**NumPy**

**General Notes**

* Must make sure numpy is installed (pip3 install numpy, at cmd line prompt)
* Must import the numpy module
  + import numpy as np
* Data in each column must be the same type
  + 2d numpy is a list of lists
  + each list can only include one data type
  + use **pandas** if you need different data types

**Subsetting**

* 0 indexed
* 2D arrays
  + *my\_array*[2,4] # accesses the data in the third row “2” and the fifth column “4”
  + *my\_array*[:,4] # accesses all of the rows in the fifth column
  + *my\_array*[0,:] # accesses the entire first row (all columns)
* Using conditions for indexes
  + [*var condition value*]
    - will select all the data where the *value* of the specified *var* meets the *condition*
      * gk\_heights = np\_heights[np\_position == “GK”]

**Functions**

* np.mean(*set*) # calculates the mean for your set
  + Examples:
    - np.mean(*my\_array*[:,1]) # mean for the second column
* np.median(*set*)
  + Examples:
    - np.median(*my\_array*[:,6]) # median for the seventh column
* np.std(*set*) standard deviation
  + Examples:
    - np.std(*my\_array*[:,3] # std for the fourth column
* np.round(*set*, *places*)
* np.sum(*set*)
* np.sort(*set…*)
* np.transpose(*np\_array*)
  + a plot of a numpy array will often not look like what you want using matplotlib.pyplot
  + Try transposing first if you don’t get the desired result
    - *numpy\_array\_t* = np.transpose(*numpy\_array*)
    - plt.plot(*numpy­\_array\_t*)
* np.column\_stack((*col1, col2)[, (more here I think for extra dimensions)]*)
  + creates a np\_array (or matrix) by joining columns of data together
* np.corrcoef(*set1, set2*) # checks for correlation between *set1* and *set2*
  + Output
  + Examples:
    - np.corrcoef(*my\_array*[:,0], *my­\_array*[:,1]) # correlation between first and second columns

**Logical Operators** # and, or, and not do not work on numpy arrays

* logical\_and()
  + np.logical\_and(*my\_array* condition1, *my\_array* condition2)
    - np.logical\_and(my\_array > 23, my\_array < 30) # returns True/False array for each value
* logical\_or()
* logical\_not()
* can include these logical\_operators as the index value to return the actual values rather than an array of True/False # *my\_array*[np.logical\_or(*my\_array >= 1, my\_array < 10*)]

**Loops**

* For Loops
  + 1D arrays
    - for *value* in *array*:

*statements* referencing *value*

* + - for 2D arrays, this code will take each sub-array as a *value*, which may/not be what you want
  + 2D arrays
    - for *value* in np.nditer(*array*): # need nditer function

*statements* referencing *value*

* + - for 2D arrays, this code iterates through each value in the sub arrays one at a time in order

**Random Numbers**

* np.random.rand()
  + the random package in numpy will produce a random number between 0-1
* np.random.randint(*start, stop*)
  + generates a random integer where *start <= int < stop*
  + coin = np.random.randint(0, 2) if coin == 0: print(“heads”) else: print(“tails”)
* can specify a seed first
  + np.random.seed(*value*), then call the np.random.rand() function after setting the *seed*
    - seed should be an integer of our choosing
  + beware, calling np.random.rand() multiple times with the same seed will produce the same random numbers!
    - however, sometimes you want reproducibility, and supplying the same seed will ensure you get the same random numbers
    - setting the see will not generate the same random number every time, but it will result in the same sequence of random numbers when calling np.random repeatedly

**Generating Random Data**

* np.random.normal(*dist\_mean, dist\_std, num\_samps*)
  + creates a “random” “normal” distribution around the *dist\_mean* (distribution mean) with a specified *std* (standard deviation), and produces a specific *num\_samps* (number of samples)
  + can combine with the np.round(*np.rand…*, *places*) to round these data