



데이터분석 with 파이썬

Lesson 01_Numpy 기초문법2

윤수연

목차

- Numpy 인덱싱과 슬라이싱
- Numpy 2차원 배열
- Numpy 난수 생성

Numpy 문법 : 인덱싱과 슬라이싱

- 다음과 같은 성적이 저장된 1차원 배열에서 요소들을 꺼내는 방법

```
>>> import numpy as np  
>>> grades=np.array([88, 72, 93, 94])
```

- Numpy 배열에서 특정한 요소를 추출하려면 인덱스를 사용

- 파이썬 리스트와 마찬가지로 인덱스는 0부터 시작한다.
- 따라서 인덱스로 2를 지정하면 93이 출력

```
>>> grades[2]  
93
```

- 마지막 요소에 접근하려면 인덱스로 -1을 주면 된다

```
>>> grades[-1]  
94
```

Numpy 문법 : 인덱싱과 슬라이싱

인덱싱은 특정한 요소를 얻는 방법이다

0 *1* *2* *3*
grades=[88, 72, 93, 94]
>>> grades[2]
[93]

슬라이싱은 요소 집합을 선택하는 방법이다

0 *1* *2* *3*
grades=[88, 72, 93, 94]
>>> grades[1:3]
[72, 93]

Numpy 문법 : 인덱싱과 슬라이싱

■ Numpy 배열에서는 다음과 같이 슬라이싱도 가능하다

- 0에서 2까지의 슬라이스는 다음과 같이 얻을 수 있다.

```
>>> grades[ 1 : 3]  
array([72, 93])
```

■ 다음과 같이 시작 인덱스나 종료 인덱스는 생략 가능

- 파이썬 리스트와 동일

```
>>> grades [:2]  
array([88, 72])
```

Numpy 문법 : 논리적인 인덱싱

■ 논리적인 인덱싱(logical indexing)이란 어떤 조건을 주어서 배열에서 원하는 값을 추려내는 것

- 예 : 사람들의 나이가 저장된 넘파이 배열 ages가 있다고 가정 할 때

```
>>> ages=np.array([18, 19, 25, 30, 28])
```

- ages에서 20살 이상인 사람만 고르려고 하면 다음과 같은 조건식을 코딩

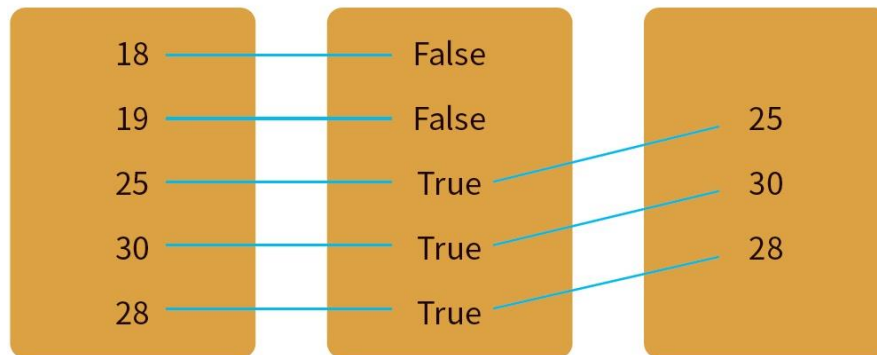
```
>>> y = ages > 20  
>>> y  
array([False, False, True, True, True])
```

- 결과는 부울형의 넘파이 배열이 된다.

Numpy 문법 : 논리적인 인덱싱 - Boolean

- 실제로는 배열 중에서 부울형의 결과를 뽑아내는 연산식이 많이 사용된다.
- 이 때는 앞의 부울형 배열을 인덱스로 하여 배열 ages에서 보내면 된다.

```
>>> ages[ ages > 20 ]  
array([25, 30, 28])
```



조건을 주어서 배열 중에서 원하는 요소들을 선택할 수 있습니다.



Numpy 문법 : 논리적인 인덱싱 - Boolean

- **Numpy 배열 a의 원소들이 50을 초과하는지 검사**
 - 초과하면 True 값을, 초과하지 않으면 False 값을 배열 b에 순서대로 저장

```
a=np.array([10, 20, 30, 40, 50, 60, 70])  
b = a > 50  
print(b)  
array[False, False, False, False, False, True, True]
```


Numpy 문법 : 논리적인 인덱싱 - Boolean

- **Quiz1 BMI가 20 이상인 사람 출력하기**
 - 우리는 다음과 같이 각 실험 대상자의 BMI를 계산하였다.

```
import numpy as np

heights = [ 1.83, 1.76, 1.69, 1.86, 1.77, 1.73 ]
weights = [ 86, 74, 59, 95, 80, 68 ]

np_heights = np.array(heights)
np_weights = np.array(weights)

bmi = np_weights/(np_heights**2)
print(bmi)
```

- BMI 수치가 25 이상인 사람들을 다음과 같이 출력하는 코딩을 완성하시오.

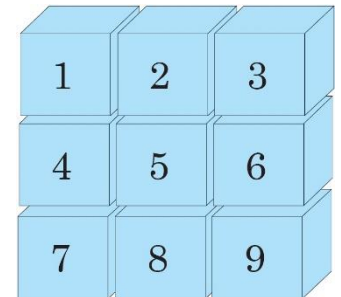
```
array([25.68007405, 27.45982194, 25.53544639])
```

Numpy 문법 : Numpy 2차원 배열

- 넘파이를 사용하면 2차원 배열도 쉽게 만들수 있다.
- 2차원 배열은 숫자들이 2차원 형태로 나열된 것이다.
- 파이썬은 2차원 리스트를 먼저 생성한 후에 이것을 넘파이의 2차원 배열로 변경해보자
- 파이썬의 2차원 리스트는 "리스트의 리스트" 라고 할 수 있다
- 수학에서의 행렬과는 약간의 차이가 있다.

```
>>> import numpy as np
>>> y = [[1,2,3], [4,5,6], [7,8,9]]
>>> y
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> ny = np.array(y)
>>> ny
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```



1	2	3
4	5	6
7	8	9

Numpy 문법 : Numpy 2차원 배열

- 첫번째 print() 문은 2차원 배열의 첫번째 원소인 1차원 배열 출력
- 두번째 print() 문은 첫번째 1차원 배열에 저장된 원소들의 평균값 (mean)을 실수 형태로 출력
- 세번째 print() 문은 x 배열 전체의 원소들의 평균값을 구해 출력
- 네번째 print() 문은 x 배열의 모양(형식)을 출력

출력	<pre>x = np.array([[1, 3, 5], [2, 4, 6]]) print(x[1]) # [2 4 6] 출력 print(x[1].mean()) # 4.0출력 print(x.mean()) # 3.5 출력 print(x.shape) # (2, 3) 출력</pre>
----	--

Numpy 문법 : Numpy 2차원 배열

■ 2차원 list

■ Quiz 2

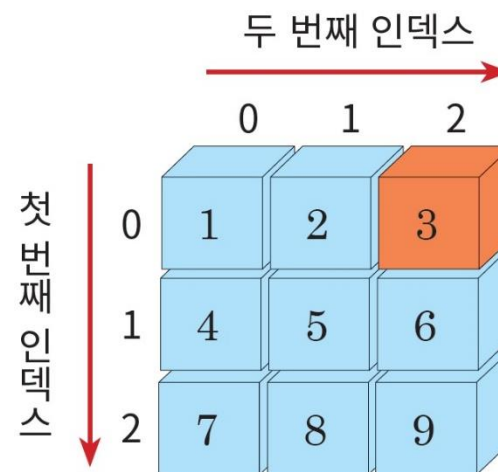
- 다음의 list1 은 파이썬의 2차원 리스트를 나타낸 것이다. 다음의 명령문을 실행할 때 출력되는 결과를 적으시오.

명령문	<pre>list1 = [[1, 11], [2, 12], [3, 13]] print(list1[1][1])</pre>
결과	

Numpy 문법 : Numpy 2차원 배열의 인덱스

- 2차원 배열에서 특정한 위치에 있는 요소는 어떻게 꺼낼까?
- 2차원 배열도 인덱스를 사용한다.
- 다만 2차원이기 때문에 인덱스가 2개 필요
- 첫 번째 인덱스는 행 번호
- 두 번째 인덱스는 열 번호

```
>>> ny[0][2]  
3
```



Numpy 문법 : Numpy 2차원 배열

■ `arrange()` 함수

- `arrange` 함수를 사용하면 특정한 범위의 정수를 가지는 넘파이 배열을 쉽게 만들 수 있다.

`np.arrange(start, stop, step)`



시작값



종료값



간격

Numpy 문법 : Numpy 2차원 배열 `arrange()`

- 다음과 같은 성적이 저장된 1차원 배열에서 요소들을 꺼내는 방법

```
>>> import numpy as np  
>>> np.arange(5)  
array([0, 1, 2, 3, 4])
```

- 시작값을 지정하려면 다음과 같이 한다.

```
>>> np.arange(1, 6)  
array([1, 2, 3, 4, 5])
```

- 종료되는 값을 지정하려면 다음과 같이 한다.

```
>>> np.arange(1, 10, 2)  
array([1, 3, 5, 7, 9])
```

Numpy 문법 : Numpy 2차원 배열

■ linspace() 함수

- linspace 함수는 상당히 많이 사용되는 함수이다.
- linspace()는 시작값부터 종료값까지 균일한 간격으로 지정된 개수 만큼의 배열을 생성한다.

```
np.linspace(start, stop, step)
```

시작값

종료값

개수

Numpy 문법 : Numpy 2차원 배열 linspace()

- `linspace(0, 10, 100)` 이라고 호출하면 0에서 10까지 100개의 수들이 생성된다.

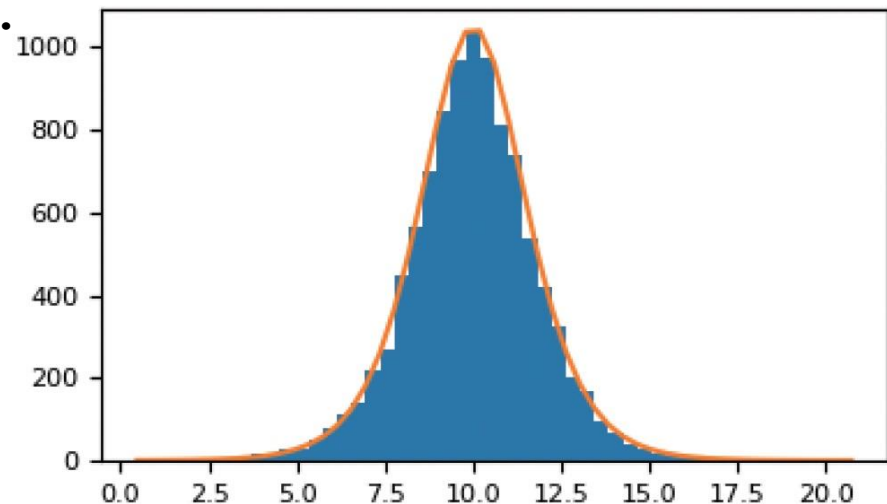
```
>>> np.linspace(0, 10, 100)
array([ 0.          , 0.1010101 , 0.2020202 , 0.3030303 , 0.4040404 ,
        ...
        8.08080808, 8.18181818, 8.28282828, 8.38383838, 8.48484848,
        8.58585859, 8.68686869, 8.78787879, 8.88888889, 8.98989899,
        9.09090909, 9.19191919, 9.29292929, 9.39393939, 9.49494949,
        9.5959596 , 9.6969697 , 9.7979798 , 9.8989899 , 10.          ])
```

- `logspace()`는 로그 스케일로 수들이 생성된다.
- 이것들은 모두 MATLAB 에서 유래한 함수들이다
- 형식은 `logspace(x,y,n)`이며, 10의 x승~10의 y승 사이의 n 개의 수들을 생성한다.

```
>>> np.logspace(0, 5, 10)
array([1.00000000e+00, 3.59381366e+00, 1.29154967e+01, 4.64158883e+01,
        1.66810054e+02, 5.99484250e+02, 2.15443469e+03, 7.74263683e+03,
        2.78255940e+04, 1.00000000e+05])
```

Numpy 문법 : 난수 생성

- 데이터가 저장되면 제일 먼저 하여야 하는 것은 데이터를 분석하는 것이다.
- 넘파이 배열은 데이터가 많지 않아 한 눈에 파악되지만, 실제 상업적인 목적으로 수집된 데이터는 상당히 크다.
- 예를 들어 서울시에 거주하는 성인 10000명의 키와 몸무게를 넘파이 배열에 저장한다고 가정했을 때, 2차원 배열로 저장한다면 10000개의 데이터를 어떻게 생성할 것인가?
- 실제 서울 시민 데이터를 입력할 수 있지만 어떤 확률분포에서 난수를 생성하여 실험데이터로 사용할 수도 있다.
- 오른쪽 그래프는 정규분포에서 생성된 난수를 표시한 것이다.



Numpy 문법 : 난수 생성

■ 시드 설정하기

- 컴퓨터 프로그램에서 발생하는 무작위 수는 사실 엄격한 의미의 무작위 수가 아니다
- 어떤 특정한 시작 숫자를 정해 주면 컴퓨터가 정해진 알고리즘에 의해 마치 난수처럼 보이는 수열을 생성
- 생성된 난수는 다음번 난수 생성을 위한 시드값이 된다.
- 파이썬에서 시드를 설정하는 명령은 seed이다. 인수로는 0과 같거나 큰 정수를 넣어준다

```
np.random.seed(0)
```

Numpy 문법 : 난수 생성

- 넘파이에서 난수의 seed를 설정하는 문장은 다음과 같다.

```
>>> np.random.seed(100)
```

- 시드가 설정되면 다음과 같은 문장을 수행하여 5개의 난수를 얻을 수 있다
 - 난수는 0.0에서 1.0 사이의 값으로 생성된다.

```
>>> np.random.rand(5)  
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])
```

- 난수로 이루어진 2차원 배열(크기=5*3)을 얻으려면 다음과 같다.

```
>>> np.random.rand(5, 3)  
array([[0.12156912, 0.67074908, 0.82585276],  
       [0.13670659, 0.57509333, 0.89132195],  
       [0.20920212, 0.18532822, 0.10837689],  
       [0.21969749, 0.97862378, 0.81168315],  
       [0.17194101, 0.81622475, 0.27407375]])
```

Numpy 문법 : 난수 생성

■ 어떤 범위에 있는 난수를 생성하려면 다음과 같은 수식을 만들 수 있다.

- 10에서 20사이에 있는 난수 5개 생성하려면

```
>>> a=10 ; b=20
>>> (b-a)*np.random.rand(5)+a
array([14.31704184, 19.4002982, 18.17649379, 13.3611195, 11.75410454])
```

■ 정수 난수가 필요하다면 randint()를 사용한다

- randint(a, b)는 정수 a와 정수 b 사이의 난수를 생성하여 반환하다.
- 주사위 10번 던지는 시뮬레이션 코딩을 작성한다면

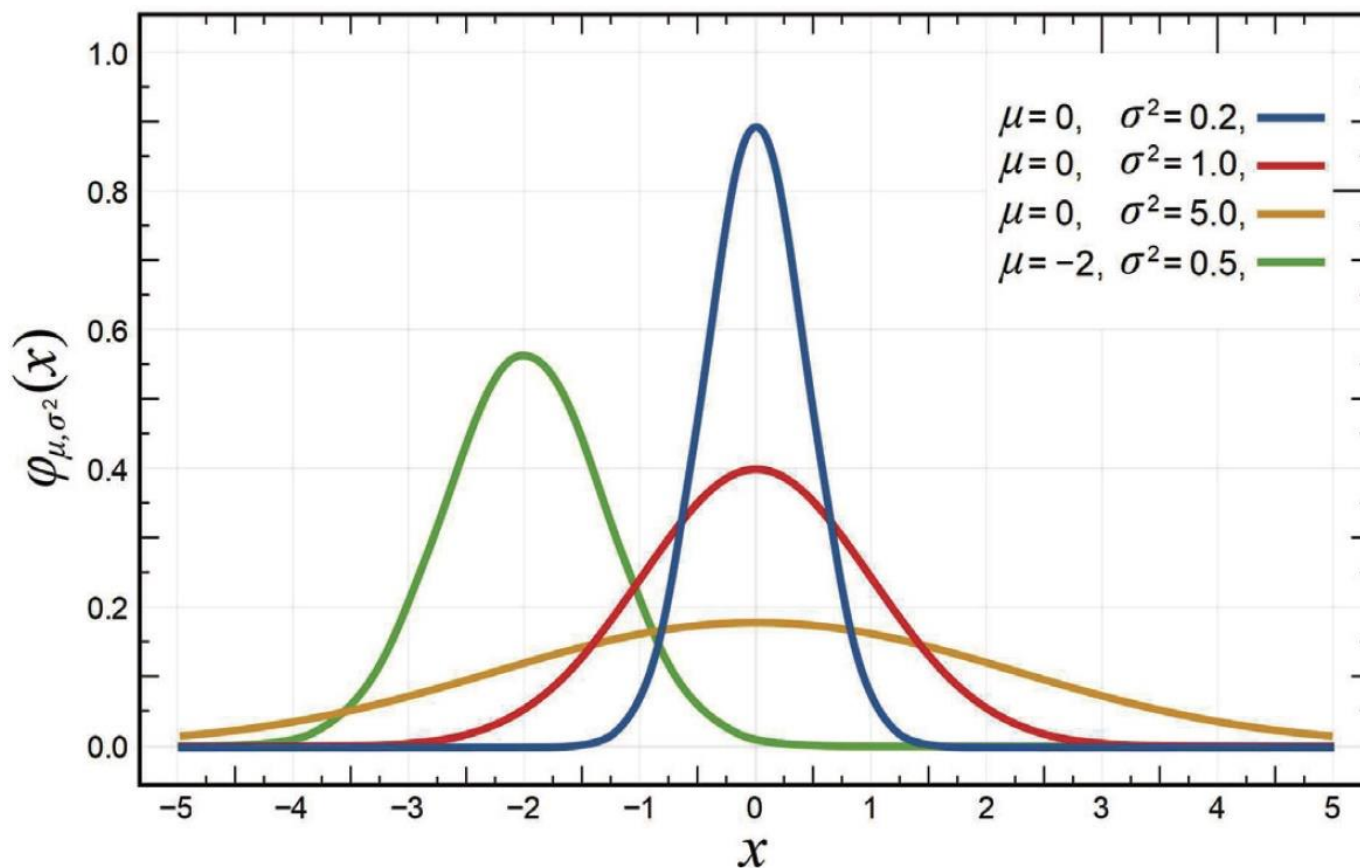
```
>>> np.random.randint(1,7, size=10)
array([4, 3, 1, 1, 2, 6, 6, 2, 6])
```

- 크기가 4*7인 2차원 배열을 1부터 10사이의 정수 난수로 채우려면 다음과 같다.

```
>>> np.random.randint(1, 11, size=(4, 7))
array([[10, 2, 6, 9, 8, 5, 3],
       [7, 3, 2, 9, 5, 3, 2],
       [3, 1, 6, 2, 9, 8, 2],
       [7, 5, 2, 8, 3, 3, 6]])
```

Numpy 문법 : 정규분포 난수 생성

- 앞에서 생성한 난수는 균일한 확률 분포에서 만들어진다.
- 정규분포란 그림과 같은 형태를 가지는 확률 분포 함수 이다.



(이미지 출처: 위키 백과)

Numpy 문법 : 정규분포 난수 생성

■ 넘파이에서 준비한 함수는 `randn()` 이다

- 정규분포에서 난수 5개 생성하려면

```
>>> np.random.randn(5)
array( [ 0.78148842, -0.65438103, 0.04117247, -0.20191691, -0.87081315 ] )
```

■ 2차원 배열 형태의 난수를 생성하려면 다음과 같이 적어준다

```
>>> np.random.randn(5, 4)
array([[ 0.22893207, -0.40803994, -0.10392514,  1.56717879],
       [ 0.49702472,  1.15587233,  1.83861168,  1.53572662],
       [ 0.25499773, -0.84415725, -0.98294346, -0.30609783],
       [ 0.83850061, -1.69084816,  1.15117366, -1.02933685],
       [-0.51099219, -2.36027053,  0.10359513,  1.73881773]])
```

- 위의 정규분포는 평균값이 0이고 표준편차가 1.0이다
- 만약 평균값과 표준편차를 다르게 하려면 다음과 같다.

```
>>> m = 10; sigma = 2
>>> m + sigma*np.random.randn(5)
array([ 8.56778091, 10.84543531,  9.77559704,  9.09052469,  9.48651379])
```

Numpy 문법 : 정규분포 그래프 그리기

■ 정규분포 그래프를 그리기 위해 다음과 같이 코딩해보자

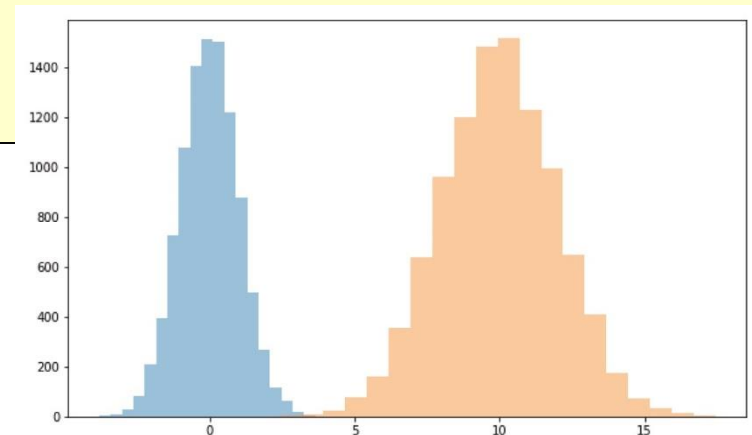
```
import numpy as np
import matplotlib.pyplot as plt

m = 10; sigma = 2
x1 = np.random.randn(10000)
x2 = m+sigma*np.random.randn(10000)
```

```
plt.figure(figsize=(10,6))
plt.hist(x1, bins=20, alpha=0.4)
plt.hist(x2, bins=20, alpha=0.4)
plt.show()
```

20개의 상자를 이용하여 히스토그램 계산

- `np.random.randn(10000)`은 평균이 0인 정규분포에서 100000 개의 난수를 생성한다.
- 수식 `m+sigma*np.random.randn(10000)`은 평균이 `m`이고 표준편차가 `sigma`인 정규분포에서 난수를 생성한다. 이 2개의 정규분포를 그래프로 그리면 그림과 같다.





THANK YOU FOR
YOUR ATTENTION