Author : Sanjoy Biswas

Topic : NumPy Tutorial: Data Analysis with NumPY

Email : sanjoy.eee32@gmail.com

Numpy and Array Basics

The numpy library is one of the core packages in Python's data science software stack. Many other Python data analysis libraries require numpy as a prerequisite, because they use its array data structure as a building block. The Kaggle Python environment has numpy available by default; if you are running Python locally, the Anaconda Python distribution comes with numpy as well.

Numpy implements a data structure called the N-dimensional array or ndarray. ndarrays are similar to lists in that they contain a collection of items that can be accessed via indexes. On the other hand, ndarrays are homogeneous, meaning they can only contain objects of the same type and they can be multi-dimensional, making it easy to store 2-dimensional tables or matrices.

Import NumPY Library

In [1]:

```
import numpy as np
```

Creating a NumPy Array

Basic ndarray

In [2]:

```
a = [10,20,30,40]
```

In [3]:

```
type(a)
```

Out[3]:

```
list
```

In [4]:

```
x = np.array(a)
```

In [5]:

```
print(x)
```

```
[10 20 30 40]
```

In [6]:

```
type(x)
```

Out[6]:

```
numpy.ndarray
```

In [7]:

```
a = [1,2,3,4]
b = [10,20,30,40]
x = np.array([a,b])
```

In [8]:

```
print(x)
```

```
[[ 1  2  3  4]
 [10 20 30 40]]
```

In [9]:

```
a = [1,2,3,4]
b = [10,20,30]
x = np.array([a,b])
```

In [10]:

```
print(x)
```

```
[list([1, 2, 3, 4]) list([10, 20, 30])]
```

In [11]:

```
x = np.array([1,2,3,4], dtype = np.float32)
print(x)
```

```
[1. 2. 3. 4.]
```

In [12]:

```
x = np.zeros((2,3))
print(x)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

Array of zeros

In [13]:

```
x = np.zeros((4,4))
print(x)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

Array of ones

In [14]:

```
np.ones((3,3))
```

Out[14]:

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

In [15]:

```
np.ones(5, dtype = np.int32)
```

Out[15]:

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

Random numbers in ndarrays

In [16]:

```
x = np.random.rand(2,5)
```

In [17]:

```
print(x)
```

```
[[0.38607292 0.65668231 0.27302497 0.55370173 0.20555002]
 [0.7579577  0.12896021 0.14645863 0.70445747 0.09258144]]
```

In [18]:

```
np.random.rand??
```

In [19]:

```
np.random.randint(1,20, size = (3,3))
```

Out[19]:

```
array([[ 9,  8, 13],
       [14,  7,  7],
       [ 4, 12, 10]])
```

In [20]:

```
np.random.randint??
```

In [21]:

```
np.random.randn(2,5)
```

Out[21]:

```
array([[-0.15935158,  0.31854443,  1.59474656,  1.77768865,  0.84622622],
       [-0.75920145,  0.20301125,  0.54237881, -0.18939443,  0.4309023 ]])
```

In [22]:

```
np.random.randn??
```

In [23]:

```
np.random.choice([2,3,4,7,8,10])
```

Out[23]:

```
3
```

An array of your choice

In [24]:

```
np.full((3,4),10)
```

Out[24]:

```
array([[10, 10, 10, 10],
       [10, 10, 10, 10],
       [10, 10, 10, 10]])
```

In [25]:

```
np.ones((3,3))*10
```

Out[25]:

```
array([[10., 10., 10.],
       [10., 10., 10.],
       [10., 10., 10.]])
```

Identity matrix

In [26]:

```
x = np.eye(4, k = -1)
print(x)
```

```
[[0. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

In [27]:

```
np.eye??
```

Evenly spaced ndarray

In [28]:

```
np.arange(5,11)
```

Out[28]:

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

In [29]:

```
np.arange(0,11,2)
```

Out[29]:

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

In [30]:

```
np.linspace(0,1,5)
```

Out[30]:

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

$\frac{end-start}{n-1}$

Shape and Reshaping of NumPy Array

Dimension of nparray

In [31]:

```
a = [10,20,30,40,50]
x = np.array(a)
print(x)
```

```
[10 20 30 40 50]
```

In [32]:

```
x.ndim
```

Out[32]:

```
1
```

In [33]:

```
b = [1,2,3,4,5]
y = np.array([a,b])
print(y)
```

```
[[10 20 30 40 50]
 [ 1  2  3  4  5]]
```

In [34]:

```
y.ndim
```

Out[34]:

```
2
```

Shape of NumPy array

In [35]:

```
y.shape
```

Out[35]:

```
(2, 5)
```

In [36]:

```
y.size
```

Out[36]:

```
10
```

Reshaping a NumPy array

In [37]:

```
a = np.array([3,6,9,12])
print(a)
```

```
[ 3  6  9 12]
```

In [38]:

```
np.reshape(a, (2,2))
```

Out[38]:

```
array([[ 3,  6],
       [ 9, 12]])
```

In [39]:

```
a.reshape(2,2)
```

Out[39]:

```
array([[ 3,  6],
       [ 9, 12]])
```

In [40]:

```
x = np.arange(1,16)
x
```

Out[40]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

In [41]:

```
x.size
```

Out[41]:

```
15
```

In [42]:

```
x.reshape(5,3)
```

Out[42]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15]])
```

Transpose of a NumPy array

In [43]:

```
x = np.arange(1,16)
print(x)
```

```
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15]
```

In [44]:

In [44]:

```
y = x.reshape(3,5)
print(y)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]]
```

In [45]:

```
z = np.transpose(y)
print(z)
```

```
[[ 1  6 11]
 [ 2  7 12]
 [ 3  8 13]
 [ 4  9 14]
 [ 5 10 15]]
```

In [46]:

```
y.transpose()
```

Out[46]:

```
array([[ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14],
       [ 5, 10, 15]])
```

Indexing and Slicing of NumPy Array

1D NumPy arrays

In [47]:

```
a = np.arange(1,10)
```

In [48]:

```
print(a)
```

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

In [49]:

```
a[2:6]
```

Out[49]:

```
array([3, 4, 5, 6])
```

2-D NumPy arrays

In [50]:

```
x = np.arange(1,10).reshape(3,3)
print(x)
```

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

In [51]:

```
x[1,2]
```

Out[51]:

```
6
```

In [52]:

```
x[1][2]
```

Out[52]:

```
6
```

In [53]:

```
x[1:3,1:3]
```

Out[53]:

```
array([[5, 6],
       [8, 9]])
```

Stacking and Concatenating NumPy Arrays

Stacking

In [54]:

```
a = np.arange(1,6)
b = np.arange(6,11)
```

In [55]:

```
print(a)
```

```
[1 2 3 4 5]
```

In [56]:

```
print(b)
```

```
[ 6  7  8  9 10]
```

In [57]:

```
x = np.vstack((a,b))
```

In [58]:

```
print(x)
```

```
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

In [59]:

```
x = np.hstack((a,b))
```

In [60]:

```
print(x)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

Concatenating

In [61]:

```
a = np.arange(1,10).reshape(3,3)
```

In [62]:

```
b = np.arange(10,19).reshape(3,3)
```

In [63]:

```
print(a)
```

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

In [64]:

```
print(b)
```

```
[[10 11 12]
 [13 14 15]
 [16 17 18]]
```

In [65]:

```
np.concatenate((a,b))
```

Out[65]:

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15],
       [16, 17, 18]])
```

In [66]:

```
np.concatenate((a,b), axis = 1)
```

Out[66]:

```
array([[ 1,  2,  3, 10, 11, 12],
       [ 4,  5,  6, 13, 14, 15],
       [ 7,  8,  9, 16, 17, 18]])
```

Broadcasting in NumPy arrays

In [67]:

```
a = np.arange(10,21,2)
```

In [68]:

```
b = np.array([[2],[2]])
```

In [69]:

```
print(a)
```

```
[10 12 14 16 18 20]
```

In [70]:

```
print(b)
```

```
[[2]
 [2]]
```

In [71]:

```
z = a+b
print(z)
```

```
[[12 14 16 18 20 22]
 [12 14 16 18 20 22]]
```

In [72]:

```
a = np.arange(10,21,2)
```

In [73]:

```
a+2
```

Out[73]:

```
array([12, 14, 16, 18, 20, 22])
```

In [74]:

```
a - b
```

Out[74]:

```
array([[ 8, 10, 12, 14, 16, 18],
       [ 8, 10, 12, 14, 16, 18]])
```

In [75]:

```
a*b
```

Out[75]:

```
array([[20, 24, 28, 32, 36, 40],
       [20, 24, 28, 32, 36, 40]])
```

In [76]:

```
a = np.ones((3,3))
b = np.array([2])
print(a)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

In [77]:

```
print(b)
```

```
[2]
```

In [78]:

```
a+b
```

Out[78]:

```
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

In [79]:

```
a-b
```

Out[79]:

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

Arithmetic Operations on NumPy Array

In [80]:

```
a = np.eye(4)
print(a)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

In [81]:

```
print(a+5)
```

```
[[6. 5. 5. 5.]
 [5. 6. 5. 5.]
 [5. 5. 6. 5.]
 [5. 5. 5. 6.]]
```

In [82]:

```
a - 1
```

Out[82]:

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

In [83]:

```
a*2
```

Out[83]:

```
array([[2., 0., 0., 0.],
       [0., 2., 0., 0.],
       [0., 0., 2., 0.],
       [0., 0., 2., 0.],
       [0., 0., 0., 2.]])
```

In [84]:

```
a/0.5
```

Out[84]:

```
array([[2., 0., 0., 0.],
       [0., 2., 0., 0.],
       [0., 0., 2., 0.],
       [0., 0., 0., 2.]])
```

In [85]:

```
np.sin(a)
```

Out[85]:

```
array([[0.84147098, 0.        , 0.        , 0.        ],
       [0.        , 0.84147098, 0.        , 0.        ],
       [0.        , 0.        , 0.84147098, 0.        ],
       [0.        , 0.        , 0.        , 0.84147098]])
```

In [86]:

```
np.cos(a)
```

Out[86]:

```
array([[0.54030231, 1.        , 1.        , 1.        ],
       [1.        , 0.54030231, 1.        , 1.        ],
       [1.        , 1.        , 0.54030231, 1.        ],
       [1.        , 1.        , 1.        , 0.54030231]])
```

In [87]:

```
np.sqrt(a)
```

Out[87]:

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

Aggregate Function

Mean, Median and Standard deviation

In [88]:

```
a = np.arange(5,15,2)
```

In [89]:

```
print(a)
```

```
[ 5  7  9 11 13]
```

In [90]:

```
np.mean(a)
```

Out[90]:

```
9.0
```

In [91]:

```
np.median(a)
```

Out[91]:

```
9.0
```

In [92]:

```
np.std(a)
```

Out[92]:

```
2.8284271247461903
```

Min-Max values

In [93]:

```
a = np.array([[1,6],[4,3]])
print(a)
```

```
[[1 6]
 [4 3]]
```

In [94]:

```
print(np.min(a, axis = 0))
```

```
[1 3]
```

In [95]:

```
print(np.min(a, axis = 1))
```

```
[1 3]
```

In [96]:

```
print(np.max(a, axis = 1))
```

```
[6 4]
```

In [97]:

```
print(np.max(a, axis = 0))
```

```
[4 6]
```

Sorting

In [98]:

```
a = np.array([1,4,3,10,20])
```

In [99]:

In [99]:

```
x = np.sort(a, kind = 'mergesort')
```

In [100]:

```
print(x)
```

```
[ 1  3  4 10 20]
```

Matrix Multiplication

In [101]:

```
A = np.arange(0,9).reshape(3,3)
B = np.ones((3,3))
```

In [102]:

```
print(A)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [103]:

```
print(B)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

In [104]:

```
np.dot(A,B)
```

Out[104]:

```
array([[ 3.,  3.,  3.],
       [12., 12., 12.],
       [21., 21., 21.]])
```

In [105]:

```
A.dot(B)
```

Out[105]:

```
array([[ 3.,  3.,  3.],
       [12., 12., 12.],
       [21., 21., 21.]])
```

In [ ]:

```

```