L3. Python

UCLA Masters of Applied Economics

Fall 2018

Melody Y. Huang

From last time…

- We learned about dictionaries, sets, lists, and list comprehension
- We demonstrated how dictionaries can be very computationally efficient and we can use this to speed up a lot of time consuming operations!
- Today we will introduce some new objects that can help us with our future analysis

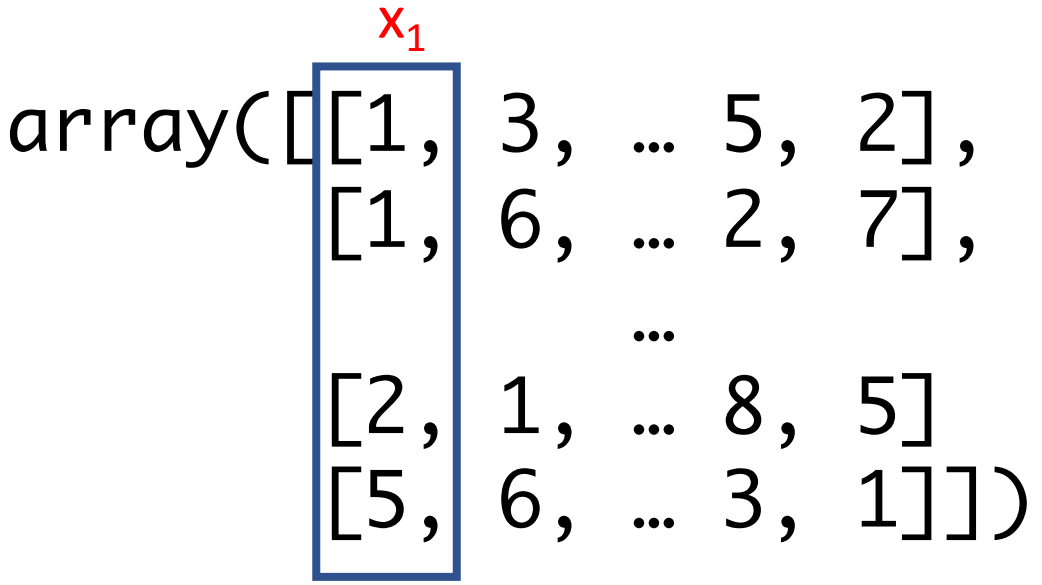
Basics of NumPy

- What is NumPy?
  - Allows for matrix representation in Python
  - We call these objects arrays in Python

- When would we ever use this?
  - Bread and butter of machine learning!
  - Complex data can be represented in this form
    - Ex: images, audio clips, etc.
  - Much of the packages that implement data science analytic tools on Python require the input of NumPy arrays

Intuition

- What does a NumPy array represent in the context of data analytics?



Creating an Array

- An NumPy (abbreviated as np) array can be constructed from a regular Python list, tuple, or Pandas series

- The input MUST be a container that has your data

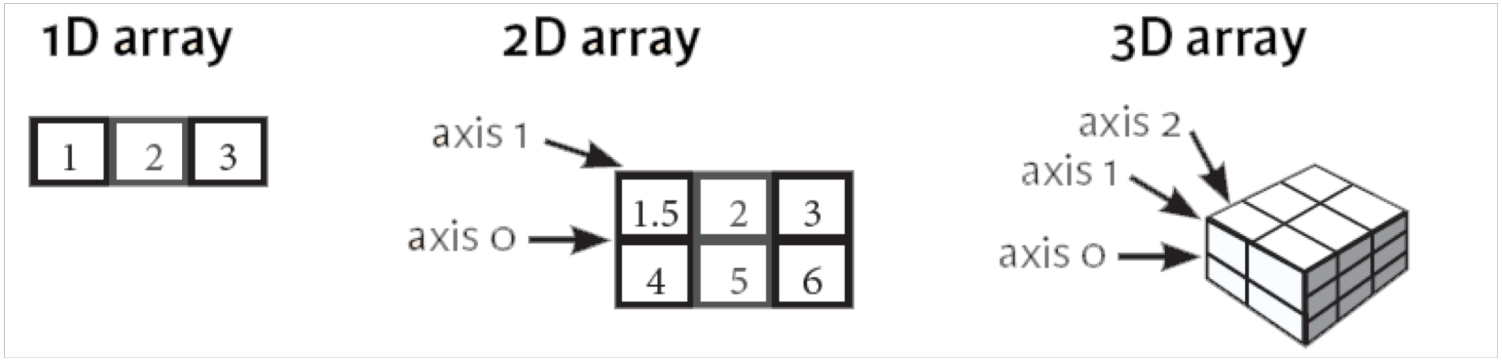
- Example:

```
np.array(1,2,3,4,5) #will NOT work
np.array([1,2,3,4,5]) #works
```

Creating an Array (cont.)

- What if you have lists within lists?
  - np.array() will convert these lists within lists into n-dimensional arrays (n = subgroups)
  - Example:

```
print np.array([[1,3,5,2], [1,6,2,7]])
array([[1, 3, 5, 2],
       [1, 6, 2, 7]])
```



Attributes in Numpy Objects

- Important Attributes:
  - ndim: # of axes of the array
  - shape: dimensions of the array
    - In the form of (row, column)
  - size: total number of elements in the array
  - dtype: Python types of the elements in the array (i.e., float, integers, etc.)

Array Types

- In Numpy, we have more flexibility to work with different numerical types:
  - int8
  - int16
  - int32
  - uint8
  - uint16
  - unint32… etc.

Note:
Unsigned ints hold only positive values (> 0), while signed ints have both positive and negative values

- To initialize an array to be a certain type:

```
np.array([1,3,4,5], dtype=np.uint8)
```

Helpful Functions

- Array of all zeroes

```
np.zeros((n,m))
```

- Array of all ones

```
np.ones((n,m))
```

- Constant arrays

```
np.full((n, m), c) #c is constant
```

- Identity Matrix:

```
np.eye(n) #Creates nxn identity matrix
```

- Random values:

```
np.random.random((n,m))
```

Indexing

- This is where NumPy arrays get potentially complicated
- Because you have a variety of different dimensions, when you try to retrieve data from your matrix, you have to specify all dimensions
- Example:

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
a[:,1:3] #will retrieve the 2nd and 3rd columns
```

Elementwise Operations

- Basic operations:
  - Adding a scalar will just add the scalar to each element in the matrix
  - All arithmetic operates elementwise (same with multiplying b a scalar)

- Array multiplication will **not** be matrix multiplication!
  - For matrix multiplication: `np.dot(x1, x2)`

SciPy - Linear Algebra

- We can pair NumPy with Python's SciPy library to perform a lot of different linear algebra operations!

- Some sample operations from SciPy:
  - Computing the inverse
  - Computing determinants
  - Computing eigenvalues

Exercises

1. Create a 3x3 matrix with values ranging from 0 to 8.
2. Create a 10x10 array with random values and find the minimum and maximum values
3. Create a null vector (vector of all zeroes) of size 10 but the fifth value which is 1
4. Create a random vector of size 30 and find the mean value

Solutions

```
import numpy as np
np.arange(9).reshape(3,3)
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

Solutions (cont.)

```
Z = np.random.random(100).reshape(10,10)
np.min(Z)
np.max(Z)
#Other sample stats:
print(np.mean(Z))
print(np.median(Z))
print(np.std(Z))
```

Solutions (cont.)

```
Z = np.zeros(10)
Z[4] = 1
print(Z)

Z = np.random.random(30)
np.mean(Z)
```

Pandas

- Now that we have introduced matrices, we can introduce data frames
- Python has no built-in data frame function
- Therefore, we must use the library Pandas

What is Pandas?

- Python's response to R's DataFrame object
- Combines some functionalities from DataFrames in R, as well as the dplyr library (SQL-like join functions)
- Allows for writing data into CSV and text files (also can write data frames into Excel, SQL data bases, and HDF5, which is commonly used in big data)
- Handling NA's
- Also commonly used for time series

Basic Syntax

- Create a Pandas Data Frame from scratch:

```
series1 = pd.Series([1,3,5,np.nan,6,8])
#Results in one column
```

- To create a multiple column data frame:

```
df1 = pd.DataFrame({"column1":[3,6,1,7],
                    "column2":[1,7,2,7]})
```

Analogies to R's Data Frame:

R	Pandas
head(df)	df.head()
tail(df)	df.tail()
summary(df)	df.describe()
df$column1	df['column1']
df[3,]	df.iloc[2]
na.omit(df)	df.dropna(how='any')

Some Nice Additional Features

- Allows for you to easily shift your series as needed
- Example:

```
series1 = pd.Series([1,3,5,np.nan,6,8])
```

1	3	5	NA	6	8

```
print(series1.shift(1))
```

NA	1	3	5	NA	6

Additional Features (cont.)

- Can easily join data frames together using merge:

df1

studentID	name
23095	Jill
10956	Heather
24096	Brad

df2

studentID	grade
23095	A
10956	B
24096	A-

```
pd.merge(df1, df2, on='studentID')
```

Additional Features (cont.)

- Adding rows: `df1.append(df2)`
- Adding columns: `pd.concat(df1, df2)`