Numpy Basics

Welcome to section of Numpy and Pandas. This is the most used Python libraries for data science. NumPy consists of a powerful data structure called multidimensional arrays. Pandas is another powerful Python library that provides fast and easy data analysis platform.

NumPy is a library written for scientific computing and data analysis. It stands for numerical python and also known as array oriented computing.

The most basic object in NumPy is the ndarray, or simply an array which is an n-dimensional, homogeneous array. By homogenous, we mean that all the elements in a NumPy array have to be of the same data type, which is commonly numeric (float or integer).

Why Numpy?

convenience & speed

Numpy is much faster than the standard python ways to do computations.

Vectorised code typically does not contain explicit looping and indexing etc. (all of this happens behind the scenes, in precompiled C-code), and thus it is much more concise.

Also, many Numpy operations are implemented in C which is basically being executed behind the scenes, avoiding the general cost of loops in Python, pointer indirection and per-element dynamic type checking. The speed boost depends on which operations you're performing.

NumPy arrays are more compact than lists, i.e. they take much lesser storage space than lists

In [ ]:
```
```

In [7]:
```
import numpy
```

In [2]:
```
numpy.array([1,2,3])
```

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

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

In [8]:
```
a = np.array([1,2,3])
```

In [9]:
```
a
```

Out[9]:
```
array([1, 2, 3])
```

In [12]:
```
b = np.array([[1,2,3],[4,5,6]])
```

In [13]:
```
b
```

Out[13]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [14]:
```
b.shape
```

Out[14]:
```
(2, 3)
```

In [56]:
```
a.shape
```

Out[56]:
```
(100,)
```

In [58]:
```
b.dtype
```

Out[58]:
```
dtype('int64')
```

In [59]:
```
b.ndim
```

Out[59]:
```
2
```

In [17]:
```
print(type(a))
```

```
<class 'numpy.ndarray'>
```

In [18]:
```
print(type(b))
```

```
<class 'numpy.ndarray'>
```

In [19]:
```
np.arange(10)
```

Out[19]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Performance measurement

I mentioned that the key advantages of numpy are convenience and speed of computation.

You'll often work with extremely large datasets, and thus it is important point for you to understand how much computation time (and memory) you can save using numpy, compared to standard python lists.

In [20]:
```
c = range(10000)
%timeit [i**3 for i in c]
```

```
4.59 ms ± 160 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [21]:
```
c_numpy = np.arange(10000)
%timeit c_numpy**3
```

```
24 µs ± 474 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Still not convinced? want to see one more intresting example

In [22]:
```
l1 = range(10000)
l2 = [i**2 for i in range(10000)]
```

In [23]:
```
%timeit list(map(lambda x, y: x*y, l1, l2))
```

```
2.19 ms ± 140 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [24]:
```
a1 = np.array(l1)
b1 = np.array(l2)
```

In [25]:
```
%timeit a1*b1
```

```
8.39 µs ± 177 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [28]:
```
a1
```

Out[28]:
```
array([       0,        1,        4, ..., 99940009, 99960004, 99980001])
```

In [29]:
```
a1 * a1
```

Out[29]:
```
array([       0,        1,        4, ..., 99940009, 99960004, 99980001])
```

so I can do everything without even writing a loop? yes... ohh wao

Creating Numpy array

There are multiple ways to create numpy array. Lets walk over them

In [105…
```
np.arange(2,12)
```

Out[105…
```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

In [106…
```
np.arange(2,12,2)
```

Out[106…
```
array([ 2,  4,  6,  8, 10])
```

In [35]:
```
np.zeros((3,2))
```

Out[35]:
```
array([[ 0.,  0.],
       [ 0.,  0.],
       [ 0.,  0.]])
```

In [36]:
```
np.ones((3,2))
```

Out[36]:
```
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
```

In [38]:
```
np.eye(3)
```

Out[38]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

In [40]:
```
np.full((3,3),2)
```

Out[40]:
```
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
```

In [92]:
```
np.full((3,3),2.2, dtype= np.int)
```

Out[92]:
```
array([[2, 2, 2],
       [2, 2, 2],
       [2, 2, 2]])
```

In [135…
```
np.diag([1,2,3,4,5])
```

Out[135…
```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0],
       [0, 0, 0, 0, 5]])
```

In [142…
```
v = np.array([1,2,3])
np.tile(v,(3,1)) # stack 3 copies of v on top of each other
```

Out[142…
```
array([[1, 2, 3],
       [1, 2, 3],
       [1, 2, 3]])
```

In [ ]:
```

```

In [44]:
```
# between 0 and 1
np.random.random()
```

Out[44]:
```
0.8105331407200129
```

In [46]:
```
# so let say I want a random value between 2 and 50
50*np.random.random()+2
```

Out[46]:
```
39.48149909935893
```

In [48]:
```
np.random.random([3,3])
```

Out[48]:
```
array([[ 0.66403971,  0.9379972 ,  0.43803717],
       [ 0.7747613 ,  0.44762134,  0.96971174],
       [ 0.06266723,  0.48139333,  0.7745762 ]])
```

In [51]:
```
# 100 values between 1 and 50
a = np.linspace(1,50,100)
```

In [52]:
```
a
```

Out[52]:
```
array([  1.        ,   1.49494949,   1.98989899,   2.48484848,
         2.97979798,   3.47474747,   3.96969697,   4.46464646,
         4.95959596,   5.45454545,   5.94949495,   6.44444444,
         6.93939394,   7.43434343,   7.92929293,   8.42424242,
         8.91919192,   9.41414141,   9.90909091,  10.4040404 ,
        10.8989899 ,  11.39393939,  11.88888889,  12.38383838,
        12.87878788,  13.37373737,  13.86868687,  14.36363636,
        14.85858586,  15.35353535,  15.84848485,  16.34343434,
        16.83838384,  17.33333333,  17.82828283,  18.32323232,
        18.81818182,  19.31313131,  19.80808081,  20.3030303 ,
        20.7979798 ,  21.29292929,  21.78787879,  22.28282828,
        22.77777778,  23.27272727,  23.76767677,  24.26262626,
        24.75757576,  25.25252525,  25.74747475,  26.24242424,
        26.73737374,  27.23232323,  27.72727273,  28.22222222,
        28.71717172,  29.21212121,  29.70707071,  30.2020202 ,
        30.6969697 ,  31.19191919,  31.68686869,  32.18181818,
        32.67676768,  33.17171717,  33.66666667,  34.16161616,
        34.65656566,  35.15151515,  35.64646465,  36.14141414,
        36.63636364,  37.13131313,  37.62626263,  38.12121212,
```

```
       38.61616162,  39.11111111,  39.60606061,  40.1010101 ,
       40.5959596 ,  41.09090909,  41.58585859,  42.08080808,
       42.57575758,  43.07070707,  43.56565657,  44.06060606,
       44.55555556,  45.05050505,  45.54545455,  46.04040404,
       46.53535354,  47.03030303,  47.52525253,  48.02020202,
       48.51515152,  49.01010101,  49.50505051,  50.        ])
```

In [98]:
```
#memory used by each array element in bytes
a.itemsize
```

Out[98]:
```
8
```

In [101…
```
np.arange(24)
```

Out[101…
```
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])
```

In [107…
```
np.arange(18).reshape(2,3,3)
```

Out[107…
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

In [109…
```
# -1 will automatically adjust dimention
np.arange(18).reshape(2,3,-1)
```

Out[109…
```
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8]],

       [[ 9, 10, 11],
        [12, 13, 14],
        [15, 16, 17]]])
```

accessing Numpy array element

In [80]:
```
a = np.array([2,4,6,8,10,12,14,16])
```

In [81]:
```
a
```

Out[81]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16])
```

In [82]:
```
a[2]
```

Out[82]:
```
6
```

In [83]:
```
a[[2,4,6]]
```

Out[83]:
```
array([ 6, 10, 14])
```

In [84]:
```
a[2:]
```

Out[84]:
```
array([ 6,  8, 10, 12, 14, 16])
```

In [85]:
```
a[2:5]
```

Out[85]:
```
array([ 6,  8, 10])
```

In [86]:
```
a[0::2]
```

Out[86]:
```
array([ 2,  6, 10, 14])
```

Lets check the same for 2 D array

In [122…
```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

In [123…
```
a
```

Out[123…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [124…
```
a[2,2]
```

Out[124…
```
9
```

In [127…
```
a > 2
```

Out[127…
```
array([[False, False,  True],
       [ True,  True,  True],
       [ True,  True,  True]], dtype=bool)
```

In [128…
```
a[a > 2]
```

Out[128…
```
array([3, 4, 5, 6, 7, 8, 9])
```

In [129…
```
a[(a > 2) & (a < 5)]
```

Out[129…
```
array([3, 4])
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

subset of numpy array

In [161…
```
a = np.arange(10)
```

In [162…
```
b = a
```

In [62]:
```
b
```

Out[62]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [63]:
```
b[0] = 11
```

In [64]:
```
b
```

Out[64]:
```
array([11,  1,  2,  3,  4,  5,  6,  7,  8,  9])
```

In [163…
```
# Notice a is also changed
a
```

Out[163…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [164…
```
np.shares_memory(a,b)
```

Out[164…
```
True
```

In [165…
```
a = np.arange(10)
```

In [166…
```
b = a.copy()
```

In [68]:
```
b[0] = 11
```

In [167…
```
b
```

Out[167…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [168…
```
np.shares_memory(a,b)
```

Out[168…
```
False
```

In [70]:
```
a
```

Out[70]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

More operations

In [171…
```
a = np.array([[1,2,3],[4,5,6]])
```

In [113…
```
a
```

Out[113…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [114…
```
a.T
```

Out[114…
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [172…
```
b = np.array([[7,8,9],[10,11,12]])
```

In [116…
```
a
```

Out[116…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [170…
```
b
```

Out[170…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [174…
```
a == b
```

Out[174…
```
array([[False, False, False],
       [False, False, False]], dtype=bool)
```

In [ ]:
```

```

In [118…
```
np.vstack((a,b))
```

Out[118…
```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

In [169…
```
np.hstack((a,b))
```

Out[169…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

MAthmatical operation

In [ ]:
```

```

In [190…
```
a = np.arange(1,10)
```

In [131…
```
np.sin(a)
```

Out[131…
```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427,
       -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

In [132…
```
np.cos(a)
```

Out[132…
```
array([ 0.54030231, -0.41614684, -0.9899925 , -0.65364362,  0.28366219,
        0.96017029,  0.75390225, -0.14550003, -0.91113026])
```

In [133…
```
np.exp(a)
```

Out[133…
```
array([  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 [191…
```
np.sum(a)
```

Out[191…
```
45
```

In [193…
```
np.median(a)
```

Out[193…
```
5.0
```

In [194…
```
a.std()
```

Out[194…
```
2.5819888974716112
```

In [147…
```
a = np.arange(1,10).reshape(3,3)
```

In [148…
```
np.linalg.det(a)
```

Out[148…
```
-9.5161973539299405e-16
```

In [149…
```
np.linalg.inv(a)
```

Out[149…
```
array([[  3.15251974e+15,  -6.30503948e+15,   3.15251974e+15],
       [ -6.30503948e+15,   1.26100790e+16,  -6.30503948e+15],
       [  3.15251974e+15,  -6.30503948e+15,   3.15251974e+15]])
```

In [156…
```
np.linalg.eig(a)
```

Out[156…
```
(array([  1.61168440e+01,  -1.11684397e+00,  -9.75918483e-16]),
 array([[-0.23197069, -0.78583024,  0.40824829],
        [-0.52532209, -0.08675134, -0.81649658],
        [-0.8186735 ,  0.61232756,  0.40824829]]))
```

In [157…
```
a
```

Out[157…
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [158…
```
b = a.T
```

In [159…
```
b
```

Out[159…
```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

In [160…
```
np.dot(a,b)
```

Out[160…
```
array([[ 14,  32,  50],
       [ 32,  77, 122],
       [ 50, 122, 194]])
```

In [187…
```
a = np.array([1,1,0], dtype = bool)
b = np.array([1,0,1], dtype = bool)
```

In [176…
```
np.logical_or(a,b)
```

Out[176…
```
array([ True,  True,  True], dtype=bool)
```

In [188…
```
np.logical_and(a,b)
```

Out[188…
```
array([ True, False, False], dtype=bool)
```

In [189…
```
np.all(a == a)
```

Out[189…
```
True
```

In [179…
```
a = np.array([[1,2],[3,4]])
```

In [180…
```
a
```

Out[180…
```
array([[1, 2],
       [3, 4]])
```

In [181…
```
a.sum()
```

Out[181…
```
10
```

In [182…
```
a.sum(axis=0)
```

Out[182…
```
array([4, 6])
```

In [183…
```
a.sum(axis=1)
```

Out[183…
```
array([3, 7])
```

In [185…
```
a.max()
```

Out[185…
```
4
```

In [186…
```
a.argmax()
```

Out[186…
```
3
```

In [195…
```
a
```

Out[195…
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [197…
```
a.shape
```

Out[197…
```
(9,)
```

In [200…
```
a[:,np.newaxis].shape # adds a new axis -> 2D
```

Out[200…
```
(9, 1)
```

In [201…
```
np.sort(a)
```

Out[201…
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [202…
```
np.argsort(a)
```

Out[202…
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [ ]: