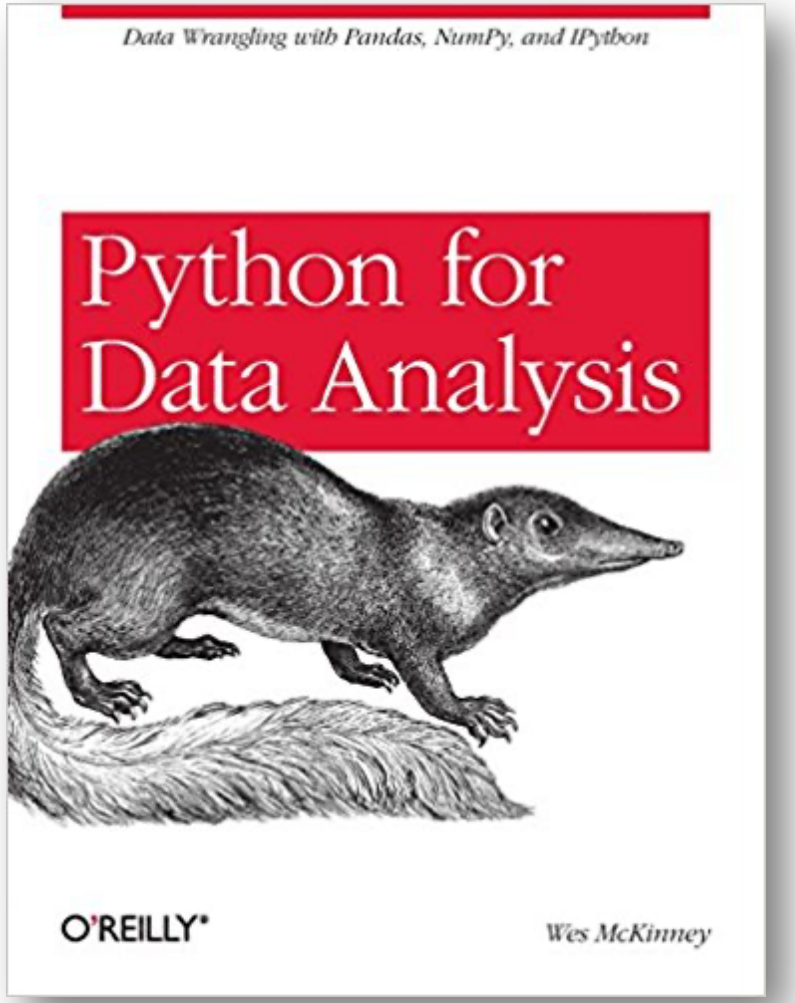
Textbooks



http://bedford-computing.co.uk/learning/wp-content/uploads/2015/10/Python-for-Data-Analysis.pdf

Why Learn Python for Data Mining?

Pros

- Open Source – free to install
- Awesome online community
- Very easy to learn
- Very easy to debug
- Is getting a common language for data science and production of web based analytics products.

Cons

It is an interpreted language rather than compiled language – hence might take up more CPU time. However, given the savings in programmer time (due to ease of learning), it is still a good choice.

Python versions!

- Awesome community support! This is something you'd need in your early days. Python 2 was released in late 2000 and has been in use for more than 15 years.
- Plenty of third-party libraries! Though many libraries have provided 3.x support but still a large number of modules work only on 2.x versions. If you plan to use Python for specific applications like web-development with high reliance on external modules, you might be better off with 2.7.
- Some of the features of 3.x versions have backward compatibility and can work with 2.7 version.

This is one of the most debated topics in Python. There is no clear winner between Python 2.7 and Python 3.X but I suppose the bottom line is that you should focus on learning Python as a language. Shifting between versions should just be a matter of time.

Interactive “Shell”

IDLE (GUI), Python (command line), Jupyter Notebook

Type statements or expressions at prompt:

```
>>> print "Hello, world!"
Hello, world!
>>> x = 12**2
>>> x/2
72
>>> # this is a comment
```

Variable Naming Conventions

1. Names *must* start with a letter (Python requirement) and *should not* begin with an underscore (style requirement).
2. Names are case sensitive but avoid distinguishing variable names only by case (style requirement).
3. For variable names composed of multiple words separate each word by capitalizing the first letter of each word (save for the first word) or by using an underscore. Either approach is acceptable but don’t mix and match (style requirement).
4. Can't be a keyword (Python requirement, see next slide).

Python Keywords

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Numbers

- Int – 42
- Float – 2.171892
- Complex – 4 + 3j
- Bool – True of False

Number Type Conversion

- Type **int(x)** to convert x to a plain integer.
- Type **long(x)** to convert x to a long integer.
- Type **float(x)** to convert x to a floating-point number.
- Type **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
- Type **complex(x, y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions

```
a = int("34")                    # a = 34
b = long("0xfe76214", 16)        # b = 266822164L (0xfe76214L)
b = float("3.1415926")           # b = 3.1415926
```

Popular Functions in Python Math Module

ceil(x)	Returns the smallest integer greater than or equal to x.
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN

Popular Functions in Python Math Module

log(x[, base])	Returns the logarithm of x to the base (defaults to e)
log2(x)	Returns the base-2 logarithm of x
log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
cos(x)	Returns the cosine of x
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

Strings

- "hello"+"world" "helloworld"# concatenation
- "hello"*3 "hellohellohello“ # repetition
- "hello"[0] "h" # indexing
- "hello"[-1] "o" # (from end)
- "hello"[1:4] "ell" # slicing
- len("hello") 5 # size
- "e" in "hello" 1 # search

List

- Flexible arrays

```
a = [99, "bottles of beer", ["on", "the", "wall"]]
```

- Same operators as for strings

```
a+b, a*3, a[0], a[-1], a[1:], len(a)
```

- Item and slice assignment

```
a[0] = 98
a[1:2] = ["bottles", "of", "beer"]        # [98, "bottles", "of", "beer", ["on", "the", "wall"]]
del a[-1]                                 # [98, "bottles", "of", "beer"]
```

More List Operations

```
>>> a = range(5)                # [0,1,2,3,4]
>>> a.append(5)                 # [0,1,2,3,4,5]
>>> a.pop()             # [0,1,2,3,4]
5
>>> a.insert(0, 42)             # [42,0,1,2,3,4]
>>> a.pop(0)                    # [0,1,2,3,4]
42
>>> a.reverse()                 # [4,3,2,1,0]
>>> a.sort()            # [0,1,2,3,4]
```

Control Structures

if *condition*:
    statements
[elif *condition*:
    statements] ...
else:
    statements

while *condition*:
    statements

for *var* in *sequence*:
    statements

break

continue

Example

```
value1 = 11
if value1 == 1:
    print "value equals 1"
    print "more can come in this block"
else:
    print value1
```

```
sentence = ['Marry','had','a','little','lamb']
for word in sentence:
        print word, len(word)


for i in range(10):
        print i


while True:
        pass


for i in xrange(10):
        if i == 3: continue
        if i == 5: break
        print i
```

Essential Python Libraries for Data Mining

- NumPy
- SciPy
- Pandas
- Matplotlib
- SciKit-Learn
- …

Loading Python Libraries

In [ ]:

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Press Shift+Enter to execute the *jupyter* cell

NumPy

- Introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects

- Provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance

- Many other python libraries are built on NumPy

➢ **Vectorization** (**mathematics**) **In mathematics**, especially in linear algebra and matrix theory, the **vectorization** of a matrix is a linear transformation which converts the matrix into a column vector

Example

- Lists ok for storing small amounts of one-dimensional data

```
>>> a = [1,3,5,7,9]
>>> print(a[2:4])
[5, 7]
>>> b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
>>> print(b[0])
[1, 3, 5, 7, 9]
>>> print(b[1][2:4])
[6, 8]
```

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

- But, can't use directly with arithmetic operators (+, -, *, /, …)
- Need efficient arrays with arithmetic and better multidimensional tools
- **Numpy -** Similar to lists, but much more capable, except fixed size

Numpy – Creating Vectors From Lists

```
# as vectors from lists

>>> a = numpy.array([1,3,5,7,9])

>>> b = numpy.array([3,5,6,7,9])

>>> c = a + b

>>> print c

[4, 8, 11, 14, 18]

>>> type(c)

(<type 'numpy.ndarray'>)

>>> c.shape
```

Numpy – Creating Matrices

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = numpy.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # get type of an array
int64
```

```
# or directly as matrix
>>> M = array([[1, 2], [3, 4]])
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

Numpy – Matrices Use

```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # this is just like a list of lists
[1 2 3]
>>> print(a[1, 2])  # arrays can be given comma separated indices
9
>>> print(a[1, 1:3])  # and slices
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```

Numpy – Creating Arrays

```
# a diagonal matrix
>>> numpy.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> b = numpy.zeros(5)
>>> print(b)
[ 0.  0.  0.  0.  0.]
>>> b.dtype
dtype('float64')
```

```
>>> n = 1000
>>> my_int_array = numpy.zeros(n, dtype=numpy.int)
>>> my_int_array.dtype
dtype('int32')

>>> c = numpy.ones((3,3))
>>> c
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Numpy – Array Creation and Use

```
>>> d = numpy.arange(5)  # just like range()
>>> print(d)
[0 1 2 3 4]
>>> d[1] = 9.7
>>> print(d)  # arrays keep their type even if elements changed
[0 9 2 3 4]
>>> print(d*0.4)  # operations create a new array, with new type
[ 0.   3.6  0.8  1.2  1.6]
>>> d = numpy.arange(5, dtype=numpy.float)
>>> print(d)
[ 0.  1.  2.  3.  4.]
```

Numpy – Array Creation and Use

Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

```
>>> numpy.random.rand(5,5)
array([[ 0.51531133,  0.74085206,  0.99570623,  0.97064334,  0.5819413 ],
       [ 0.2105685 ,  0.86289893,  0.13404438,  0.77967281,  0.78480563],
       [ 0.62687607,  0.51112285,  0.18374991,  0.2582663 ,  0.58475672],
       [ 0.72768256,  0.08885194,  0.69519174,  0.16049876,  0.34557215],
       [ 0.93724333,  0.17407127,  0.1237831 ,  0.96840203,  0.52790012]])
```

Numpy – Array Mathematics

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

Numpy – Matrix Algebra

```
>>> A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
>>> v1 = arange(0, 5)
>>> A
array([[ 0, 1, 2, 3, 4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24],
       [30, 31, 32, 33, 34],
       [40, 41, 42, 43, 44]])
>>> v1
array([0, 1, 2, 3, 4])
```

dot product is an algebraic operation that takes two equal-length sequences of numbers

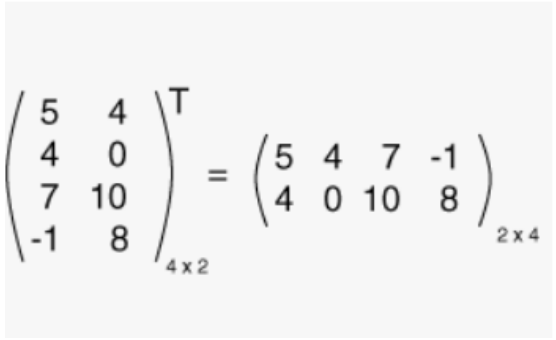
```
>>> np.dot(A,A)
array([[ 300,  310,  320,  330,  340],
       [1300, 1360, 1420, 1480, 1540],
       [2300, 2410, 2520, 2630, 2740],
       [3300, 3460, 3620, 3780, 3940],
       [4300, 4510, 4720, 4930, 5140]])

>>> np.dot(A,v1)
array([ 30, 130, 230, 330, 430])

>>> np.dot(v1,v1)
30
```

Numpy – Matrix Algebra

```
>>> M = np.matrix(A)
>>> v = np.matrix(v1).T
>>> v
matrix([[0],
        [1],
        [2],
        [3],
        [4]])
>>> M*v
matrix([[ 30],
        [130],
        [230],
        [330],
        [430]])
```



```
>>> v.T * v   # inner product
matrix([[30]])

# standard matrix algebra applies
>>> v + M*v
matrix([[ 30],
        [131],
        [232],
        [333],
        [434]])
```

Pandas

- Adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- Allows handling missing data

http://pandas.pydata.org/

Matplotlib

- Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- A set of functionalities similar to those of MATLAB
- Line plots, scatter plots, barcharts, histograms, pie charts etc.
- Relatively low-level; some effort needed to create advanced visualization

https://matplotlib.org/

SciPy

- Collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- Part of SciPy Stack
- Built on NumPy

https://www.scipy.org/scipylib/

SciKit-Learn

- Provides machine learning algorithms: classification, regression, clustering, model validation etc.
- Built on NumPy, SciPy and matplotlib

http://scikit-learn.org/

Other References

- Python Crash Course Numpy Slides
- Getting Started With Python Programming Slides
- Guido van Rossum - Introduction to Python Slides
- Katia Oleinik - Python for Data Analysis Slides, Boston University
- Sven Schmit - Introduction to Scientific Python, Stanford University