Introduction to Python Data Analysis

Stephen Weston   Robert Bjornson

Yale Center for Research Computing
Yale University

January 2017

Python for data analysis

Python is more of a general purpose programming language than R or Matlab. It has gradually become more popular for data analysis and scientific computing, but additional modules are needed. Some of the more popular modules are:

- **NumPy** N-dimensional array
- **SciPy** Scientific computing (linear algebra, numerical integration, optimization, etc)
- **Matplotlib** 2D Plotting (similar to Matlab)
- **IPython** Enhanced Interactive Console
- **Sympy** Symbolic mathematics
- **Pandas** Data analysis (provides a data frame structure similar to R)

NumPy, SciPy and Matplotlib are used in this presentation.

Creating N-dimensional arrays using NumPy

There are many ways to create N-dimensional arrays

```
import numpy as np
# Create 2X3 double precision array initialized to all zeroes
a = np.zeros((2,3), dtype=np.float64)

# Create array initialized by list of lists
a = np.array([[0,1,2],[3,4,5]], dtype=np.float64)

# Create array by reading CSV file
a = np.genfromtxt('data.csv', dtype=np.float64, delimiter=',')

# Create array using "arange" function
a = np.arange(6, dtype=np.float64).reshape(2,3)
```

Get values from N-dimensional array

NumPy provides many ways to extract data from arrays

```
# Print single element of 2D array
print a[0,0]       # a scalar, not an array

# Print first row of 2D array
print a[0,:]       # 1D array

# Print last column of array
print a[:,-1]      # 1D array

# Print sub-matrix of 2D array
print a[0:2,1:3]  # 2D array
```

Modifying N-dimensional arrays

NumPy uses the same basic syntax for modifying arrays

```
# Assign single value to single element of 2D array
a[0,0] = 25.0

# Assign 1D array to first row of 2D array
a[0,:] = np.array([10,11,12], dtype=np.float64)

# Assign 1D array to last column of 2D array
a[:,-1] = np.array([20,21], dtype=np.float64)

# Assign 2D array to sub-matrix of 2D array
a[0:2,1:3] = np.array([[10,11],[20,21]], dtype=np.float64)
```

Modifying arrays using broadcasting

```
# Assign scalar to first row of 2D array
a[0,:] = 10.0

# Assign 1D array to all rows of 2D array
a[:,:] = np.array([30,31,32], dtype=np.float64)

# Assign 1D array to all columns of 2D array
a[:,:] = np.array([40,41], dtype=np.float64).reshape(2,1)

# Assign scalar to sub-matrix of 2D array
a[0:2,1:3] = 100.0
```

Arithmetic on arrays

Operate on arrays using binary operators and NumPy functions

```
# Create 1D array
a = np.arange(4, dtype=np.float64)

# Add 1D arrays elementwise
a + a

# Multiply 1D arrays elementwise
a * a

# Sum elements of 1D array
a.sum()

# Compute dot product
np.dot(a, a)    # same as: (a * a).sum()

# Compute cross product
np.dot(a.reshape(4,1), a.reshape(1,4))
```

NumPy views

Views are arrays that share memory with another array.

- views can make your program more memory and CPU efficient
- views are explicitly generated via the view method
- reshape and transpose implicitly return views of the original array
- arrays generated by slicing are views of the original
- use the copy method to avoid sharing memory
- set the writeable flag to make a view read-only (a.flags.writeable)

NumPy views continued

```
>>> a = np.arange(10)
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
>>> b.__array_interface__['data'][0]
4330625040
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
>>> b.__array_interface__['data'][0]
4330625040
>>> c = b.copy()
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
>>> b.__array_interface__['data'][0]
4330625040
>>> c = b.copy()
>>> c.flags.owndata, c.__array_interface__['data'][0]
(True, 4301585776)
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
>>> b.__array_interface__['data'][0]
4330625040
>>> c = b.copy()
>>> c.flags.owndata, c.__array_interface__['data'][0]
(True, 4301585776)
>>> d = a
```

NumPy views continued

```
>>> a = np.arange(10)
>>> b = a[2::2]
>>> a.flags.owndata, b.flags.owndata
(True, False)
>>> b[0] = 100
>>> a
array([  0,   1, 100,   3,   4,   5,   6,   7,   8,   9])
>>> a.__array_interface__['data'][0]
4330625024
>>> b.__array_interface__['data'][0]
4330625040
>>> c = b.copy()
>>> c.flags.owndata, c.__array_interface__['data'][0]
(True, 4301585776)
>>> d = a
>>> d is a
True
```

Mini NumPy Cookbook

matrix size	a.shape
transpose	a.T
extract diagonal elements	np.diag(a)
matrix multiply	a.dot(b)
element-wise multiply	a * b
column sums	np.sum(a, axis=0)
row sums	np.sum(a, axis=1)
element sum	np.sum(a, axis=None)
column means	np.mean(a, axis=0)
column max	np.max(a, axis=0)
column min	np.min(a, axis=0)
element-wise maximum	np.maximum(a, b)
element-wise minimum	np.minimum(a, b)
identity matrix	np.eye(n)
random matrix	np.random.rand(m, n)
set random seed	np.random.seed(i)
solve ax=b	np.linalg.solve(a, b)
fourier transform	np.fft.fft(a)

SciPy Linear Algebra functions

```
import numpy as np
from scipy import linalg
a = np.array([[1, 2], [3, 4]], dtype=np.float64)

# Compute the inverse matrix
linalg.inv(a)

# Compute singular value decomposition
linalg.svd(a)

# Compute eigenvalues
linalg.eigvals(a)
```

2D plotting using Matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0.0, 2.0, 20)

plt.plot(x, np.sqrt(x), 'ro')  # red circles
plt.show()

plt.plot(x, np.sqrt(x), 'b-')  # blue lines
plt.show()

# Three plots in one figure
plt.plot(x, x, 'g--', x, np.sqrt(x), 'ro', x, np.sqrt(x), 'b-')
plt.show()
```