APANPS5210_002_2018_1 - PYTHON FOR DATA ANALYSIS

March 2nd

- Numpy
- Lab

Introduction

Create a new notebook for your code-along:

From your directory, type:

```
jupyter notebook
```

Or launch from Anaconda-navigator From the Dashboard, open a new notebook. Make a copy JIC.

Introduction to Numpy

- Overview
- ndarray
- Indexing and Slicing

More info: http://wiki.scipy.org/Tentative_NumPy_Tutorial (http://wiki.scipy.org/Tentative_NumPy_Tutorial)

Numpy Overview

- Why Python for Data? Numpy brings *decades* of C math into Python!
- Numpy provides a wrapper for extensive C/C++/Fortran codebases, used for data analysis functionality
- NDAarray allows easy vectorized math and broadcasting (i.e. functions for vector elements of different shapes)

```
In [10]: import numpy as np
```

Creating ndarrays

An array object represents a multidimensional, homogeneous array of fixed-size items.

How to create some toy arrays .

```
In [11]: # Note the way each array is printed:
         a=np.zeros((3))
         a

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

```
In [12]: b = np.ones((2,3))
         b

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

```
In [13]: # arange() is like range()
         c = np.arange(0,11,1)
         c

Out[13]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [14]: emty=np.empty((1,10))  # very small numbers meaningless
         emty

Out[14]: array([[  6.79038654e-313,   6.79038653e-313,   2.37663529e-312,
                   2.35541533e-312,   2.14321575e-312,   8.48798317e-313,
                   1.06099790e-312,   1.08221785e-312,   8.70018274e-313,
                   2.07955588e-312]])
```

```
In [15]: ones_diag=np.eye(4)
         ones_diag

Out[15]: array([[ 1.,  0.,  0.,  0.],
                [ 0.,  1.,  0.,  0.],
                [ 0.,  0.,  1.,  0.],
                [ 0.,  0.,  0.,  1.]])
```

```
In [16]: e=np.array(['a','a','b','c'])
         e

Out[16]: array(['a', 'a', 'b', 'c'],
               dtype='<U1')
```

```
In [17]: #f=np.ndarray(['a','b','c']) #
         #ndarray is reserved as a low level function for numbers only .
         # np.array is a wrapper and is preferred
```

```
In [18]: ## Arithmetic in arrays is element wise
In [19]: a = np.array( [20,30,40,50] )
         b = np.arange( 4 )
         b

Out[19]: array([0, 1, 2, 3])
```

```
In [20]: c = a-b
         print(c)
         c= a+3
         c

         [20 29 38 47]

Out[20]: array([23, 33, 43, 53])
```

```
In [21]: print(b)
         b**2

         [0 1 2 3]

Out[21]: array([0, 1, 4, 9], dtype=int32)
```

```
In [22]: b/3

Out[22]: array([ 0.        ,  0.33333333,  0.66666667,  1.        ])
```

Indexing, Slicing and Iterating

```
In [23]: # one-dimensional arrays work like lists: 0 - 9
         a = np.arange(10)**2
```

```
In [24]: a

Out[24]: array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81], dtype=int32)
```

```
In [25]: a[2:5]

Out[25]: array([ 4,  9, 16], dtype=int32)
```

```
In [26]: # Selecting an element
         a[2]

Out[26]: 4
```

```
In [27]: [ -x**2 for x in a if x >4]

Out[27]: [-81, -256, -625, -1296, -2401, -4096, -6561]
```

Multidimensional arrays use tuples with commas for indexing with (row,column) conventions beginning, as always in Python, from 0

```
In [28]: np.random.seed(123)
         b = np.random.randint(1,100,size=(4,4))
         b

Out[28]: array([[67, 93, 99, 18],
                [84, 58, 87, 98],
                [97, 48, 74, 33],
                [47, 97, 26, 84]])
```

```
In [29]: b[0],b[0,:],b[0,0:4]

Out[29]: (array([67, 93, 99, 18]), array([67, 93, 99, 18]), array([67, 93, 99, 18]))
```

```
In [30]: print(b[:,1]) # cols
         print( b[1:3,1:3]) #submatrix

         [93 58 48 97]
         [[58 87]
          [48 74]]
```

```
In [31]: # Guess the output
         print(b[2,3])
         print(b[0,0])

         33
         67
```

```
In [32]: b[0:4,1],b[:,1],b[0:3,1]

Out[32]: (array([93, 58, 48, 97]), array([93, 58, 48, 97]), array([93, 58, 48]))
```

```
In [33]: #Selecting ONE element from a 2 d array

         print(b[0][1])
         print(b[0,1])

         93
         93
```

```
In [34]: np.random.seed(12345)# setting the seed. It's ramdom but alwys the same.
         data = np.random.randn(2, 3)
         data.shape,data.dtype,data

Out[34]: ((2, 3), dtype('float64'), array([[-0.20470766,  0.47894334, -0.51943872],
                 [-0.5557303 ,  1.96578057,  1.39340583]]))
```

```
In [35]: [np.log(x) for x in b ]

Out[35]: [array([ 4.20469262,  4.53259949,  4.59511985,  2.89037176]),
          array([ 4.4308168 ,  4.06044301,  4.46590812,  4.58496748]),
          array([ 4.57471098,  3.87120101,  4.30406509,  3.49650756]),
          array([ 3.8501476 ,  4.57471098,  3.25809654,  4.4308168 ])]
```

Data Types for ndarrays¶

```
In [36]: arr1 = np.array([1, 2, 3], dtype=np.float64)
         arr2 = np.array([1, 2, 3], dtype=np.int32)# set to int
         arr1.dtype,arr2.dtype

Out[36]: (dtype('float64'), dtype('int32'))
```

```
In [37]: # our first method  as.type() conversion
         arr = np.array([1, 2, 3, 4, 5])
         print(arr.dtype)
         float_arr = arr.astype(np.float64) #astype
         float_arr.dtype

         int32

Out[37]: dtype('float64')
```

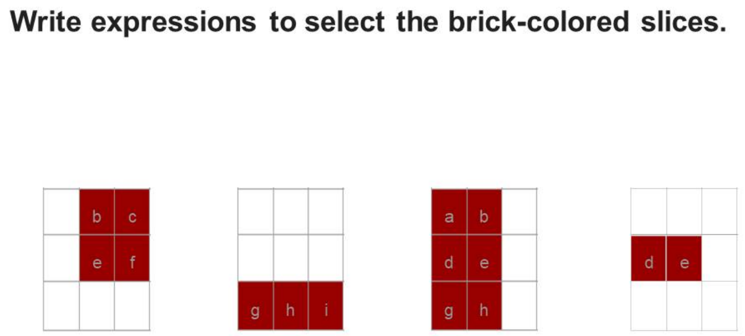
```
In [38]: #Selecting element from 2d array
         arr2d=np.array([["a","b","c"],["d","e","f"],["g","h","i"]])
         arr2d

Out[38]: array([['a', 'b', 'c'],
                ['d', 'e', 'f'],
                ['g', 'h', 'i']],
               dtype='<U1')
```

In Class assignment

Don't execute the following cell !

In [39]:
```
from IPython.display import Image
Image('C://Users//jpsabini//Columbia2018//ColumbiaApan5000Spring18/Presentation2.jpg',width=400,height=200)
```

Out[39]:



Write expressions to select the brick-colored slices.

In [40]:
```
#Submit your ans to Canvas
```

In [41]:
```
# Copy by assignment isn't safe/
X=arr2d
keepitsafe=X[:,1]
keepitsafe
X[:,1]=9
print('X is ',X)
print('-----------')
print(keepitsafe,end='\n')
arr2d  # was affected
```

```
X is  [['a' '9' 'c']
 ['d' '9' 'f']
 ['g' '9' 'i']]
-----------
['9' '9' '9']
```

Out[41]:
```
array([['a', '9', 'c'],
       ['d', '9', 'f'],
       ['g', '9', 'i']],
      dtype='<U1')
```

Methods as verbs and Attributes as nouns

copy method

In [42]:

```
arr2d=np.array([["a","b","c"],["d","e","f"],["g","h","i"]])
arr2d
X=arr2d.copy()

X[:,1]=9
print(X)
print('-----------')

print(arr2d)
```

```
[['a' '9' 'c']
 ['d' '9' 'f']
 ['g' '9' 'i']]
-----------
[['a' 'b' 'c']
 ['d' 'e' 'f']
 ['g' 'h' 'i']]
```

In [43]:

```
#np dota np.
```

In [44]:

```
#arr dot arr.
```

In [45]:

```
grid=np.linspace(start=1.,stop=10.0,num=20)
grid
```

Out[45]:

```
array([  1.        ,   1.47368421,   1.94736842,   2.42105263,
         2.89473684,   3.36842105,   3.84210526,   4.31578947,
         4.78947368,   5.26315789,   5.73684211,   6.21052632,
         6.68421053,   7.15789474,   7.63157895,   8.10526316,
         8.57894737,   9.05263158,   9.52631579,  10.        ])
```

In [46]:

```
# How would you change the above to yield 1.0 1.5 2.0 2.5 etc??
```

In [47]:

```
# Boolean Indexing
# Say we have 7 sensors and 10 rows of out put fr each sensor.
```

In [48]:
```
names = np.array(['Sensor1', 'Sensor2', 'Sensor1', 'Sensor4', 'Sensor5', 'Sens
or6', 'Sensor6'])
np.random.seed(123)
data = np.random.randn(7, 10)
names
print(data)
data.shape
```

```
[[-1.0856306   0.99734545  0.2829785  -1.50629471 -0.57860025  1.65143654
  -2.42667924 -0.42891263  1.26593626 -0.8667404 ]
 [-0.67888615 -0.09470897  1.49138963 -0.638902   -0.44398196 -0.43435128
   2.20593008  2.18678609  1.0040539   0.3861864 ]
 [ 0.73736858  1.49073203 -0.93583387  1.17582904 -1.25388067 -0.6377515
   0.9071052  -1.4286807  -0.14006872 -0.8617549 ]
 [-0.25561937 -2.79858911 -1.7715331  -0.69987723  0.92746243 -0.17363568
   0.00284592  0.68822271 -0.87953634  0.28362732]
 [-0.80536652 -1.72766949 -0.39089979  0.57380586  0.33858905 -0.01183049
   2.39236527  0.41291216  0.97873601  2.23814334]
 [-1.29408532 -1.03878821  1.74371223 -0.79806274  0.02968323  1.06931597
   0.89070639  1.75488618  1.49564414  1.06939267]
 [-0.77270871  0.79486267  0.31427199 -1.32626546  1.41729905  0.80723653
   0.04549008 -0.23309206 -1.19830114  0.19952407]]
```

Out[48]:
```
(7, 10)
```

In [49]:
```
names == 'Sensor2'
```

Out[49]:
```
array([False,  True, False, False, False, False, False], dtype=bool)
```

In [50]:
```
data[names == 'Sensor2']
```

Out[50]:
```
array([[-0.67888615, -0.09470897,  1.49138963, -0.638902  , -0.44398196,
        -0.43435128,  2.20593008,  2.18678609,  1.0040539 ,  0.3861864 ]])
```

In [51]:
```
data[1,:]
```

Out[51]:
```
array([-0.67888615, -0.09470897,  1.49138963, -0.638902  , -0.44398196,
       -0.43435128,  2.20593008,  2.18678609,  1.0040539 ,  0.3861864 ])
```

In [52]:
```
Sensor1=names=='Sensor1'
print(Sensor1)
data[~(Sensor1) ]
```

```
[ True False  True False False False False]
```

Out[52]:
```
array([[-0.67888615, -0.09470897,  1.49138963, -0.638902  , -0.44398196,
        -0.43435128,  2.20593008,  2.18678609,  1.0040539 ,  0.3861864 ],
       [-0.25561937, -2.79858911, -1.7715331 , -0.69987723,  0.92746243,
        -0.17363568,  0.00284592,  0.68822271, -0.87953634,  0.28362732],
       [-0.80536652, -1.72766949, -0.39089979,  0.57380586,  0.33858905,
        -0.01183049,  2.39236527,  0.41291216,  0.97873601,  2.23814334],
       [-1.29408532, -1.03878821,  1.74371223, -0.79806274,  0.02968323,
         1.06931597,  0.89070639,  1.75488618,  1.49564414,  1.06939267],
       [-0.77270871,  0.79486267,  0.31427199, -1.32626546,  1.41729905,
         0.80723653,  0.04549008, -0.23309206, -1.19830114,  0.19952407]])
```

```
In [53]: mask = ((names == 'Sensor1') |(names == 'Sensor5')) #logic bool
         print(mask)
         data[mask]

         [ True False  True False  True False False]

Out[53]: array([[-1.0856306 ,  0.99734545,  0.2829785 , -1.50629471, -0.57860025,
                  1.65143654, -2.42667924, -0.42891263,  1.26593626, -0.8667404 ],
                [ 0.73736858,  1.49073203, -0.93583387,  1.17582904, -1.25388067,
                 -0.6377515 ,  0.9071052 , -1.4286807 , -0.14006872, -0.8617549 ],
                [-0.80536652, -1.72766949, -0.39089979,  0.57380586,  0.33858905,
                 -0.01183049,  2.39236527,  0.41291216,  0.97873601,  2.23814334]])

In [54]: A=np.random.randn(16)

In [55]: A=A.reshape(4,4)

In [56]: A

Out[56]: array([[ 0.46843912, -0.83115498,  1.16220405, -1.09720305],
                [-2.12310035,  1.03972709, -0.40336604, -0.12602959],
                [-0.83751672, -1.60596276,  1.25523737, -0.68886898],
                [ 1.66095249,  0.80730819, -0.31475815, -1.0859024 ]])

In [57]: A[0,0]=np.nan

In [58]: A

Out[58]: array([[        nan, -0.83115498,  1.16220405, -1.09720305],
                [-2.12310035,  1.03972709, -0.40336604, -0.12602959],
                [-0.83751672, -1.60596276,  1.25523737, -0.68886898],
                [ 1.66095249,  0.80730819, -0.31475815, -1.0859024 ]])

In [59]: np.isnan(A)

Out[59]: array([[ True, False, False, False],
                [False, False, False, False],
                [False, False, False, False],
                [False, False, False, False]], dtype=bool)

In [60]: # there is also a linear algebra module within np
         #np.linalg.

           File "<ipython-input-60-7df7fdecbe5b>", line 2
             np.linalg.
                       ^
         SyntaxError: invalid syntax
```

In [61]:

```
#new axis make a column out of a row
y = np.linspace(0, 12, 5)
print(y)
print(y[:, np.newaxis])
```

```
[  0.   3.   6.   9.  12.]
[[  0.]
 [  3.]
 [  6.]
 [  9.]
 [ 12.]]
```

In [62]:

```
A=np.array([[1,3],[2,7]])
A.shape
```

Out[62]:

```
(2, 2)
```

In [ ]:

```
### https://en.wikipedia.org/wiki/Dot_product
### A%*%b  In R
### b=[1,1]
```

In [63]:

```
b=[1,1]
x=A.dot(b)  # no of rows  by no of cols  dot no of rows by no of cols gives a
 3 by 1
print(x,x.shape)
np.dot(b,A)
```

```
[4 9] (2,)
```

Out[63]:

```
array([ 3, 10])
```

In [64]:

```
# This is an element be elemnt multiplication of a vector and Matrix
A=np.array([[1,4,3],[2,5.5,7],[8.5,7,10]])
print(A)
a=np.array([0,1,2])
x=A*a
print(x.shape)
print(x)
```

```
[[  1.    4.    3. ]
 [  2.    5.5   7. ]
 [  8.5   7.   10. ]]
(3, 3)
[[  0.    4.    6. ]
 [  0.    5.5  14. ]
 [  0.    7.   20. ]]
```

In [65]:

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

$$cos(\theta) = \frac{a \cdot b}{||(a)|| * ||(b)||}$$

```
In [66]: (np.dot(a,b)/np.linalg.norm(a)*np.linalg.norm(b))

Out[66]: 9.2951600308978009
```

end

We introduced numpy arrays and methods ad attributes (dtype, shape), boolean filtering, accessing elements, slices andso on. Please study Pandas Book on Numpy.