Numpy

Numpy (or NumPy) is an incredibly fast Linear Algebra Library for Python and it's used by any other Data Science library.

Installation Instructions

It is highly recommended that you install Python using the Anaconda distribution from the Environment tab in the Anaconda Navigator.

Using Numpy

Once installed NumPy you can import it as a library:

In [3]:
```
import numpy as np
```

Numpy Arrays

From a Python List

In [4]:
```
my_list = [1, 2, 3]
my_list
```

Out[4]:
```
[1, 2, 3]
```

In [5]:
```
np.array(my_list)
```

Out[5]:
```
array([1, 2, 3])
```

In [6]:
```
my_matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
my_matrix
```

Out[6]:
```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

In [7]:
```
np.array(my_matrix)
```

Out[7]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Built-in Methods

There are many other ways to generate Arrays

arange

Return evenly-spaced values within a given interval.

In [8]:
```
np.arange(0, 10)
```

Out[8]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [9]: np.arange(0, 11, 2)
```

```
Out[9]: array([ 0,  2,  4,  6,  8, 10])
```

zeros and ones

Generate arrays of zeros or ones

```
In [10]: np.zeros(3)
```

```
Out[10]: array([0., 0., 0.])
```

```
In [11]: np.zeros((5, 5))
```

```
Out[11]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

```
In [12]: np.ones(3)
```

```
Out[12]: array([1., 1., 1.])
```

```
In [13]: np.ones((3, 3))
```

```
Out[13]: array([[1., 1., 1.],
                [1., 1., 1.],
                [1., 1., 1.]])
```

linspace

Return evenly spaced numbers over a specified interval.

```
In [14]: np.linspace(0, 10, 3)
```

```
Out[14]: array([ 0.,  5., 10.])
```

```
In [15]: np.linspace(0, 10, 50)
```

```
Out[15]: array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
                 1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
                 2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
                 3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
                 4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
                 5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
                 6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
                 7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
                 8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
                 9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

eye

Creates an identity matrix

```
In [16]: np.eye(4)
```

```
Out[16]: array([[1., 0., 0., 0.],
                [0., 1., 0., 0.],
                [0., 0., 1., 0.],
                [0., 0., 0., 1.]])
```

Random

rand

Create an array of the given shape data from a uniform distribution in `[0, 1)`.

In [17]:
```
np.random.rand(2)
```

Out[17]:
```
array([0.36142418, 0.3337723 ])
```

In [18]:
```
np.random.rand(5, 5)
```

Out[18]:
```
array([[0.44331793, 0.74132989, 0.61869025, 0.23389701, 0.93044039],
       [0.03214202, 0.63727379, 0.90292092, 0.20701773, 0.46292102],
       [0.17762503, 0.42786384, 0.33644163, 0.32023486, 0.58649251],
       [0.91708992, 0.99219352, 0.50710684, 0.46091326, 0.28593914],
       [0.55931609, 0.80312598, 0.70020631, 0.78364126, 0.70142297]])
```

randn

Create an array of the given shape data from the "standard normal"

In [19]:
```
np.random.randn(2)
```

Out[19]:
```
array([-0.65909088,  0.4476652 ])
```

In [20]:
```
np.random.randn(5, 5)
```

Out[20]:
```
array([[-0.09292629,  2.08006322,  1.04895936,  0.2699721 , -0.40897547],
       [ 2.65214171,  0.75351175, -0.53535337, -0.14147868,  1.14375106],
       [-0.26547227, -1.38460249, -0.35079365, -0.2332907 , -1.09250361],
       [-0.07160806,  0.74491854, -1.55084443, -0.10281442, -0.49631293],
       [ 0.45831963, -0.03355836,  0.44819313, -0.54632515,  0.44169754]])
```

randint

Return random integers from `low` (inclusive) to `high` (exclusive).

In [21]:
```
np.random.randint(1, 100)
```

Out[21]:
```
11
```

In [22]:
```
np.random.randint(1, 100, 10)
```

Out[22]:
```
array([35, 94, 20, 61, 48, 36, 83, 59, 84, 66])
```

In [23]:
```
arr = np.arange(25)
ranarr = np.random.randint(0, 50, 10)
```

In [24]:
```
arr
```

Out[24]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

In [25]:
```
ranarr
```

Out[25]:
```
array([32,  9, 10, 42, 26, 11, 40,  4, 20, 12])
```

Reshape

Returns an array containing the same data with a new shape.

In [26]:
```
arr.reshape(5, 5)
```

Out[26]:
```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

max, min, argmax, argmin

Methods for finding `max` or `min` values or to find their index locations using `argmin` or `argmax`

In [27]:
```
ranarr
```

Out[27]:
```
array([32,  9, 10, 42, 26, 11, 40,  4, 20, 12])
```

In [28]:
```
ranarr.max()
```

Out[28]:
```
42
```

In [29]:
```
ranarr.argmax()
```

Out[29]:
```
3
```

In [30]:
```
ranarr.min()
```

Out[30]:
```
4
```

In [31]:
```
ranarr.argmin()
```

Out[31]:
```
7
```

Shape

Shape is an attribute that arrays have (not a method):

In [32]:
```
# Vector
arr.shape
```

Out[32]:
```
(25,)
```

In [33]:
```
# Notice the two sets of brackets
arr.reshape(1, 25)
```

Out[33]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

In [34]:
```
arr.reshape(1, 25).shape
```

Out[34]:
```
(1, 25)
```

In [35]:
```
arr.reshape(25, 1)
```

Out[35]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24]])
```

In [36]:
```
arr.reshape(25, 1).shape
```

Out[36]:
```
(25, 1)
```

dtype

You can also grab the data type of the object in the array:

In [37]:
```
arr.dtype
```

Out[37]:
```
dtype('int64')
```

NumPy Indexing and Selection

How to select elements or groups of elements from an array.

In [4]:
```
import numpy as np
```

In [5]:
```
# Creating sample array
arr = np.arange(0, 11)
```

In [6]:
```
arr
```

Out[6]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Slicing

As in python lists

In [7]:
```
# Get a value at an index
arr[8]
```

Out[7]:
```
8
```

In [8]:
```
# Get values in a range
arr[1:5]
```

Out[8]:
```
array([1, 2, 3, 4])
```

Broadcasting

Numpy arrays differ from a normal Python list because of their ability to broadcast:

In [9]:
```
# Setting a value to all these indexes (Broadcasting)
arr[0:5] = 100
arr
```

Out[9]:
```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

Let's create a new array

In [10]:
```
arr = np.arange(0, 11)
arr
```

Out[10]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [11]:
```
slice_of_arr = arr[0:6]
slice_of_arr
```

Out[11]:
```
array([0, 1, 2, 3, 4, 5])
```

In [12]:
```
slice_of_arr[:] = 99
slice_of_arr
```

Out[12]:
```
array([99, 99, 99, 99, 99, 99])
```

The changes also occur in the original array!

In [13]:
```
arr
```

Out[13]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Data is not copied, it's a view of the original array!

In [14]:
```
# To get a copy, need to be explicit
arr_copy = arr.copy()
arr_copy
```

Out[14]:
```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Indexing a 2D array (matrices)

The general format is `arr_2d[row][col]` or `arr_2d[row,col]` . Use comma notation for clarity.

In [15]:
```
arr_2d = np.array(([5, 10, 15], [20, 25, 30], [35, 40, 45]))
arr_2d
```

Out[15]:
```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

In [16]:
```
arr_2d[1]
```

Out[16]:
```
array([20, 25, 30])
```

In [17]:
```
arr_2d[1][0]
```

Out[17]:
```
20
```

In [18]:
```
# Getting individual element value
arr_2d[1,0]
```

Out[18]:
```
20
```

In [19]:
```
# 2D array slicing

# Shape (2,2) from top right corner
arr_2d[:2,1:]
```

Out[19]:
```
array([[10, 15],
       [25, 30]])
```

In [20]:
```
# bottom row
arr_2d[2]
```

Out[20]:
```
array([35, 40, 45])
```

In [21]:
```
# bottom row
arr_2d[2,:]
```

Out[21]:
```
array([35, 40, 45])
```

Fancy Indexing

In [34]:
```
# Set up a new matrix
arr2d = np.zeros((10,10))
```

In [23]:
```
# Length of array
arr_length = arr2d.shape[1]
```

In [24]:
```
# Set up array

for i in range(arr_length):
    arr2d[i] = i

arr2d
```

Out[24]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

Fancy indexing allows the following

In [25]:
```
arr2d[[2,4,6,8]]
```

Out[25]:
```
array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])
```

In [26]:
```
# in any order
arr2d[[6,4,2,7]]
```

Out[26]:
```
array([[6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])
```

More Indexing Help

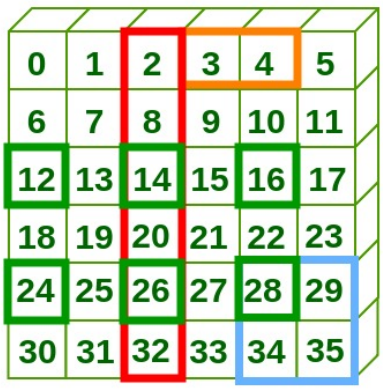
Indexing a 2d matrix can be a bit confusing at first, especially when you start to add in step size:

```
>>> a[0,3:5]
array( [3,4] )

>>> a[4:, 4:]
array( [ 28, 29],
       [ 34, 35] ] )

>>> a[ :, 2]
array( [2, 8, 14, 20, 26, 32] )

>>> a[2 : : 2, : : 2]
array( [ 12, 14, 16],
       [ 24, 26, 28] ] )
```



Selection

Selecting over comparison operators

In [27]:
```
arr = np.arange(1,11)
arr
```

Out[27]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [28]:
```
arr > 4
```

Out[28]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [29]:
```
bool_arr = arr > 4
```

In [30]:
```
bool_arr
```

Out[30]:
```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

In [31]:
```
arr[bool_arr]
```

Out[31]:
```
array([ 5,  6,  7,  8,  9, 10])
```

In [32]:
```
arr[arr>2]
```

Out[32]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

In [33]:
```
x = 2
arr[arr>x]
```

Out[33]:
```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

NumPy Operations

Arithmetic

You can perform array arithmetic

In [1]:
```
import numpy as np
arr = np.arange(0, 10)
```

In [2]:
```
arr + arr
```

Out[2]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [3]:
```
arr * arr
```

Out[3]:
```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

In [4]:
```
arr - arr
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [5]:
```
# Warning on division by zero, but not an error!
# Just replaced with nan (Not a Number)
arr / arr
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: RuntimeWarning:
invalid value encountered in true_divide
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[5]:
```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

In [6]:
```
# Also warning, but not an error instead inf (infinity)
1 / arr
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: RuntimeWarning:
divide by zero encountered in true_divide
```

Out[6]:
```
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
       0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111])
```

In [7]:
```
arr ** 3
```

Out[7]:
```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

Universal Array Functions

Numpy comes with [universal array functions (http://docs.scipy.org/doc/numpy/reference/ufuncs.html)](http://docs.scipy.org/doc/numpy/reference/ufuncs.html), which are mathematical operations you can use to perform the operation across the array.

In [13]:
```
# Square Roots
np.sqrt(arr)
```

Out[13]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

In [14]:
```
# exponential (e^)
np.exp(arr)
```

Out[14]:
```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

In [15]:
```
np.max(arr) # same as arr.max()
```

Out[15]:
```
9
```

In [16]:
```
np.sin(arr)
```

Out[16]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [17]:
```
np.log(arr)
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning:
divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

Out[17]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
```

NumPy Exercises

Write the code to reproduce the expected output that follows the code cell.

Import NumPy as np

```
In [1]: import numpy as np
```

Create an array of 10 zeros

```
In [2]: np.zeros(10)
Out[2]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

Create an array of 10 ones

```
In [3]: np.ones(10)
Out[3]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

Create an array of 10 fives

```
In [4]: np.ones(10) * 5
Out[4]: array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

```
array([ 5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.,  5.])
```

Create an array of the integers from 10 to 50

```
In [5]: np.arange(10, 51)
Out[5]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
               27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
               44, 45, 46, 47, 48, 49, 50])
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
       27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
       44, 45, 46, 47, 48, 49, 50])
```

Create an array of all the even integers from 10 to 50

```
In [6]: np.arange(10, 51, 2)
```

```
Out[6]: array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
               44, 46, 48, 50])
```

```
array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
       44, 46, 48, 50])
```

Create a 3x3 matrix with values ranging from 0 to 8

```
In [7]: np.arange(9).reshape(3, 3)
```

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

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

Create a 3x3 identity matrix

```
In [8]: np.eye(3)
```

```
Out[8]: array([[1., 0., 0.],
               [0., 1., 0.],
               [0., 0., 1.]])
```

```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Use NumPy to generate a random number between 0 and 1

```
In [9]: np.random.rand(1)
```

```
Out[9]: array([0.45867295])
```

Use NumPy to generate an array of 25 random numbers sampled from a standard normal distribution

```
In [10]: np.random.randn(25)
```

```
Out[10]: array([ 0.79653559, -0.44319015,  0.95030889, -0.05770164,  1.13451699,
                -0.9737383 ,  0.17506728,  1.09478002,  1.13297785,  0.6581298 ,
                 0.38173961, -1.89487283, -1.08458719,  1.52402873,  1.08490149,
                 1.62586319,  0.56694894,  0.72246269,  0.6583532 , -1.26692379,
                 1.8798305 , -0.74896597, -1.02234438,  1.21546514,  1.43000788])
```

Create the following matrix:

In [11]:
```
np.arange(1, 101).reshape(10, 10) / 100
```

Out[11]:
```
array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
       [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
       [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
       [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
       [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
       [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
       [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
       [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
       [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
       [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1.  ]])
```

```
array([[ 0.01,  0.02,  0.03,  0.04,  0.05,  0.06,  0.07,  0.08,  0.09,  0.1 ],
       [ 0.11,  0.12,  0.13,  0.14,  0.15,  0.16,  0.17,  0.18,  0.19,  0.2 ],
       [ 0.21,  0.22,  0.23,  0.24,  0.25,  0.26,  0.27,  0.28,  0.29,  0.3 ],
       [ 0.31,  0.32,  0.33,  0.34,  0.35,  0.36,  0.37,  0.38,  0.39,  0.4 ],
       [ 0.41,  0.42,  0.43,  0.44,  0.45,  0.46,  0.47,  0.48,  0.49,  0.5 ],
       [ 0.51,  0.52,  0.53,  0.54,  0.55,  0.56,  0.57,  0.58,  0.59,  0.6 ],
       [ 0.61,  0.62,  0.63,  0.64,  0.65,  0.66,  0.67,  0.68,  0.69,  0.7 ],
       [ 0.71,  0.72,  0.73,  0.74,  0.75,  0.76,  0.77,  0.78,  0.79,  0.8 ],
       [ 0.81,  0.82,  0.83,  0.84,  0.85,  0.86,  0.87,  0.88,  0.89,  0.9 ],
       [ 0.91,  0.92,  0.93,  0.94,  0.95,  0.96,  0.97,  0.98,  0.99,  1.  ]])
```

Create an array of 20 linearly spaced points between 0 and 1:

In [12]:
```
np.linspace(0, 1, 20)
```

Out[12]:
```
array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

```
array([ 0.        ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.        ])
```

Numpy Indexing and Selection

In [13]:
```
mat = np.arange(1,26).reshape(5,5)
mat
```

Out[13]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

In [14]:
```
mat[2:,1:]
```

Out[14]:
```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

```
array([[12, 13, 14, 15],
       [17, 18, 19, 20],
       [22, 23, 24, 25]])
```

In [15]:
```
mat[3,4]
```

Out[15]:
```
20
```

```
20
```

In [16]:
```
mat[:3,1:2]
```

Out[16]:
```
array([[ 2],
       [ 7],
       [12]])
```

```
array([[ 2],
       [ 7],
       [12]])
```

In [17]:
```
mat[4,:]
```

Out[17]:
```
array([21, 22, 23, 24, 25])
```

```
array([21, 22, 23, 24, 25])
```

In [18]:
```
mat[3:5,:]
```

Out[18]:
```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

```
array([[16, 17, 18, 19, 20],
       [21, 22, 23, 24, 25]])
```

Methods

Get the sum of all the values in `mat`

In [19]:
```
mat.sum()
```

Out[19]:
```
325
```

```
325
```

Get the standard deviation of the values in `mat`

```
In [20]: mat.std()
Out[20]: 7.211102550927978
```

7.2111025509279782

Get the sum of all the columns in `mat`

```
In [21]: mat.sum(axis=0)
Out[21]: array([55, 60, 65, 70, 75])
```

array([55, 60, 65, 70, 75])