빅 데이터 분석
python 기초와 numpy

August 2020
Prepared by Prof. Youn-Sik Hong

본 강의 자료는 "Python for Data Analysis"(2nd Ed.)
by William McKinney 내용을 참고하여 작성하였음

목차

Python 자료 형 : python_type. ipynb

tuple, list, dictionary, set

- tuple: 콤마(,)로 원소를 구분
  - immutable(값을 변경할 수 없음)
- list: 대괄호로 묶음 - [ ]
  - mutable(값을 변경할 수 있음)
- dictionary: 중괄호로 묶음 - { }
  - key-value 쌍으로 표현. key는 변경되지 않는 값.
- set: 소괄호로 묶음 - ( )
  - 집합 연산(union, intersection, difference 등)

Python 함수 : python_func. ipynb

- 정의
  - def … return
- 여러 개 값을 반환
  - return a, b, c
- **lambda expression**
  - 매개변수가 1개인 간단한 함수 선언
  - sum = lambda x: x + x

Python 함수 응용 – 여러 개 함수 적용 (1/3)

```
states = ['Alabama', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
          'south   carolina##', 'West  virginia?']
```

```
def clean_strings(strings):
    result = []
    for value in strings:
        value = value.strip()  # 공백문자 제거
        value = re.sub('[!#?]','', value) # 특수문자 제거
        value = value.title() # 첫글자를 대문자로 변환
        value = re.sub('[ ]+',' ', value) # 2개 이상 공백문자가 있을 경우 1개만 남김
        result.append(value)
    return result
```

```
clean_strings(states)
```

```
['Alabama',
 'Georgia',
 'Georgia',
 'Georgia',
 'Florida',
 'South Carolina',
 'West Virginia']
```

Python 함수 응용 – 여러 개 함수 적용 (2/3) : 함수 리스트

```
states = ['Alabama', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
          'south   carolina##', 'West  virginia?']
```

```
def remove_specialchars(value):
    return re.sub('[?!#]', '', value)
def redundant_spaces(value):
    return re.sub('[ ]+',' ', value)

apply_ops = [str.strip, remove_specialchars, str.title, redundant_spaces]

def clean_strings2(strings, ops):
    result = []
    for value in strings:
        for func in ops:
            value = func(value)
        result.append(value)
    return result

clean_strings2(states, apply_ops)
```

Python 함수 응용 – 여러 개 함수 적용 (3/3) : map

```
states = ['Alabama', 'Georgia!', 'Georgia', 'georgia', 'FlOrlda',
          'south   carolina##', 'West  virginia?']
```

```
def remove_specialchars(value):
    return re.sub('[?!#]', '', value)
def redundant_spaces(value):
    return re.sub('[ ]+',' ', value)
def clean_strings3(strings, ops):
    result = []
    for value in map(ops, strings):
        result.append(value)
    return result

states = clean_strings3(states, str.strip)
states = clean_strings3(states, remove_specialchars)
states = clean_strings3(states, str.title)
states = clean_strings3(states, redundant_spaces)
```

Python 함수 – lambda 함수

- 함수 이름 대신 **lambda** 키워드 사용
  - 단일 매개변수 및 한 문장으로 이루어진 간단한 함수
- 함수의 형식 매개변수를 lambda 함수로 정의하는 방식을 주로 사용

```
def boo(x):
    return x*2

mboo = lambda x: x*2

boo_lst = list(range(1,4))
def apply_to_list(lst, f):
    return [f(x) for x in lst]

apply_to_list(boo_lst, mboo)
apply_to_list(boo_lst, lambda x:x*2)
```

Python 유용한 함수 : python_util. ipynb

enumerate, sorted, zip, reversed

- enumerate
  - for 루프에서 index와 value을 함께 반환
- sorted
  - 정렬된 자료를 반환. <cf.>sort
- zip
  - 여러 개 자료 구조를 짝지어 tuple의 list를 만듦.
- reversed
  - 순차 자료의 순서를 반대로 해서 나열.

Python 파일 입출력 : python_file. ipynb

- 파일 I/O 관련 메소드
  - read, readlines
  - write, writelines
  - close, flush

- 파일 I/O 모드
  - r, w, x, a, r+, b, t

- 파일 인코딩
  - utf-8, unicode

목차

Numpy : Numerical Python

- **Numpy, Scipy, and Matplotlib provide MATLAB-like functionality in python.**
  - 과학 계산용 패키지 대부분은 **NumPy** 배열 객체**(ndarray)**를 데이터 호환 목적으로 사용

- **Numpy Features:**
  - Typed multi-dimensional arrays
  - Fast numerical computations
  - High-level math functions

- **Python does numerical computations slowly.**
  - Numpy는 python보다 더 적은 메모리를 사용하면서도 최소 10배 이상 빠른 속도로 배열 연산을 처리

np is faster than python – numpy-01. ipynb

```
import numpy as np
my_arr = np.arange(1000000)
my_list = list(range(1000000))
```

1 million elements

NumPy : arange
Python : range

```
%time for _ in range(10): my_arr2 = my_arr * 2
```

```
Wall time: 25.9 ms
```

```
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
Wall time: 905 ms
```

Numpy ndarray (1/2): n-dimensional Array Object

ndarray 속성 및 연산

```
import numpy as np
data = np.random.randn(2, 3)
data
```

```
array([[ 0.80575002,  1.31087794,  1.37496429],
       [-1.04023463, -0.87827823, -0.01420351]])
```

```
data.shape
```

```
(2, 3)
```

```
data.dtype
```

```
dtype('float64')
```

```
type(data)
```

```
numpy.ndarray
```

```
data + data
```

```
array([[ 1.61150004,  2.62175588,  2.74992858],
       [-2.08046926, -1.75655645, -0.02840702]])
```

```
data * 10
```

```
array([[  8.0575002 ,  13.10877938,  13.7496429 ],
       [-10.40234632,  -8.78278225,  -0.14203511]])
```

Numpy ndarray (2/2)

np.random.randn

- 정규 분포(**normal distribution**)를 의미
  - 모든 원소는 실수(**float**)
- 정규 가우스 분포: 평균=0, 표준편차=1

```
arr = np.random.randn(100)
```

```
arr.mean()
```

```
0.029474975104755677
```

```
arr.std()
```

```
1.0227020633018393
```

ndarray 특징

1. Arrays can have any number of dimensions, including zero (a scalar).
2. Arrays are typed: **np.uint8, np.int64, np.float32, np.float64**.
3. Each element of the array has the same type.

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]],dtype=np.float32)
print (a.ndim, a.shape, a.dtype)
```

```
2 (2, 3) float32
```

ndarray 생성 (1/3) – np.array

- Python 리스트(또는 tuple)와 np.array를 사용

```
# create an array from a list
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

```
array([6. , 7.5, 8. , 0. , 1. ])
```

```
type(arr1)
```

```
numpy.ndarray
```

```
type(data1)
```

```
list
```

```
# convert lists of the same length to a 2D array
data2 = [[1,2,3,4],[5,6,7,8]]
arr2 = np.array(data2)
arr2
```

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

```
arr2.ndim
```

```
2
```

```
arr2.shape
```

```
(2, 4)
```

ndarray 생성 (2/3)

- np.arange, np.linspace 와 같은 함수를 사용

```
np.linspace(0,10,25)
```

```
array([ 0.        ,  0.41666667,  0.83333333,  1.25      ,  1.66666667,
        2.08333333,  2.5       ,  2.91666667,  3.33333333,  3.75      ,
        4.16666667,  4.58333333,  5.        ,  5.41666667,  5.83333333,
        6.25      ,  6.66666667,  7.08333333,  7.5       ,  7.91666667,
        8.33333333,  8.75      ,  9.16666667,  9.58333333, 10.        ])
```

```
np.logspace(0,10,10, base=np.e)
```

```
array([1.00000000e+00, 3.03773178e+00, 9.22781435e+00, 2.80316249e+01,
       8.51525577e+01, 2.58670631e+02, 7.85771994e+02, 2.38696456e+03,
       7.25095809e+03, 2.20264658e+04])
```

```
np.diag([1,2,3])
```

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

ndarray 생성 (3/3)

- **np.zeros, np.empty, np.ones 와 같은 함수를 사용**

```
import numpy as np
np.zeros(10)
```

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

```
np.zeros((3,6))
```

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

```
np.empty((2,3,2))
```

```
array([[[1.27080098e-311, 3.16202013e-322],
        [0.00000000e+000, 0.00000000e+000],
        [0.00000000e+000, 1.95035900e+160]],

       [[5.24824275e+174, 1.21608390e-046],
        [1.31532202e-047, 2.90005442e-057],
        [2.69685993e+184, 7.49259746e-067]]])
```

```
np.ones(10)
```

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

```
one_list = np.ones(10)
```

```
type(one_list)
```

```
numpy.ndarray
```

Create an uninitialized array with garbage values.

ndarray 차원 변환

- **reshape : 1차원 배열을 다차원 배열로 변환**

```
arr = np.arange(32).reshape((8,4))
arr
```

```
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]])
```

1차원 배열 원소 = (다차원 배열의) 행x열

Select by Boolean value – numpy-02. ipynb

```
names = np.array(['Kim', 'Lee', 'Park', 'Kim', 'Park', 'Lee', 'Lee'])
data = np.random.randn(7,4)
```

```
names
```

```
array(['Kim', 'Lee', 'Park', 'Kim', 'Park', 'Lee', 'Lee'], dtype='<U4')
```

```
data
```

```
array([[ 0.48408458,  1.24158315,  1.24129628,  0.47844047],
       [ 0.01351501, -0.3823599 , -0.20404109, -0.87467514],
       [ 0.68669715,  0.19150034, -0.8000639 ,  0.42169539],
       [ 0.27083698,  0.22715289, -0.36498433,  1.474622  ],
       [-0.34285793, -0.76967845, -2.40397905,  0.56079246],
       [ 0.041964  ,  0.41468742,  0.0697782 , -0.66825039],
       [-0.57673388,  0.46331817,  0.34054792, -0.23458025]])
```

배열 data에서 'Kim'에 대응되는 행(row)인 0, 3을 찾고 싶다. 'Lee'는 행 1,5,6, 'Park'은 행 2,4.

```
names == 'Kim'
```

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

```
data[names == 'Kim']
```

```
array([[ 0.48408458,  1.24158315,  1.24129628,  0.47844047],
       [ 0.27083698,  0.22715289, -0.36498433,  1.474622  ]])
```

Practice #1

■ **Self learning : Do it yourself !**

```
data[names == 'Kim', 2:]
#data[names == 'Kim', 3]

names != 'Kim'
#~(names == 'Kim')
data[~(names == 'Kim')]

cond = names == 'Kim'
data[~cond]

mask = (names == 'Kim') | (names == 'Park')
mask
data[mask]

data[data<0] = 0  # 모든 음수 값을 0으로 바꿈

data[names != 'Lee'] = 7
```

Transpose (전치) – numpy-03. ipynb

```
arr = np.arange(15).reshape((3,5))
arr
```

2D array

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

```
arr.T
```

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

```
arr.T.dot(arr)
np.matmul(arr.T, arr)
```

```
arr = np.random.randn(6, 3)
arr
```

```
array([[ 1.28068039, -0.13038423,  0.96057337],
       [ 0.19074038, -0.79647797, -1.36782827],
       [ 0.16365705, -0.58790259, -1.32225372],
       [-0.05589944,  0.72981198,  0.30708075],
       [ 0.60411168, -0.58191187, -0.56321216],
       [-0.9082198 , -0.26939451, -1.2242103 ]])
```

```
np.dot(arr.T, arr)
```

```
array([[ 2.89624666, -0.56278185,  1.5073346 ],
       [-0.56278185,  1.94082702,  2.62320444],
       [ 1.5073346 ,  2.62320444,  6.45220765]])
```

Mathematical and statistical method (1/2)

```
arr = np.random.randn(5, 4)
arr
```

```
array([[-1.41214623, -0.21592616,  2.82336698, -0.01749237],
       [ 0.5461368 ,  0.65015356, -1.0886096 ,  0.56510676],
       [ 1.65811068,  0.84214537, -0.92751196, -0.04823559],
       [-1.2259034 , -0.73683956, -1.83191825, -0.05519403],
       [-0.42349617, -0.37118824,  1.56575017, -0.89049822]])
```

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

```
-0.02970947323206206
```

```
arr.sum()
```

```
arr.mean(axis=1)
```

axis=1, find the sum of each row

```
array([ 0.29445056,  0.16819688,  0.38112713, -0.96246381, -0.02985812])
```

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

axis=0, find the sum of each column

```
array([-0.85729832,  0.16834497,  0.54107734, -0.44631345])
```

Mathematical and statistical method (2/2)

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7])
```

```
arr.cumsum()
```

```
array([ 0,  1,  3,  6, 10, 15, 21, 28], dtype=int32)
```

Return the cumulative sum of the elements along a given axis.

```
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
arr
```

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

```
arr.cumsum(axis=0)
```

```
array([[ 0,  1,  2],
       [ 3,  5,  7],
       [ 9, 12, 15]], dtype=int32)
```

```
arr.cumprod(axis=1)
```

```
array([[  0,   0,   0],
       [  3,  12,  60],
       [  6,  42, 336]], dtype=int32)
```

Return the cumulative product of the elements along each row(axis=1)

Expressing Conditional Logic as Array Operations

```
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])
```

```
result = [(x if c else y) for x, y, c in zip(xarr, yarr, cond)]
result
```

```
[1.1, 2.2, 1.3, 1.4, 2.5]
```

Python only

```
result = np.where(cond, xarr, yarr)
print(result)
```

```
[1.1 2.2 1.3 1.4 2.5]
```

NumPy version

Positive numbers are replaced by 2, negative numbers are replaced by -2.

```
arr = np.random.randn(4,4)
arr
```

```
array([[ 0.39697634,  1.22585531,  0.23271269,  0.45859679],
       [ 0.81604988, -1.7047198 ,  0.78379346, -0.56051444],
       [-0.07244697, -1.31533048, -0.04294008, -0.80956578],
       [-1.23594955,  1.66882977,  0.36851927,  0.43390901]])
```

```
arr > 0
```

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

```
np.where(arr > 0, 2, -2)
```

```
array([[ 2,  2,  2,  2],
       [ 2, -2,  2, -2],
       [-2, -2, -2, -2],
       [-2,  2,  2,  2]])
```