Lecture 4

May 18, 2023

1 Lecture 4

1.1 Numpy contd.

1.1.1 More functions

[2]:
```python
import numpy as np
```

[3]:
```python
x = [1, 4, 9, 16, 25]
print(x)
```

```
[1, 4, 9, 16, 25]
```

[4]:
```python
np.sqrt(x)
```

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

[5]:
```python
np.log(x)
```

[5]:
```
array([0.        , 1.38629436, 2.19722458, 2.77258872, 3.21887582])
```

[6]:
```python
np.exp(x)
```

[6]:
```
array([2.71828183e+00, 5.45981500e+01, 8.10308393e+03, 8.88611052e+06,
       7.20048993e+10])
```

[8]:
```python
a = np.arange(10)
a = a.reshape(5, 2)
print(a)
```

```
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

[9]:
```python
a1 = a.T
print(a1)
```

```
[[0 2 4 6 8]
 [1 3 5 7 9]]
```

[10]:

```
arr1 = np.array([1.23, 4.67, 3.56, 4.5, 8.79, 3.12])
print(arr1)
```

```
[1.23 4.67 3.56 4.5  8.79 3.12]
```

[11]:

```
np.floor(arr1)
```

[11]:

```
array([1., 4., 3., 4., 8., 3.])
```

[12]:

```
np.ceil(arr1)
```

[12]:

```
array([2., 5., 4., 5., 9., 4.])
```

[14]:

```
np.round(arr1)   # <0.5  and >=0.5
```

[14]:

```
array([1., 5., 4., 4., 9., 3.])
```

1.1.2 Stacking

[16]:

```
#Stacking function
arr1 = np.arange(10).reshape(2, 5)
arr2 = np.arange(20).reshape(4, 5)

print(arr1)
print(arr2)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

[18]:

```
#1) Vertical stacking - No. of columns must be same
np.vstack((arr1, arr2))
```

[18]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

[19]:

```
#2) Horizontal stacking - No. of rows must be same
np.hstack((arr1.T, arr2.T))
```

[19]:

```
array([[ 0,  5,  0,  5, 10, 15],
       [ 1,  6,  1,  6, 11, 16],
       [ 2,  7,  2,  7, 12, 17],
       [ 3,  8,  3,  8, 13, 18],
       [ 4,  9,  4,  9, 14, 19]])
```

1.1.3 Split

[22]:

```
arr = np.arange(100).reshape(10, 10)
print(arr)
```

```
[[ 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 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 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]]
```

[30]:

```
#Vertical split - splits array into multiple array vertically (row-wise)
np.vsplit(arr, 2)  #split number should be multiple of number of rows
```

[30]:

```
[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, 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]]),
 array([[50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
        [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
        [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
        [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
        [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])]
```

[31]:

```
#Horizontal split - splits array into multiple array horizontally (column-wise)
np.hsplit(arr, 2)          #split number should be multiple of number of columns
```

[31]:

```
[array([[ 0,  1,  2,  3,  4],
        [10, 11, 12, 13, 14],
        [20, 21, 22, 23, 24],
        [30, 31, 32, 33, 34],
        [40, 41, 42, 43, 44],
        [50, 51, 52, 53, 54],
        [60, 61, 62, 63, 64],
        [70, 71, 72, 73, 74],
        [80, 81, 82, 83, 84],
        [90, 91, 92, 93, 94]]),
 array([[ 5,  6,  7,  8,  9],
        [15, 16, 17, 18, 19],
        [25, 26, 27, 28, 29],
        [35, 36, 37, 38, 39],
        [45, 46, 47, 48, 49],
        [55, 56, 57, 58, 59],
        [65, 66, 67, 68, 69],
        [75, 76, 77, 78, 79],
        [85, 86, 87, 88, 89],
        [95, 96, 97, 98, 99]]])
```

```
[27]: np.linspace(1, 100, 10)  #Linearly spaced numbers
```

```
[27]: array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

```
[28]: np.linspace(1, 100, 10, retstep=True)  #retstep gives difference between two consecutive numbers
```

```
[28]: (array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.]), 11.0)
```

1.1.4 EYE - identity matrix

```
[29]: #Creates an identity matrix

      np.eye(5)
```

```
[29]: 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.]])
```

1.1.5 Random

```
[33]: # Rand function
      np.random.rand(4)    #Random numbers between 0 and 1
```

```
[33]: array([0.94603218, 0.44523807, 0.47724374, 0.15691597])
```

```
[37]: # randint function - between (low(inclusive), high(exclusive))
      print(np.random.randint(0, 100))

      print(np.random.randint(0, 100, 10))
```

```
55
[91 36 34 19 66 74 74 67 54 82]
```

1.1.6 Other important functions

[38]:

```
arr = np.random.randint(0, 100, 10)
print(arr)
```

```
[91 96 22 24 67 88 59 53 82 78]
```

[40]:

```
# mean
arr.mean()
```

[40]:

```
66.0
```

[41]:

```
#copy
b = arr.copy()
print(b)
```

```
[91 96 22 24 67 88 59 53 82 78]
```

[42]:

```
#cumsum - Cumulative sum

print(arr.cumsum())
```

```
[ 91 187 209 233 300 388 447 500 582 660]
```

[43]:

```
#max

print(arr.max())
```

```
96
```

[46]:

```
#min

print(arr.min())
```

```
22
```

[45]:

```
#argsort - sort in ascending order but gives the index positions
arr.argsort()
```

[45]:

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

1.2 Pandas

[49]:

```
#import pandas library
import pandas as pd
import numpy as np
```

1.2.1 Series

A series is an one dimensional labelled array that can hold data of any type

[50]:

```
s = pd.Series(np.random.rand(5))
print(s)
```

```
0    0.877075
1    0.789421
2    0.211327
3    0.430693
4    0.386360
dtype: float64
```

[51]:

```
#indexing
print(s[3])
print(s[4])
```

```
0.43069290977179175
0.38635962298647597
```

[53]:

```
#Slicing [start:end:stepsize]
print(s[2:])
```

```
2    0.211327
3    0.430693
4    0.386360
dtype: float64
```

[54]:

```
#update value
s[0] = 560
print(s)
```

```
0    560.000000
1      0.789421
2      0.211327
3      0.430693
4      0.386360
dtype: float64
```

[57]:

```
#Changing index
s.index = ['a', 'b', 'c', 'd', 'e']
print(s)

#Can be set during intialization too
s = pd.Series(np.random.rand(5), index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

```
a    0.692837
b    0.060494
c    0.291181
d    0.525858
e    0.387971
dtype: float64
```

```
a    0.544902
b    0.321337
c    0.756062
d    0.397758
e    0.332198
dtype: float64
```

[58]:

```
#argmax

s.argmax()
```

[58]:

```
2
```

[59]:

```
#argmin

s.argmin()
```

[59]:

```
1
```

[60]:

```
#mean

s.mean()
```

[60]:

```
0.47045129301048194
```

[61]:

```
#median

s.median()
```

[61]:

```
0.3977575336741862
```

[62]:

```
#max

s.max()
```

[62]:

```
0.7560620028498166
```

[63]:

```
#min

s.min()
```

[63]:

```
0.32133658387528974
```

[64]:

```
#Converting numpy array into Series

arr = np.array(range(4))
print(arr)
arr_series = pd.Series(arr)
```

```
print(arr_series)
```

```
[0 1 2 3]
0    0
1    1
2    2
3    3
dtype: int64
```

[65]:

```
#Using Dictionary
d = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5}
print(d)
d_series = pd.Series(d)
print(d_series)
```

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

[66]:

```
#Checking type
type(d_series)
```

[66]:

```
pandas.core.series.Series
```