2-NumPy

September 13, 2019

___ ## Pedram Jahangiry (Fall 2019)

1 NumPy

Topics to be covered:

1. Numpy arrays
2. Numpy indexing and extraction
3. Numpy operations

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

1.1 1. NumPy Arrays

[4]:
```
my_list = [0,1,2,3]
my_list
```

[4]:
```
[0, 1, 2, 3]
```

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

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

[6]:
```
my_matrix = [[1,2,3],[4,5,6]]
my_matrix
```

[6]:
```
[[1, 2, 3], [4, 5, 6]]
```

[11]:
```
np.array(my_matrix)
```

[11]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

[12]:
```
# Built in methods
# np.arange() :Return evenly spaced values within a given interval.
np.arange(0,4)
```

[12]:
```
array([0, 1, 2, 3])
```

[13]:
```
np.arange(0,10,3)
```

[13]:
```
array([0, 3, 6, 9])
```

[14]:
```
# Zeros and ones : Generate arrays of zeros or ones
np.zeros(5)
```

[14]:
```
array([0., 0., 0., 0., 0.])
```

[15]:
```
np.zeros((5,5))
```

[15]:
```
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.]])
```

[16]:
```
np.ones(5)
```

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

[17]:
```
np.ones((5,5))
```

[17]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

[18]:
```
# linspace : Return evenly spaced numbers over a specified interval.
np.linspace(0,20,5)
```

[18]:
```
array([ 0.,  5., 10., 15., 20.])
```

[19]:
```
# eye: Creates identity matrix
np.eye(5)
```

[19]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

[20]:
```
# rand : create random samples from a uniform distribution
np.random.rand(5)
```

[20]:
```
array([0.74412674, 0.4708743 , 0.67942014, 0.93936755, 0.20810138])
```

[21]:
```
np.random.rand(5,5)
```

[21]:
```
array([[0.27284824, 0.55840574, 0.98721906, 0.58581059, 0.50727531],
       [0.16236248, 0.34150367, 0.78465915, 0.81185737, 0.43495061],
       [0.65176496, 0.48507315, 0.17914585, 0.2290497 , 0.7445111 ],
       [0.36971418, 0.13355033, 0.61175082, 0.1711799 , 0.59379592],
       [0.81048788, 0.20122472, 0.10459302, 0.54537236, 0.0975716 ]])
```

[22]:
```
# randn : create random samples from standard normal distribution
np.random.randn(5)
```

[22]:
```
array([-0.48086967,  0.11367305,  0.73001448, -0.67785124, -0.64725918])
```

[23]:
```
np.random.randn(5,5)
```

[23]:
```
array([[-0.71350404, -0.28165882,  0.5579421 , -0.68325667, -1.5250754 ],
       [-0.13861949, -0.76485286,  0.8746773 , -2.29237738,  0.08701108],
       [-0.74077738,  1.70693218, -0.63901472, -0.63612506, -0.50749737],
       [-1.33379258,  1.50455475, -0.04011188, -1.31740649, -0.45616237],
       [-0.6453803 ,  1.53224062,  0.32501494, -0.41251124, -1.40849415]])
```

[24]:
```
# randit(a,b) : create random sample of integers from a (including a) to b␣
 ↪(excluding b)
np.random.randint(1,5)
```

[24]:
```
2
```

[25]:
```
np.random.randint(1,5,20)
```

[25]:
```
array([1, 2, 4, 1, 4, 4, 3, 1, 3, 2, 3, 4, 4, 3, 1, 4, 4, 1, 1, 3])
```

[21]:
```
# seed is used to fix the random state.
np.random.seed(100)
np.random.randn(5)
```

[21]:
```
array([-1.74976547,  0.3426804 ,  1.1530358 , -0.25243604,  0.98132079])
```

[22]:
```
np.random.seed(100)
np.random.randn(4)
```

[22]:
```
array([-1.74976547,  0.3426804 ,  1.1530358 , -0.25243604])
```

[26]:
```
# array methods

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

[26]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[27]:
```
# reshape
new_array = my_array.reshape(3,3)
new_array
```

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

[28]:
```
new_array.shape
```

[28]:
```
(3, 3)
```

[29]:
```
new_array.dtype
```

[29]:
```
dtype('int32')
```

[30]:
```
type(new_array)
```

[30]:
```
numpy.ndarray
```

[31]:
```
my_array = np.append(my_array, [100,-100])
my_array
```

[31]:
```
array([   1,    2,    3,    4,    5,    6,    7,    8,    9,  100, -100])
```

[32]:
```
my_array.max()
```

[32]:
```
100
```

[33]:
```
my_array.argmax()
```

[33]:
```
9
```

[34]:
```
my_array.min()
```

[34]:
```
-100
```

[35]:
```
my_array.argmin()
```

[35]:
```
10
```

1.2 2. Numpy indexing and extraction

[36]:
```
my_array
```

[36]:
```
array([   1,    2,    3,    4,    5,    6,    7,    8,    9,  100, -100])
```

[37]:
```
# extraction is very similar to list extraction
my_array[9]
```

[37]:
```
100
```

[38]:
```
my_array[6:9]
```

[38]:
```
array([7, 8, 9])
```

[39]:
```
# With NumPy arrays, you can broadcast a single value across a larger set of␣
↪values. This is not possible using lists.

my_array[0:5]=100
my_array
```

[39]:
```
array([ 100,  100,  100,  100,  100,    6,    7,    8,    9,  100, -100])
```

[40]:
```
my_list = list(range(1,10))
my_list
```

[40]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[41]:
```
my_list[0:5]
```

[41]:
```
[1, 2, 3, 4, 5]
```

[42]:
```
my_list[0:2]=100
```

```
␣
↪-------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)

<ipython-input-42-a37bc34a487d> in <module>
----> 1 my_list[0:2]=100


TypeError: can only assign an iterable
```

[43]:
```
my_list[0:2]=[100,100,100]
my_list
```

[43]:
```
[100, 100, 100, 3, 4, 5, 6, 7, 8, 9]
```

[44]:
```
# matrix: Note that matrix indexing in python is slightly different than R, Matlab or other programmings.

my_matrix= np.arange(0,6).reshape(2,3)
my_matrix
```

[44]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[45]:
```
# Format is matrix[row][col] or matrix[row,col]

# extracting the first row
my_matrix[0]
```

[45]:
```
array([0, 1, 2])
```

[46]:
```
# extracting the first column
my_matrix[:,0]
```

[46]:
```
array([0, 3])
```

[47]:
```
my_matrix[0][1] == my_matrix[0,1]
```

[47]:
```
True
```

[48]:
```
my_matrix[:2,1:]
```

[48]:
```
array([[1, 2],
       [4, 5]])
```

Use google image to get help on "numpy array indexing"

1.2.1 Extracting with conditional selection

[49]:
```
my_array
```

[49]:
```
array([ 100,  100,  100,  100,  100,    6,    7,    8,    9,  100, -100])
```

[50]:
```
my_array > 50
```

[50]:
```
array([ True,  True,  True,  True,  True, False, False, False, False,
        True, False])
```

[51]:
```
my_array[my_array>50]
```

[51]:
```
array([100, 100, 100, 100, 100, 100])
```

[58]:
```
my_matrix
```

[58]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[53]:
```
my_matrix > 1
```

[53]:
```
array([[False, False,  True],
       [ True,  True,  True]])
```

[56]:
```
my_matrix[my_matrix > 1]
```

[56]:
```
array([2, 3, 4, 5])
```

1.3 3. Numpy operations

[59]:
```
arr = np.arange(0,5)
arr
```

[59]:
```
array([0, 1, 2, 3, 4])
```

[53]:
```
arr + arr
```

[53]:
```
array([0, 2, 4, 6, 8])
```

[54]:
```
arr ** arr
```

[54]:
```
array([  1,   1,   4,  27, 256], dtype=int32)
```

[55]:
```
arr/arr
```

```
C:\Users\jahan\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
RuntimeWarning: invalid value encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[55]:
```
array([nan,  1.,  1.,  1.,  1.])
```

[56]:
```
1/arr
```

```
C:\Users\jahan\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
RuntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

[56]:
```
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ])
```

[57]:
```
# Square Roots
np.sqrt(arr)
```

[57]:
```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ])
```

[58]:
```
# Exponential
np.exp(arr)
```

[58]:
```
array([ 1.        ,  2.71828183,  7.3890561 , 20.08553692, 54.59815003])
```

[59]:
```
# Natural Logarithm
np.log(arr)
```

```
C:\Users\jahan\Anaconda3\lib\site-packages\ipykernel_launcher.py:2:
RuntimeWarning: divide by zero encountered in log
```

[59]:
```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436])
```

[63]:
```
# summary statistics on arrays
arr
```

[63]:
```
array([0, 1, 2, 3, 4])
```

[61]:
```
arr.sum()
```

[61]:
```
10
```

[62]:
```
arr.mean()
```

[62]:
```
2.0
```

[63]:
```
arr.var()
```

[63]:
```
2.0
```

[64]:
```
arr.std()
```

[64]:
```
1.4142135623730951
```

1.4 Axis Logic

When working with 2-dimensional arrays (matrices) we have to consider rows and columns. This becomes very important when we get to the section on pandas. In array terms, axis 0 (zero) is the vertical axis (rows), and axis 1 is the horizonal axis (columns). These values (0,1) correspond to the order in which arr.shape values are returned.

Let's see how this affects our summary statistic calculations from above.

[64]:
```
my_matrix
```

[64]:
```
array([[0, 1, 2],
       [3, 4, 5]])
```

[65]:
```
# axis 0 (zero) is the vertical axis (rows), and axis 1 is the horizonal axis␣
↪(columns)
# again, note that the logic is different from R, Matlab or etc
```

```
my_matrix.sum(axis=0)
```

[65]:
```
array([3, 5, 7])
```

[66]:
```
my_matrix.sum(1)
```

[66]:
```
array([ 3, 12])
```

[67]:
```
my_matrix * my_matrix  # note that this is not a matrix multiplication
```

[67]:
```
array([[ 0,  1,  4],
       [ 9, 16, 25]])
```

[68]:
```
# To do a matrix multiplication use np.dot (equivalent to %*% in R)
np.dot(my_matrix,my_matrix.T)
```

[68]:
```
array([[ 5, 14],
       [14, 50]])
```

[69]:
```
# inverse of a matrix
A = np.array([[1,2],[3,4]])
A
```

[69]:
```
array([[1, 2],
       [3, 4]])
```

[70]:
```
A_inv = np.linalg.inv(A)
A_inv
```

[70]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```