Re-indexing

The index of a Pandas data-structure is the key that controls:

- how the data is displayed and ordered,
- how to align and combine different datasets.

The index can be:

- shuffled (and the values will follow),
- overwritten,
- transformed,
- set to the values of any of the columns of a DataFrame ,
- made of multiple sub-indices.

157

# Re-indexing Series

## RE-INDEXING

```
>>> index = ['a', 'b', 'c', 'd']
>>> s = Series(range(4), index=index)

# Select a different set of indices
>>> s.reindex(['c', 'b', 'a', 'e'])
c    2
b    1
a    0
e    NaN

# Sort by values. See s.sort_index()
# to sort based on index value.
>>> s.order(ascending=False)
d    3
c    2
b    1
a    0
```

## ALIGNMENT OF 2 SERIES

```
>>> s = Series(range(4), index=index)
>>> s2 = s.iloc[:-2]

>>> s2
a    0
b    1

# Operations automatically align on
# the index (different from NumPy)
>>> s + s2
a    0
b    2
c    NaN
d    NaN
```

158

# Re-indexing DataFrames

## RE-INDEXING DATAFRAMES

```
>>> df
      A    B   flags
a  0  NaN  False
b  1 -0.9  False
c  2 -1.7  False
d  3  1.1   True
>>> df.reindex(['c', 'a', 'b'])
      A    B   flags
c  2 -1.7  False
a  0  NaN  False
b  1 -0.9  False
# Sort a DF by a (list of) column(s)
>>> df.sort('B')
      A    B   flags
c  2 -1.7  False
b  1 -0.9  False
d  3  1.1   True
a  0  NaN  False
```

## INDEX TO/FROM A COLUMN

```
# Set dataframe column as index
>>> df2 = df.set_index('A')
>>> df2
      B   flags
A
0  NaN  False
1 -0.9  False
2 -1.7  False
3  1.1   True
# Opposite operation
>>> df2.reset_index()
      A    B   flags
0  0  NaN  False
1  1 -0.9  False
2  2 -1.7  False
3  3  1.1   True
```

159

# Dealing with date & time

## CREATING DATE/TIME INDEXES

```
# The index can be a list of
# dates+times locations that can be
# automatically generated
>>> date_range('1/1/2000', periods=4)
<class
'pandas.tseries.index.DatetimeIndex'>
[2000-01-01 00:00:00, ..., 2000-01-04
00:00:00]
Length: 4, Freq: D, Timezone: None
# Specify frequency: us,ms,S,T,H,D,B,
# W,M,3min, 2h20min, 2W, ...
>>> r=date_range('1/1/2000', periods=72,
...     freq='H')
>>> i=date_range('1/1/2000', periods=4,
...     freq=datetools.YearEnd())
>>> i=date_range('1/1/2000', periods=4,
...     freq='3min')
>>> ts=Series(range(4), index=i)
2000-01-01 00:00:00    0
2000-01-01 00:03:00    1
2000-01-01 00:06:00    2
2000-01-01 00:09:00    3
Freq: 3T
```

## UP/DOWN-SAMPLING

```
>>> ts.resample('T')
2000-01-01 00:00:00    0
2000-01-01 00:01:00    NaN
2000-01-01 00:02:00    NaN
2000-01-01 00:03:00    1
2000-01-01 00:04:00    NaN
2000-01-01 00:05:00    NaN
2000-01-01 00:06:00    2
2000-01-01 00:07:00    NaN
2000-01-01 00:08:00    NaN
2000-01-01 00:09:00    3
Freq: T
# Group hourly data into daily
>>> ts2 = Series(randn(72), index=r)
>>> ts2.resample('D', how='mean',
...     closed='left', label='left')
01-Jan-2000    0.397501
02-Jan-2000    0.186568
03-Jan-2000    0.327240
Freq: D
```

160

# Dealing with date & time II

## TIME ALIGNMENT

```
# Data alignment based on time is one
# of Panda's most celebrated features
>>> daily = date_range('2000-01-01',
...     freq='D', periods=5)
>>> df = DataFrame(random.rand(5),
...     index=daily, columns=['A'])
>>> df
          A
2000-01-01  0.954140
2000-01-02  0.511243
2000-01-03  0.979188
2000-01-04  0.793727
2000-01-05  0.238190
>>> bidaily = pd.date_range('2000-01-01',
...     freq='2D', periods=3)
>>> df2 = pd.DataFrame(np.random.rand(3),
...     index=bidaily, columns=['B'])
>>> df2
          B
2000-01-01  0.007215
2000-01-03  0.797108
2000-01-05  0.440173
```

## TIME ALIGNMENT (cont.)

```
>>> concat([df, df2], axis=1)
          A      B
2000-01-01  0.954140  0.007215
2000-01-02  0.511243    NaN
2000-01-03  0.979188  0.797108
2000-01-04  0.793727    NaN
2000-01-05  0.238190  0.440173
```

161

# Give it a try!

Download the Apple stock prices from 2010:

```
import pandas.io.data as web
aapl =
web.get_data_yahoo('AAPL', '1/1/2010')
```

1. Print the data from the last 4 weeks  
(see the `.last` method)
  
2. Extract the adjusted close column (“Adj Close”),  
resample the full data to a monthly period and plot. Do  
this 3 times, using the min, max, and mean of the  
resampling window.

# Dealing with missing data

163

# Dealing with missing data

## PANDAS PHILOSOPHY

- To signal a missing value, Pandas stores a NaN (Not a Number) value defined in NumPy (`np.nan`).
- Unlike other packages (like NumPy), most operators in Pandas will ignore NaN values in a Pandas datastructure.

```
>>> import numpy as np
>>> a = np.array([1,2,3,np.nan])
>>> a.sum()
nan
>>> s = Series(a)
>>> s.sum()
6
```

164

# Dealing with missing data

## FIND MISSING VALUES

```
>>> df
      s1    s2
a    1    NaN
b    NaN   NaN
c    3    3.5
d    4    4.5
# Boolean mask for all null values:
# np.nan and None .
# Use notnull method for the inverse
>>> df.isnull()
      s1    s2
a  False  True
b  True  True
c  False False
d  False False
```

## REMOVE/REPLACE NaN

```
# Replace missing values manually
>>> df[isnull(df)] = 0.
```

```
# Inverse operation
>>> df[df == 0] = np.nan
# Fill na from previous value
>>> df.fillna(method='ffill')
      s1    s2
a    1    NaN
b    1    NaN
c    3    3.5
d    4    4.5
# Remove all rows w/ missing values
>>> df.dropna(how='all')
      s1    s2
a    1    NaN
c    3    3.5
d    4    4.5
>>> df.dropna(how='any')
      s1    s2
c    3    3.5
d    4    4.5
# Interpolate NaNs away
>>> df.interpolate()
      s1    s2
a    1    NaN
b    2    NaN
c    3    3.5
d    4    4.5
```

165

# Give it a try!

Download the Apple stock prices from 2010:

```
import pandas.io.data as web
aapl =
web.get_data_yahoo('AAPL', '1/1/2010')
```

1. Extract the adjusted close column and plot it.
2. Resample to a 6 hours period, then interpolate through the missing values using cubic interpolation. Plot the resulting time series.

# Computations and statistics

167

## Computations with DataFrames

**Rule 1:** Mathematical operators (+ - \* / exp, log, ...) apply element by element, on the values.

**Rule 2:** Reduction operations (mean, std, skew, kurt, sum, prod, ...) are applied column by column.

**Rule 3:** Operations between multiple Pandas object implement auto-alignment based on index first.

168

# Computations with Pandas

```
# Computations are applied
# column-by-column
>>> df
   A      B   Flags
a  0    NaN  False
b  1   -0.9  False
c  2   -1.7  False
d  3    1.1  True

>>> df.sum()
A      6.0
B     -1.5
Flags  1.0
dtype: float64

# Adding a series or re-scaling
>>> row = df.iloc[1]
>>> df - row
   A      B   Flag
a -1    NaN  False
b  0     0  False
c  1   -0.8  False
d  2    2.0  True
```

## DATAFRAME REDUCTION

```
# 'apply' a custom function to
# columns. The function receives a
# column (Series) and returns a value
>>> f = lambda x: x.max() - x.min()
>>> df.apply(f, axis=0)
A      3.0
B      2.8
Flags  1.0
```

## DATAFRAME TRANSFORMATION

```
# applymap is similar but receives a
# value and return a value.
>>> df.applymap(lambda x: len(str(x)))
   A      B   Flags
a  1      3      5
b  1      4      5
c  1      4      5
d  1      3      4
```

169

# Statistical Analysis

## DESCRIPTIVE STATS

```
>>> df
   A      B   Flag
a  0    NaN  False
b  1   -0.9  False
c  2   -1.7  False
d  3    1.1  True

# Descriptive stats available:
# count, sum, mean, median, min, max,
# abs, prod, std, var, skew, kurt,
# quantile, cumsum, cumprod, cummax
# Stats on DF are column per column
>>> df.mean()
A      1.50
B     -0.50
flag   0.25

>>> df.mean(axis=1)
a    0.000000
b    0.033333
c    0.100000
d    1.700000

# min/max location (Series only)
>>> df['B'].argmin()
'c'
```

## DATAFRAME REDUCTION

```
>>> df.describe()
   A      B   Flag
count  4.000000  3.000000  4
mean   1.500000 -0.500000  0.25
std    1.290994  1.442221  0.5
min    0.000000 -1.700000  False
25%   0.750000 -1.300000  0
50%   1.500000 -0.900000  0
75%   2.250000  0.100000  0.25
max    3.000000  1.100000  True
```

## WINDOWED STATS

```
# Available rolling stats discoverable
# in IPython using
>>> pd.rolling_<TAB>
# Includes std, min, max, count, sum
# quantile, kurt, skew, ...
# For example,
>>> t = pd.rolling_mean(s, window=20)
# Custom function on ndarray possible
>>> f = lambda x: return x.mean()
>>> t == pd.rolling_apply(s, 20, f)
True
```

170

# Correlations

## CORRELATIONS

```
# Correlation of Series
>>> ts.corr(ts2)
0.06666666666666693

# Pair-wise correlations of the columns.
# Optional argument: 'method', one of
# {'pearson', 'kendall', 'spearman'}
>>> corr_matrix = df.corr()

# Pair-wise covariance of the columns
>>> cov_matrix = df.cov()
```



For more stats, see statsmodels.

171

# Data Filtering and Aggregation

172

# Split, apply and combine

## RATIONALE

It is often necessary to apply different operations on different subgroups

- Traditionally handled by SQL-based systems
- Pandas provides in-memory, sql-like set of operations

General ‘framework’: split, apply, combine (Hadley Wickham, R programmer):

- Splitting the data into groups (based on some criterion, e.g. column value)
- Applying a function to each group independently
- Combine the results back into a data structure (e.g. dataframe)

173

# Data aggregation: Split

## SPLIT WITH groupby()

```
>>> df
      A     B   Flag
a  0  NaN  False
b  1 -0.9  False
c  2 -1.7  False
d  3  1.1  True
e  4  0.5  True
# Group data by one column's value
>>> gb = df.groupby('Flag')
# gb is a groupby object
>>> gb.groups
{False: ['a', 'b', 'c'], True: ['d', 'e']}
# gb = iterator of tuples with
# group name and sub part of df
>>> for value, subdf in gb:
    print value
    print subdf
```

	A	B	Flag
a	0	NaN	False
b	1	-0.9	False
c	2	-1.7	False
d	3	1.1	True
e	4	0.5	True

	A	B	Flag
d	3	1.1	True
e	4	0.5	True

# Displays a subplot per group.

```
>>> gb.boxplot(column=["A", "B"])
```

## groupby() ON THE INDEX

```
>>> df2 = df.reset_index()
>>> even = lambda x: x%2 == 0
>>> gb2 = df2.groupby(even)
>>> gb2.groups
{False: [1, 3], True: [0, 2, 4]}
```

174

# Data aggregation: Apply

Three ways to apply: `aggregate` (or equivalently `agg`) if each series in each group is turned into one value, `transform` if each series in each group is modified but retains its length, or `apply` in the most general case.

## APPLY WITH `aggregate()` or `agg()`

```
>>> gb.sum()
      A      B
Flag
False  3 -2.6
True   7  1.6
# More flexible but slower
>>> summed = gb.aggregate(np.sum)
# Given a list or dict
>>> gb.agg([np.mean, np.std])
      A          B
mean      std  mean      std
Flag
False    1.0  1.000000 -1.3  0.565685
True     3.5  0.707107   0.8  0.424264
```

```
>>> gb.agg({'A': 'sum', 'B': 'std'})
```

	A	B
Flag		
False	3	0.565685
True	7	0.424264

## APPLY WITH `transform()`

```
>>> f = lambda x: x - x.mean()
>>> gb.transform(f)
      A      B
a -1.0  NaN
b  0.0  0.4
c  1.0 -0.4
d -0.5  0.3
e  0.5 -0.3
```

175

# Data aggregation II

## APPLY WITH `apply()`

```
# Computations from values in groups can be turned into a DF of calcs
>>> desc = lambda x: x.describe()
>>> gb['A'].apply(desc).unstack()
      count  mean      std  min   25%   50%   75%  max
Flag
False      3    1.0  1.000000    0   0.50   1.0   1.50    2
True       2    3.5  0.707107    3   3.25   3.5   3.75    4
>>> f = lambda group: DataFrame({'original':group,
                                 'demeaned': group - group.mean()})
>>> gb['A'].apply(f)
      demeaned  original
a         -1.0        0
b          0.0        1
c          1.0        2
d         -0.5        3
e          0.5        4
```

176

# Combining tables

177

# Merging

## Definition

`pandas.merge` connects DataFrames based on one or more keys (close to SQL join).

Let's assume we are running a restaurant, and store customer information and orders coming in in different tables:

```
>>> customers = DataFrame({'id': range(3), 'name': ['john', 'alex', 'lucy']})  
>>> orders = DataFrame({'id': [1, 0, 1, 2], 'order': ['pasta', 'salad',  
                  'coke', 'fries']})
```

Let's now assume we want to connect customer names to their order. We need to use their id to make that connection:

```
>>> merge(customers, orders, on='id')  
      id  name  order  
0    0  john  salad  
1    1  alex  pasta  
2    1  alex  coke  
3    2  lucy  fries
```

178

# Merging (Cont.)

## OUTER vs INNER JOINS

```
# Assume a mysterious order comes in
>>> orders = orders.append({'id': 3, 'order': 'pasta'}, ignore_index=True)

>>> merge(customers, orders, on='id')      >>> merge(customers, orders, on='id',
           id  name  order                  how='outer')
           0   0   john  salad
           1   1   alex  pasta
           2   1   alex  coke
           3   2   lucy  fries
                           id  name  order
                           0   0   john  salad
                           1   1   alex  pasta
                           2   1   alex  coke
                           3   2   lucy  fries
                           4   3   NaN  pasta
```

Merge method	SQL Join Name	Description
inner (default)	INNER JOIN	Use intersection of keys from both frames
outer	FULL OUTER JOIN	Use union of keys from both frames
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only

179

# Data summarization

180

# Pivot tables

## PIVOTING

```
# Repeating columns can be viewed as
# an additional axis
>>> df

      date variable    value
0 2000-01-03        A  0.469112
1 2000-01-04        A -0.282863
2 2000-01-05        A -1.509059
3 2000-01-03        B -1.135632
4 2000-01-04        B  1.212112
5 2000-01-05        B -0.173215

>>> df.pivot(index='date',
             columns='variable', values='value')

variable      A      B
date
2000-01-03  0.469112 -1.135632
2000-01-04 -0.282863  1.212112
2000-01-05 -1.509059 -0.173215
```

181

# Pivot tables II

## PIVOT\_TABLE

```
# Another way to reshape a DF and
# aggregate at the same time
>>> df

      A   B   C   D
0  foo one small  1
1  foo one large  2
2  foo one large  2
3  foo two small  3
4  foo two small  3
5  bar one large  4
6  bar one small  5
7  bar two small  6
8  bar two large  7

>>> table= df.pivot_table(
             index=['A', 'B'], columns=['C'],
             values='D', aggfunc=np.sum)
>>> table

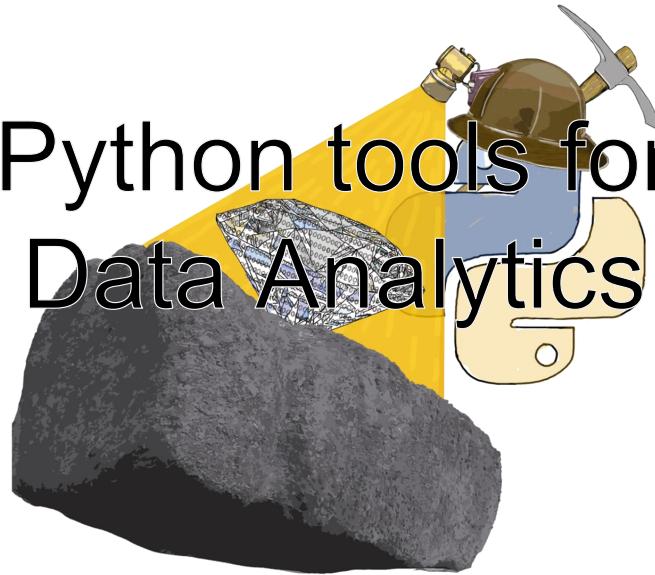
           small  large
foo  one    1     4
      two    6    NaN
bar  one    5     4
      two    6     7

# The values arg is optional
>>> df["E"] = randn(9)
>>> df.pivot_table(index=['A', 'B'],
                     columns=['C'])

           D          E
C      large  small
A   B
bar  one     4     5  1.683667 -1.979804
      two     7     6 -1.790215 -0.595985
foo  one     2     1  1.256463 -0.305674
      two    NaN     3    NaN  1.172797
```

182

# Python tools for Data Analytics



183

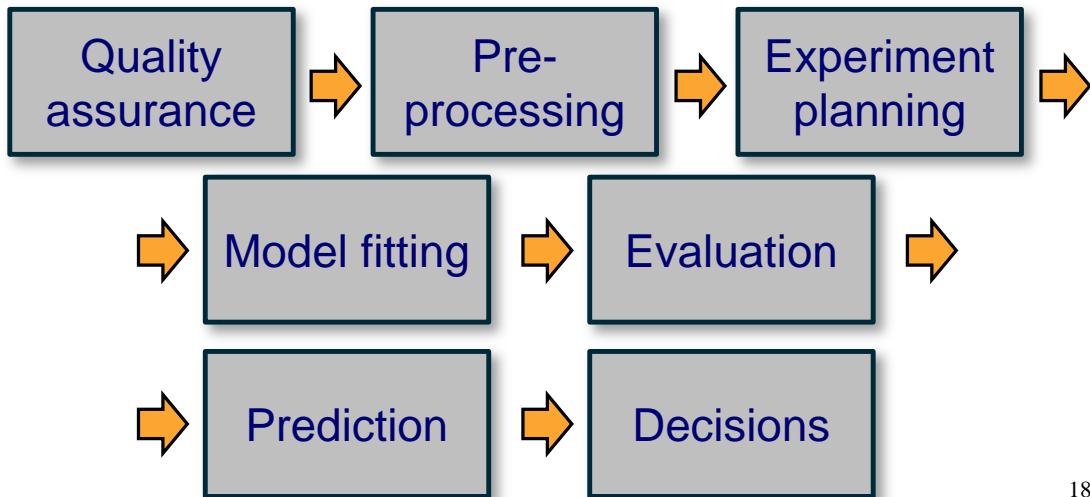
## Outline

- The data analytics workflow
- Handling data and quality checking
  - Reading data from multiple sources
  - Visual exploration
  - Statistics, hypothesis testing
- Advanced analytics with Machine Learning
  - Introduction to sklearn
  - Cleaning and preprocessing data
  - Regression problems
  - Model selection and cross-validation
  - Classification
- Natural Language Processing

184

# Data analytics workflow

The final goal of data analytics is to build a model with minimum prediction error, in order to take decisions with maximum predicted benefit.



185

## Before we start...

- Version check for the packages needed in this section:

Package name	Version
<code>sqlalchemy</code>	$\geq 1.0.6$
<code>seaborn</code>	$\geq 0.6.0$
<code>sklearn</code> <code>(scikit_learn)</code>	$\geq 0.15.2$
<code>mdp</code>	$\geq 3.3$
<code>nltk</code>	$\geq 3.0.1$
<code>gensim</code> <sup>(*)</sup>	$\geq 0.9.1$

Example:

```
>>> import sklearn
>>> sklearn.__version__
'0.16.0'
```

(\*) Install using: `pip install gensim`  
from a terminal

- Use Canopy's Package Manager to install / update these packages (Tools > Package Manager)

186

# Data access

From file, the web, SQL, ...

187

## Outline

- Importing files
  - Low-level I/O in Python
  - Pandas I/O
  - Common file formats
- Databases
  - DB-API 2.0
  - SQL Alchemy
  - SQL queries in Pandas

188

# Importing files

189

# Reading files

## FILE INPUT EXAMPLE

```
>>> results = []
>>> f = open('c:\\\\rcs.txt','r')
# Read all the lines.
>>> lines = f.readlines()
>>> f.close()
# Discard the header.
>>> lines = lines[1:]

>>> for line in lines:
...     # split line into fields
...     fields = line.split()
...     # convert text to numbers
...     freq = float(fields[0])
...     vv = float(fields[1])
...     hh = float(fields[2])
...     # group & append to results
...     all = [freq,vv,hh]
...     results.append(all)
... < hit return >
```

## PRINTING THE RESULTS

```
>>> for i in results: print i
[100.0, -20.30..., -31.20...]
[200.0, -22.70..., -33.60...]
```

## EXAMPLE FILE: RCS.TXT

#freq (MHz)	vv (dB)	hh (dB)
100	-20.3	-31.2
200	-22.7	-33.6



See demo/reading\_files directory  
for code.

190

# More compact version

## ITERATING ON A FILE AND LIST COMPREHENSIONS

```
>>> results = []
>>> f = open('c:\\rcs.txt', 'r')
>>> f.readline()
'#freq (MHz)  vv (dB)  hh (dB)\n'
>>> for line in f:
...     all = [float(val) for val in line.split()]
...     results.append(all)
... < hit return >
>>> for i in results:
...     print i
... < hit return >
>>> f.close()
```

## EXAMPLE FILE: RCS.TXT

```
#freq (MHz)  vv (dB)  hh (dB)
100        -20.3    -31.2
200        -22.7    -33.6
```

191

# Writing files

## FILE OUTPUT

```
>>> # Mode 'w': create new file:
>>> f = open('a.txt', 'w')
>>> f.write('Hello, world!')
>>> f.close()
>>> open('a.txt').read()
'Hello, world!'
>>> # Use the 'with' statement:
>>> with open('a.txt', 'w') as f:
....     f.write('Wow!')
....
>>> open('a.txt').read()
'Wow!'
>>> # Mode 'a': append to file:
>>> with open('a.txt', 'a') as f:
....     f.write(' Boo.')
....
>>> open('a.txt').read()
'Wow! Boo.'
```

## WRITE AND READ

```
>>> f = open('a.txt', 'w+')
>>> f.write('12 34 56')
>>> f.seek(3)
>>> f.read(2)
'34'
>>> f.close()
```

192

# Pandas IO

Pandas provides a high-level interface to and from many file formats used in data science:

- .txt, CSV, json, HTML, clipboard, Excel (.xls .xlsx), pickle, HDF5, SQL, R (exp.), Stata .dta, ...

For any given format, there is

- a `read_**` function,
- a `to_**` method attached to all Pandas data objects.

You might need to install other libraries for some of the formats (Pandas will warn you if that is the case).

193

## Pandas' `read_table` example

### FEATURES

`read_table` can read tabular text (for example CSV files) into a DataFrame and implements the following:

- detect comments, headers and footers
- specify which column is the index
- specify the column names or which line is the column name,
- parse dates stored in one or several columns,
- handle multiple codes for missing data,
- read data by chunk (large files),
- custom value parsing based on column
- ...

### EXAMPLE

```
# Historical_data.csv
Date,AAPL,GOOGL,MSFT,PG,XOM
2005-01-03,64.78,197.4,26.8,-,51.02
2005-01-04,63.79,201.4,26.87,55.12,50.34
2005-01-05,64.46,193.45,26.84,55.28,49.83
...
```

```
>>> read_table('historical_data.csv',
                 sep=',', header=1,
                 index_col=0,
                 na_values=['-'],
                 parse_dates=True)
```

Date	AAPL	GOOGL	MSFT	PG	XOM
2005-01-03	64.78	197.40	26.80	NaN	51.02
2005-01-04	63.79	201.40	26.87	55.12	50.34
2005-01-05	64.46	193.45	26.84	55.28	49.83
					194

# Reading large files in chunks

Pandas supports reading potentially very large files in chunks, e.g.:

```
>>> chunks = []
>>> reader = pd.read_csv('contributions_2012.csv',
...                      chunksize=100000)
>>> for table in reader:
...     new_yorkers = table['contbr_city'] == 'NEW YORK'
...     chunks.append(table[new_yorkers])
>>> new_york_contributions = pd.concat(chunks)
>>> print len(new_york_contributions)
25858

>>> print new_york_contributions.iloc[143]
cmte_id           C00431171
cand_id           P80003353
cand_nm          Romney, Mitt
contbr_st            NY
contbr_occupation    EXECUTIVE
contb_receipt_amt       2500
contb_receipt_dt      22-JUN-11
...

```

195

# Pandas IO summary

## READING

Format	Method, Function, Class
txt, csv	read_table, read_csv
pickle	read_pickle
HDF5	read_hdf, HDFStore
SQL	read_sql_table
Excel	read_excel
R (exp.)	rpy.common.load_data

## WRITING

Format	Method, Function, Class
txt, csv	to_string, to_csv
html	to_html
pickle	to_pickle
HDF5	to_hdf, HDFStore
Excel	to_excel, ExcelWriter
R (exp.)	rpy.common.convert_to_r_dataframe

## EXAMPLES

```
# Excel
>>> writer = ExcelWriter('out.xlsx')
>>> df1.to_excel(writer, 'Sheet1')
>>> df2.to_excel(writer, 'Sheet2')
>>> writer.save()
>>> read_excel("out.xlsx", "Sheet2")

# HDF5
>>> stor = HDFStore('foo.h5')
>>> stor['ser1'] = s
>>> s2 = stor['ser1']
>>> stor.close()
>>> s3 = read_hdf('foo.h5', 'ser1')

# Scrape tables from HTML webpages
>>>
read_html("http://www.bloomberg.com/markets")
```

196

# Other common file formats

Many other file formats are supported in the scientific Python ecosystem:

File format	Package name(s)	Description
JSON	json	Human-friendly, lightweight data-interchange format.
XML	xml.etree.ElementTree, lxml	eXtensible Markup Language.
SAS	sas7bdat (to_dataframe)	SAS' own file format.
Matlab <= 7.1	scipy.io (loadmat, savemat)	Matlab's own file format.
HDF5, Matlab > 7.1	pytables, h5py	Hierarchical Data Format designed to store large amounts of numerical data.
NetCDF	netCDF4, scipy.io.netcdf	Format for array-based numerical data.
wav	scipy.io.wavfile	Audio files.
jpeg, png, ...	PIL, scipy.misc.pilutil, skimage	Common image formats.

197

# Databases

198

# Outline

- Learn how to access data in SQL databases
  - Low-level access with DB-API 2.0
  - SQLAlchemy, core expression language
  - SQL queries from Pandas

199

## DB-API 2.0

Python defines a common API for all Python DB libraries, called DB-API 2.0 and described in PEP 249. <https://www.python.org/dev/peps/pep-0249/>

Database	Module or Modules for Python 2.X
Oracle	<b>cx_Oracle</b> , DCOracler2, mxODBC, pyodbc
PostgreSQL	psycopg2, PyGreSQL, pyPgSQL, mxODBC, pyodbc, pg8000
MySQL	MySQLdb, mxODBC, pyodbc, myconnpj
Sqlite	<b>sqlite3</b> (included in standard library)
Microsoft SQL Server	adodbapi, pymssql, mssql, mxODBC, <b>pyodbc</b>
Ingres	ingresdbi
IBM DB2	lqm_db, PyDB2, ceODBC, mxODBC, pyodbc
Sybase ASE	Sybase, mxODBC
Sybase SQL Anywhere	mxODBC, sqlanydb
SAP DB	sdb.dbapi, sapdbapi, mxODBC, sdb.sql, sapdb
Informix	InformixDB, mxODBC,
Firebird	KInterbasdb

200

# DB-API 2.0

## CONNECT

```
import <somedbmodule> as db
# Connect to the database.
# Extra arguments typically include user and password.
conn = db.connect(<dsn>, ...)
```

## EXECUTE STATEMENTS

```
# Get a cursor object.
c = conn.cursor()

c.execute("CREATE TABLE users(name TEXT, email TEXT, age INT, city TEXT)")
c.execute("INSERT INTO users VALUES ('Mike','mike@mike.com',23,'London')")
c.execute("INSERT INTO users VALUES ('Kim','kim@kim.net',34,'New York')")
conn.commit()

c.execute("SELECT * FROM users WHERE city='London' ORDER BY age")
for row in c: # Fetch records using the cursor as an iterator.
    print row
single = c.fetchone() # Fetch one record.
list_of_lists = c.fetchall() # Fetch all rows in one list.
```

## CLOSE

```
c.close()
conn.close() # Closing without committing causes a rollback.
```

201

# DB-API 2.0: Dynamic queries

Typically, your queries will need to use variables from Python variables.

## DON'T DO THIS! INSECURE AND INEFFICIENT

```
>>> city = "London"
# This is dangerous!
>>> c.execute("SELECT * FROM users WHERE city='{}'".format(city))
>>> c.fetchall()
[(u'Mike', u'mike@mike.com', 23, u'London')]

# For example, this malicious input will reveal all users.
>>> city = "' OR '1'='1"
>>> c.execute("SELECT * FROM users WHERE city='{}'".format(city))
>>> c.fetchall()
[(u'Mike', u'mike@mike.com', 23, u'London'),
 (u'Kim', u'kim@kim.net', 34, u'New York')]
```

## USE PLACEHOLDERS

```
>>> city = ("London",)
>>> c.execute("SELECT * FROM users WHERE city=?", city)

>>> row = ("John", "john@corp.com", 63, "London")
>>> c.execute("INSERT INTO users VALUES(?, ?, ?, ?)", row)
# Some modules use a different place-holder instead of ?,
# see db.paramstyle.
```

202

## DB-API 2.0: Transactions

- A transaction is sequence of operations performed as a single logical unit of work.
- Handled through the `Connection` object, using the `commit` and `rollback` connection methods.
- Use `Connection` as a context manager to group statements into a transaction, which is committed on successful exit.

```
connection = db.connect(<dsn>)
with connection:
    #(Almost) everything in the context manager happens
    # within a transaction.
    c = connection.cursor()
    c.execute("INSERT INTO users VALUES"
              " ('Mike','mike@mike.com',23,'London')")
    c.execute("INSERT INTO users VALUES"
              " ('Kim','kim@kim.net',34,'New York')")
```

203

## DB-API 2.0: Transactions

### Interactive notebook demo

DB API 2.0 - `transaction.ipynb`

204

# SQLAlchemy

- Abstracts over differences of DB backends.
- Used by famous Python users:  
Mozilla, reddit, Dropbox, Uber, ...

Two main components:

- Core Expression Language
  - Abstracts differences between databases: can emit queries for most DB backends.
  - SQL commands are built using Python statements instead of SQL strings.
  - Efficient and mostly transparent .
- Object Relational Mapper
  - Maps DB rows to Python objects.

205

# SQLAlchemy

## Interactive notebook demo

`SQLA-core_expression_language.ipynb`

206

## SQLA: Creating a table

- `create_engine` returns an engine that can connect to the database.
- A `Metadata` object contains information about the schema of the tables in the database.

```
from sqlalchemy import (create_engine, Column, Integer,
                        MetaData, String, Table)

# "sqlite://" is the equivalent of ":memory:" in DB-API
engine = create_engine("sqlite://")

metadata = MetaData()
table = Table("example", metadata,
              Column("id", Integer),
              Column("value", String))

# This creates the table (if it does not exist already).
metadata.create_all(engine)
```

207

## SQLA: Inserting data

- `Table.insert` method to insert data into an existing table
- To pass values, use a tuple or dictionary mapping column name to value (less error prone)

```
with engine.connect() as conn:
    conn.execute(table.insert(),
                 [{"id": 1, "value": "some value"},
                  {"id": 2, "value": "another value"}])
```

208

# SQAL: Selecting data

- Use the function `select` to create a “select” query
- Execute the query using a `Connection` object

```
from sqlalchemy import select
# Select the whole table.
with engine.connect() as conn:
    res = conn.execute(select([table]))
    for row in res:
        print(row)
(1, u'some value')
(2, u'another value')

# Select a few columns.
with engine.connect() as conn:
    res = conn.execute(select([table.c.id, table.c.value]))
    print(res.fetchall())
[(1, u'some value'), (2, u'another value')]
```

209

# SQAL: Selecting data

The selection object returned by `select` provides methods to perform more complex SQL queries: `where`, `order_by`, `count`, `group_by`, ...

```
from sqlalchemy import func, select
# Select users older than 20, sorted by name.
with engine.connect() as conn:
    s = (select([users.c.name]).where(users.c.age > 20) .
         order_by(users.c.name))
    res = conn.execute(s)

# Create a list of number of users per country.
with engine.connect() as conn:
    s = (select([func.count(users.c.id), users.c.country]) .
         group_by(users.c.country))
    res = conn.execute(s)
```

210

# SQL queries from Pandas

Pandas' `read_sql` can be used to execute SQL queries and automatically wrap result in a `DataFrame`:

```
# Load fuel consumption data.
table = pd.read_csv("vehicles.csv")

# Store data frame in the database in a table called
# 'vehicles'.
# ('db' is an SQLA engine or a DBAPI2 connection.)
table.to_sql("vehicles", db, if_exists="replace")
# `if_exists` is one of 'fail', 'replace', or 'append'

# Request data for vehicles using Diesel, from 1990.
# Can pass an SQL query (string) or an SQLA `Select` object
query = "SELECT year, comb08, highway08, city08 " \
        "FROM vehicles WHERE fuelType1='Diesel' AND year > \
1990"
diesel_consumption = pd.read_sql(query, db)

diesel_consumption.head()
```

	year	comb08	highway08	city08
0	1993	19	22	18
1	1993	17	20	15
2	1993	17	20	15
3	1993	17	20	15
4	1993	17	20	15

211

# SQL and Pandas: workflow

It is in general much more efficient to execute computations on the SQL side and import the result in Pandas than vice versa.

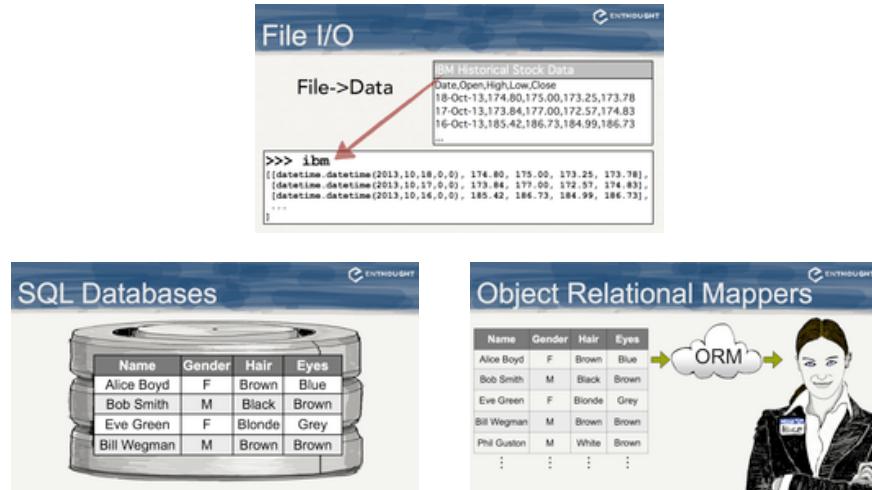
The typical workflow is:

1. Run complex query on DB to select interesting subset of data (the full data might not fit in memory).
2. Import result in Pandas and analyze and plot.

212

# Further learning online

Additional material and exercises available online:



The image contains three separate screenshots from Enthought's training materials:

- File I/O:** A screenshot of a Jupyter notebook cell showing code reading historical stock data from a file. The code is: 

```
>>> ibm
```

 followed by a list of tuples representing stock prices over time.
- SQL Databases:** An illustration of a stack of four cylinders representing a database system, with a table overlaid showing columns: Name, Gender, Hair, Eyes. Data rows include Alice Boyd (F, Brown, Blue), Bob Smith (M, Black, Brown), Eve Green (F, Blonde, Grey), and Bill Wegman (M, Brown, Brown).
- Object Relational Mappers:** A diagram showing the relationship between a database table and an object. The table has columns: Name, Gender, Hair, Eyes. The object is a person with the same attributes. A cloud labeled "ORM" connects the two.

# Visual exploration

# Visually exploring data

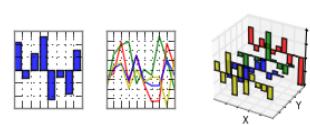
- *Exploratory analysis*: Visually explore data and formulate hypotheses
- Quality control: any obvious problem?
- Typical steps in visual exploration:
  - Look at distribution of variables
  - Look at relations between variables
  - Look at panels of plots with grouped data

215

# Tools



`import matplotlib.pyplot as plt`



`import pandas as pd`

## Seaborn: statistical data visualization

`import seaborn as sns`

216

# Visualization

## Interactive notebook demo

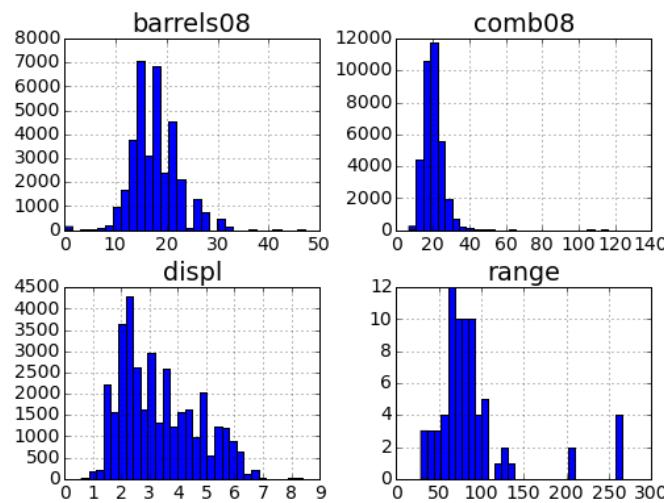
Vehicles\_exploration.ipynb

Stocks\_visual\_exploration.ipynb

217

## Distribution of individual variables

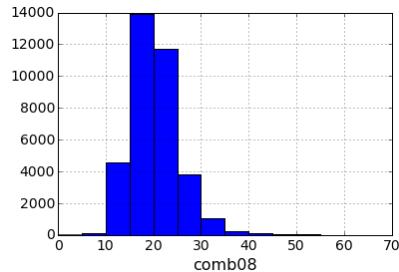
```
vehicles.hist(column=['comb08', 'range', 'barrels08', 'displ'],
              bins=30)
```



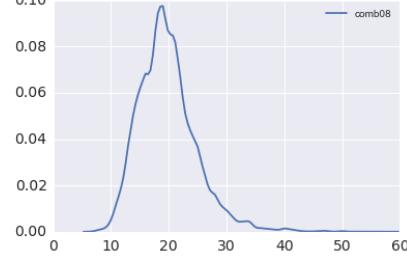
218

# Distribution of individual variables

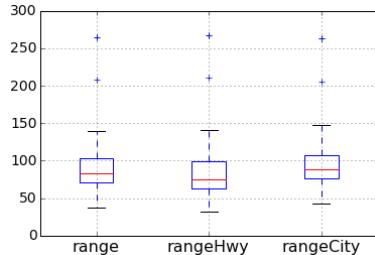
`hist` shows a classical histogram of the data



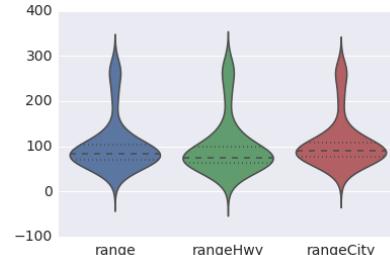
`sns.kdeplot` displays an estimate of the probability density



`boxplot` summarizes each data distribution with 5 points



`sns.violinplot` shows an estimate of the shape of the density

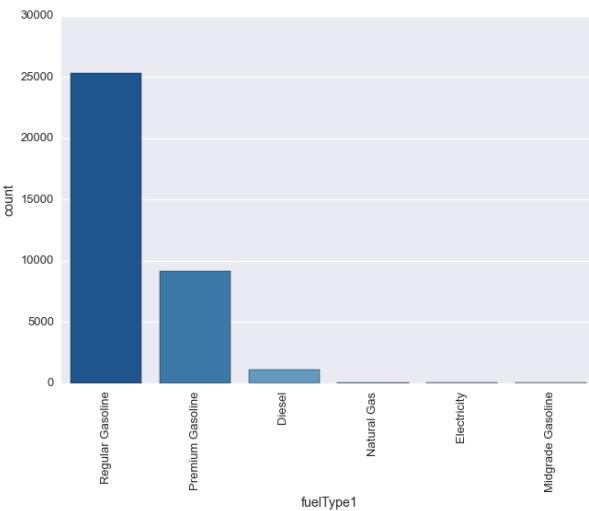


219

# countplot

`countplot` is used to display the distribution of categorical data.

```
sns.countplot('fuelType1', data=vehicles)
```

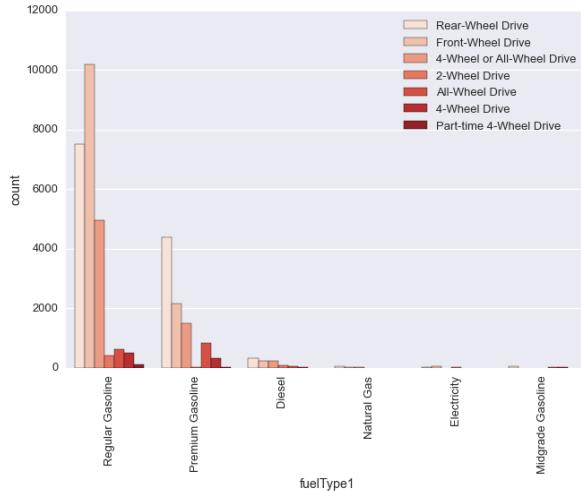


220

# countplot

`countplot`, and many other Seaborn functions, accept a `hue` keyword that subdivides the plots by another categorical variable.

```
sns.countplot('fuelType1', hue='drive', data=vehicles)
```

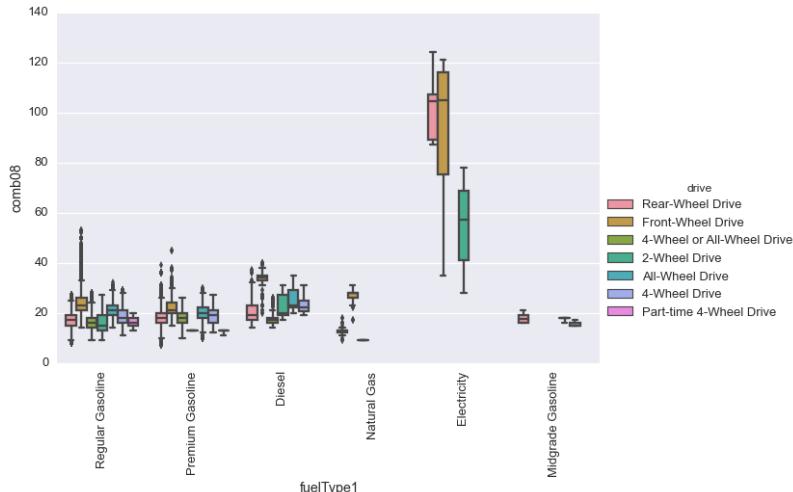


221

# factorplot

`factorplot` is a more flexible plot that allows displaying the distribution of any quantity, grouped by a categorical variable.

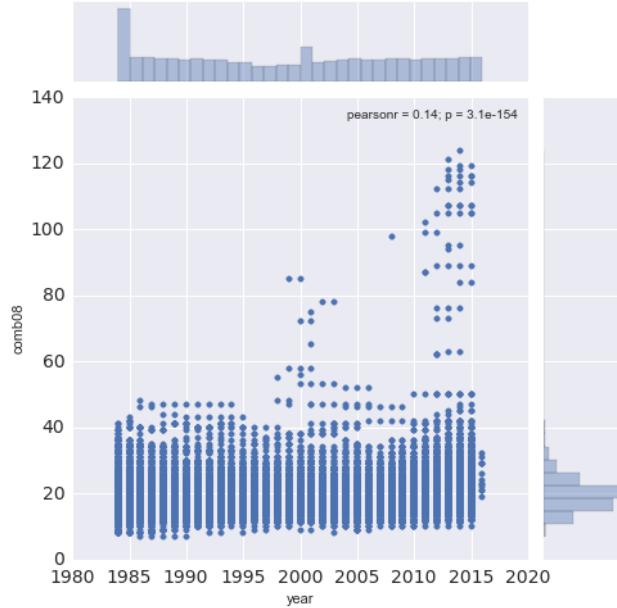
```
sns.factorplot(x='fuelType1', y='comb08', hue='drive',
                 kind='box', data=vehicles)
```



222

# Joint distributions of variables

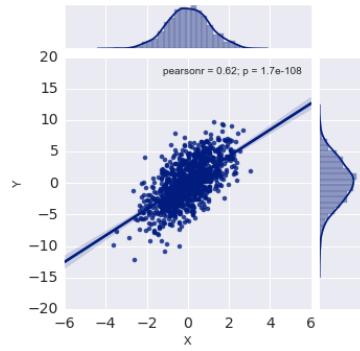
```
sns.jointplot("year", "comb08", data=vehicles)
```



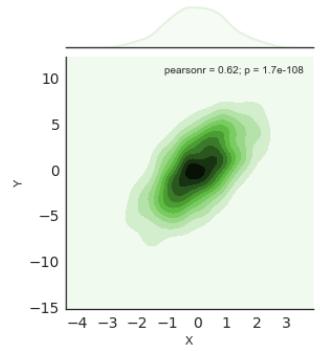
223

# Jointplot styles

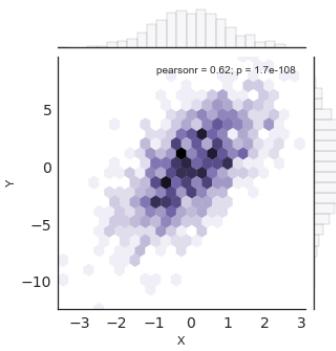
`kind='reg'`



`kind='kde'`



`kind='hex'`



224

# Small multiples

*"At the heart of quantitative reasoning is a single question: Compared to what? Small multiple designs, multivariate and data bountiful, answer directly by visually enforcing comparisons of changes, of the differences among objects, of the scope of alternatives. For a wide range of problems in data presentation, small multiples are the best design solution."*

*Edward Tufte (Envisioning Information, p. 67)*

Key idea: Visualize a grid of plots showing the same information, with data split in multiple groups.

Aliases: Trellis plot, panel chart

225

# FacetGrid

A `FacetGrid` object is responsible for creating the grid of plots. Each plot in the grid will contain data grouped by the `DataFrame` columns corresponding to its row and column:

```
grid = sns.FacetGrid(  
    dataframe, row='col_name',  
    col='col_name', hue='col_name')
```

`FacetGrid.map` draws to all plots of the grid:

```
grid.map(plot_func, 'x_col_name', 'y_col_name')
```

226

# Example 1: plotting 5 dimensions

Consumption over time, grouped by make and no. of cylinders:

```
grid = sns.FacetGrid(
    top_makes, row='make', col='cylinders', hue='fuelType1')
grid.map(plt.scatter, "year", "comb08", alpha=.5)
grid.add_legend()
```

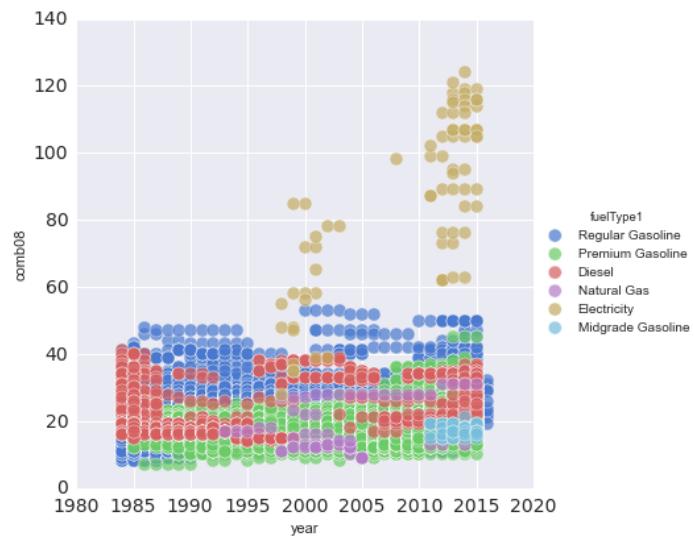


227

# Example 2: A grid of one

Consumption over time, colored by primary fuel type:

```
grid = sns.FacetGrid(vehicles, hue="fuelType1")
grid.map(plt.scatter, "year", "comb08", alpha=.7)
grid.add_legend()
```

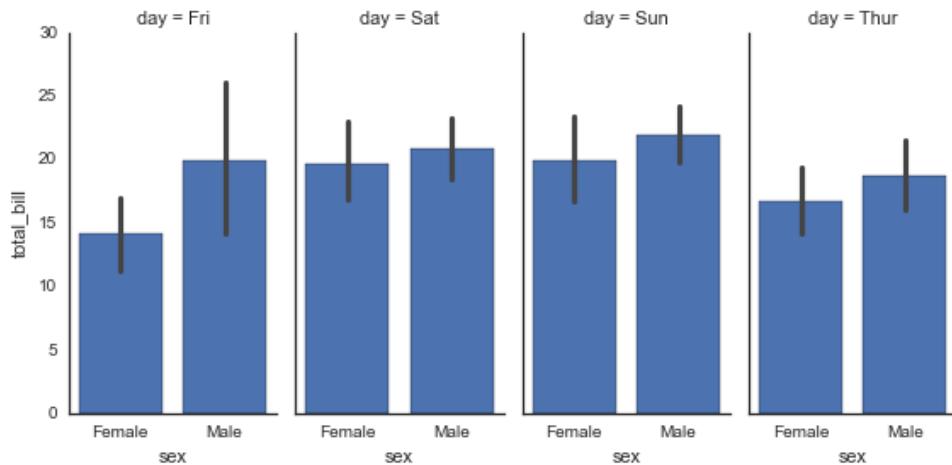


228

## Example 3

Total restaurant bill, grouped by day of the week and sex:

```
g = sns.FacetGrid(tips, col="day", aspect=.5)
g.map(sns.barplot, "sex", "total_bill");
```



229

## PairGrid

A `PairGrid` object is another plot grid, where each entry corresponds to a pair of a columns in a `DataFrame`. The plot displays *all* the data from the two columns.

```
grid = sns.PairGrid(data, hue='col_name', vars=None)
```

`PairGrid` methods allow drawing in all plots...

```
grid.map(plot_func)
```

on/off the diagonal only...

```
grid.map_diag(plot_func)
grid.map_offdiag(plot_func)
```

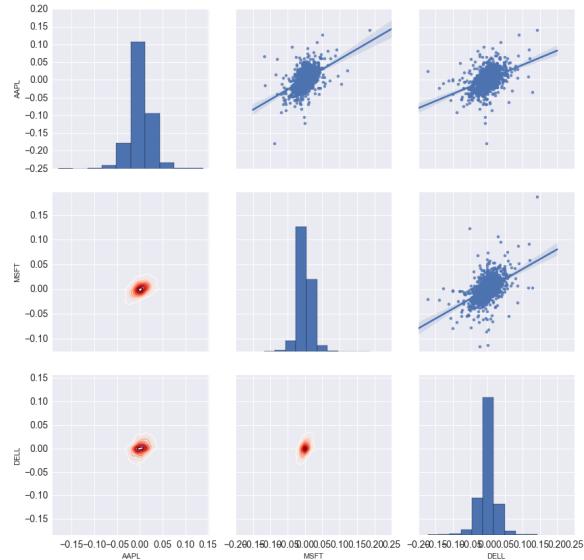
or only on the upper / lower half of the grid

```
grid.map_upper(plot_func)
grid.map_lower(plot_func)
```

230

# Example

```
grid = sns.PairGrid(returns, vars=['AAPL', 'MSFT', 'DELL'])
grid.map_upper(sns.regplot)
grid.map_diag(plt.hist)
grid.map_lower(sns.kdeplot, linewidth=3.0, cmap='Reds')
```



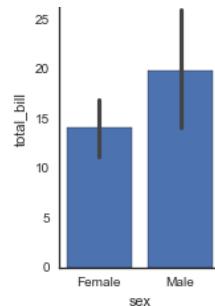
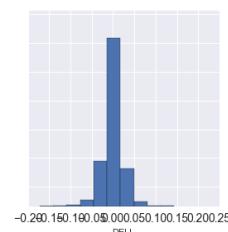
231

# Grid-plot compatible plots

From Matplotlib:

`plt.hist`

`plt.scatter`



From Seaborn:

`sns.barplot`

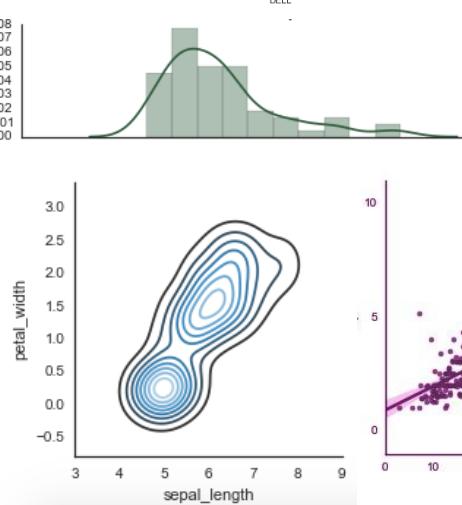
`sns.regplot`

`sns.kdeplot`

`sns.distplot`

`sns.pointplot`

...



232

# Statistics

233

## Outline

- Introduction
- Fraud detection:
  - Hypothesis testing
  - Statistical tests

234

# Data Science and statistics

- All Data Science methods are, in one way or another, statistical algorithms.
- A solid understanding of basic statistical ideas is necessary to:
  - Quality check the input data,
  - Verify that the data match the assumptions of each algorithm,
  - Check that the results are meaningful.
- We will have a look at one key idea in statistics, hypothesis testing, by developing a fraud detection algorithm.

235

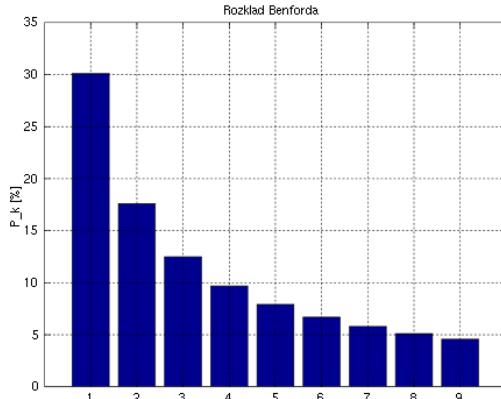
# Benford's law

- Which number is more likely to appear in an accounting book, 135.2 or 294.1?
- What is the distribution of the first digit of numbers picked at random from newspapers?
- Is the digit 2 more likely to be preceded by a 5, or a 1?

236

# Benford's law

- One answer to all these questions: Benford's law
- The distribution of the leading digit in many, common cases is not uniform:



$$P(d) = \log_{10}(d+1) - \log_{10}(d) = \log_{10}\left(1 + \frac{1}{d}\right).$$

237

## Example: Fraud in Arizona

The table lists the checks that a manager in the office of the Arizona State Treasurer wrote to divert funds for his own use. The vendors to whom the checks were issued were fictitious.

Date of Check	Amount
October 9, 1992	\$ 1,927.48
	27,902.31
October 14, 1992	86,241.90
	72,117.46
	81,321.75
	97,473.96
October 19, 1992	93,249.11
	89,658.17
	87,776.89
	92,105.83
	79,949.16
	87,602.93
	96,879.27
	91,806.47
	84,991.67
	90,831.83
	93,766.67
	88,338.72
	94,639.49
	83,709.28
	96,412.21
	88,432.86
	71,552.16
<b>TOTAL</b>	<b>\$ 1,878,687.58</b>

Source: "I've got your number",  
Journal of accountancy, May 1999

# Fraud detection demo

## Fraud detection demo:

`benfords_hypothesis_testing.ipynb`

239

# Hypothesis testing

## Hypothesis testing checklist

## Example: one-sample t-test

Which effect to test?	Mean is different from $\mu_0$
What would be true if the effect is <i>not</i> present? <b>“null hypothesis”</b>	Mean is equal to $\mu_0$
How to measure the effect? <b>“statistic”</b>	$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$
Compute distribution of statistics under the null hypothesis (table or formula)	t-distribution Depends on the number of data points available
Compare statistic of observed data to that distribution. Is it highly unlikely? If so, discard null hypothesis	P-value Probability of a result as extreme or more as the one observed, under the null hypothesis

240

# Multiple measurements problem

- If we execute  $n$  tests, we increase the probability of finding significant results by chance.



- In our case, if we investigate fraud in 1000 companies, and set a threshold of  $P < 0.01$  for detection, at least 10 companies will be flagged as fraudulent even though they are not.

241

# Type I and Type II error

	Null hypothesis is true	Null hypothesis is false
Reject	Type I error “False positive”	All good
Do not reject	All good	Type II error “False negative”

- Type I error: incorrect rejection of a true null hypothesis
- Rate of Type I errors is fixed before the experiment is made, and is the threshold to which the P-value is compared
- Type II error: failure to reject a false null hypothesis
- Rate of Type II errors is usually high when the number of data points is low, or when a test is very generic

242

# t-tests

t-Tests: effect on the mean of one or two populations.

t-test variant	Null hypothesis	Assumptions	scipy.stats function
One-sample t-test	Data has given mean.	1. Independent samples 2. Normal distribution <sup>[2]</sup>	<code>ttest_1samp(data, mu0)</code>
Two-sample t-test, independent samples	Two independent sets of data have the same mean.	1. Independent samples 2. Normal distributions <sup>[2]</sup> 3. If equal_var=True, variances are the same	<code>ttest_ind(data_1, data_2, equal_var=True)</code>
Two-sample t-test, paired <sup>[1]</sup> samples	Two related sets of data have the same mean.	1. Independent samples 2. Normal distributions of pair differences <sup>[2]</sup> 3. Variance is the same	<code>ttest_rel(data_1, data_2)</code>
All t-test functions return a tuple, (t, p) t: value of the t statistics, p: <b>two-tailed</b> p-value of the test. (Divide by two to get one-tailed p-value.)		<p>[1] E.g., repeated measurements (performance of students over time)</p> <p>[2] The t-test is robust to violation of this assumption, but the power is reduced<sup>243</sup></p>	

# More Hypothesis Tests

Test	Null hypothesis	scipy.stats function
ANOVA	Multiple samples $x_1, x_2, \dots$ have the same mean. Generalizes t-test to multiple group.	<code>f_oneway(x1, x2)</code>
Wilcoxon rank-sum test	Two independent samples are drawn from the same distribution. Alternative to t-test for non-normal distributions.	<code>ranksums(x, y)</code>
Kolmogorov-Smirnov test	Compare the distribution of a continuous variable to a known distribution. H0: the distr are identical	<code>kstest(x, 'norm')</code>
chi-square test	Compare the distribution of a categorical variable to a known distribution. H0: the distr are identical	<code>chisquare(x, f_exp=[16, 16, 8])</code>

# Further learning online

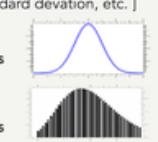
Additional material and exercises available online:

**Statistics Overview**

scipy.stats



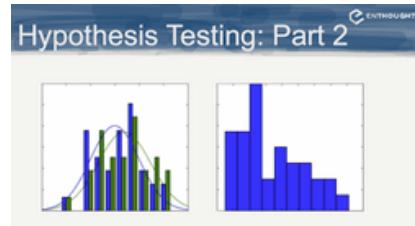
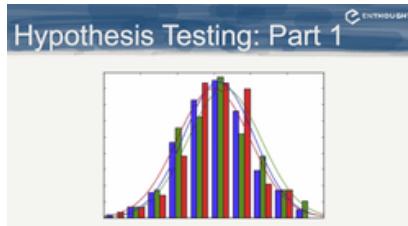
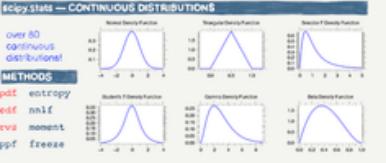
- Descriptive Statistics [ mean, standard deviation, etc. ]
- Continuous Distributions
- Discrete Distributions



**Probability Distributions**

scipy.stats — CONTINUOUS DISTRIBUTIONS

- over 80 continuous distributions!
- METHODS
- pdf entropy
- cdf nnlf
- rva moment
- ppf freeze



# Advanced data analytics with Machine Learning

# Introduction

- *Machine Learning* is about creating **models** of data that can be **tuned** from observed data and used to make **predictions** about new, partially observed data

247

# Generalist Python tools



- **sklearn 0.15**: Very active and complete project, contains 99% of what you will need for general ML



- MDP 3.4: Older project, we'll use it for a couple of features that are unavailable in sklearn



- Orange: Alternative ML library with GUI. Not as well integrated with scientific stack



- PyMC: To define probabilistic models, and run inference using Markov Chain Monte Carlo methods

248

# Specialist Python tools

Other libraries worth mentioning for special needs:

**gensim**

- gensim: Topic modeling, natural language models

**NLTK**

- NLTK: More natural language processing



- NeuroLab: Neural network algorithms

249

# Main classes of ML problems

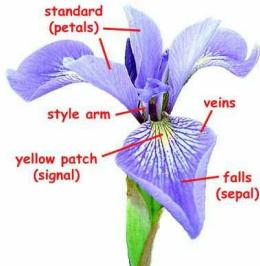
- **Supervised learning:** labelled data
  - **Classification**  
*Identify person from photograph*  
*Classify spam in Inbox*
  - **Regression**  
*Learn relation between location and house price*  
*Learn relation between product properties and user rating*
- **Unsupervised learning:** unlabelled data
  - **Clustering**  
*Identify groups of stars with similar characteristics*
  - **Density estimation**  
*Remove redundancy from data*  
*Given mixture of sounds, separate the sources (“blind source separation”)*
- **Reinforcement learning:** learn through feedback  
*Learning to play backgammon based on game outcome*

250

# sklearn basic concepts: data

- Large number of models and utility functions with consistent API
- **Data** in sklearn is represented as a 2D numpy array: samples x features

*The Iris data set*



Samples

Features

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

251

# sklearn basic concepts: models

- *Estimators* are model classes with shared API
- All estimators (**x** are input features, **y** are target values)
  - `.fit(x[, y])` Fits the model parameters given input and target data
- Supervised estimators
  - `.predict(x_new)` Predict output given input data
  - `.score(x_new[, y_new])` Evaluate fit on new data with 0 – 1 score
  - `.predict_proba(x_new)` Probability of assignment for categorical output
- Unsupervised estimators
  - `.transform(x)` Transform input based on model

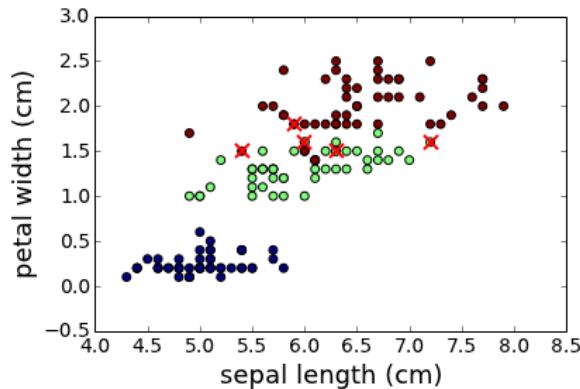
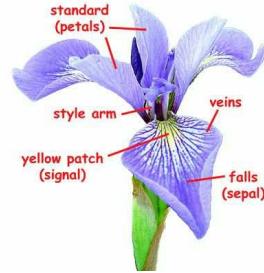
252

# Example

```
from sklearn import datasets, svm
iris = datasets.load_iris()

model = svm.LinearSVC()
model.fit(iris.data, iris.target)

predicted = model.predict(iris.data)
```



253

# Navigating sklearn

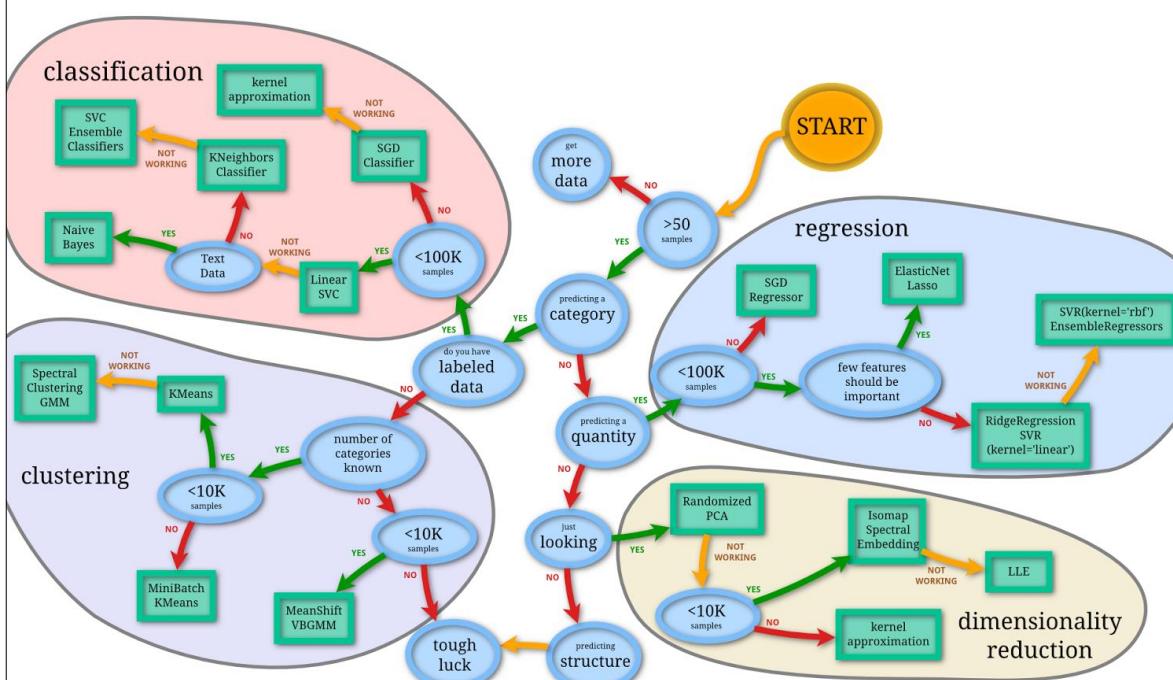


Diagram by Andreas Muller, 2013, released under CC0

254

## Additional Resources

- (book) C.M.Bishop, *Pattern recognition and Machine Learning*
- (book) Trevor Hastie et al, *The Elements of Statistical Learning*
- (video) Jake Vanderplas, *An Introduction to scikit-learn: Machine Learning in Python*,  
<http://bit.ly/1I0LOGD>
- (videos) Gaël Varoquaux, Jake Vanderplas, Olivier Grisel, *An introduction to scikit-learn (I) and (II)*,  
[https://conference.scipy.org/scipy2013/tutorials\\_schedule.php](https://conference.scipy.org/scipy2013/tutorials_schedule.php)
- (videos) Andrew Ng, *Machine Learning*, Coursera lecture, <https://www.coursera.org/course/ml>

255

## Preprocessing

257

## Key idea

ML algorithms make assumptions about their input data. One should prepare the data set to match those expectations, without distorting it.

258

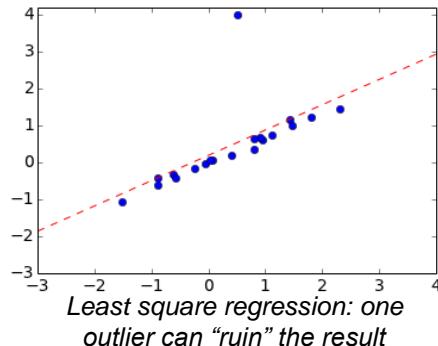
## Preprocessing, definition

- Everything that needs to be done *before* starting with the experiments
  - Removing outliers
  - Normalization
  - Dealing with missing values
  - Dimensionality reduction
  - Feature selection
- Critical to the success of ML

259

# The problem with outliers

- Many statistics (mean, std dev, correlations) are sensitive to outliers, and thus so are all methods that depend on them)
- Defining “outliers” can be subjective and removing them could bias your results. Beware of removing “black swan” data points!
- One cannot simply use a canned set of rules to identify outliers, we need to look at the data

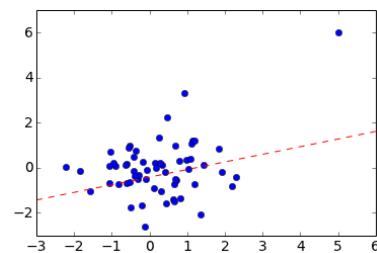


Least square regression: one outlier can “ruin” the result

260

# (Not) removing outliers

- The only outliers you can feel confident removing are obvious errors; requires expert knowledge of domain:  
*Adult weighing 3kg; oil price of -2.3\$; dog aged 300 yrs*
- Consider removing the entire feature instead of the individual points.
- On the other hand, if the presence of the outliers alone creates a significant result, they *must* be dropped (“leverage”).

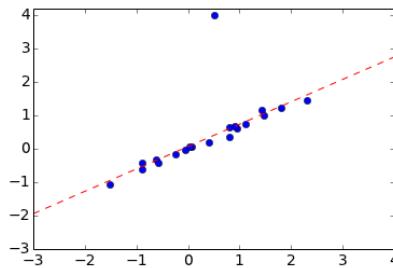


One outlier can create a result

261

# Outliers advice

- Use robust statistics (e.g., median) and ML methods if you can.  
*Warning: this could be conceptually equivalent to removing the outliers*



*Linear regression using a robust loss function*

- Remember that the goal is robust prediction, so include the outlier diagnostic in the prediction workflow.
- Use quantitative recipes to remove outliers, so that the same rules can be applied on new data, and when doing prediction.

262

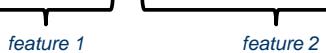
# Data normalization

- Some algorithms are sensitive to the scale of the data. If features represent different quantities, they are often normalized to zero mean, unit std dev.  
*E.g.: age, weight, height features; not pixel intensities*
- Data is often transformed non-linearly to make the distribution more Gaussian, or make regression problems linear.  
*E.g.: human perception of the intensity of sound or light is linearly related to the log of intensity (dB scale); multiplicative laws like the Cobb-Douglas production law, output = productivity \* labor<sup>α</sup> \* capital<sup>β</sup>*

263

# sklearn normalization functions

Normalization functions are defined in `sklearn.preprocessing`

<code>scale(x, axis=0)</code>	Rescale data to zero mean and unit variance over the given axis (over features, by default).
<code>normalize(x, norm='l2', axis=1)</code>	Normalize data to unit norm along given axis. norm can be either 'l2' or 'l1'
<code>enc = OneHotEncoder() enc.fit_transform(labels)</code>	Transform categorical features encoded as integers to a binary one-of-K format (e.g., {1, 2, 3} is transformed to {[1, 0, 0], [0, 1, 0], [0, 0, 1]}). Encoding of multiple features are concatenated, e.g.: <pre>&gt;&gt;&gt; enc = preprocessing.OneHotEncoder() &gt;&gt;&gt; dat = [[0, 3], [1, 0], [0, 1], [1, 2]] &gt;&gt;&gt; enc.fit(dat) &gt;&gt;&gt; enc.transform([[0, 3]]).toarray() array([[ 1.,  0.,  0.,  0.,  0.,  1.]])</pre> 

264

# Dealing with missing values

- The majority of (classical) ML methods cannot deal with missing values.
- Not much one can do about it, in general:
  - Discard entire sample or feature if you can afford it.
  - Replace missing values with mean, median, or most common value for that feature.

```
>>> from sklearn.preprocessing import Imputer

>>> x = np.array([[1, 2], [np.nan, 3], [7, np.nan]])

# Strategy can be 'mean', 'median', or 'most_frequent'
>>> imp = Imputer(missing_values='NaN', strategy='mean', axis=0)
>>> imp.fit_transform(x)
array([[1., 2.], [4., 3.], [7., 2.5]])
```

265

# Dimensionality reduction

Two main reasons to transform the vector of input features to a smaller-dimensional version:

1. Memory/speed constraints.
2. Features are highly correlated, which leads the ML method to be under-constrained, and convergence to be slow.

266

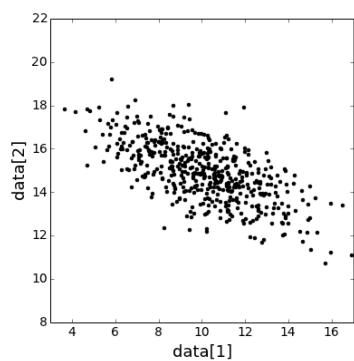
# PCA

- The main technique used for dimensionality reduction is Principal Component Analysis (PCA)
- Given D-dimensional data, we are interested in the K orthogonal directions that **contain most of the variance**.
- Alternative view: We want to find a K-dimensional subspace such that **the error that we make when projecting the data is minimal**.
- We can fix K, or pick K so that a fixed percentage of the total variance is preserved after reduction.

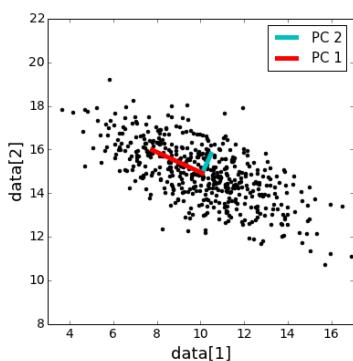
267

# PCA

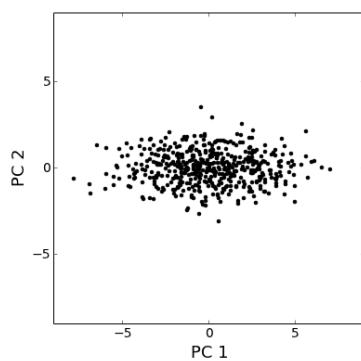
Original data



Principal Components



Transformed data



The first component  
explains 87% of the total  
variance, the second 13%.

268

## decomposition.PCA

- sklearn provides a PCA estimator in `decomposition.PCA`

## Interactive notebook demo

`Wind PCA.ipynb`

269

## Final notes about PCA

- PCA is sensitive to the scale of the features (e.g. transform meters to millimeters to increase variance by one million), so PCA is often preceded by normalization.
- In *whitening*, the transformed data is rescaled so that it has unit variance.

270

## Regression problems

272

# Regression

The goal of regression is to predict the value of *target* variables,  $t$ , given the value of *input* variables,  $\mathbf{x}$ .

Outline:

- Linear models
- Support Vector Regression
- Decision trees
- Overfitting and regularization
- Bias vs. variance

273

## Linear models for regression

- The vast majority of regression problems are based on a simple linear model.
- Simplest version: linear regression  
$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \dots + w_D x_D$$

For example:

$x_1$  = alcohol,  $x_2$  = pH,  $x_3$  = sulphates  
 $y$  = wine quality score

274

## Linear model, non-linear problem

- In general, we introduce non-linear *basis functions*

$$\phi_1(\mathbf{x}) \dots \phi_M(\mathbf{x})$$

and write

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

- Introducing a constant term,  $\phi_0(\mathbf{x}) = 1$ , we get rid of the “special” bias term

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x})$$

275

## Example: Polynomial regression

- For polynomial regression of order  $P$ , the basis functions correspond to all monomials up to that order.
- For example, for quadratic regression and  $D=2$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_1 x_2 + w_5 x_2^2$$

and so the basis functions are

$$\phi_0(\mathbf{x}) = 1, \phi_1(\mathbf{x}) = x_1, \phi_2(\mathbf{x}) = x_2$$

$$\phi_3(\mathbf{x}) = x_1^2, \phi_4(\mathbf{x}) = x_1 x_2, \phi_5(\mathbf{x}) = x_2^2$$

276

# Issues with polynomials

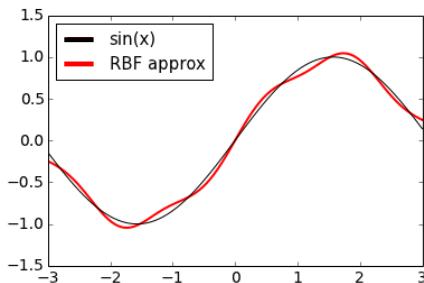
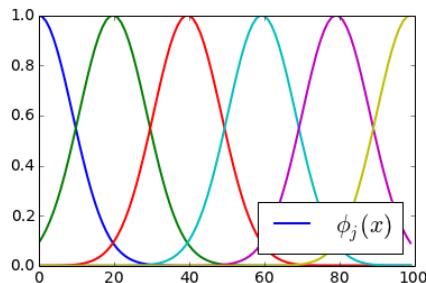
- The basis functions are global functions of  $x$ , which means that changes in one region affect all the coefficients (e.g., outliers)
- The number of basis functions grows very fast with  $D$  and  $P$ , in the order of  $O((P+1)^{(D-1)})$

277

# Example: Radial basis functions

- Gaussian-shaped, local basis functions
- Defined by centers,  $\mu_j$ , and width,  $\sigma$

$$\phi_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2\sigma^2}\right)$$



278

## Fitting a linear model

- Fit weights to minimize sum of square errors for  $N$  training data points

$$E_D(\mathbf{w}) = \sum_{n=1}^N \left( t_n - \underbrace{\mathbf{w}^T \phi(\mathbf{x}_n)}_{\text{Predicted output}} \right)^2$$

Expected output  
(Ground truth)

279

## Learning methods

- Batch learning: a closed-form solution exists, which allows to find a global optimum efficiently

```
from sklearn import linear_model
lm = linear_model.LinearRegression()
```

- Batch learning requires  $O(N^*D^2)$  space in memory
- For large data sets, we need to use an online learning method: using stochastic gradient descent

```
lm = linear_model.SGDRegressor()
```

280

# Evaluating the results

- Mean squared error (MSE),  $E_D(\mathbf{w})$ 
  - Available in `sklearn.metrics` as `mean_squared_error`
- $R^2$ , Coefficient of determination
  - Standard measure of “goodness-of-fit” in regression problems, ranges from 0 (bad) to 1 (good)
$$R^2 = 1 - \frac{\sum(t_i - y_i)^2}{\sum(t_i - \text{avg}(t))^2} = 1 - \frac{\text{unexplained variance}}{\text{total variance}}$$
  - Available as `.score(x, y)` method in the `sklearn` regression methods

281

# Support Vector Regressor

- Based on the theory of Support Vector Machines (more on this later).
- For the present purposes:
  - linear regression method, with non-linear basis functions.
  - scales with the number of samples, not the number of basis function (see *kernel trick*).

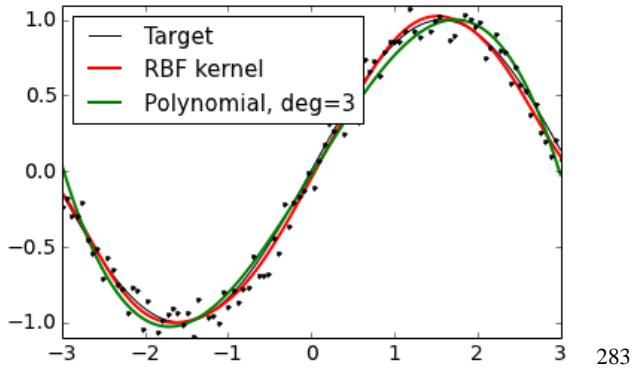
282

## sklearn.svm.SVR

```
from sklearn import svm

regr = svm.SVR(kernel='rbf', gamma=1.0, C=1., epsilon=0.1)
regr.fit(x, t)
rbf_t = regr.predict(x)

regr = svm.SVR(kernel='poly', degree=3, C=1., epsilon=0.1,
                coef0=1.0)
regr.fit(x, t)
poly_t = regr.predict(x)
```



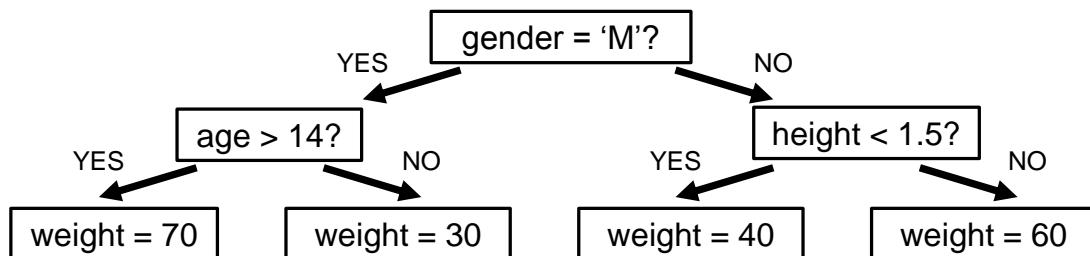
283

## SVR options

- Available basis functions (*kernels*):
  - `kernel='linear'`
  - `kernel='rbf'`  
`gamma=1/n_samples` controls the width of the RBFs
  - `kernel='poly'`  
`degree=3` controls the degree of the polynomial  
`coef0=1.0` controls the constant term in the polynomial
  - `kernel='sigmoid'`  
`gamma=1/n_samples` controls the width of the sigmoids
- For all kernels:
  - `C=1.0` regularization term (inverse, large=low reg, small=high reg)
  - `epsilon=0.1` tolerance region (where errors are ignored)

# Decision trees

- Decision trees predict the target value by applying a set of YES/NO rules to the feature variables



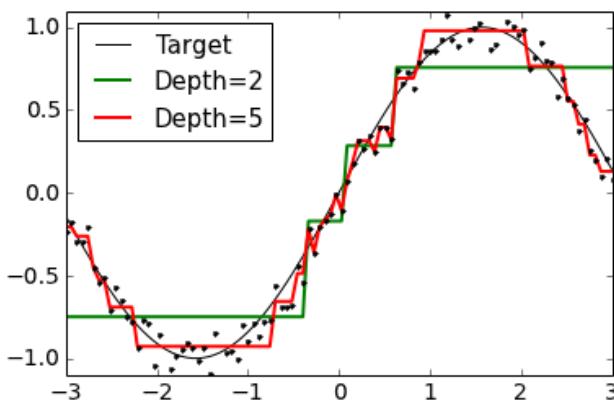
- The depth of the tree is the main parameter controlling the complexity of the model

285

## Example

```

from sklearn import tree
regr = tree.DecisionTreeRegressor(max_depth=2)
regr.fit(x, t)
predicted_t = regr.predict(x)
  
```



286

# Decision trees pros/cons

## Pros

- Handles categorical features as well as real-valued ones.
- Does not require much preprocessing.

## Cons

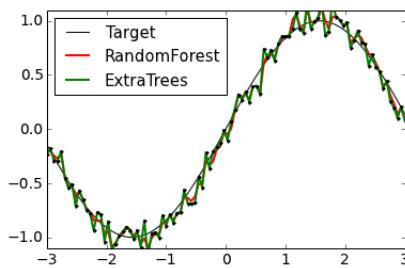
- The number of samples should roughly double for each level of the tree
- Likely to overfit if number of features is large
- Binary rules do not conceptually fit some problems, e.g. where the correlations arise from physical processes.

287

# Beyond trees: random forests

- Build a forest of decision trees learned on a subset of the data, and inject some randomness when learning the rules; prediction is average predictions over trees.
- Individual trees less accurate, compensated by averaging.
- Overfitting less likely due to bootstrapping, randomness.

```
from sklearn import ensemble
rf = ensemble.RandomForestRegressor()
et = ensemble.ExtraTreesRegressor()
```



288

# Regression demo

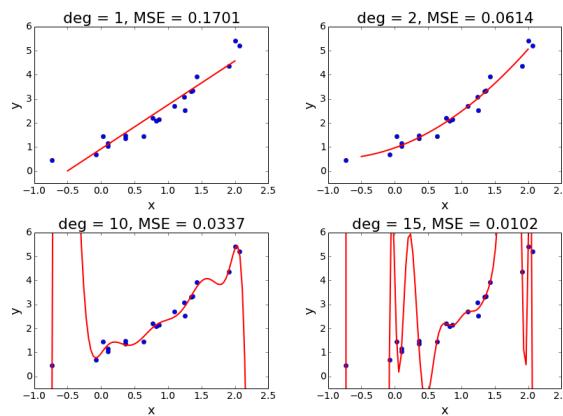
## Interactive notebook demo

Regression.ipynb

289

# Overfitting

Fitting a quadratic function with polynomial of increasing degree.



Increasing the number of parameters in a model will *always* improve the fit, but eventually we will start fitting noise.

290

# Regularization

- To avoid overfitting, we can add a penalty term, so that weights are only set if necessary:

$$E_D(\mathbf{w}) + \alpha E_W(\mathbf{w})$$

- $\alpha$  is called *regularization coefficient*
- Common regularization term is

$$E_W(\mathbf{w}) = \frac{1}{2} \sum_{j=1}^M w_j^2$$

for which a closed-form solution exists.

- In sklearn:

```
lm = linear_model.Ridge(alpha=0.5)
```

291

# Lasso

- Using the regularization term

$$E_W(\mathbf{w}) = \sum_{j=1}^M |w_j|$$

we obtain the *Lasso* linear model.

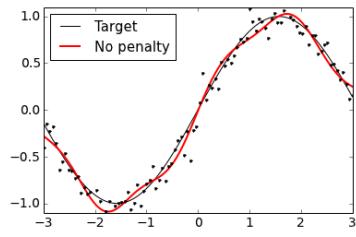
- The penalty is much stronger for small weights, leading to *sparse* solution with unnecessary weights set to exactly 0.

```
lm = linear_model.Lasso(alpha=0.1)
```

292

# Effect of regularization

No penalty

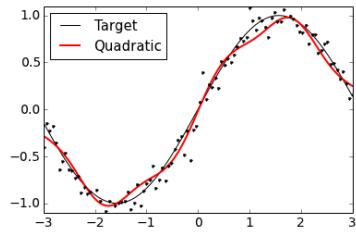


Weights: [-0.36 -1.17 -0.77  
0.4 0.8 -0.01]  
MSE: 0.014

Not symmetric, slightly overfitting.

Ridge

(quadratic penalty,  
 $\alpha=1.0$ )

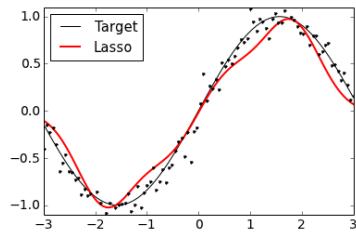


Weights: [-0.18 -0.94 -0.56,  
0.56 0.93 0.18]  
MSE: 0.015

Symmetric, all good.

Lasso

(linear penalty,  $\alpha=0.02$ )



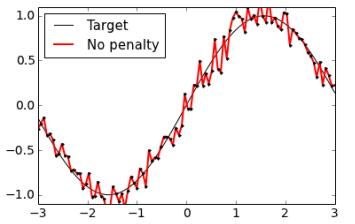
Weights: [-0.0 -0.94 -0.41  
0.41 0.94 0.0]  
MSE: 0.025

Sparse solution, at the cost  
of some accuracy.

293

# Overfitting revisited

The regularization term essentially limits  
the **effective model complexity**.



Sensitive to  
specific data set  
**High variance**

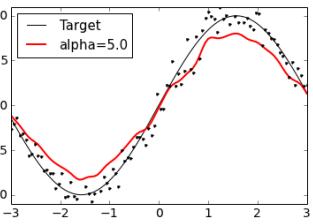
$$\mathbb{E}[(y(\mathbf{x}, \mathbf{w}) - \mathbb{E}[y(\mathbf{x}, \mathbf{w})])^2]$$

Individual  
solution

Average  
solution



Trade-off



Off-target  
**High bias**

$$(\mathbb{E}[y(\mathbf{x}, \mathbf{w})] - h(\mathbf{x}))^2$$

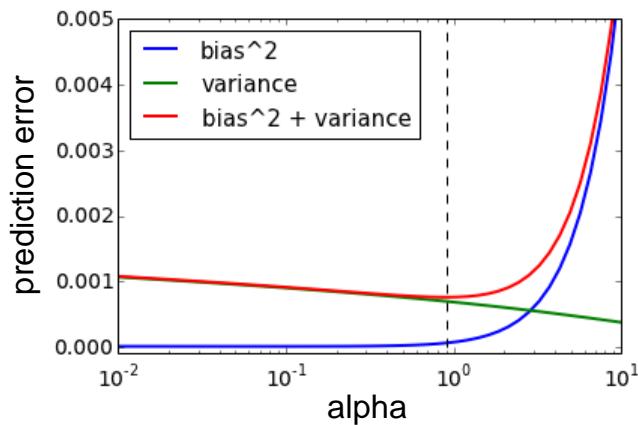
Average  
solution

Real  
function

294

## Bias/variance trade-off

The smallest prediction error will fall somewhere between the two regularization extrema of high variance and high bias.



In this example, the predicted error is minimal for  $\alpha=0.91$ .

295

## Overfitting “solutions”

- Get more data (the “Big Data” solution)  
*Can be impossible*
- Limit the complexity of the model  
(e.g., number of basis functions)  
*Inflexible function spaces*
- Set regularizer  
*Which value is optimal?*

Selecting the right model complexity is the goal of model selection.

296

# Model Selection

298

## Definition

Model selection:

- Tune *complexity parameters*, or *hyperparameters* (number of basis functions, regularization parameter, etc.)
- Compare different kind of models

Goal: achieve the best **predictive performance**

299

# Train, test, and validation sets

- When adjusting *hyperparameters*, there is the risk of overfitting over the test set: another multiple comparisons problem!
- One solution is to put aside a third set of data, called the *validation set*.
  - Train on the *training set*
  - Tune on the *validation set*
  - Evaluate on the *test set* once the hyperparameters are fixed
- Issue: shortage of data!

300

# Cross-validation

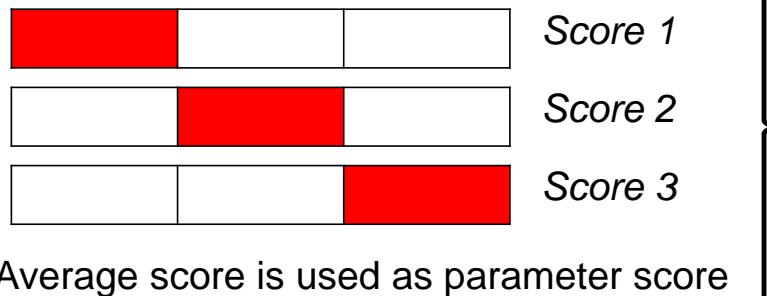
- The training data is partitioned several times in two non-overlapping sets; one is used for training, the second for validation.
- Still need a test set for the final evaluation, but don't need a dedicated validation set any longer.
- We can use more of the data, at the cost of multiple estimations.

301

# Cross-validation



Multiple evaluations on partitions



Average score is used as parameter score

Repeat for  
each setting  
of the  
complexity  
parameters

Test data

Final evaluation  
on test data

302

# K-Fold

- In the *K-Fold* method, we partition the data in K contiguous groups

```
from sklearn.cross_validation import KFold
data = np.array(['A', 'B', 'C', 'D', 'E', 'F'])

# K=3
cv = KFold(len(data), n_folds=3)
for train_idx, test_idx in cv:
    print 'Train', data[train_idx], '; Test', data[test_idx]

Train ['C' 'D' 'E' 'F'] ; Test ['A' 'B']
Train ['A' 'B' 'E' 'F'] ; Test ['C' 'D']
Train ['A' 'B' 'C' 'D'] ; Test ['E' 'F']
```

303

## Leave-one-out

- For K=N, we obtain the leave-one-out method

```
from sklearn.cross_validation import LeaveOneOut
data = np.array(['A', 'B', 'C', 'D', 'E', 'F'])

cv = LeaveOneOut(len(data))
for train_idx, test_idx in cv:
    print 'Train', data[train_idx], '; Test', data[test_idx]

Train ['B' 'C' 'D' 'E' 'F'] ; Test ['A']
Train ['A' 'C' 'D' 'E' 'F'] ; Test ['B']
Train ['A' 'B' 'D' 'E' 'F'] ; Test ['C']
Train ['A' 'B' 'C' 'E' 'F'] ; Test ['D']
Train ['A' 'B' 'C' 'D' 'F'] ; Test ['E']
Train ['A' 'B' 'C' 'D' 'E'] ; Test ['F']
```

304

## Shuffle split

- *Shuffle split*, splits the data into random partitions
- Better control on number of iterations, but some examples might be reused / not used at all

```
from sklearn.cross_validation import ShuffleSplit
data = np.array(['A', 'B', 'C', 'D', 'E', 'F'])

cv = ShuffleSplit(len(data), n_iter=3, test_size=0.25)
for train_idx, test_idx in cv:
    print 'Train', data[train_idx], '; Test', data[test_idx]

Train ['B' 'F' 'D' 'A'] ; Test ['C' 'E']
Train ['B' 'C' 'D' 'F'] ; Test ['E' 'A']
Train ['D' 'A' 'E' 'F'] ; Test ['B' 'C']
```

305

# Support for cross-validation

`cross_val_score` fits an estimator on cross-validation sets and evaluates the results using a scoring function

```
>>> from sklearn.cross_validation import cross_val_score
# By default, KFold is used for CV.
# The scores signs are flipped so larger is better.
>>> K = 4
>>> cross_val_score(estimator, x, t, cv=K,
                     scoring='mean_squared_error')
array([-0.01427696, -0.01344068, -0.01461322, -0.01378299])

# One can specify an alternative CV method
>>> cv = ShuffleSplit(len(t), n_iter=6, test_size=0.25)
>>> scores = cross_val_score(estimator, x, t, cv=cv,
                           scoring='mean_squared_error')
```

306

# Available scoring functions

Scoring	Function
<b>Classification</b>	
'accuracy'	<code>sklearn.metrics.accuracy_score</code>
'average_precision'	<code>sklearn.metrics.average_precision_score</code>
'f1'	<code>sklearn.metrics.f1_score</code>
'precision'	<code>sklearn.metrics.precision_score</code>
'recall'	<code>sklearn.metrics.recall_score</code>
'roc_auc'	<code>sklearn.metrics.roc_auc_score</code>
<b>Clustering</b>	
'adjusted_rand_score'	<code>sklearn.metrics.adjusted_rand_score</code>
<b>Regression</b>	
'mean_squared_error'	<code>sklearn.metrics.mean_squared_error</code>
'r2'	<code>sklearn.metrics.r2_score</code>

It is also possible, but rarely necessary, to define a custom score function.

307

# Models with integrated CV

- A small number of models have built-in CV:

```
>>> lm = linear_model.RidgeCV(alphas=alphas)
>>> lm.fit(phi_x, t)
>>> print 'optimal alpha =', lm.alpha_
optimal alpha = 0.790
```

```
>>> lm = linear_model.LassoCV(alphas=alphas)
>>> lm.fit(phi_x, t)
>>> print 'optimal alpha =', lm.alpha_
optimal alpha = 0.00054
```

308

# Demo

## Interactive notebook demo

Cross-validated regularization.ipynb

Grid search cross-validation.ipynb

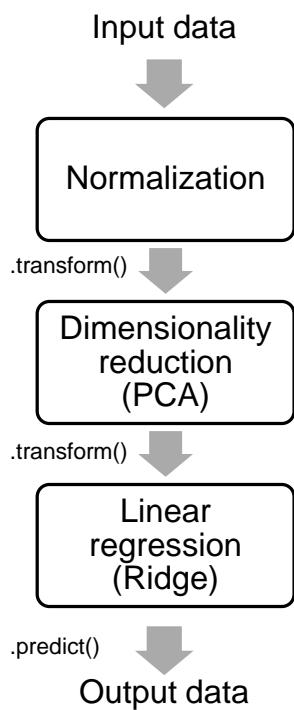
309

# Convenience objects

- Pipeline: treat entire data analysis pipelines as a single estimator
- GridSearchCV: automatic cross-validation on grids of parameters

310

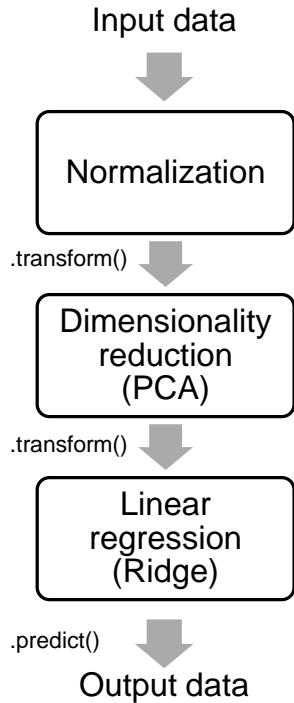
# Creating a Pipeline



For data analysis workflows requiring multiple steps, sklearn provides a “Pipeline” object to group the estimator together, and minimize bookkeeping.

311

# Creating a Pipeline



```

from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR

# Each element in the pipeline is a tuple
# ('name', estimator)
pipeline = Pipeline(
    [('scaler', StandardScaler()),
     ('pca', PCA(n_components=0.9)),
     ('regressor', SVR(C=100.0, gamma=0.1))])

# The pipeline can be used as a single estimator.
pipeline.fit(x_train, t_train)
y_test = pipeline.predict(x_test)
print 'R2 = ', pipeline.score(x_test, t_test)

# How to refer to a single step in the pipeline?
pipeline.step[1]
pipeline.named_steps['pca']
    
```

312

# Cross-validation on a grid

GridSearchCV can be used to easily define a cross-validation workflow on a full grid of hyperparameters.

Each grid is represented by a dictionary:

	kernel = 'rbf'		
c \ gamma	0.1	0.01	
1	'rbf', 0.1, 1	'rbf', 0.01, 1	
10	'rbf', 0.1, 10	'rbf', 0.01, 10	
100	'rbf', 0.1, 100	'rbf', 0.01, 100	

```
{'kernel': ['rbf'], 'gamma': [0.1, 0.01], 'C': [1, 10, 100]}
```

313

# Cross-validation on a grid

`GridSearchCV` runs a cross-validation iteration for each combination of parameters in a set of grids.

```
from sklearn.grid_search import GridSearchCV

# Multiple grids can be specified in a list.
params_grid = [
    {'kernel': ['linear'], 'C': [1, 10, 100]},
    {'kernel': ['rbf'], 'gamma': [0.1, 0.01],
     'C': [1, 10, 100]}
]
estimator = SVR()
grid_search = GridSearchCV(estimator, params_grid, cv=4)

# Run the cross-validation iterations.
grid_search.fit(x, t)

# After the run, grid_search can be used as a proxy for
# the estimator with the best combination of parameters.
y_test = grid_search.predict(new_x)
```

314

# Cross-validation on a grid

After a run, a set of attributes of the `GridSearchCV` object holds the result of the cross-validation iterations:

- `grid_scores_`** List of score objects for all parameter combinations. Each object has the attributes:
  - `parameters`: dict of parameter settings
  - `mean_validation_score`: mean score over the cross-validation folds
  - `cv_validation_scores`: list of scores for each fold
- `best_estimator_`** Estimator that had the best score on the left out data (re-fitted on the whole training data).
- `best_params_`** Dict holding the parameters for the best estimator.
- `best_score_`** Score of `best_estimator` on the left out data. (Not the final score after re-fitting.)

315

## Cross-validation on a grid

The combination of the Pipeline and GridSearchCV objects allows optimizing complex workflows.

```
pipeline = Pipeline([
    ('scaler', preprocessing.StandardScaler()),
    ('regressor', SVR())]

# A naming convention is used to refer to the parameters
# of a specific estimator in the pipeline.
param_grid = [ {'regressor__kernel': ['rbf'],
                'regressor__gamma': gammas,
                'regressor__C': Cs}]

grid_search = GridSearchCV(pipeline , param_grid, cv=4)
grid_search.fit(x_train, t_train)
```

316

## Classification problems

317

# Classification

The goal of classification is to assign input vectors,  $\mathbf{x}$ , to one of  $K$  discrete and disjoint classes  $C_k$ .

2 2 2 2 2 2	→	2
3 3 3 3 3 3	→	3
4 4 4 4 4 4	→	4
5 5 5 5 5 5	→	5

*Handwritten digits classification*

Other examples: *spam classification; language identification; face recognition*

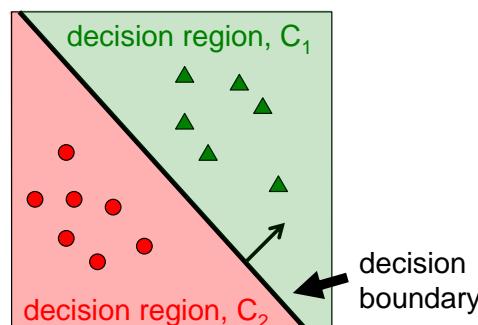
318

# Linear classifiers

- The basic linear model for binary classification is

$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + w_0)$$

- $f$  is a non-linear *activation function*
- Input  $\mathbf{x}$  is classified in  $C_1$  if  $y(\mathbf{x}) \geq 0$ , else in  $C_2$
- The *decision boundary*  $y(\mathbf{x}) = 0$  is linear



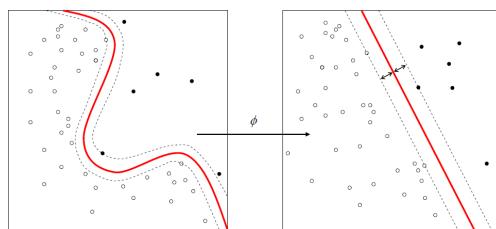
319

# Basis functions

- As for regression, we can use a linear model for non-linear classification by introducing M basis functions  $\phi_0(\mathbf{x}) = 1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots$

$$y(\mathbf{x}) = f(\mathbf{w}^T \Phi(\mathbf{x}))$$

- A key idea is that a linear decision boundary in the basis function space is equivalent to a non-linear boundary in the original input space

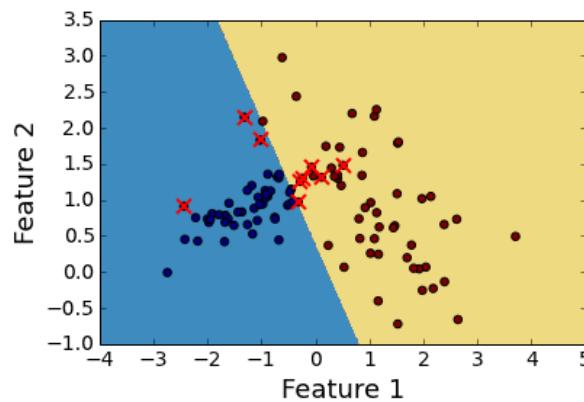


320

# Learning linear classifiers

- Parameters are learned by minimizing a classification error function
- Efficient linear learning implementation available in `linear_models.SGDClassifier()`

```
from sklearn import linear_model
clf = linear_model.SGDClassifier(n_iter=500)
clf.fit(x, t)
y = clf.predict(x)
```



321

# Multiple classes, K>2

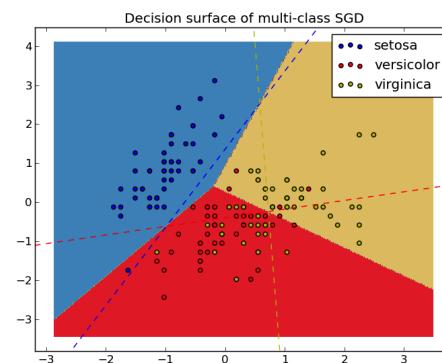
- **One-vs-all:** Train classifiers fitting  $C_k$  against  $C_{j \neq k}$ 
  - Relatively efficient (requires fitting K classifiers)
  - Easy to interpret: one can examine the classifier for class  $k$  to learn about that class
  - Most common strategy
- **One-vs-one:** Train one classifier for each pair of classes; take a majority vote
  - Requires fitting  $K * (K-1) / 2$  classifiers
  - Scales better since each classifier is learned on a subset of the input data

322

## Example

SGDClassifier supports one-vs-all classification

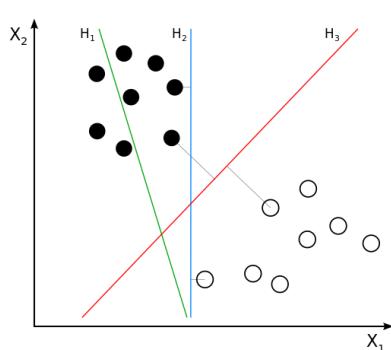
```
iris = datasets.load_iris()
x = iris.data[:, :2]
t = iris.target
clf = SGDClassifier(n_iter=100)
clf.fit(x, t)
y = clf.predict(x)
```



323

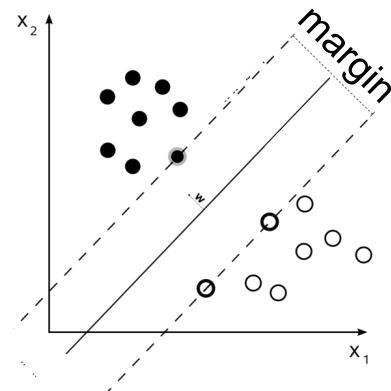
# Maximum margin classifiers

An infinite number of decision boundary may separate two classes



$H_2$  and  $H_3$  both separate the black and the white classes

The most robust boundary (in terms of predictive error) is the one with maximum *margin*

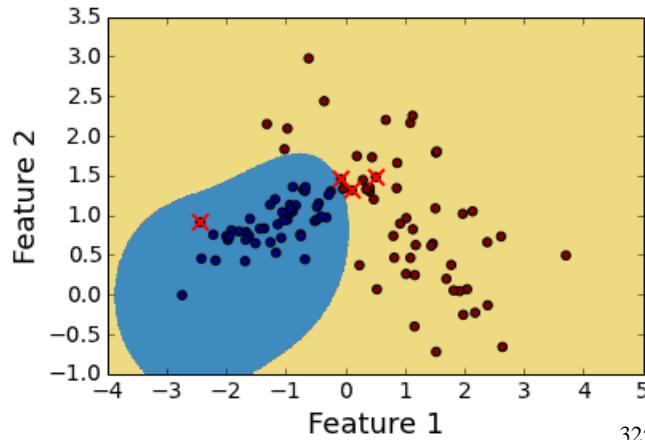


324

# Support Vector Machines

- Support Vector Machines
  - Maximum margin classifiers
  - Use the so-called *kernel trick* to efficiently handle basis functions (including infinite-dim basis function spaces)

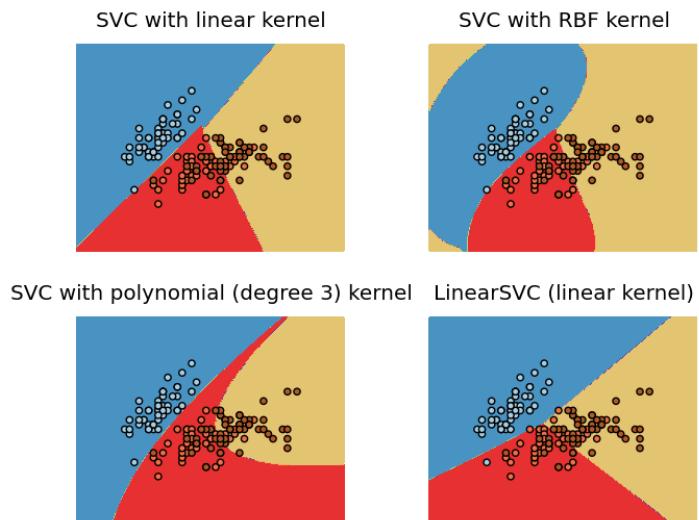
```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(x, t)
y = clf.predict(x)
```



325

# Multi-class SVM

SVC supports one-vs-one classification. sklearn also contains an implementation based on another C++ library, LinearSVC, which supports one-vs-all classification



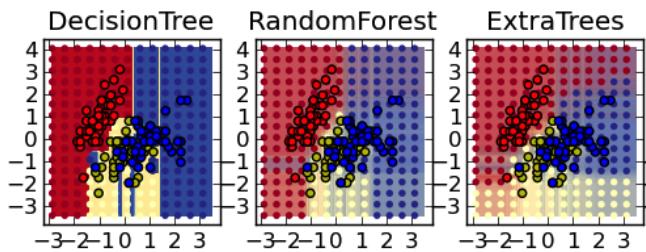
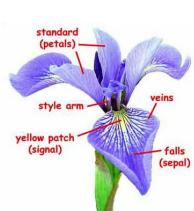
326

# Decision trees

Just like for regression, decision trees can be used as rule-based classifiers

```
from sklearn import tree
from sklearn import ensemble

clf = tree.DecisionTreeClassifier(max_depth=3)
clf = ensemble.RandomForestClassifier(n_estimators=10)
clf = ensemble.ExtraTreesClassifier(n_estimators=10)
```



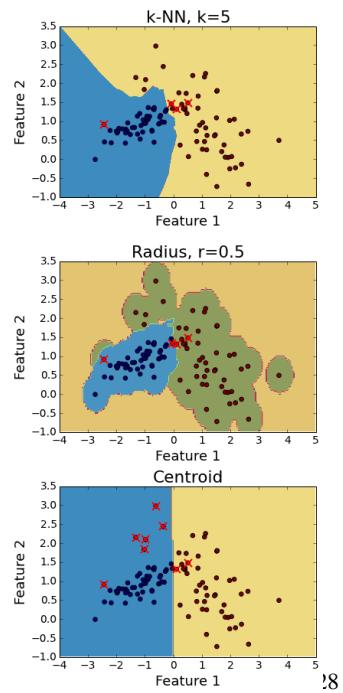
327

# Instance-based classifiers

- Compare new samples with the training samples, or a representative set of samples.
- Nearest-neighbors classifiers: classify sample based on the label of the nearest training samples.

```
from sklearn import neighbors

clf = neighbors.KNeighborsClassifier(
    n_neighbors=5)
clf = neighbors.RadiusNeighborsClassifier(
    radius=0.5, outlier_label=3)
clf = neighbors.NearestCentroid()
clf.fit(x, t)
y = clf.predict(x)
```



!8

## Demo

### Interactive notebook demo

Classification.ipynb

# Evaluating classifiers (binary)

		Predicted	
		P	N
True	P	True positive (TP)	False negative (FN)
	N	False positive (FP)	True negative (TN)

*confusion  
matrix*

Measure	Formula	Interpretation
Precision	$TP / (TP + FP)$	Percentage of P predictions that were correct.
Recall / Sensitivity	$TP / (TP + FN)$	Percentage of P samples that were predicted as P.
Accuracy	$(TP + TN) / \text{Total}$	Percentage of correct predictions.

330

## Example: Diagnostic test

		Predicted	
		P	N
True	P	500	100
	N	500	10 000

P: has Ebola

N: has not

Measure	Formula	Interpretation
Precision	50%	Percentage of P predictions that were correct.
Recall / Sensitivity	83%	Percentage of P samples that were predicted as P.
Accuracy	95%	Percentage of correct predictions.

331

# Evaluating classifiers (multi-class)

- Standard measures of performance:
  - **Accuracy:** Fraction of correct predictions  
`.score(x, y_true)` method of classifiers objects
  - **Precision:** Average precision over classes, weighted by number of true instances of each class.  
**Available in** `sklearn.metrics` **as**  
`precision_score(y_true, y_pred, average='weighted')`
  - **Recall:** Average recall over classes, weighted by number of true instances of each class.  
**Available in** `sklearn.metrics` **as**  
`recall_score(y_true, y_pred, average='weighted')`
  - `sklearn.metrics.confusion_matrix(y_true, y_pred)`  
**returns the confusion matrix for inspection**

332

## Example

```

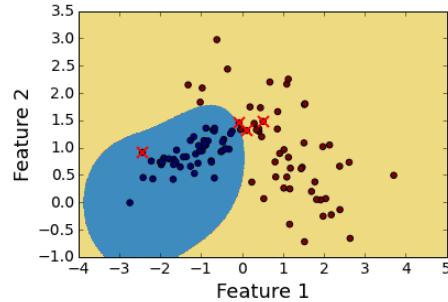
from sklearn import metrics

svc = svm.SVC(kernel='rbf')
svc.fit(x, t)
y = svc.predict(x)

print 'Confusion matrix'
print metrics.confusion_matrix(t, y)
Confusion matrix
[[46  3]
 [ 1 50]]

print 'Accuracy =', svc.score(x, t)
print 'Precision =', metrics.precision_score(t, y)
print 'Recall =', metrics.recall_score(t, y)
Accuracy = 0.96
Precision = 0.943396226415
Recall = 0.980392156863

```



333

# Parallel sklearn

335

## sklearn tasks on multiple cores

Tasks that are embarrassingly parallel in sklearn offer an optional argument, `n_jobs`, that represents the number of parallel processes on which to run the task.

Notable examples:

- Classification and regression using random forests
- Performing cross-validation on a grid of parameters with `GridSearchCV`
- SGD classification in multi-class problems (one vs all)

336

# Demo

## Interactive notebook demo

Parallel Wine Regression.ipynb

337

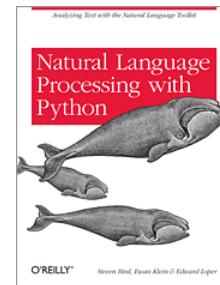
## Natural Language Processing

338

# Python packages for NLP

## Natural Language ToolKit (NLTK 3.0)

- Mature, leading library for natural language processing
- Text normalization and segmentation
- Part-Of-Speech tagging
- Grammatical and semantic analysis
- Easy access to corpora and lexical resources



## gensim

- Topic modeling

## Other useful packages:

- FuzzyWuzzy (Approximate string matching)
- Jellyfish , Metaphone (Phonetic analysis)

339

# Before we start...

Download all corpora and lexical resources for NLTK with:

```
import nltk
nltk.download_shell()
```

then press 'd' and download the following corpora:

```
movie_reviews
stopwords
wordnet
book
```

340

# Text modeling workflow

## 1. Preparing text

- Tokenization
- Normalization of words, removing punctuation and stopwords
- Stemming

} nltk

## 2. Extract features

- “Bag of words” counts, binary
- TF-IDF
- POS tags, semantic features

} sklearn  
} nltk

## 3. Classification / Topic modeling

- Naïve Bayes, Decision Trees
- LDA, HDP

} sklearn  
} gensim

341

# Preparing text

342

# Preparing text (1)

```
text = (`There lie the woods of Lothl\xc3\xb3rien! ' said Legolas. "
      ``That is the fairest of all the dwellings of my people. `")
```

## 1. DECODE TO UNICODE

```
>>> text = text.decode('utf8')
>>> print text
`There lie the woods of Lothlórien! ' said Legolas. `That is the
fairest of all the dwellings of my people.'
```

## 2. TOKENIZE

```
>>> import nltk
>>> sentences = nltk.sent_tokenize(text)
[u`There lie the woods of Lothl\xf3rien!', 
 u''' said Legolas.", 
 u``That is the fairest of all the dwellings of my people.'']
>>> tokens = [w for sent in sentences
              for w in nltk.word_tokenize(sent)]
>>> print tokens
[u`There', u'lie', u'the', u'woods', u'of', u'Lothl\xf3rien',
u'!', u'''', u'said', u'Legolas', u'.', u``That', u'is', u'the',
u'fairest', u'of', u'all', u'the', u'dwellings', u'of', u'my',
u'people.'']
```

343

# Tokenizers

NLTK provides several tokenizers:

1. **nltk.word\_tokenize(text)**  
Recommended, most complex, works on single sentences.
2. **nltk.wordpunct\_tokenize(text)**  
Extracts entire words and punctuation (30x faster).
3. **nltk.regexp\_tokenize(text, '\w+')**  
Simple, extracts words and discards punctuation.

(More in `nltk.tokenize`)

	<code>word_tokenize</code>	<code>wordpunct_tokenize</code>	<code>regexp, '\w+'</code>
<b>X-ray</b>	=> ['X-ray']	['X', '-', 'ray']	['X', 'ray']
<b>Dr. Seuss</b>	=> ['Dr.', 'Seuss']	['Dr', '.', 'Seuss']	['Dr', 'Seuss']
<b>gotta</b>	=> ['got', 'ta']	['gotta']	['gotta']
<b>won't</b>	=> ['wo', "n't"]	['won', "'", 't']	['won', 't']
<b>we'll</b>	=> ['we', "ll"]	['we', "'", 'll']	['we', 'll']

# Preparing text (2)

```
text = ("`There lie the woods of Lothl\xc3\xb3rien! ' said Legolas. "
       "`That is the fairest of all the dwellings of my people. `")
```

## 3. NORMALIZE TEXT

```
>>> # Normalize case
>>> [w.lower() for w in tokens]
[u'`there', u'lie', u'the', u'woods', u'of', u'lothl\xf3rien', u'!',
 u'!', u'said', u'legolas', u'.', u'`that', u'is', u'the', u'fairest',
 u'of', u'all', u'the', u'dwellings', u'of', u'my', u'people.`']

>>> # Remove punctuation
>>> from string import punctuation
>>> [w.lower() for w in tokens if w not in punctuation]
[u'`there', u'lie', u'the', u'woods', u'of', u'lothl\xf3rien',
 u'said', u'legolas', u'`that', u'is', u'the', u'fairest', u'of',
 u'all', u'the', u'dwellings', u'of', u'my', u'people.']
>>> normalized =[w.strip(punctuation).lower()
                  for w in tokens if w not in punctuation]
[u'there', u'lie', u'the', u'woods', u'of', u'lothl\xf3rien',
 u'said', u'legolas', u'that', u'is', u'the', u'fairest', u'of',
 u'all', u'the', u'dwellings', u'of', u'my', u'people']
```

345

# Preparing text (3)

```
text = ("`There lie the woods of Lothl\xc3\xb3rien! ' said Legolas. "
       "`That is the fairest of all the dwellings of my people. `")
```

## 4. REMOVE STOPWORDS

*Stopwords* are words that occur with high frequency in a language.

```
>>> from nltk.corpus import stopwords
>>> clean = [w for w in normalized
              if w not in stopwords.words('english')]
[u'lie', u'woods', u'lothl\xf3rien', u'said', u'legolas', u'fairest',
 u'dwellings', u'people']
```

## 5. STEMMING

```
>>> stemmer = nltk.PorterStemmer()
>>> print [stemmer.stem(w) for w in clean]
[u'lie', u'wood', u'lothl\xf3rien', u'said', u'legola', u'fairest',
 u'dwell', u'peopl']
```

346

# Stemmers

Stemming is the process of reducing a word to a “base form”. It is not a well-defined process, and different stemmers will provide slightly different answers.

For example:

```
>>> p = nltk.PorterStemmer()
>>> [p.stem(w) for w in ['multiply', 'multiplied', 'multiplication']]
[multipli, multipli, multipli]
>>> [p.stem(w) for w in ['lie', 'lies', 'lying']]
[lie, lie, lie]

>>> l = nltk.LancasterStemmer()
>>> [l.stem(w) for w in ['multiply', 'multiplied', 'multiplication']]
[multiply, multiply, multiply]
>>> print [l.stem(w) for w in ['lie', 'lies', 'lying']]
[lie, lie, lying]
```

More stemmers are available in `nltk.stem`.

347

# Extracting text features

348

# Extracting features

The most common feature set used to classify a text is known as “**bag of words**”: the order of the words is discarded, and the word count is retained.

CountVectorizer extracts tokens from a list of documents, and transforms each document into a vector of counts or binary indicators for each token.

```
from sklearn.feature_extraction.text import CountVectorizer
CountVectorizer(
    stop_words='english', ← Remove stop words if set to 'english'.
    lowercase=True, ← Transform all words to lowercase?
    binary=False, ← Return a word counts vector if False, or else a
)                                binary vector indicating which words appear in
                                the document.
```

See the documentation for many more options.

349

# Common usage

```
documents = [
    'TITLE: The Tiger Snake is a very venomous and dangerous snake.',
    'TITLE: Python is a great programming language!',
    'TITLE: A Python is a constricting snake.',
    'TITLE: Perl used to be a popular language among geeks.',
]

vectorizer = CountVectorizer(stop_words='english',
                            lowercase=True, binary=False)
features = vectorizer.fit_transform(documents)

print vectorizer.get_feature_names()
[u'constricting', u'dangerous', u'geeks', u'great', u'language',
u'perl', u'popular', u'programming', u'python', u'snake', u'tiger',
u'title', u'used', u'venomous']

print features.toarray() # toarray(): `features` is a sparse matrix
[[0 1 0 0 0 0 0 0 2 1 1 0 1]
 [0 0 0 1 1 0 0 1 1 0 0 1 0 0]
 [1 0 0 0 0 0 0 1 1 0 1 0 0 0]
 [0 0 1 0 1 1 1 0 0 0 0 1 1 0]]
```

350

# Customizing the vectorizer

```
new_doc = 'TITLE: Pythons are non-venomous snakes, they
constrict!'
vectorizer.transform([new_doc]).toarray()
[[0 0 0 0 0 0 0 0 0 1 0 1]]
```

The vectorizer found “title” and “venomous”, but missed “Pythons” “snakes”, and “constrict”! We need to use a stemmer.

```
def stemming_analyzer(raw):
    tokens = nltk.wordpunct_tokenize(raw)
    # ... Normalize and remove stopwords ...
    return [stemmer.stem(w) for w in tokens]

vectorizer = CountVectorizer(analyzer=stemming_analyzer)
features = vectorizer.fit_transform(documents)
print vectorizer.get_feature_names()
['among', 'constrict', 'danger', 'geek', 'great', 'languag',
'perl', 'popular', 'program', 'python', 'snake', 'tiger', 'titl',
'use', 'venom']

vectorizer.transform([new_doc]).toarray()
```

351

# TF-IDF features

All the documents in our example started with “TITLE”. In most cases, we want to discount very common terms, as they are not informative.

**TF-IDF (Term Frequency-Inverse Document Frequency)** is a common statistics used to reflect how important a word is in a document.

TF(word) = Count of word in doc / Count of all words in doc

IDF(word) =  $\log(\frac{\text{Number of docs}}{\text{Number of docs containing word}})$

TF-IDF = TF \* IDF

This is a heuristic formula, there are several variants (accessible through optional arguments).

352

# TF-IDF vectorizer

sklearn provides a TF-IDF vectorizer:

```
vectorizer = TfidfVectorizer(analyzer=stemming_analyzer)
features = vectorizer.fit_transform(documents)

zip(vectorizer.idf_, vectorizer.get_feature_names())
[...
(1.5108256237659907, 'python'),
(1.5108256237659907, 'snake'),
(1.9162907318741551, 'tiger'),
(1.0, 'titl'),
(1.9162907318741551, 'use'),
(1.9162907318741551, 'venom')]

vectorizer.transform([new_doc]).toarray()
[ 0, 0.53 0, 0, 0, 0, 0, 0, 0.42, 0.42, 0, 0.28, 0, 0.53]

# For reference, the CountVectorizer returned:
# [[0 1 0 0 0 0 0 0 1 1 0 1 0 1]]
```

353

# Other text features

Alternative / complementary text features:

- Treat collocations as single terms  
*New York, Dow Jones, ice cream, red wine*
- Part-Of-Speech frequency  
*E.g., for sentiment analysis: more emotional text uses more adjective, verbs*
- Use semantics to smooth differences in vocabulary  
*E.g., word2vec, hypernyms*

```
>>> from nltk.corpus import wordnet
>>> motorcar = wordnet.synset('car.n.01')
>>> truck = wordnet.synset('truck.n.01')
>>> print motorcar.hypernyms()
[Synset('motor_vehicle.n.01')]
>>> print truck.hypernyms()
[Synset('motor_vehicle.n.01')]
```

354

# Text classification

355

## Text classification

Suitable for text classification:

- Decision-tree-based classifiers
- Naïve Bayes classifier

These classifiers handle discrete and binary distributions effortlessly, and are thus ideal to use with Bag-Of-Words features.

356

# Naive Bayes classifiers

We want to classify a document based on its words content.

*E.g., classify a movie in a good or bad category, based on a review.*

Applying Bayes' rule:

$$P(\text{class}|\text{words}) = \frac{1}{Z} P(\text{class}) P(\text{words}|\text{class})$$

Prior, frequency of class in training set
Likelihood, we'll never have enough data to estimate this term!

357

# Naive Bayes classifiers

We want to classify a document based on its set of words.

*E.g., classify a movie in a good or bad category, based on a review.*

Applying Bayes' rule:

$$P(\text{class}|\text{words}) = \frac{1}{Z} P(\text{class}) P(\text{words}|\text{class})$$

The "Naive" part: we assume all the words are independent, given a class
Prior, again
These terms are now easy to estimate

$$\approx \frac{1}{Z'} P(\text{class}) \prod_i P(\text{word}_i|\text{class})$$

358

# Naive Bayes classifiers

We want to classify a document based on its set of words.

*E.g., classify a movie in a good or bad category, based on a review.*

Applying Bayes' rule:

$$P(\text{class}|\text{words}) = \frac{1}{Z} P(\text{class}) P(\text{words}|\text{class})$$

The “Naive” part: we assume all the words are independent, given a class →

$$\approx \frac{1}{Z'} \underbrace{P(\text{class})}_{\text{Prior, again}} \prod_i \underbrace{P(\text{word}_i|\text{class})}_{\text{These terms are now easy to estimate}}$$

*Smoothing parameter:* to avoid probability 0 for a word that has not been observed in a class, add a small constant to all word frequencies.

359

# Naive Bayes in sklearn

In `sklearn.naive_bayes`:

- **MultinomialNB** : For count data, in practice used with TF-IDF features, as well
- **BernoulliNB** : For binary data (e.g., word indicators)
- **GaussianNB** : For continuous variables. Each of the likelihood terms is assumed to be a Gaussian distribution.

360

# Naive Bayes in action (1)

## FITTING THE CLASSIFIER TO OUR EXAMPLE

```
from sklearn.naive_bayes import MultinomialNB

documents = [
    'TITLE: The Tiger Snake is a very venomous and dangerous snake.',
    'TITLE: Python is a great programming language!',
    'TITLE: A Python is a constricting snake.',
    'TITLE: Perl used to be a popular language among geeks.',
]
labels = ['snakes', 'cs', 'snakes', 'cs']

vectorizer = TfidfVectorizer(stop_words='english')
features = vectorizer.fit_transform(documents)
classifier = MultinomialNB()
classifier.fit(features, labels)
```

361

# Naive Bayes in action (2)

## MOST RELEVANT FEATURES

```
feature_names = np.asarray(vectorizer.get_feature_names())
for idx, class_ in enumerate(classifier.classes_):
    order = np.argsort(-classifier.feature_log_prob_[idx])
    top5 = feature_names[order[:5]]
    print 'Top 5 features for "{}":\n{}'.format(class_, top5)
Top 5 features for "cs":
[u'language' u'great' u'programming' u'title' u'geeks']
Top 5 reatures for "snakes":
[u'snake' u'constricting' u'title' u'python' u'dangerous']
```

## PREDICTING ON NEW DOCUMENTS

```
new_docs = [
    'TITLE: Pythons are non-venomous snakes, they constrict!',
    'TITLE: Python vs Perl -- war in geek-land'
]

new_features = vectorizer.transform(new_docs)
print classifier.predict(new_features)
['snakes', 'cs']
```

362

# Demo

## Interactive notebook demo

Movie\_reviews\_classification.ipynb

363

## Topic modeling

364

# Topic modeling with gensim

Topic models are machine learning algorithms for discovering a set of topics underlying a set of text documents.

The `gensim` library contains several topic modeling methods, including:

- Latent Dirichlet Allocation (LDA)
- Hierarchical Dirichlet Process (HDP)

365

## LDA, HDP models

- In the **Latent Dirichlet Allocation (LDA)** model, each document is assumed to be derived from a small set of topics, each associated with a distribution of words.
- The **Hierarchical Dirichlet Process (HDP)** model is an extension of LDA, in which the overall number of topics does not need to be specified: it is learned as a parameter, and is potentially infinite.

366

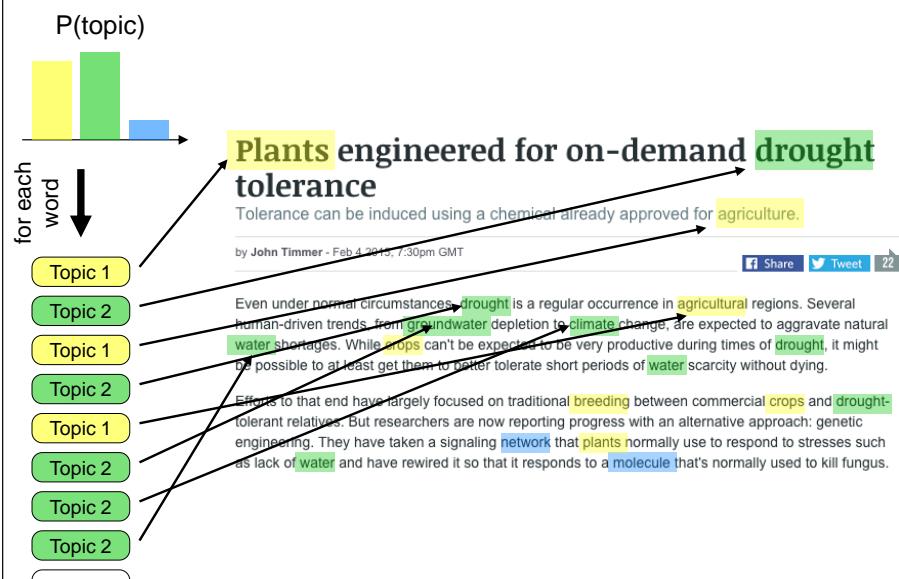
# LDA model

for each document

Topic 1  
0.09 plants  
0.03 agriculture  
0.01 crops  
...

Topic 2  
0.05 water  
0.04 climate  
0.02 drought  
...

Topic 3  
0.05 neuron  
0.03 brain  
0.03 network  
...



367

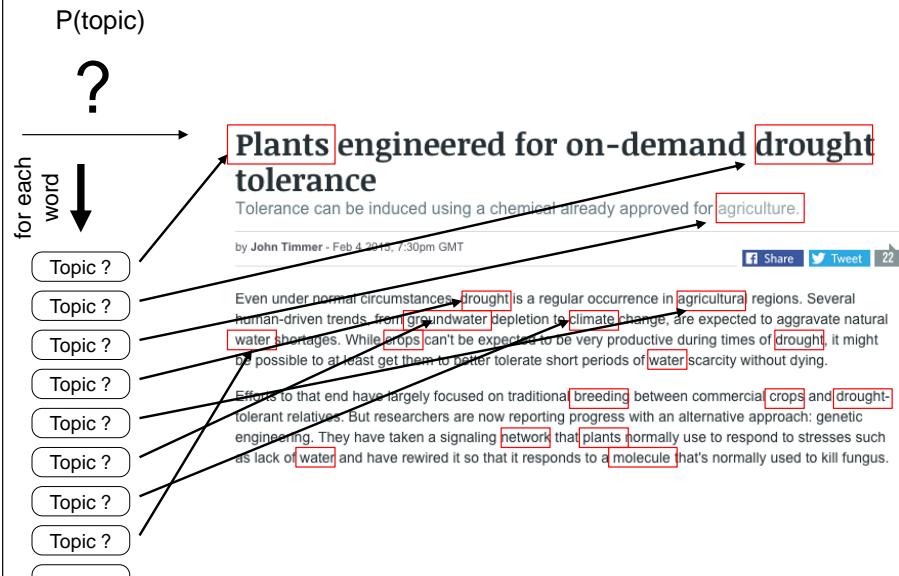
# LDA learning problem

for each document

Topic 1  
?

Topic 2  
?

Topic 3  
?



368

# Demo

## Interactive notebook demo

Topic\_modeling.ipynb

369

# Application of topic models

Topic models are used to:

- Visualize and navigate documents
- Search/classify documents: compute similarity in topics space, rather than words space
- First, automatic pass for manual classification

370

## Stay in touch!



@enthought



facebook.com/Enthought



**SciPy**



**EuroSciPy**

Please complete the online survey!

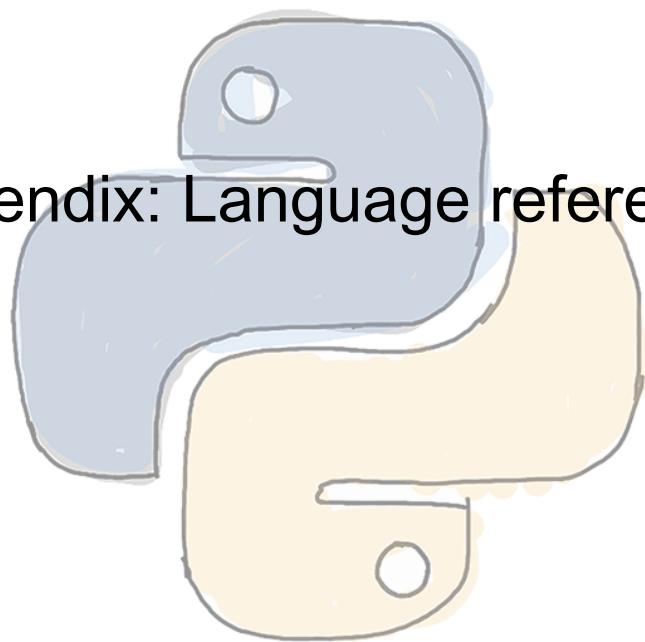
(link on course web page)

371

# Appendix

373

Appendix: Language reference



374

# String Formatting - Format Spec

```
>>> 'price: ${0:=-7.2f}'.format(3.4)
'price: $    3.40'
```

The *format spec* is a sequence of characters including:

- the *alignment* option,
- the *sign* option,
- the *width* (and *.precision*) option
- the *type code*.

## ALIGNMENT OPTION

### Char Meaning

<	Left aligned.
>	Right aligned.
=	(For numeric types only.) Pad after the sign but before the digits (e.g. +000000120).
^	Center within the available space.

If an alignment character is given, it may be preceded by a *fill character*.

## SIGN OPTION

### For numbers only.

### Char Meaning

+	Include a sign for positive and negative number.
-	Indicate sign for negative numbers only (default)
space	Include a leading space for positive numbers.

## STRING TYPE CODES

Type	Meaning
s	String. This is the default, and may be omitted.

## INTEGER TYPE CODES

Type	Meaning
b	Binary format.
c	Character; converts int to unicode char.
d	Decimal integer (base 10).
o	Octal (base 8).
x	Hex (base 16), lower case.
X	Hex (base 16), upper case.
n	Number; same as 'd', but uses current locale.
None	Same as 'd'.

## FLOATING POINT TYPE CODES

Type	Meaning
e	Scientific notation.
E	Scientific notation, with upper case 'E'.
f	Fixed point.
F	Fixed point; same as 'f'.
g	General format.
G	General format; same as 'g', with upper case 'E' when necessary.
n	Number; same as 'g', but uses current locale.
%	Percentage. Multiplies by 100 and displays with 'f', followed by a percent sign.
None	Same as 'g'.

375

# String formatting with % (older)

## FORMAT OPERATOR %

```
# the % operator formats values
# to strings using C conventions.
>>> s = "some numbers:"
>>> x = 1.34
>>> y = 2
>>> t = "%s %f, %d" % (s,x,y)
>>> print t
some numbers: 1.340000, 2

>>> y = -2.1
>>> print "%f\n%f" % (x,y)
1.340000
-2.100000

>>> print "% f\n% f" % (x,y)
1.340000
-2.100000

>>> print "%4.2f" % x
1.34
```

## CONVERSION CODES

Conversion	Meaning
d or i	Signed integer decimal
o	Unsigned octal
u	Unsigned decimal
x	Unsigned hexadecimal (lowercase)
X	Unsigned hexadecimal (uppercase)
e	Floating point exponential format (lowercase)
E	Floating point exponential format (uppercase)
F or f	Floating point decimal format
G or g	Floating point format or exponential
c	Single character
r	Converts object using repr()
s	Converts object using str()

## CONVERSION FLAGS

Flag	Meaning
0	The conversion will be zero padded for numeric values.
-	The converted value is left adjusted (overrides the "0" conversion if both are given).
<space>	(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
+	A sign character ("+" or "-") will precede the conversion (overrides a "space" flag).

376

# Common methods for lists

## `some_list.append( x )`

Add the element x to the end of the list some\_list.

## `some_list.count( x )`

Count the number of times x occurs in the list.

## `some_list.extend( sequence )`

Concatenate sequence onto this list.

## `some_list.index( x )`

Return the index of the first occurrence of x in the list.

## `some_list.insert( index, x )`

Insert x before the specified index.

## `some_list.pop( index )`

Return the element at the specified index. Also, remove it from the list.

## `some_list.remove( x )`

Delete the first occurrence of x from the list.

## `some_list.reverse( )`

Reverse the order of elements in the list.

## `some_list.sort( key )`

By default, sort the elements in ascending order. If a key function is given, apply it to each element to determine the value for sorting.

37'

# Common methods for dictionaries

## `some_dict.clear( )`

Remove all key/value pairs from the dictionary, some\_dict.

## `some_dict.copy( )`

Create a copy of the dictionary

## `x in some_dict`

Test whether the dictionary contains the key x.

## `some_dict.keys( )`

Return a list of all the keys in the dictionary.

## `some_dict.values( )`

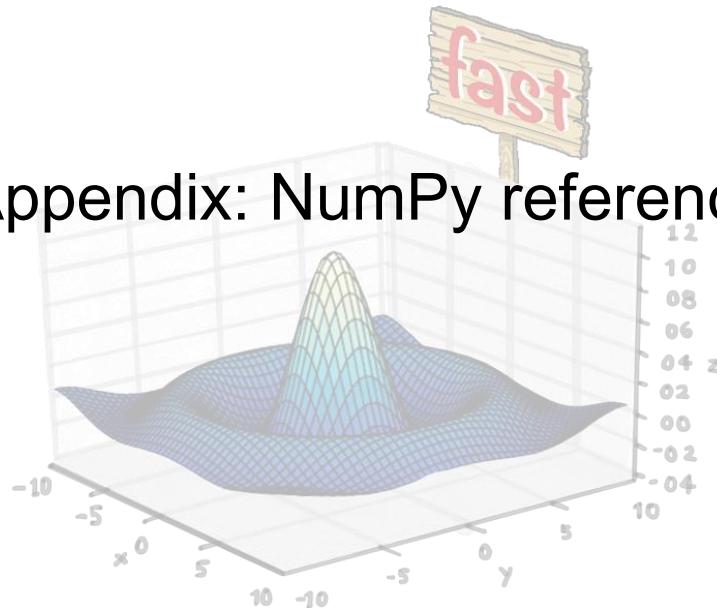
Return a list of all the values in the dictionary.

## `some_dict.items( )`

Return a list of all the key/value pairs in the dictionary.

378

# Appendix: NumPy reference



379

## NumPy dtypes

Type	Available NumPy types	Code	Comments
Boolean	bool	b	Elements are 1 byte in size.
Integer	int8, int16, int32, int64, int128, int	i	int defaults to the size of long in C for the platform.
Unsigned Integer	uint8, uint16, uint32, uint64, uint128, uint	u	uint defaults to the size of unsigned long in C for the platform.
Float	float16, float32, float64, float, longfloat	f	float is always a double precision floating point value (64 bits). longfloat represents large precision floats. Its size is platform dependent.
Complex	complex64, complex128, complex, longcomplex	c	The real and imaginary elements of a complex64 are each represented by a single precision (32 bit) value for a total size of 64 bits.
Strings	str, unicode	S or a, U	For example, dtype='S4' would be used for an array of 4-character strings.
DateTime	datetime64, timedelta64	None	Allow operations between dates and/or times. New in 1.7.
Object	object	o	Represent items in array as Python objects.
Records	void	v	Used for arbitrary data structures.

# Summary of (most) array attributes/methods (1/4)

<b>BASIC ATTRIBUTES</b>	
a.dtype	Numerical type of array elements: float 32, uint8, etc.
a.shape	Shape of array (m, n, o, ...)
a.size	Number of elements in entire array
a.itemsize	Number of bytes used by a single element in the array
a nbytes	Number of bytes used by entire array (data only)
a.ndim	Number of dimensions in the array
<b>SHAPE OPERATIONS</b>	
a.flat	An iterator to step through array as if it were 1D
a.flatten()	Returns a 1D copy of a multi-dimensional array
a.ravel()	Same as flatten(), but returns a "view" if possible
a.resize(new_size)	Changes the size/shape of an array in place
a.swapaxes(axis1, axis2)	Swaps the order of two axes in an array
a.transpose(*axes)	Swaps the order of any number of array axes
a.T	Shorthand for a.transpose()
a.squeeze()	Removes any length==1 dimensions from an array

381

# Summary of (most) array attributes/methods (2/4)

<b>FILL AND COPY</b>	
a.copy()	Returns a copy of the array
a.fill(value)	Fills an array with a scalar value
<b>CONVERSION/COERCION</b>	
a.tolist()	Converts array into nested lists of values
a.tostring()	Raw copy of array memory into a Python string
a.astype(dtype)	Returns array coerced to the given type
a.byteswap(False)	Converts byte order (big <->little endian)
a.view(type_or_dtype)	Creates a new ndarray that sees the same memory but interprets it as a new datatype (or subclass of ndarray)
<b>COMPLEX NUMBERS</b>	
a.real	Returns the real part of the array
a.imag	Returns the imaginary part of the array
a.conjugate()	Returns the complex conjugate of the array
a.conj()	Returns the complex conjugate of the array (same as conjugate)

382

# Summary of (most) array attributes/methods (3/4)

<b>SAVING</b>	
a.dump(file)	Stores binary array data to <i>file</i>
a.dumps()	Returns a binary pickle of the data as a string
a.tofile(fid, sep="", format="%s")	Formatted ASCII output to a file
<b>SEARCH/SORT</b>	
a.nonzero()	Returns indices for all non-zero elements in the array
a.sort(axis=-1)	Sort the array elements in place, along <i>axis</i>
a.argsort(axis=-1)	Finds indices for sorted elements, along <i>axis</i>
a.searchsorted(b)	Finds indices where elements of <i>b</i> would be inserted in <i>a</i> to maintain order
<b>ELEMENT MATH OPERATIONS</b>	
a.clip(low, high)	Limits values in the array to the specified range
a.round(decimals=0)	Rounds to the specified number of digits
a.cumsum(axis=None)	Cumulative sum of elements along <i>axis</i>
a.cumprod(axis=None)	Cumulative product of elements along <i>axis</i>

383

# Summary of (most) array attributes/methods (4/4)

## REDUCTION METHODS

All the following methods “reduce” the size of the array by 1 dimension by carrying out an operation along the specified axis. If axis is None, the operation is carried out across the entire array.

a.sum(axis=None)	Sums values along axis
a.prod(axis=None)	Finds the product of all values along axis
a.min(axis=None)	Finds the minimum value along axis
a.max(axis=None)	Finds the maximum value along axis
a.argmin(axis=None)	Finds the index of the minimum value along axis
a.argmax(axis=None)	Finds the index of the maximum value along axis
a.ptp(axis=None)	Calculates a.max(axis) – a.min(axis)
a.mean(axis=None)	Finds the mean (average) value along axis
a.std(axis=None)	Finds the standard deviation along axis
a.var(axis=None)	Finds the variance along axis
a.any(axis=None)	True if any value along axis is non-zero (logical OR)
a.all(axis=None)	True if all values along axis are non-zero (logical AND)

384

# Trig and math Functions

## TRIGONOMETRIC

```
sin(x)      sinh(x)
cos(x)      cosh(x)
arccos(x)   arccosh(x)
arctan(x)   arctanh(x)
arcsin(x)   arcsinh(x)
arctan2(x,y)
```

## OTHERS

```
exp(x)      log(x)
log10(x)    sqrt(x)
absolute(x) conjugate(x)
negative(x) ceil(x)
floor(x)    fabs(x)
hypot(x,y)  fmod(x,y)
maximum(x,y) minimum(x,y)
```

## VECTOR OPERATIONS

```
dot(x,y)    vdot(x,y)
inner(x,y)  outer(x,y)
cross(x,y)  kron(x,y)
tensordot(x,y[,axis])
```

## hypot(x,y)

Element by element distance  
calculation using  $\sqrt{x^2 + y^2}$

385

# Other array functions

## TYPE HANDLING

```
iscomplexobj real_if_close isnan
iscomplex    isscalar      nan_to_num
isrealobj   isneginf     common_type
isreal      isposinf     typename
imag        isinf
real        isfinite
```

## OTHER USEFUL FUNCTIONS

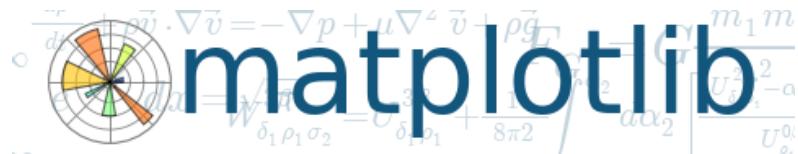
```
fix         unwrap       roots
mod         sort_complex poly
amax        trim_zeros   any
amin        fliplr       all
ptp         flipud      disp
sum         rot90       unique
cumsum      eye         nansum
prod        diag        nanmax
cumprod     select      nanargmax
diff        extract     nanargmin
angle       insert      nanmin
```

## SHAPE MANIPULATION

```
atleast_1d   hstack      hsplit
atleast_2d   vstack      vsplit
atleast_3d   dstack      dsplit
expand_dims  column_stack split
apply_over_axes squeeze
apply_along_axis
```

386

# Appendix: matplotlib reference



387

<http://matplotlib.org/>



[home](#) | [search](#) | [examples](#) | [gallery](#) | [docs](#) »

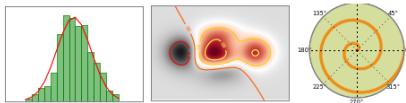
ENTHOUGHT

[modules](#) | [index](#)

## intro

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and [ipython](#) shell (ala MATLAB® or Mathematica®), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the [screenshots](#), [thumbnail](#) gallery, and [examples](#) directory



For example, using "ipython -pylab" to provide an interactive environment, to generate 10,000 gaussian random numbers and plot a histogram with 100 bins, you simply need to type

```
x = randn(10000)
hist(x, 100)
```

For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users. The pylab mode provides all of the [pyplot](#) plotting functions listed below, as well as non-plotting functions from [numpy](#) and [matplotlib.mlab](#).

## plotting commands

Function	Description
<a href="#">acorr</a>	plot the autocorrelation function

## News

Please [donate](#) to support matplotlib development.

matplotlib 1.0.1 is available for download. See [what's new](#) and [tips on installing](#)

Sandro Tosi has a new book [Matplotlib for python developers](#) also at [amazon](#).

Build websites like matplotlib's, with [sphinx](#) and extensions for mpl plots, math, inheritance diagrams -- try the [sampledoc](#) tutorial.

## Videos

Watch the [SciPy 2009 intro](#) and [advanced](#) matplotlib tutorials

Watch a [talk](#) about matplotlib presented at [NIPS 08 Workshop MLOSS](#) and one presented at [ChiPy](#).

## Toolkits

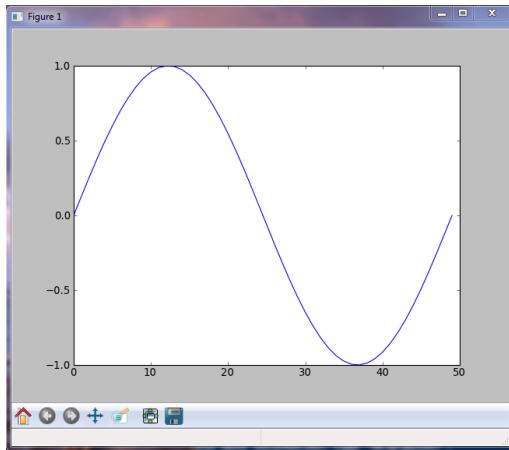
There are several matplotlib addon [toolkits](#), including the projection and mapping toolkit

388

# Line Plots

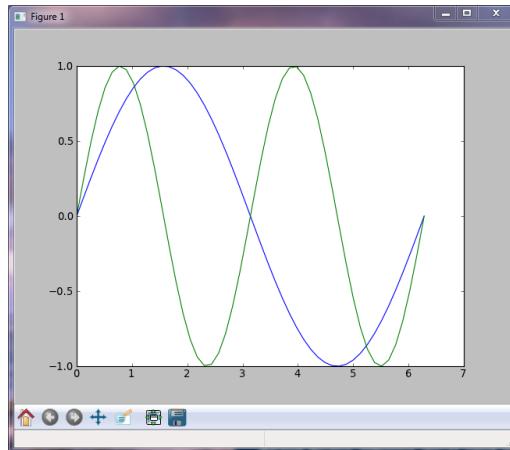
## PLOT AGAINST INDICES

```
>>> x = linspace(0,2*pi,50)
>>> plot(sin(x))
```



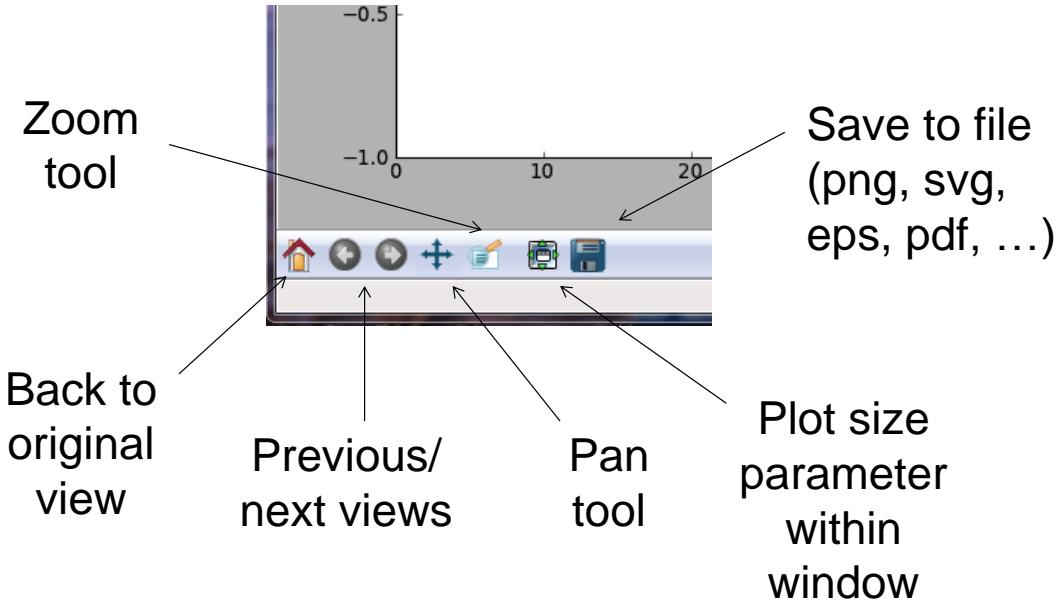
## MULTIPLE DATA SETS

```
>>> plot(x, sin(x),
...         x, sin(2*x))
```



389

# Matplotlib Menu Bar

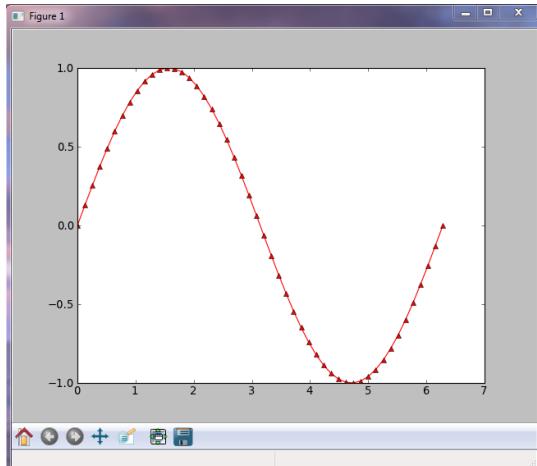


390

# Line Plots

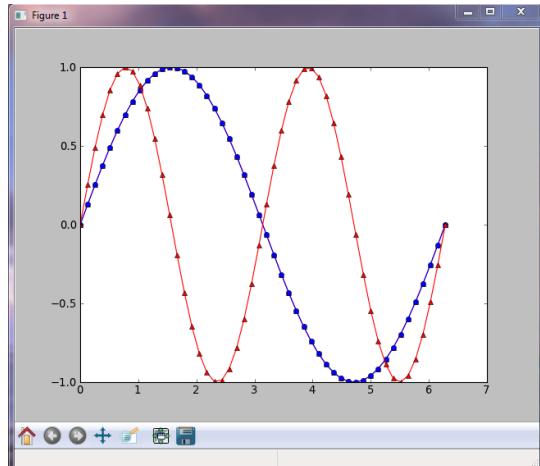
## LINE FORMATTING

```
# red, dot-dash, triangles
>>> plot(x, sin(x), 'b-o',
...       x, sin(2*x), 'r^-')
```



## MULTIPLE PLOT GROUPS

```
>>> plot(x, sin(x), 'b-o',
...       x, sin(2*x), 'r^-')
```

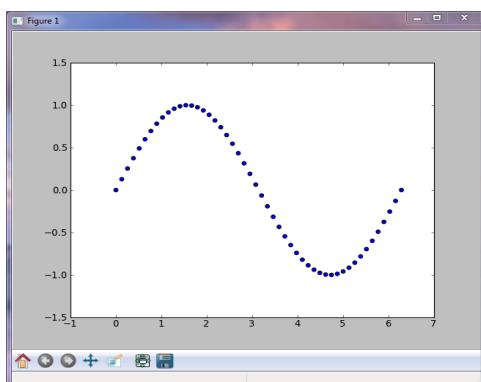


391

# Scatter Plots

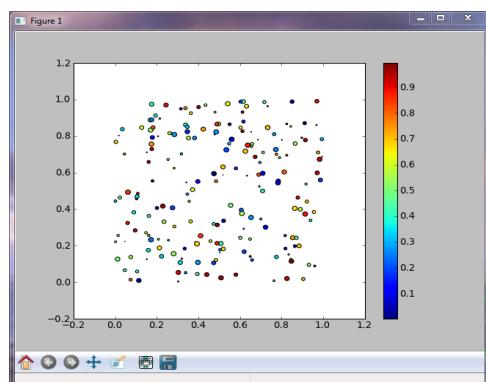
## SIMPLE SCATTER PLOT

```
>>> x = linspace(0,2*pi,50)
>>> y = sin(x)
>>> scatter(x, y)
```



## COLORMAPPED SCATTER

```
# marker size/color set with data
>>> x = rand(200)
>>> y = rand(200)
>>> size = rand(200)*30
>>> color = rand(200)
>>> scatter(x, y, size, color)
>>> colorbar()
```



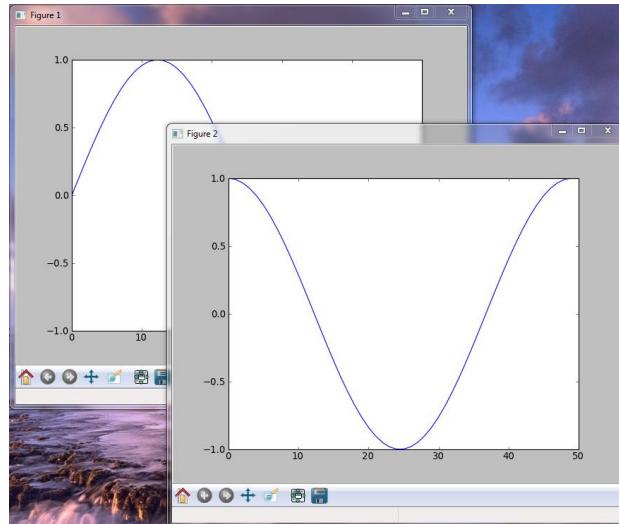
92

# Multiple Figures

```
>>> t = linspace(0,2*pi,50)
>>> x = sin(t)
>>> y = cos(t)

# Now create a figure
>>> figure()
>>> plot(x)

# Now create a new figure.
>>> figure()
>>> plot(y)
```



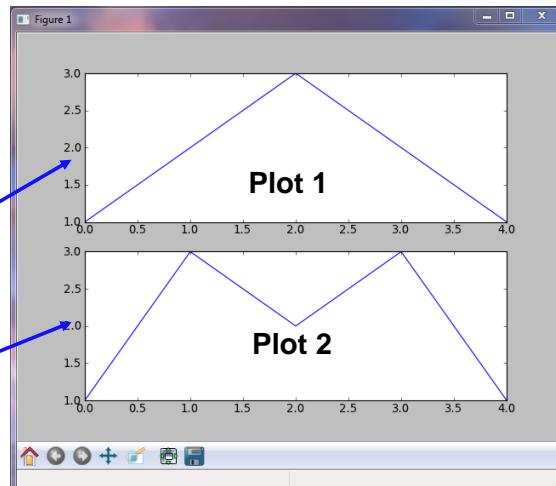
393

# Multiple Plots Using subplot

```
>>> x = array([1,2,3,2,1])
>>> y = array([1,3,2,3,1])

# To divide the plotting area
    columns
    |
>>> subplot(2, 1, 1)
    |
    rows      active plot
    |
>>> plot(x)

# Now activate a new plot
# area.
>>> subplot(2, 1, 2)
    |
    |
>>> plot(y)
```



If this is used in a python script, a call to the function show() is required.

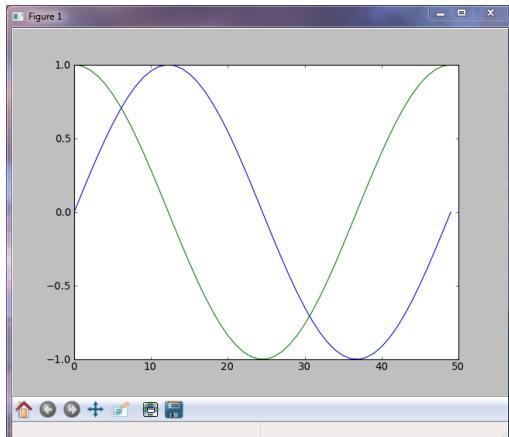
394

# Adding Lines to a Plot

## MULTIPLE PLOTS

```
# By default, previous lines
# are "held" on a plot.

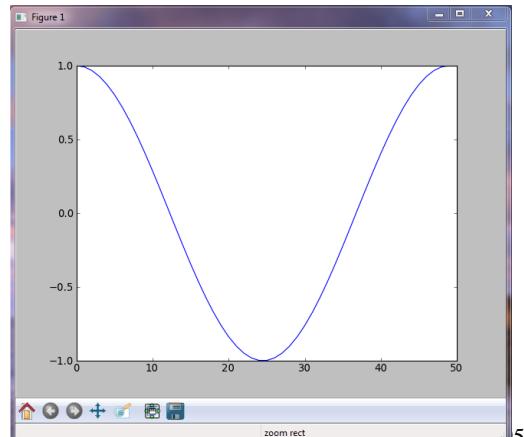
>>> plot(sin(x))
>>> plot(cos(x))
```



## ERASING OLD PLOTS

```
# Set hold(False) to erase
# old lines

>>> plot(sin(x))
>>> hold(False)
>>> plot(cos(x))
```

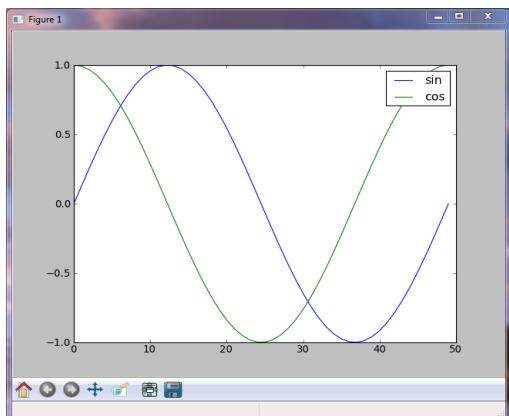


# Legend

## LEGEND LABELS WITH PLOT

```
# Add labels in plot command.

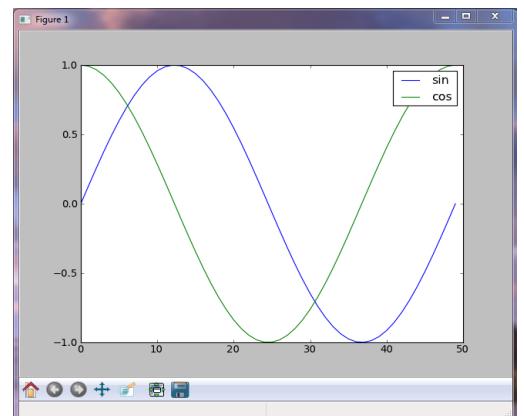
>>> plot(sin(x), label='sin')
>>> plot(cos(x), label='cos')
>>> legend()
```



## LABELING WITH LEGEND

```
# Or as a list in legend().

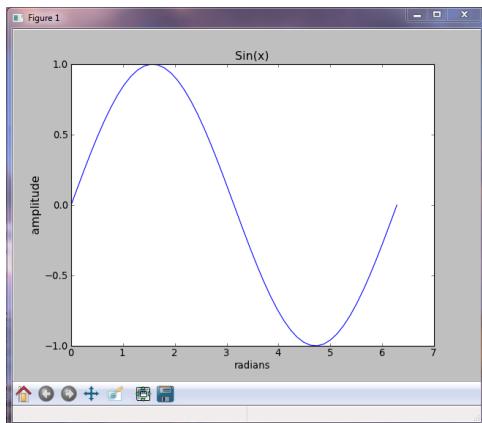
>>> plot(sin(x))
>>> plot(cos(x))
>>> legend(['sin', 'cos'])
```



# Titles and Grid

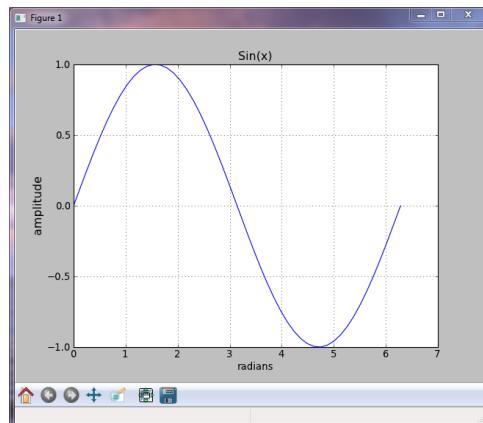
## TITLES AND AXIS LABELS

```
>>> plot(x, sin(x))
>>> xlabel('radians')
# Keywords set text properties.
>>> ylabel('amplitude',
...         fontsize='large')
>>> title('Sin(x)')
```



## PLOT GRID

```
# Display gridlines in plot
>>> grid()
```

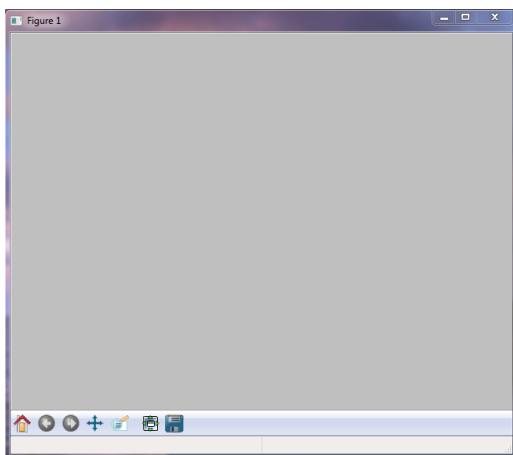


97

# Clearing and Closing Plots

## CLEARING A FIGURE

```
>>> plot(x, sin(x))
# clf will clear the current
# plot (figure).
>>> clf()
```



## CLOSING PLOT WINDOWS

```
# close() will close the
# currently active plot window.
>>> close()

# close('all') closes all the
# plot windows.
>>> close('all')
```

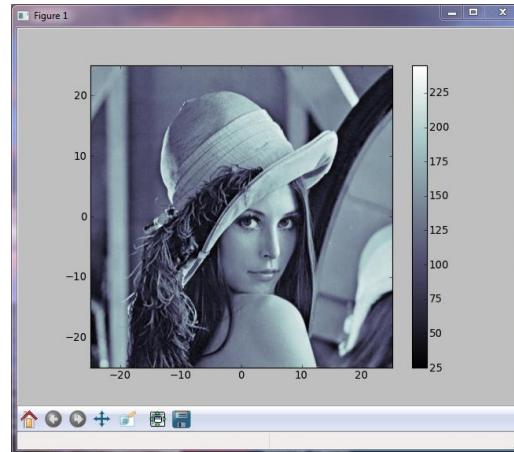
398

# Image Display

```
# Get the Lena image from scipy.
>>> from scipy.misc import lena
>>> img = lena()

# Display image with the jet
# colormap, and setting
# x and y extents of the plot.
>>> imshow(img,
...         extent=[-25,25,-25,25],
...         cmap = cm.bone)

# Add a colorbar to the display.
>>> colorbar()
```



399

# Plotting from Scripts

## INTERACTIVE MODE

```
# In IPython, plots show up
# as soon as a plot command
# is called.
>>> figure()
>>> plot(sin(x))
>>> figure()
>>> plot(cos(x))
```

## NON-INTERACTIVE MODE

```
# script.py
# In a script, you must call
# the show() command to display
# plots. Call it at the end of
# all your plot commands for
# best performance.
figure()
plot(sin(x))
figure()
plot(cos(x))

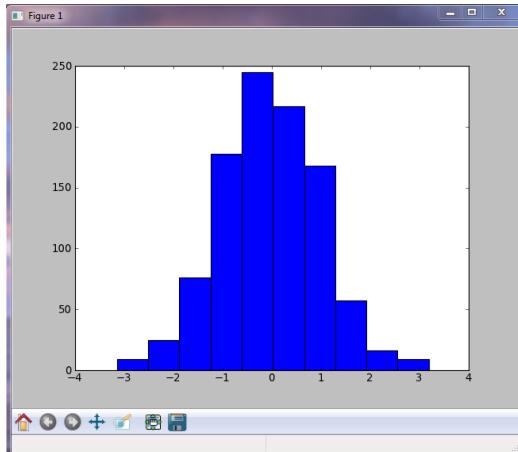
# Plots will not appear until
# this command is issued.
show()
```

400

# Histograms

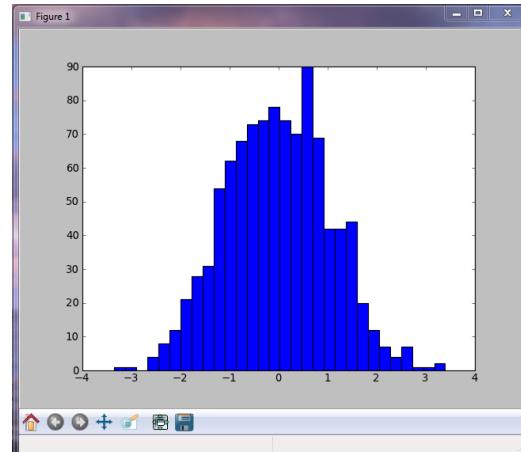
## HISTOGRAM

```
# plot histogram
# defaults to 10 bins
>>> hist(randn(1000))
```



## HISTOGRAM 2

```
# change the number of bins
>>> hist(randn(1000), 30)
```

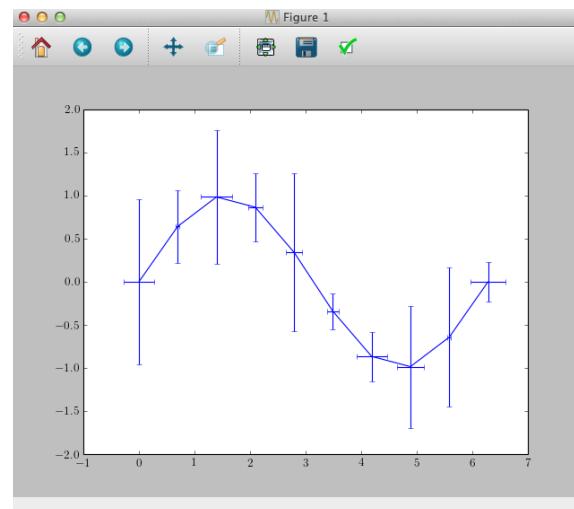


401

# Plots with error bars

## ERRORBAR

```
# Assume points are known
# with errors in both axis
>>> x = linspace(0,2*pi,10)
>>> y = sin(x)
>>> yerr = rand(10)
>>> xerr = rand(10)/3
>>> errorbar(x, y, yerr, xerr)
```



402

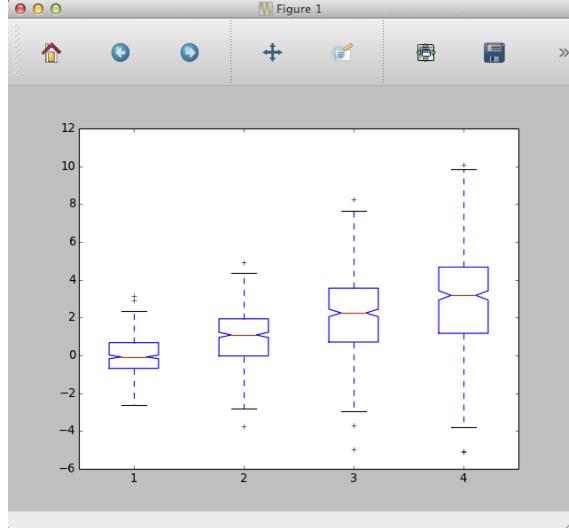
# Plots with error bars II

## BOXPLOT

```
# Assume 4 experiments have measured some
# quantities
# Data creation
>>> from numpy.random import normal
>>> ex = [normal(i, 1+i/2, size=(500,))
          for i in np.arange(4.)]

# Plot creation
>>> positions = np.arange(len(ex))+1
>>> boxplot(ex, sym='k+', notch=True,
            positions=positions)

# Interpretation
# Red line = median
# Notch = median @95% CL
# Box = [25%-75%]
# Whiskers = 1.5 * box_extent
```

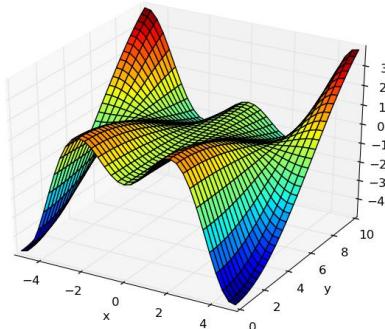


403

# 3D Plots with Matplotlib

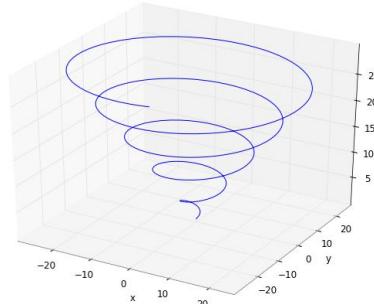
## SURFACE PLOT

```
>>> from mpl_toolkits.mplot3d import
Axes3D
>>> x, y = mgrid[-5:5:35j, 0:10:35j]
>>> z = x*sin(x)*cos(0.25*y)
>>> fig = figure()
>>> ax = fig.gca(projection='3d')
>>> ax.plot_surface(x, y, z,
...                   rstride=1, cstride=1,
...                   cmap=cm.jet)
>>> xlabel('x'); ylabel('y')
```



## PARAMETRIC CURVE

```
>>> from mpl_toolkits.mplot3d import
Axes3D
>>> t = linspace(0, 30, 1000)
>>> x, y, z = [t*cos(t), t*sin(t), t]
>>> fig = figure()
>>> ax = fig.gca(projection='3d')
>>> ax.plot(x, y, z)
>>> xlabel('x')
>>> ylabel('y')
```



404

# Surface Plots with mlab

```
# Create 2D array where values
# are radial distance from
# the center of array.
>>> from numpy import mgrid
>>> from scipy import special
>>> x,y = mgrid[-25:25:100j,
...                 -25:25:100j]
>>> r = sqrt(x**2+y**2)
# Calculate Bessel function of
# each point in array and scale.
>>> s = special.j0(r)*25

# Display surface plot.
>>> from mayavi import mlab
>>> mlab.surf(x,y,s)
>>> mlab.scalarbar()
>>> mlab.axes()
```

