Machine Learning Class

판다스 & 넘피 강의자료

Kim, Chang Yeong

00. Introduction

분석에 특화된 모듈(라이브러리)

- NumPy
  - 고성능 과학계산을 위한 데이터 분석 라이브러리
- Pandas
  - 행과 열로 구성된 표 형식의 데이터를 지원하는 라이브러리
- Matplotlib
  - 2D 그래프로 시각화가 가능한 라이브러리

01. Pandas

Pandas 모듈

- Series Class : 1차원
  - 인덱스(index) + 값(value)
- DataFrame Class : 2차원
  - 표와 같은 형태

01. Pandas

Pandas 사용하기

- import pandas as pd
  - Pandas 모듈(라이브러리)를 import하고 앞으로 pd라는 이름으로 부른다.

01. Pandas

Series

Series 생성

- obj = pd.Series([4,7,-5,3])

```
0    4
1    7
2   -5
3    3
dtype: int64
```

01. Pandas

Series

인덱스 지정하여 생성하기

- obj2 = pd.Series([4,7,-5,3], index=['d','b','a','c'])

```
d    4
b    7
a   -5
c    3
dtype: int64
```

01. Pandas

Series

Series 값 확인

- obj2.values

```
array([ 4,  7, -5,  3], dtype=int64)
```

Series 인덱스 확인

- obj2.index

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

Series 타입 확인

- obj2.dtype

```
dtype('int64')
```

01. Pandas

Series

각 도시의 2015년 인구 데이터를 시리즈로 만들어 보세요.

- s = pd.Seires([9904312, 3448737, 2890451,2466052],
  index=["서울", "부산", "인천", "대구"])

```
서울     9904312
부산     3448737
인천     2890451
대구     2466052
dtype: int64
```

01. Pandas

Series

Series에 이름 지정

- s.name = “인구”
- s.index.name = “도시”

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

01. Pandas

Series

Series연산

- s / 1000000

```
도시
서울    9.904312
부산    3.448737
인천    2.890451
대구    2.466052
Name: 인구, dtype: float64
```

01. Pandas

Series

Series 인덱싱

- s[1], s["부산"]

```
(3448737, 3448737)
```

- s[3], s["대구"]

```
(2466052, 2466052)
```

- s[[0,3,1]]
- s[["서울", "대구", "부산"]]

```
도시
서울    9904312
대구    2466052
부산    3448737
Name: 인구, dtype: int64
```

01. Pandas

Series

Series Boolean 인덱싱

- s>=2500000

```
도시
서울      True
부산      True
인천      True
대구     False
Name: 인구, dtype: bool
```

```
도시
서울    9904312
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

01. Pandas

Series

인구수가 250만 이상의 도시

- s[s>=2500000]

```
도시
서울    9904312
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

인구수가 500만 이하의 도시

- s[s<=5000000]

```
도시
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

인구수가 250만 이상 500만 이하의 도시

- s[(s>=2500000) & (s<=5000000)]

```
도시
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

01. Pandas

Series

Series 슬라이싱

- s[1 : 3]

```
도시
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

- s["부산" : "대구"]

```
부산    3448737
인천    2890451
대구    2466052
dtype: int64
```

01. Pandas

Series

딕셔너리 객체로 Series 생성 (2010년 인구수)

- data = {"서울": 9631482, "부산": 3393191,
  "인천": 2632035, "대전": 1490158}

- s2 = pd.Series(data)

```
대전     1490158
부산     3393191
서울     9631482
인천     2632035
dtype: int64
```

01. Pandas

Series

2015년도와 2010년도의 인구 증가를 계산

- ds = s - s2

```
대구          NaN
대전          NaN
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

01. Pandas

Series

- ds.notnull()

```
도시
서울     True
부산     True
인천     True
대구    False
대전    False
Name: 인구, dtype: bool
```

- ds[ds.notnull()]

```
도시
서울    9904312.0
부산    3448737.0
인천    2890451.0
Name: 인구, dtype: float64
```

01. Pandas

Series

- ds.isnull()

```
도시
서울    False
부산    False
인천    False
대구     True
대전     True
Name: 인구, dtype: bool
```

- ds[ds.isnull()]

```
도시
대구   NaN
대전   NaN
Name: 인구, dtype: float64
```

01. Pandas

Series

2015년도와 2010년도의 인구 증가율(%)를 계산

- rs = (s – s2)/s2*100
- rs[rs.notnull()]

```
부산     1.636984
서울     2.832690
인천     9.818107
dtype: float64
```

01. Pandas

Series

Series 데이터 갱신, 추가, 삭제

- rs["부산"] = 1.6
- rs["대구"] = 1.41
- del rs["서울"]
- rs[rs.notnull()]

```
대구      1.410000
부산      1.600000
인천      9.818107
dtype: float64
```

01. Pandas

DataFrame

DataFrame을 만드는 방법

- data = {
  "2015": [9904312, 3448737, 2890451, 2466052],
  "2010": [9631482, 3393191, 2632035, 2431774]
  }
- df = pd.DataFrame(data)

	2010	2015
0	9631482	9904312
1	3393191	3448737
2	2632035	2890451
3	2431774	2466052

01. Pandas

DataFrame

DataFrame 인덱스 수정

- df.index = [“서울”, “부산”, “인천“, “대구“]

	2010	2015
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

01. Pandas

DataFrame

DataFrame 인덱스 지정하여 생성

- ind = [“2015”, “2010”]
- col = [“서울”, “부산“, “인천“, “대구“]
- df2 = pd.DataFrame(data, index = ind, columns = col)

	서울	부산	인천	대구
2015	9904312	3448737	2890451	2466052
2010	9631482	3393191	2632035	2431774

01. Pandas

DataFrame

df2.T

	2015	2010
서울	9904312	9631482
부산	3448737	3393191
인천	2890451	2632035
대구	2466052	2431774

01. Pandas

DataFrame

DataFrame 값 확인

- df.values

```
array([[9904312, 9631482],
       [3448737, 3393191],
       [2890451, 2632035],
       [2466052, 2431774]], dtype=int64)
```

DataFrame 인덱스 확인

- df.index

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
```

DataFrame 컬럼 확인

- df.columns

```
Index(['2015', '2010'], dtype='object')
```

01. Pandas

DataFrame

DataFrame 열 인덱스

- df[“2015”]

```
서울      9904312
부산      3448737
인천      2890451
대구      2466052
Name: 2015, dtype: int64
```

- df[[“2010”]]

	2015
서울	9904312
부산	3448737
인천	2890451
대구	2466052

- df[[“2010”, ”2015”]]

	2010	2015
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

01. Pandas

DataFrame

DataFrame 열 인덱스

- df[["2015","2010"]]

	2015	2010
서울	9904312	9631482
부산	3448737	3393191
인천	2890451	2632035
대구	2466052	2431774

- df[["2010", "2015"]]

	2010	2015
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

01. Pandas

DataFrame

"2005"라는 컬럼명으로 2005년 인구수 대입

- df ["2005"] = [9762546, 3512547, 2517680, 2456016]

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547
인천	2890451	2632035	2517680
대구	2466052	2431774	2456016

01. Pandas

DataFrame

DataFrame 행 인덱싱

- df[0:1]

	2015	2010	2005
서울	9904312	9631482	9762546

DataFrame 행 인덱싱

- df["서울":"인천"]

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547
인천	2890451	2632035	2517680

01. Pandas

DataFrame

loc(), iloc() 함수

- loc()
  - 실제 인덱스를 사용하여 행을 가지고 올 때 사용
    > df.loc[행,열]
    > df.loc[“서울”]
    > df.loc[“서울”:”부산”,”2015”:”2010”]

```
2015    9904312.0
2010    9631482.0
2005    9762546.0
Name: 서울, dtype: float64
```

	2015	2010
서울	9904312	9631482
부산	3448737	3393191

01. Pandas

DataFrame

loc(), iloc() 함수

- iloc()

  - numpy의 array인덱싱 방식으로 행을 가지고 올 때 사용

  >df.iloc[3]

```
2015    2890451.0
2010    2632035.0
2005    2517680.0
Name: 인천, dtype: float64
```

01. Pandas

DataFrame

Pandas Boolean 인덱싱

- df ["2010"] >=2500000

```
서울      True
부산      True
인천      True
대구     False
Name: 2010, dtype: bool
```

01. Pandas

DataFrame

csv파일 불러오기

```
population_number = pd.read_csv("population_number.csv",encoding="euc-kr")
```

	도시	지역	2015	2010	2005	2000
0	서울	수도권	9904312	9631482.0	9762546.0	9853972
1	부산	경상권	3448737	NaN	NaN	3655437
2	인천	수도권	2890451	2632035.0	NaN	2466338
3	대구	경상권	2466052	2431774.0	2456016.0	2473990

01. Pandas

DataFrame

csv파일 불러오기

```
pd.read_csv("population_number.csv",index_col="도시", encoding="euc-kr",)
```

도시	지역	2015	2010	2005	2000
서울	수도권	9904312	9631482.0	9762546.0	9853972
부산	경상권	3448737	NaN	NaN	3655437
인천	수도권	2890451	2632035.0	NaN	2466338
대구	경상권	2466052	2431774.0	2456016.0	2473990

01. Pandas

DataFrame

count 함수

- 데이터 개수를 셀 수 있다.

```
population_number.count()
```

```
도시        4
지역        4
2015    3
2010    2
2005    4
2000    4
dtype: int64
```

01. Pandas

DataFrame

value_counts 함수

- 값이 숫자, 문자열, 카테고리 값인 경우에 각각의 값이 나온 횟수를 셀 수 있다.

➢ s2 = pd.Series(np.random.randint(6, size=100))

➢ s2.tail()

```
Out[18]: 95    5
         96    5
         97    5
         98    3
         99    4
         dtype: int32
```

```
Out[19]: 4    25
         3    18
         5    17
         0    16
         1    15
         2     9
         dtype: int64
```

01. Pandas

DataFrame

```
population_number["2015"].value_counts()
```

```
2466052    1
2890451    1
3448737    1
9904312    1
Name: 2015, dtype: int64
```

```
population_number["2010"].value_counts()
```

```
9631482.0    1
2431774.0    1
2632035.0    1
Name: 2010, dtype: int64
```

01. Pandas

DataFrame

정렬

```
population_number["2010"].sort_values()
```

```
도시
대구     2431774.0
인천     2632035.0
서울     9631482.0
부산           NaN
```

01. Pandas

DataFrame

정렬

```
population_number["2010"].sort_values(ascending=False)
```

```
도시
서울     9631482.0
인천     2632035.0
대구     2431774.0
부산           NaN
Name: 2010, dtype: float64
```

01. Pandas

DataFrame

DataFrame을 2010년 인구수 기준으로 정렬

```
population_number.sort_values(by="2010")
```

도시	지역	2015	2010	2005	2000
대구	경상권	2466052	2431774.0	2456016.0	2473990
인천	수도권	2890451	2632035.0	NaN	2466338
서울	수도권	9904312	9631482.0	9762546.0	9853972
부산	경상권	3448737	NaN	NaN	3655437

01. Pandas

DataFrame

DataFrame을 지역, 2010년 인구수 기준으로 정렬

```
population_number.sort_values(by=["지역","2010"])
```

도시	지역	2015	2010	2005	2000
대구	경상권	2466052	2431774.0	2456016.0	2473990
부산	경상권	3448737	NaN	NaN	3655437
인천	수도권	2890451	2632035.0	NaN	2466338
서울	수도권	9904312	9631482.0	9762546.0	9853972

01. Pandas

DataFrame

다음 데이터 프레임 만들기

과목	1반	2반	3반	4반
수학	45	44	73	39
영어	76	92	45	69
국어	47	92	45	69
사회	92	81	85	40
과학	11	79	47	26

01. Pandas

DataFrame

학급별 총계

```
score.sum()
```

```
1반    271
2반    388
3반    295
4반    243
dtype: int64
```

학급별 순위

```
score.sum().sort_values()
```

```
4반    243
1반    271
3반    295
2반    388
dtype: int64
```

01. Pandas

DataFrame

과목별 총계

```
score.sum(axis=1)
```

```
과목
수학    201
영어    282
국어    253
사회    298
과학    163
dtype: int64
```

01. Pandas

DataFrame

과목별 합계 DataFrame에 추가하기

```
score["합계"]=score.sum(axis=1)
```

과목	1반	2반	3반	4반	합계
수학	45	44	73	39	201
영어	76	92	45	69	282
국어	47	92	45	69	253
사회	92	81	85	40	298
과학	11	79	47	26	163

01. Pandas

DataFrame

max(), min() 함수

- max() : 가장 큰 값

```
score.max()
```

```
1반      92.0
2반      92.0
3반      85.0
4반      69.0
합계     298.0
평균      74.5
dtype: float64
```

- min() : 가장 작은 값

```
score.min()
```

```
1반      11.00
2반      44.00
3반      45.00
4반      26.00
합계     163.00
평균      40.75
dtype: float64
```

max(), min() 함수

- max() : 가장 큰 값

```
score.max(axis=1)
```

```
과목
수학      201.0
영어      282.0
국어      253.0
사회      298.0
과학      163.0
반평균     239.4
dtype: float64
```

- min() : 가장 작은 값

```
score.min(axis=1)
```

```
과목
수학      39.0
영어      45.0
국어      45.0
사회      40.0
과학      11.0
반평균     48.6
dtype: float64
```

01. Pandas

DataFrame

```
maxArr = score.loc[:"과학",:"4반"].max(axis=1)
maxArr
```

```
과목
수학    73.0
영어    92.0
국어    92.0
사회    92.0
과학    79.0
dtype: float64
```

01. Pandas

DataFrame

```
minArr = score.loc[:"과학",:"4반"].min(axis=1)
minArr
```

```
과목
수학    39.0
영어    45.0
국어    45.0
사회    40.0
과학    11.0
dtype: float64
```

01. Pandas

DataFrame

fillna 함수

- 결측치(NaN)를 원하는 값으로 바꿀 수 있다.

➢ df3.fillna(value=0)

Out[126]:

	A	B	C
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

01. Pandas

DataFrame

concat

```
df1 = pd.Series([275,260,250],index=["유재석","신동엽","강호동"])
df1.name = "발사이즈"
df1
```

```
df2 = pd.Series([250,245],index=["송은이","김숙"])
df2.name = "발사이즈"
df2
```

concat

```
1 df3 = pd.concat([df1,df2])
2 df3
```

```
유재석    275
신동엽    260
강호동    250
송은이    250
김숙      245
Name: 발사이즈, dtype: int64
```

01. Pandas

DataFrame

concat

```
1 df4 = pd.Series([0.2,1.8,1.5],index=["유재석","신동엽","강호동"])
2 df4.name = "시력"
3 df4
```

```
유재석    0.2
신동엽    1.8
강호동    1.5
Name: 시력, dtype: float64
```

01. Pandas

DataFrame

concat

```
df5 = pd.concat([df1,df4],axis=1)
df5
```

	발사이즈	시력
유재석	275	0.2
신동엽	260	1.8
강호동	250	1.5

02. Numpy

NumPy의 주요 기능

- 빠르고 효율적인 벡터 산술연산을 제공하는 다차원배열 제공 (ndarray 클래스)
- 반복문 없이 전체 데이터 배열 연산이 가능한 표준 수학 함수 (sum(), sqrt(), median(), mean())
- 선형대수, 난수 생성, 푸리에 변환

02. Numpy

모듈(라이브러리) 사용하기

```
import numpy as np
```

- Numpy 모듈(라이브러리)를 import하고 앞으로 np라는 이름으로 부른다.

02. Numpy

numpy.ndarray 클래스

- 동일한 자료형을 가지는 값들이 배열 형태로 존재함
- N 차원 형태로 구성이 가능하다
- 각 값들은 양의 정수로 색인(index)가 부여되어 있다
- Numpy에서 차원(Dimension)을 rank, axis라고 부르기도 한다
- ndarray를 줄여서 array로 표현한다

02. Numpy

array생성 하기 : 1차원

```
list = [1,2,3,4,5]
list
```

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

```
arr = np.array(list)
arr
```

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

02. Numpy

array생성 하기 : 2차원

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

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

02. Numpy

배열의 크기 확인하기

```
arr
```

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

```
arr2
```

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

```
arr.shape
```

```
(3,)
```

```
arr2.shape
```

```
(2, 3)
```

02. Numpy

배열의 전체 요소 개수 확인하기

```
1 array = np.arange(1,6)
2 print(array.size)
```

```
5
```

02. Numpy

배열의 타입 확인

```
arr.dtype
```

```
dtype('int32')
```

```
arr2.dtype
```

```
dtype('int32')
```

02. Numpy

배열의 차원(Dimension) 확인

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

```
2
```

02. Numpy

특정한 값으로 배열 생성하기

0으로 초기화

```
arr_zeros = np.zeros((3,4))
arr_zeros
```

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

1로 초기화

```
arr_ones = np.ones((3,4))
arr_ones
```

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

02. Numpy

특정한 값으로 배열 생성하기

```
1 arr = np.full((5,5),8)
2 arr
```

```
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

02. Numpy

1,2,3,…,50,51이 담긴 배열 생성하기

```
list = []
for i in range(1,51):
    list.append(i)
print(list)
```

```
[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]
```

02. Numpy

1,2,3,…,50,51이 담긴 배열 생성하기

```
arr3 = np.array(list)
arr3
```

```
array([ 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])
```

```
arr4 = np.arange(1,51)
arr4
```

02. Numpy

타입 지정하여 배열 생성하기

```
arr_type = np.array([1.2,2.3,3.4], dtype=np.int64)
arr_type
```

```
array([1, 2, 3], dtype=int64)
```

02. Numpy

타입변경하기

```
1 arr_type = arr_type.astype("float64")
```

```
1 arr_type
```

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

```
1 arr_type.dtype
```

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

02. Numpy

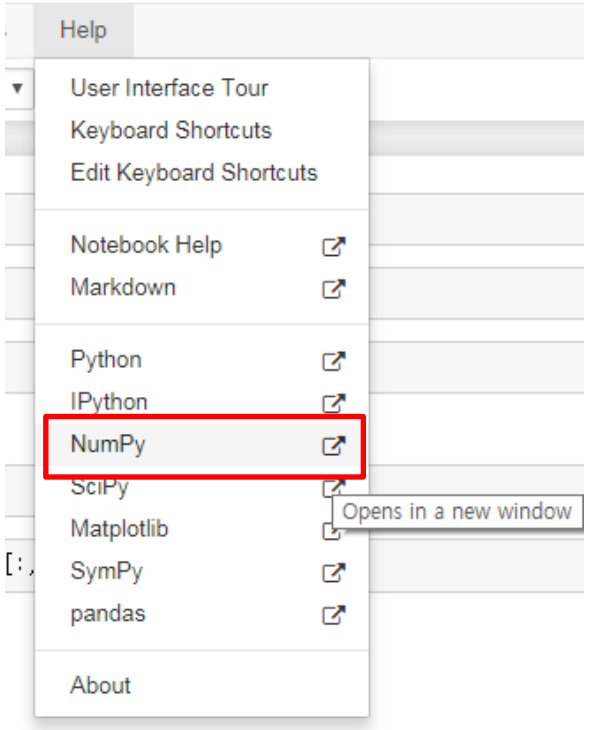
랜덤 값 배열 생성하기

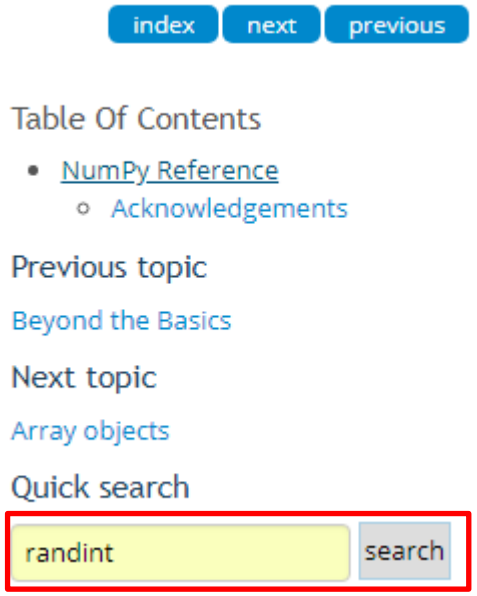
```
1 arr = np.random.rand(2,3)
2 arr
```

```
array([[0.57594363, 0.87443897, 0.20125904],
       [0.63298338, 0.50849171, 0.84489493]])
```

02. Numpy

랜덤 값 int 배열 생성하기





02. Numpy

랜덤 값 int 배열 생성하기

numpy.random.randint

numpy.random.randint(*low, high=None, size=None, dtype='l'*)¶

Return random integers from *low* (inclusive) to *high* (exclusive).

Return random integers from the "discrete uniform" distribution of the specified dtype in the "half-open" interval [*low, high*). If *high* is None (the default), then results are from [0, *low*).

Parameters:

low : *int*
Lowest (signed) integer to be drawn from the distribution (unless `high=None`, in which case this parameter is one above the *highest* such integer).

high : *int, optional*
If provided, one above the largest (signed) integer to be drawn from the distribution (see above for behavior if `high=None`).

size : *int or tuple of ints, optional*
Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

dtype : *dtype, optional*
Desired dtype of the result. All dtypes are determined by their name, i.e., 'int64', 'int', etc, so byteorder is not available and a specific precision may have different C types depending on the platform. The default value is 'np.int'.
New in version 1.11.0.

Returns:

out : *int or ndarray of ints*
size-shaped array of random integers from the appropriate distribution, or a single such random int if *size* not provided.

Examples

```
>>> np.random.randint(2, size=10)
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])
>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

```
>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1],
       [3, 2, 2, 0]])
```

02. Numpy

랜덤 seed 지정하기

```
1 np.random.seed(1)
2 np.random.randint(1,5,6)
```

```
array([2, 4, 1, 1, 4, 2])
```

```
1 np.random.seed(1)
2 np.random.randint(1,5,6)
```

```
array([2, 4, 1, 1, 4, 2])
```

random seed를
지정할 경우 매번
같은 난수가 생성된다.

02. Numpy

array 연산 (요소별 연산)

```
arr = np.array([1,2,3])
arr
```

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

```
array([2, 4, 6])
```

```
arr*arr
```

```
array([1, 4, 9])
```

02. Numpy

array 연산 (요소별 연산)

```
1 arr = np.arange(2,10,2)
2 print(arr)
3 print(arr/2.0)
```

```
[2 4 6 8]
[1. 2. 3. 4.]
```

02. Numpy

array 인덱싱

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

```
[1 2 3]
1
1
```

02. Numpy

1차원 array 슬라이싱

```
arr1 = np.arange(10)
arr1
```

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

```
arr1[3:8]
```

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

```
arr1[3:8] = 12
```

```
arr1
```

```
array([ 0,  1,  2, 12, 12, 12, 12, 12,  8,  9])
```

02. Numpy

2차원 array 생성

```
arr2 = np.arange(50).reshape(5,10)
arr2
```

```
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]])
```

02. Numpy

2차원 array 슬라이싱

```
arr2[:2,:]
```

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

```
array([ 0, 10, 20, 30, 40])
```

02. Numpy

2차원 array 슬라이싱

```
arr2[:4,:5]
```

```
array([[ 0,  1,  2,  3,  4],
       [10, 11, 12, 13, 14],
       [20, 21, 22, 23, 24],
       [30, 31, 32, 33, 34]])
```

```
arr2[2,1]
```

```
21
```

```
arr2[[2,3],[2,3]]
```

```
array([22, 33])
```

02. Numpy

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

```
1 ar[1:,1::2]
```

```
array([[ 7,  9, 11],
       [13, 15, 17]])
```

```
1 ar[1:,1::2].T
```

```
array([[ 7, 13],
       [ 9, 15],
       [11, 17]])
```

02. Numpy

Boolean 색인

```
name = np.array(["명호","해도","은비","상호"])
name
```

```
array(['명호', '해도', '은비', '상호'],
      dtype='<U2')
```

```
bol = np.array([True, False, True, False])
bol
```

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

```
name[bol]
```

```
array(['명호', '은비'],
      dtype='<U2')
```

02. Numpy

Boolean 색인

```
score = np.array([[60,60,60],[70,70,70],[80,80,80],[90,90,90]])
```

```
1 name == '명호'
```

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

```
score[name=="명호"]
```

```
array([[60, 60, 60]])
```

02. Numpy

Boolean 색인

```
score = np.array([[60,60,60],[70,70,70],[80,80,80],[90,90,90]])
```

```
1 name == '명호'
```

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

```
score[name=="명호"]
```

```
array([[60, 60, 60]])
```

02. Numpy

Universally 함수

단일 배열에 사용하는 함수

함수	설명
abs, fabs	각 원소의 절대값을 구한다. 복소수가 아닌 경우에는 fabs로 빠르게 연산가능
sqrt	제곱근을 계산 arr ** 0.5와 동일
square	제곱을 계산 arr ** 2와 동일
Exp	각 원소에 지수 e^x를 계산
Log, log10, log2, logp	각각 자연로그, 로그10, 로그2, 로그(1+x)
sign	각 원소의 부호를 계산
ceil	각 원소의 소수자리 올림
floor	각 원소의 소수자리 버림
rint	각 원소의 소수자리 반올림. dtype 유지
modf	원소의 몫과 나머지를 각각 배열로 반환
isnan	각 원소가 숫자인지 아닌지 NaN 나타내는 불리언 배열
isfinite, isinf	배열의 각 원소가 유한한지 무한한지 나타내는 불리언 배열
cos, cosh, sin, sinh, tan, tanh	일반 삼각함수와 쌍곡삼각 함수
logical_not	각 원소의 논리 부정(not) 값 계산. -arr와 동일

02. Numpy

Universally 함수

서로 다른 배열 간에 사용하는 함수

함수	설명
add	두 배열에서 같은 위치의 원소끼리 덧셈
subtract	첫번째 배열 원소 – 두번째 배열 원소
multiply	배열의 원소끼리 곱셈
divide	첫번째 배열의 원소에서 두번째 배열의 원소를 나눗셈
power	첫번째 배열의 원소에 두번째 배열의 원소만큼 제곱
maximum, fmax	두 원소 중 큰 값을 반환. fmax는 NaN 무시
minimum, fmin	두 원소 중 작은 값 반환. fmin는 NaN 무시
mod	첫번째 배열의 원소에 두번째 배열의 원소를 나눈 나머지
greater, greater_equal, less, less_equal, equal, not_equal	두 원소 간의 〉, 〉=, 〈, 〈=, ==, != 비교연산 결과를 불리언 배열로 반환
logical_and, logical_or, logical_xor	각각 두 원소 간의 논리연산, &, |, ^ 결과를 반환

02. Numpy

sum 함수

```
1 arr=np.random.rand(5,4)
2 print(arr)
3 print(arr.sum())
```

```
[[0.27835504 0.73286567 0.59505463 0.74249546]
 [0.86184864 0.48524037 0.16205655 0.18885377]
 [0.34418343 0.88056058 0.3444266  0.8740875 ]
 [0.80453612 0.26756472 0.26483781 0.73331351]
 [0.21955057 0.75198186 0.90391999 0.55469602]]
10.99042884583773
```

02. Numpy

mean 함수

```
1 print(arr.mean())
```

```
0.5495214422918865
```

02. Numpy

sqrt 함수

```
1 arr = np.arange(1,6)
2 np.sqrt(arr)
```

```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798])
```

02. Numpy

abs 함수

```
1 arr = np.array([-1,2,-3,4,-5])
2 np.abs(arr)
```

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

02. Numpy

max, min 함수

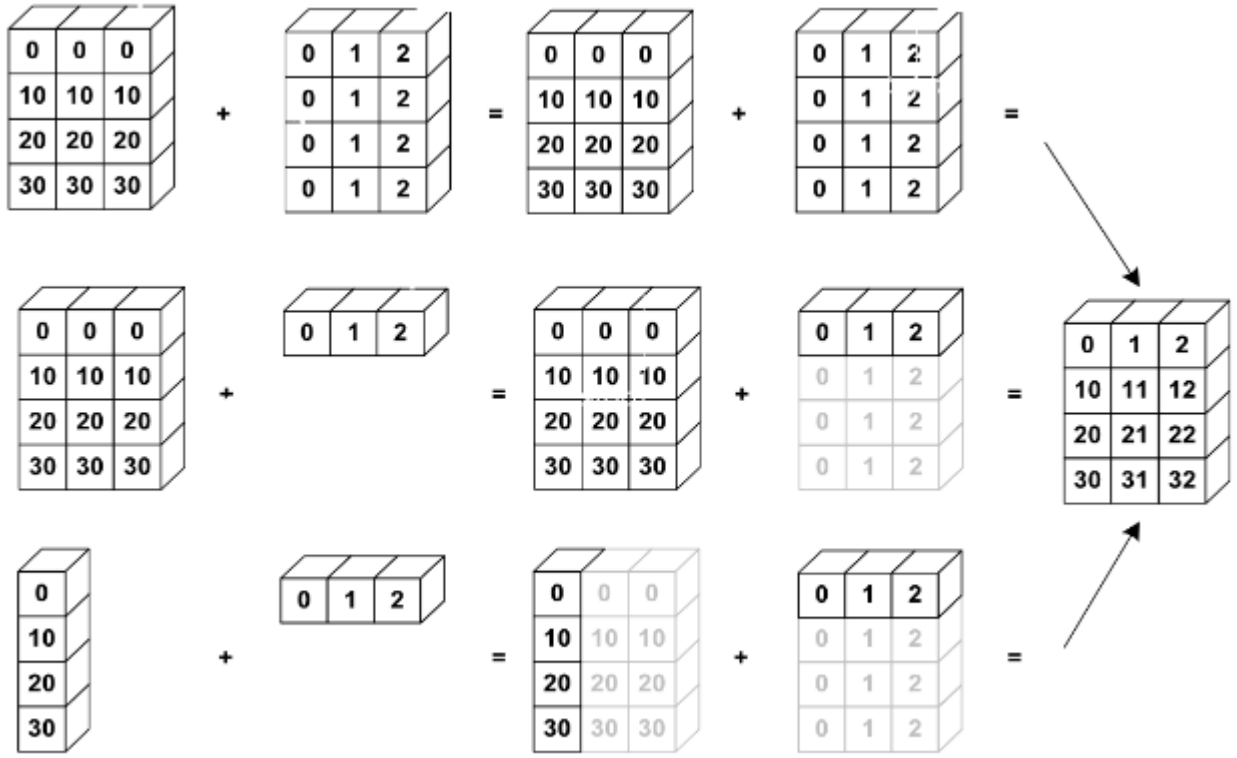
```
1 data_list = [x**2 for x in range(1,10)]
2 arr = np.array(data_list)
3 arr
```

```
array([ 1,  4,  9, 16, 25, 36, 49, 64, 81])
```

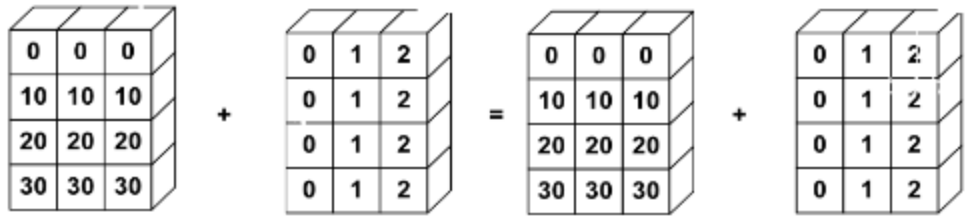
```
1 print(arr.max())
2 print(arr.min())
```

```
81
1
```

02. Numpy


0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32
0
10
20
30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=

02. Numpy



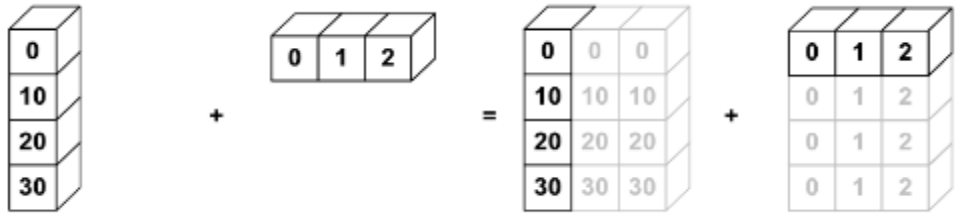

```
1 arr1 = np.array([[0]*3,[10]*3,[20]*3,[30]*3])
2 arr1
```

```
array([[ 0,  0,  0],
       [10, 10, 10],
       [20, 20, 20],
       [30, 30, 30]])
```

```
1 arr2 = np.tile([0,1,2],reps=(4,1))
2 arr2
```

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

02. Numpy




```
1 arr3 = np.array([[0,1,2,3]]).T
2 arr3
```

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

```
1 arr4 = np.array([0,1,2])
2 arr4
```

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

Thank you

Kim, Chang Yeong