Data Processing and Analysis in Python
Lecture 13
Arrays and NumPy

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

DR. ADAM LEE

Scientific Applications

- There are many third-party packages available for scientific computing that extend Python's basic math module:
  - **NumPy/SciPy** – numerical and scientific function libraries
  - **Numba** – Python compiler that support JIT compilation
  - **ALGLIB** – numerical analysis library
  - **PyGSL** – Python interface for GNU Scientific Library
  - **ScientificPython** – collection of scientific computing modules
  - …

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

SciPy Stack

- By far, the most commonly used packages are those in the SciPy stack. These packages include:
  - **NumPy** – fundamental package for scientific computing
  - **SciPy** – efficient numerical routines
  - **Matplotlib** – plotting library
  - **IPython** – interactive computing
  - **SymPy** – symbolic computation library
  - **Pandas** – data analysis library
  - …

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Install SciPy Stack

- https://scipy.org/install.html
- Mac and Linux users can install pre-built binary packages for the SciPy stack using pip
- Pip can install pre-built binary packages in the wheel package format
- Pip does not work well for Windows because the standard pip package index site, PyPI, does not yet have Windows wheels for some packages, such as SciPy

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Install SciPy Stack

- To install via pip on Mac or Linux, first upgrade pip to the latest version:

  python -m pip install --upgrade pip

- Then install the SciPy stack packages with pip by using the --user flag. This installs packages for your local user, and does not need extra permissions to write to the system directories:

  pip install --user numpy scipy matplotlib ipython sympy pandas

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy

- The fundamental package for scientific computing with Python. It contains:
  - A powerful N-dimensional array (**ndarray**) object
  - Sophisticated (broadcasting/universal) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy & Class ndarray

```
>>> import numpy
>>> import numpy as np # alias
>>> from numpy import * # import all
```

https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

```
>>> dir(numpy)
>>> dir(numpy.ndarray)
>>> help(dtype)
>>> help(ndarray)
>>> help(ndarray.dtype)
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Data Types

- NumPy supports a much greater variety of numerical types than Python does:
  - bool_
  - int_, intc, intp, u/int8, u/int16, u/int32, u/int64
  - float_, float16, float32, float64
  - complex_, complex64, complex128
- NumPy numerical types are instances of dtype (data-type) objects:
  - numpy.dtype(object, align, copy)

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Data Types

```
>>> flt = np.float32(1.0)
>>> flt
1.0

>>> arr = np.int_([1,2,4])
>>> arr
array([1, 2, 4])

>>> arr = np.arange(3, dtype=np.uint8)
array([0, 1, 2], dtype=uint8)
>>> arr.dtype
dtype('uint8')
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays

- The main feature of NumPy is an array object:
  - Arrays can be N-dimensional
  - Array elements have to be the same type
  - Array elements can be accessed, sliced, and manipulated in the same way as the lists
  - The number of elements in the array is fixed
  - Shape of the array can be changed
- Built-in NumPy array creation:
  - array(), arange(), ones(), zeros(), …

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Creation

```
>>> np.array([2,3,1,0])
array([2, 3, 1, 0])
>>> np.array([[1,2.0],[0,0],[1+1j,3.]])
array([[1.+0.j, 2.+0.j],
       [0.+0.j, 0.+0.j],
       [1.+1.j, 3.+0.j]])

>>> np.zeros((2,3)) # all zeros
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> np.ones((2,3)) # all ones
array([[1., 1., 1.],
       [1., 1., 1.]])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Creation

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2,10,dtype=np.float)
array([2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2,3,0.1)
array([2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7,
2.8, 2.9])
>>> arr = np.arange(3)
>>> arr
array([0, 1, 2])
>>> print(arr)
[0 1 2]
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Reshape

```
>>> np.arange(9).reshape(3,3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]]
>>> np.arange(8).reshape(2,2,2)
array([[[0, 1],
        [2, 3]],
       [[4, 5],
        [6, 7]]])
```

NumPy Arrays – Creation

- linspace(start, stop[, num, endpoint, retstep, dtype]) – creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values

```
>>> np.linspace(1.,4.,6)
array([1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Creation of Random Numbers

- random.random([size]) – creates arrays with random floats over the interval [0.,1.)

```
>>> np.random.random((2,3))
array([[0.96826, 0.30919, 0.58381],
       [0.56865, 0.33730, 0.41241]])
```

- random.randint(low[, high, size, dtype]) – creates arrays with random integers from low (inclusive) to high (exclusive)

```
>>> np.random.randint(1,7,(2,6))
array([[2, 4, 5, 2, 3, 6],
       [6, 3, 1, 4, 1, 5]])
```


UNIVERSITY OF MARYLAND

ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Attribute/Property

```
>>> arr = np.random.random((2,3))
array([[ 0.96826, 0.30919, 0.58381],
       [ 0.56865, 0.33730, 0.41241]])
>>> arr.ndim # number of dimensions
2
>>> arr.shape # dimensions in the array
(2, 3)
>>> arr.dtype # data type of values
dtype('float64')
>>> arr.size # total count of values
6
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Operation

- NumPy arrays are designed to support fast computation and comparisons
- Most common types of operations are:
  - Between arrays and scalars (one value at a time)
  - Unary (performed on a single array): abs, sqrt, ceil, floor, etc.
  - Binary (performed between two arrays): +, *, <, etc.
- Mathematical and statistical functions:
  - Aggregation: mean, sum, std, variance, min, max, etc.
  - Non-aggregation: cumsum , cumprod, etc.

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Arithmetic Operation

```
>>> arr = np.array([[1,2,3],[4,5,6]])
array([[1, 2, 3],
       [4, 5, 6]])
>>> arr + 1 # add a scalar value to an array
array([[2, 3, 4],
       [5, 6, 7]])
```

- A lower-dimension array can be part of the broadcast if the sizes are compatible

```
>>> arr + [10,20,30] # add a compatible array
array([[11, 22, 33],
       [14, 25, 36]])
```

NumPy Arrays – Comparison Operation

```
>>> arr = np.array([[1,2,3],[4,5,6]])

# comparison can take place with a scalar
>>> arr > 3
array([[False, False,  True],
       [ True,  True,  True]])

# or, an array with compatible size
>>> arr > [2,3,4]
array([[False, False, False],
       [ True,  True,  True]])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Universal Function

```
>>> arr = np.array([[1,2,3],[4,5,6]])

>>> np.abs(arr) # absolute value of each value
array([[1, 2, 3],
       [4, 5, 6]])
>>> np.sqrt(arr) # square root of each value
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974]])
>>> np.log(arr) # logarithm of each value
array([[0.        , 0.69314718, 1.09861229],
       [1.38629436, 1.60943791, 1.79175947]])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Aggregate Function

```
>>> arr = np.array([[1,2,3],[4,5,6]])

>>> np.sum(arr) # sum of all values
21
>>> np.mean(arr) # average of all values
3.5
>>> np.min(arr) # minimum of all values
1
# indices of maximum values along an axis
>>> np.argmax(arr,axis=1)
array([2, 2])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Non-Aggregate Function

```
>>> arr = np.array([1,2,3],[4,5,6])
>>> np.cumsum(arr) # accumulated sum of values
array([ 1,  3,  6, 10, 15, 21])
>>> np.cumprod(arr) # accumulated products
array([  1,   2,   6,  24, 120, 720])
```

NumPy Arrays versus Python Lists

- Arrays are similar to lists (e.g. mutable and iterable)
- Be sure to use arrays whenever you are performing any large scale computations or comparisons
- However, lists do not have restrictions on the size of nested sequences, whereas arrays have restrictions for constructing a useful form of the object

```
>>> arr = np.array([[1,2,3],[4,5],[6,7,8]])
array([list([1, 2, 3]), list([4, 5]), list([6,
7, 8])], dtype=object)
>>> type(arr)
<class 'numpy.ndarray'>
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays vs Lists – Execution Time

- %timeit – executing single line of code

In [1]:
```
import random
%timeit -r5 -n100 test = [random.randrange(1,7) for i in range(10000)]
```

```
8.57 ms ± 127 µs per loop (mean ± std. dev. of 5 runs, 100 loops each)
```

In [2]:
```
import numpy as np
%timeit -r5 -n100 test = [np.random.randint(1,7,10000)]
```

```
115 µs ± 6.61 µs per loop (mean ± std. dev. of 5 runs, 100 loops each)
```

- %%timeit – executing full cell of code

In [3]:
```
%%timeit
import numpy as np
[np.random.randint(1,7,10000)]
```

```
118 µs ± 5.12 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

In [4]:
```
%prun?
```

NumPy Arrays – Indexing/Subscript

- Subscripting is similar to lists:

```
>>> arr = np.array([0,1,2,3,4,5])
>>> arr[0]
0
>>> arr[-1]
5
>>> arr[2:5]
array([2, 3, 4])
```

- Create an array based on the subscripts of another array:

```
>>> arr[2,0,2,0]
array([2, 0, 2, 0])
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Iteration

- Subscripting is similar to lists:

```
>>> arr = np.array([[1,2,3],[4,5,6]])
>>> for ele in np.nditer(arr):
    print(ele, end=' ')
# iterate in C language order
>>> for ele in np.nditer(arr, order='C'):
    print(ele, end=' ')
# iterate in Fortran language order
>>> for ele in np.nditer(arr, order='F'):
    print(ele, end=' ')
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – Slicing

- Any change to list slice is not reflected in the original list:

```
>>> lst = [0,1,2,3,4,5]
>>> lst_slice = l[1:4]
>>> lst_slice[0] = -1
>>> print(lst,lst_slice)
[-1, 2, 3] [0, 1, 2, 3, 4, 5]
```

- Any change to array slice is reflected in the original array:

```
>>> arr = np.array([0,1,2,3,4,5])
>>> arr_slice = a[1:4]
>>> arr_slice[0] = -1
>>> print(arr,arr_slice)
[-1, 2, 3] [ 0, -1,  2,  3,  4,  5]
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

NumPy Arrays – View versus Copy

- No copy:

```
>>> arr = np.array([0,1,2,3,4,5])
>>> arr_assign = arr
>>> print(id(arr),id(arr_assign))
4435408896 4435408896
```

- Shallow copy:

```
>>> arr_view = arr.view()
>>> print(id(arr),id(arr_view))
4435408896 4594152192
```

- Deep copy:

```
>>> arr_copy = arr.copy()
>>> print(id(arr),id(arr_copy))
4435408896 4594152272
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS