DS-GA 1007 | Lecture 5

Programming for Data Science

Jeremy Curuksu, PhD
NYU Center for Data Science
jeremy.cur@nyu.edu

December 17, 2022

DS-GA 1007 Curriculum

Programming for Data Science:

- Introduction to Programming in Python
- Best Practice Programming and Software Engineering
- Program Efficiency
- Interacting with Programs
- **Array Manipulation for Scientific Computing**
- Data Visualization
- Advanced Data Objects ($\times 4$)
- Environments for Collaborative Programming
- Industrial Applications

Array Manipulation for Scientific Computing

Last week:

- Python Editors and IDEs
- Distributions, Packages and Virtual Environments
- OS Interfaces to Interact with Programs

Today:

- **Array Manipulation with NumPy**
- **Mathematical Operations with NumPy**

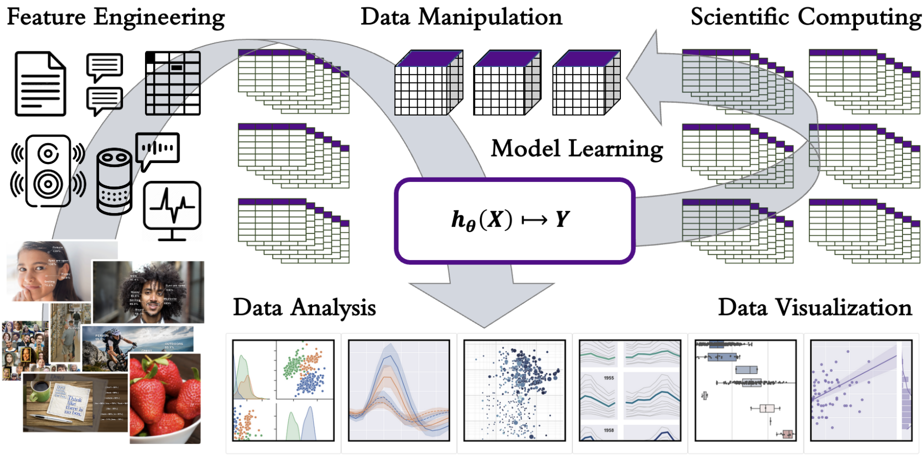
Array Manipulation for Scientific Computing

Do not forget:

- Today's lecture includes practice code examples in an accompanying Jupyter notebook

Array Manipulation for Scientific Computing

Programming for Data Science



Feature Engineering

Data Manipulation

Scientific Computing

Model Learning

$$h_\theta(X) \longmapsto Y$$

Data Analysis

Data Visualization

Programming for Data Science

- Manipulation of data objects
- Scientific computing with numerical arrays
- Feature Engineering
- Mathematical operations, linear algebra
- Modeling
- Machine Learning
- Statistical analysis
- Vizualization

Array Manipulation with NumPy

Array Manipulation with NumPy

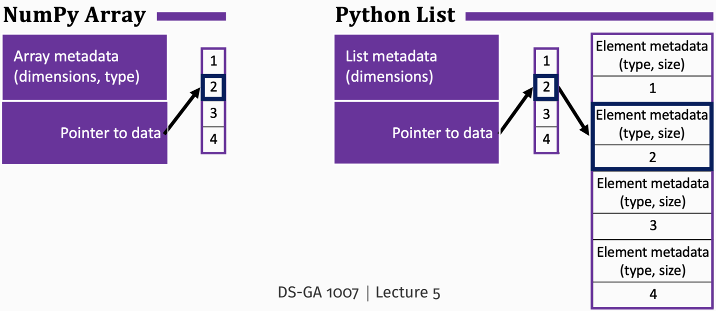
We will cover:

1. Arrays vs. List
2. Creating and reshaping arrays
3. Array attributes and built-in methods
4. Indexing arrays
5. Slicing arrays
6. Fancy indexing and slicing
7. Broadcasting

What are NumPy Arrays?

NumPy Array vs. Python List

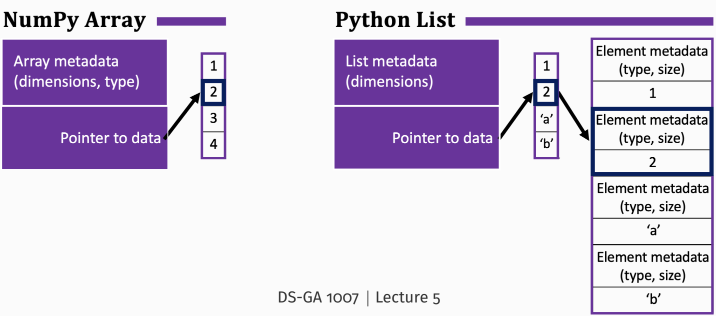
- Both are multi-element, mutable containers for data
- List elements are complete Python objects allowing for flexibility of heterogenous types
- Array elements are of fixed type allowing for more efficient storage and operations


NumPy Array
Array metadata (dimensions, type)
Pointer to data
1
2
3
4
Python List
List metadata (dimensions)
Pointer to data
1
2
3
4
Element metadata (type, size)
1
Element metadata (type, size)
2
Element metadata (type, size)
3
Element metadata (type, size)
4

What are NumPy Arrays?

NumPy Array vs. Python List

- Both are multi-element, mutable containers for data
- List elements are complete Python objects allowing for flexibility of heterogenous types
- Array elements are of fixed type allowing for more efficient storage and operations


NumPy Array
Array metadata (dimensions, type)
Pointer to data
1
2
3
4
Python List
List metadata (dimensions)
Pointer to data
1
2
'a'
'b'
Element metadata (type, size)
1
Element metadata (type, size)
2
Element metadata (type, size)
'a'
Element metadata (type, size)
'b'

Creating NumPy Arrays

There are many ways to create NumPy arrays:

```
l = [1,2,3,4,5]
np.array(l)
np.arange(0,10)
np.arange(0,10,2)
np.linspace(0,10,3)
np.zeros(5)
np.arange(0,25).reshape(5,5)
np.random.rand(5,5)
np.zeros((5,5))
np.ones((5,5))
np.eye(5)
```

Array Attributes and Methods

There are key attributes of NumPy arrays

```
x.ndim   # Number of dimensions
x.shape  # Size of each dimension
x.size   # Total size of the array
x.dtype  # Data type of the array
```

...and many, many built-in methods such as:

```
x.max()  # or np.max(a)
x.min()  # or np.min(a)
x.mean() # or np.mean(a)
x.std()  # or np.std(a)
```

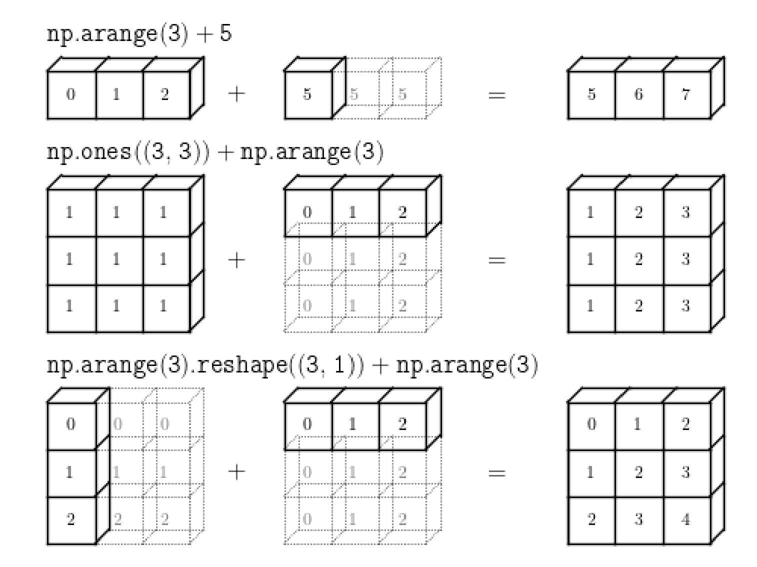
Array Indexing and Selection

Slicing, Broadcasting, and Fancy Indexing

```
a = np.arange(0,9)
a[5]    # Simple indexing
a[2:5]  # Slicing (view to access subarray, data not copied)
a[0:5]=100 # Scalar broadcasting
np.ones((3,3)) + np.arange(3) # Array broadcasting

a2 = a.reshape(3,3)
a2[1,0]       # or a2[1][0] but not recommended
a2[1:,:]      # Slicing in 2D
a2[[2,0]]     # Fancy: Select entire rows out of order
a2[:,[2,0,2]] # Fancy: Select entire columns out of order
a[a>5]        # Fancy: a > 5 is itself a Boolean array...
```

NumPy Rules of Broadcasting



Mathematical Operations with Numpy

Mathematical Operations on Arrays

NumPy Functions perform Vectorized Operations

- Arithmetic, trigonometric, and common mathematical functions

Arithmetic		**Trigonometric**		**Exp/Log/Abs**	
$+$	Addition	np.sin(x)	Sine	np.exp(x)	e^x
$-$	Substraction	np.cos(x)	Cosine	np.exp2(x)	2^x
$*$	Multiplication	np.tan(x)	Tangent	np.power(3,x)	3^x
$/$	Division			np.log(x)	$ln(x)$
$//$	Floor division			np.log2(x)	$log2(x)$
$\%$	Modulus			np.log10(x)	$log10(x)$
$**$	Exponentiation			np.abs(x)	$\lvert x \rvert$

Mathematical Operations on Arrays

NumPy Functions perform Vectorized Operations

- Basic statistics, aggregation, and comparison functions

Basic Statistics		**Aggregation**		**Comparison**	
np.mean	Mean	np.sum	Sum	==	Equality
np.std	Std Dev	np.prod	Product	! =	Inequality
np.var	Variance	np.min	Min value	>	More than
np.median	Median	np.max	Max value	>=	More or equal
np.percentile	Rank-statistics	np.argmin	Index min	<	Less than
		np.argmax	Index max	<=	Less or equal

Linear Algebra in NumPy

A NumPy Array can be used as a Matrix

Matrix and Vector Product		Other Matrix Operations*	
@	Matrix product	np.corrcoef	Correlation matrix
np.matmul	Matrix product	linalg.inv	Inverse matrix
np.dot	Dot product	linalg.norm	Norm
np.innner	Inner product	linalg.det	Determinant
np.outer	Outer product	linalg.eig	Eigenvalues

* More Linear Algebra Numpy methods are available, see link below:
`https://numpy.org/doc/stable/reference/routines.linalg.html`

A refresh in Linear Algebra

Matrix Form of a Linear Regression Model $Ax = y$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

$$(m \times n) \quad \times \quad (n \times 1) \quad = \quad (m \times 1)$$

Example: m = 100 observations, n = 10 features:

$$\begin{bmatrix} a_{1,1} & \cdots & a_{1,10} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ a_{100,1} & \cdots & a_{100,10} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_{10} \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ \vdots \\ y_{100} \end{bmatrix}$$

Now let’s practice!