Lesson 9

- Numpy

References and Resources

- http://learnpython.org/
- https://docs.python.org/3/tutorial
- https://www.w3schools.com/python/
- http://numpy.org/
- http://pandas.pydata.org/

Preliminary

- Essential libraries for Data Science:
  - Numpy, scipy, pandas, matplotlib, seaborn, sklearn, statsmodels

Libraries, packages, modules

- In your code, before each is used, usually at the top of the file
    - **`import math, array, statistics`**
    - **`from collections import deque`**

    - **`import numpy as np`**
    - **`import pandas as pd`**
    - **`import matplotlib.pyplot as plt`**

Numpy

Numpy

- https://docs.scipy.org/doc/numpy/index.html
- Uses ndarray for fast, efficient multi-dimensional arrays
- Math functions for those arrays
- Linear algebra, random numbers
- Indexing and slicing as with lists and tuples

Ndarray

- N-dimensional array
- Unlike a list, all elements are of same type
- Fast for numerical operations
- Basis for pandas and matplotlib

Random Numbers

- `import numpy as np`
- `np.random.randint(0,1000,75)    # 75 ints on 0 to 999`
- `np.random.rand(75)              # 75 floats on 0 to 1`
- `np.random.randn(75)             # 75 floats on mean=0, sd=1`

Numpy examples

```
• import numpy as np
• list1=[20, 35, 77, 42, 3, 51]
• arr=np.array(list1)     # 1-d array or vector
• print(arr)
• print(arr.min())        # 3
• print(arr.max())        # 77
• print(arr.sum())        # 228
• print(arr.mean())       # 38
• Print(arr.sd())
• print(arr.argmin())     # 4
• print(arr.argmax())     # 2
• print(arr.size)         # 6
• print(arr.dtype)        # int32
• print(arr.shape)        # 1x6
• print(type(arr))        # ndarray
```

More numpy examples: conditional selection!

```
bool_arr=arr > 20
print(bool_arr)
print(arr[bool_arr])    # conditional selection
print(arr[arr > 20])    # preferred
```

Matrices (matrixes)

```
# A 2-d array is known as a matrix
arr2d=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(type(arr2d))
print(arr2d)
print(arr2d[0][2])      # double bracket notation
print(arr2d[0,2])  # single bracket notation, preferred
print(arr2d[:2,1:])     # rows 0,1 & cols 1,2
print(arr2d[0])         # just row 0
print(arr2d[0].shape)   # 1x3
print(arr2d[:,0])       # just col 0
print(arr2d[:,0].shape) # 3x1
```

Array operations in NumPy

- https://docs.scipy.org/doc/numpy-1.15.1/reference/ufuncs.html

```
import numpy as np
ar=np.arange(6)
xa=np.linspace(0,4*np.pi,201)
print(ar+ar, ar-ar)        # array to array operations
print(ar*ar, ar/ar)        # note warning for 0/0
print(ar*100)              # scalar to array operations
print(np.exp(ar))          # see list of ufuncs in docs
print(np.log(ar))          # note first value -inf
print(1/0)                 # note that this is an error
```

More NumPy: matrix multiplication

```
# reminder number of cols in b1 must = number of rows in b2
b1=np.arange(1,5).reshape(4,1)   # 4 rows of 1 col
b2=np.arange(1,4).reshape(1,3)   # 1 row of 3 cols
b=np.dot(b1,b2)                  # 4x3
print(b1)
print(b2)
print(b)
```

Matrix mult (mat_mult.py)

```
# #cols in A must equal #rows in B, here that is 4
# result will have #rows as in A and #cols as in B, here 2x3
import numpy as np
A=np.arange(1,9).reshape(2,4)
B=np.arange(1,13).reshape(4,3)
C=A.dot(B)
print(A)
print(B)
print(C)  #  70  80  90
          # 158 184 210

# Do this by hand to gain confidence!
```

Matrices (2-D arrays) in NumPy

```
arr2d=np.array([[10,20,30],[40,50,60],[70,80,90]])
print(type(arr2d))
print(arr2d)
print(arr2d[0][2])      # double bracket notation
print(arr2d[0,2])       # single bracket notation
                        # preferred
print(arr2d[:2,1:])     # rows 0,1 & cols 1,2
print(arr2d[0])         # just row 0
print(arr2d[0].shape)   # 1x3
print(arr2d[:,0])       # just col 0
print(arr2d[:,0].shape) # 3x1
```

Numpy.linalg

- diag, dot, trace, det, eig, inv, solve …
- Standard linear algebra functions

Library np.linalg

```
import numpy as np
# Determinant
A=np.matrix([ [3, 2], [1, 6] ])
print('Det(A) = {:.2f}'.format(np.linalg.det(A)))   # 16.00

# Inverse: matrix must be square and its determinant nonzero
A=np.matrix([ [4, 3], [5, 4] ])
print(np.linalg.inv(A))          # [[4, -3], [-5, 4]]

# Verify that A x its inverse = identity matix
print(A.dot(np.linalg.inv(A)))   # Identity matrix 2 x 2
```

Examples of random draws

```
# Random uniform, normal, exponential distributions
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(51)
NPTS=10000

x1=np.random.rand(NPTS)
print(x1.mean())                       # 0.5

mu=100
sigma=15
x2 = sigma * np.random.randn(NPTS) + mu
print(x2.mean(), x2.std())             # 100, 15
plt.hist(x2,bins=50)
plt.show()
```

Exponential draw

```
mean_arr_interval = 10.0
x3 = np.random.exponential(mean_arr_interval,NPTS)
print(x3.mean())                       # 10
plt.hist(x3,bins=50)
plt.show()
```

Vectorized math (like R and Matlab)

If an np array is defined, then functions may be applied term by term

No explicit loop – it is implied

```
> NPTS=101
> n = np.arange(0, NPTS)         # a vector
> x = 2 * np.pi * n / (NPTS-1)   # also a vector
> y = np.cos(x)                  # also a vector
```

More numpy

```
import numpy as np

N = 9
x = np.linspace(0, 2*np.pi, N)
y = np.sin(x)

z = np.zeros(N)
fives = 5 * np.ones(N)

for i in range(N):
    print(f'{x[i]:.4f} {y[i]:.4f} \
    {z[i]:.1f} {fives[i]:.1f}')
```

ufuncs

- Abs, sqrt, exp, log, sin, cos, add, subtract, multiply, divide, power, maximum, minimum, …

- Vectorized functions for ndarrays