Unit 4-1 NumPy and SciPy

* What are NumPy and SciPy?
  + Although they may sound like Muppet characters, NumPy and SciPy are in fact Python packages that you'll access from your own code. Using them reduces the amount of code you'll need to write when solving challenging problems.
  + SciPy is pronounced "sigh pie," while NumPy pronunciation falls into two camps: "num pie" and "num pee." We'll leave you to battle that out with your fellow programmers.
* The NumPy Package
  + What is NumPy and why do we care?
  + NumPy is a fundamental package for scientific computing with Python.
  + It provides us with:
    - Powerful vector and matrix creation and array math operations.
    - A rich set of advanced math operations and random number generation capabilities.
  + The best part? We can access these capabilities with just a few keystrokes.
* Accessing a Package From a Python Program
  + The Python packages you'll be accessing are available in your Anaconda environment, which comes with many of the most popular ones already installed. You can access them in just two steps.
  + Step 1: Import the package.
    - import numpy as np
  + We give NumPy the alias np so that we don't need to spell out its full name when writing NumPy functions in our code. Popular packages all have commonly used abbreviations like this.
  + Step 2: Access the package's capabilities.
    - my\_array = np.array([1, 2, 3]) # Create a vector (rank 1 array) from a list
    - print(my\_array)
    - print(type(my\_array))
    - Out[]:
    - [1 2 3]
    - <class 'numpy.ndarray'>
  + There you have it! You've accessed and used a Python package.
  + Note that not all packages are available in the Anaconda package manager. Instead, you may need to manually install the packages via the command line using pip install.
* Adding Vectors in NumPy
  + Vectors are a sequence of numbers. We can use linear algebra to perform additions and other operations with vectors and matrices. NumPy allows us to use a computer to perform the same kind of calculations.
    - # Create two arrays with different values.
    - a = np.array([1, 2, 3])
    - b = np.array([4, 5, 6])
    - # Add the two vectors together.
    - a + b
    - Out[]:
    - array([5, 7, 9])
  + We can also perform other operations in a similar manner, including subtraction, multiplication, dot products, etc.
* Random Values
  + We encounter random values when it's impossible to predict future values based on previous ones.
  + A random variable contains random values.
    - Discrete random variables
      * Discrete random variables have outcomes that are "countable," such as the number of heads in a series of coin flips or the number of aces drawn in a series of hands from a deck of cards. Each outcome is called the probability mass function, or pmf.
    - Continuous random variables
      * Continuous random variables, on the other hand, have outcomes that are not countable. Say, for example, T is a random variable representing the temperature at noon. There aren't exact temperature outcomes with associated probabilities of occurrence — the number of possible outcomes is infinite.
  + np.random
  + Generating arrays of random numbers With NumPy
  + NumPy functions make it easy to generate random numbers into an array.
    - # Single array with seven values.
    - np.random.rand(7)
    - Out[]:
    - array([ 0.21508189, 0.55978152, 0.64720631, 0.57905971, 0.46305698,
    - 0.70073415, 0.73717498])
    - # Three arrays each with two values (a.k.a., matrix).
    - np.random.rand(3,2)
    - Out[]:
    - array([[ 0.14022471, 0.96360618], #random
    - [ 0.37601032, 0.25528411], #random
    - [ 0.49313049, 0.94909878]]) #random
* Generating Random Number Distributions
  + NumPy can generate random numbers according to a probability distribution. For example, we can quickly generate a normal distribution of values:
    - mean = 0
    - std = 10
    - series\_length = 20
    - normal\_series = np.random.normal(mean, std, series\_length)
    - print (normal\_series)
    - Out[]:
    - [ 1.87680468 4.92029761 -13.22123301 1.00525671 -5.88743497
    - 2.91884385 -12.3946797 6.94534691 5.43463746 -3.36199382
    - 16.9323576 10.96517156 2.69268736 -5.67856073 -4.02930
    - -2.28000912 -14.58239434 -16.22040117 7.45915422 4.32225146]
* What's the Difference Between NumPy and SciPy?
  + SciPy.org compares NumPy and SciPy this way:
    - In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic element-wise functions, etc. All numerical code would reside in SciPy.
    - However, one of NumPy’s important goals is compatibility, so NumPy tries to retain all features supported by either of its predecessors. Thus, NumPy contains some linear algebra functions, even though these more properly belong in SciPy.
    - In any case, SciPy contains more fully featured versions of the linear algebra modules, as well as many other numerical algorithms. If you are doing scientific computing with Python, you should probably install both NumPy and SciPy. Most new features belong in SciPy rather than NumPy.
* NumPy Attributes and Methods
  + An object in Python has inherent qualities it knows about itself (attributes) and functions it inherently knows how to perform (methods).
  + Attributes and methods are similar in that they both can be used at the end of a NumPy object with a similar structure.
  + For example, NumPy has a .shape attribute that allows us to get the dimensions of a NumPy array. It also has methods, such as .mean() and .median(), that perform calculations.
* Using NumPy Attributes to Find Out More About an Array
  + Determining the dimensions of an array using the .shape attribute:
    - import numpy as np
    - my\_array = np.array([[2,1],[7,4],[5,2],[6,3]])
    - print(my\_array)
    - my\_array.shape # Return the number of rows and columns.
    - [Out]
    - [[2 1]
    - [7 4]
    - [5 2]
    - [6 3]]
    - (4, 2)
* Using NumPy Methods to Calculate Basic Statistics on an Array
  + Finding the mean and median of a list:
  + import numpy as np
  + from scipy import stats
  + sample = [3, 75, 98, 2, 10, 3, 14, 99, 44, 25, 31, 100, 356, 4, 23, 55, 327, 64, 6, 20]
  + mean = np.mean(sample)
  + median = np.median(sample)
  + print("mean / median:", mean, median)
  + Out[]:
  + mean / median: 67.95 28.0
* Going Beyond NumPy to Calculate Stats on Arrays
  + NumPy doesn't include a function for calculating mode. Instead, a .mode() function is implemented in the SciPy package.
  + SciPy is easy to use, computationally optimized, and able to help us do heavy lifting. When possible, it's best to use functions from SciPy or NumPy instead of writing your own.
    - from scipy import stats
    - sample = [3, 75, 98, 2, 10, 3, 14, 99, 44, 25, 31, 100, 356, 4, 23, 55, 327, 64, 6, 20]
    - mode = stats.mode(sample)
    - print("mode, mount count: ", mode.mode, mode.count)
    - Out[]:
    - mode, mount count: [3] [2]