02. 넘파이와 판다스

소프트웨어융합대학
인공지능학부
이수미

목차

01

넘파이와 판다스 라이브러리

01. 넘파이와 판다스 라이브러리

I. 넘파이의 배열

- 넘파이(NumPy): C 언어로 구현된 파이썬 라이브러리이며 숫자 데이터를 포함한 벡터와 행렬 연산에 유용.
  - 설치되어 있지 않은 경우 pip 명령어로 넘파이 라이브러리를 설치하여 사용.

```
pip install numpy;
```

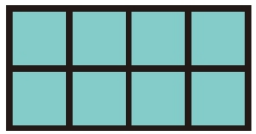
  - 배열: 넘파이의 핵심 객체. 같은 자료형인 데이터를 메모리에 물리적으로 연속 할당하여 인덱스로 데이터에 접근하기 편리.

01. 넘파이와 판다스 라이브러리

I. 넘파이의 배열

- 넘파이 배열의 차원
  - 데이터 분석에서 차원(Dimension)은 관측하고자 하는 데이터의 속성의 수 또는 측정 항목의 수를 의미.
    - » 스칼라(Scalar): 0차원 배열. 배열에서 값을 표현하는 가장 기본 단위이며, 스칼라에는 하나의 실수(Real number)를 담을 수 있음.
    - » 벡터(Vector): 1차원 배열. 스칼라(값) 여러 개를 나열한 튜플(Tuple).
    - » 행렬(Matrix): 2차원 배열. 1차원 배열을 여러 개 묶은 배열.
    - » 텐서(Tensor): 벡터의 집합. 3차원 이상의 배열은 모두 텐서.

표 6-1 넘파이 배열의 차원

차원	0차원	1차원	2차원	3차원 이상
구분	스칼라	벡터	행렬	텐서
공간 표현				

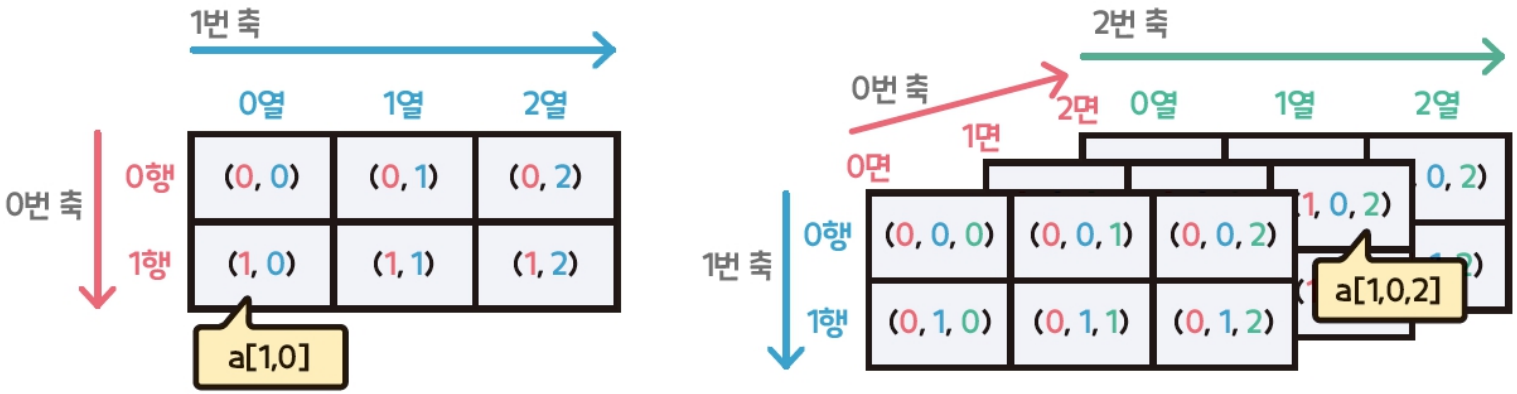
01. 넘파이와 판다스 라이브러리

I. 넘파이의 배열

- 넘파이 배열의 차원
  - 배열의 랭크(Rank)는 차원(Dimension)의 수이고, 모양(Shape)은 배열의 차원과 크기를 나타냄.
  - ndarray: 넘파이의 다차원 배열(N-dimensional array) 객체.
    ① 자료형이 모두 같은 데이터를 담은 다차원 배열.
    ② 정수 또는 실수(부동 소수점 수)를 저장.
    ③ 배열 데이터에도 순서가 있으므로 인덱싱과 슬라이싱이 가능.

I. 넘파이의 배열

- 넘파이 배열의 인덱싱과 축
  - 배열을 정수나 다른 배열, 논리값, 음이 아닌 정수의 튜플(Tuple)로 인덱싱할 수 있음.
  - 배열이 2차원이면 대괄호 안에서 행 다음에 콤마(,)를 찍고 열의 인덱스를 붙임
    3차원이면 면, 행, 열 순으로 인덱스를 붙임.
  - 배열의 축: 인덱스가 증가하는 방향. [그림 6-1] (a)에서 0번 축 방향으로 행 인덱스가 증가. (b)에서는 0번 축 방향으로 면 인덱스가 증가하며, 1번 축 방향으로는 행 인덱스, 2번 축 방향으로는 열 인덱스가 증가.



(a) 2차원 넘파이 배열의 인덱싱 (b) 3차원 넘파이 배열의 인덱싱

그림 6-1 인덱싱

01. 넘파이와 판다스 라이브러리

I. 넘파이의 배열

- 넘파이 배열의 슬라이싱
  - 슬라이싱(Slicing): 배열에서 연속한 일부분을 잘라 선택하는 것
    슬라이스: 선택된 배열 조각.
  - 튜플이나 리스트의 슬라이싱과 동일하게 인덱스와 콜론(:)을 사용.
  - 배열이 2차원 이상일 때 행의 인덱스인지 열의 인덱스인지 주의해야 함.

표 6-2 인덱싱과 슬라이싱

구분	종류	형식
인덱싱	논리값 인덱싱	조건 필터링+검색
	1차원	객체[a]
	2차원	객체[a,b]
슬라이싱	1차원	객체[a:b]
	2차원	객체[a:b, c:d]

01. 넘파이와 판다스 라이브러리

I. 넘파이의 배열

- 넘파이 배열의 슬라이싱
  - 콜론과 숫자를 함께 써서 처음과 끝 지점을 정함.
  - 열이나 행, 면 전체를 선택하려면 숫자 대신 콜론 사용.

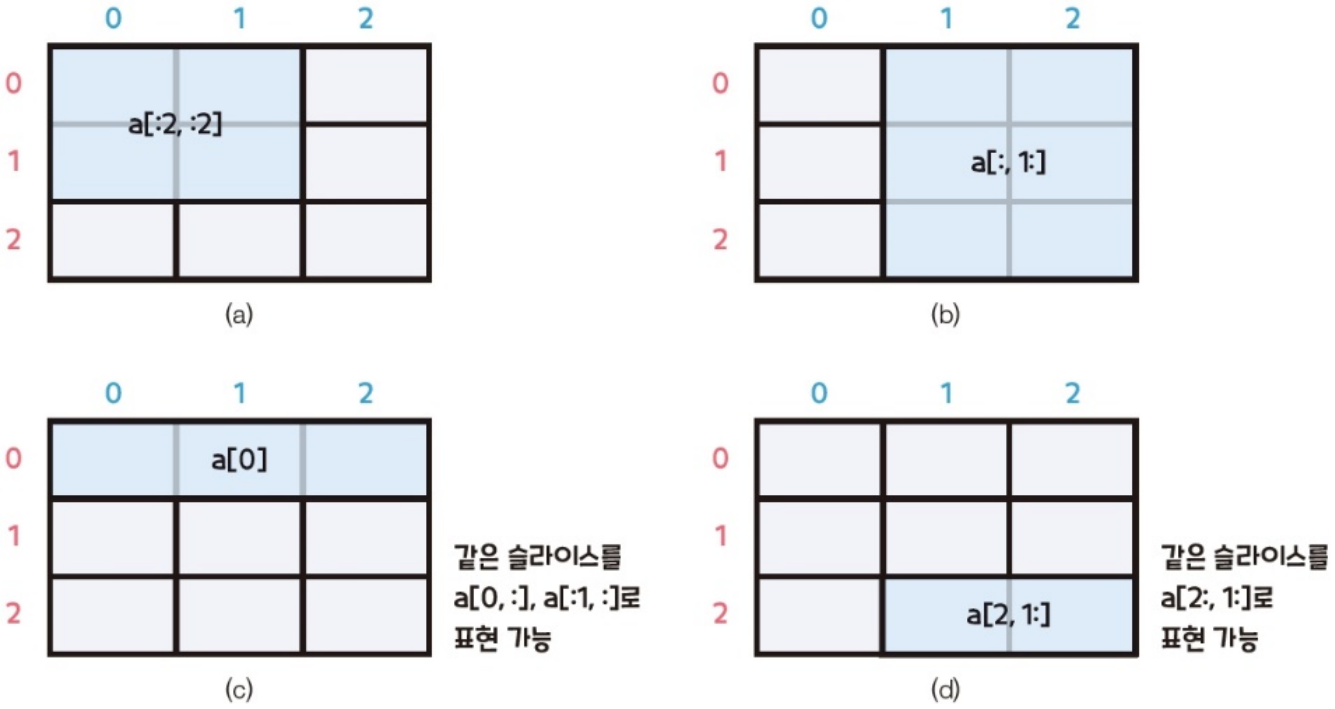


그림 6-2 넘파이 배열의 슬라이싱

II. 판다스의 시리즈와 데이터프레임

- 판다스(Pandas): 데이터프레임 자료구조를 제공하는 파이썬의 핵심 패키지.
  - 설치되어 있지 않으면 명령어 pip를 사용.

```
pip install pandas;
```

- 시리즈(Series): 인덱스와 값이 한 쌍을 이루는 1차원 자료구조 객체
  - 리스트는 값만 있고 인덱스가 0부터 자동 생성되는 반면,
    시리즈는 사용자가 직접 인덱스를 정할 수 있음.

II. 판다스의 시리즈와 데이터프레임

- 데이터프레임(Dataframe): 판다스의 기본 구조인 자료구조 객체.
  - 시리즈 여러 개를 묶어서 데이터프레임을 만드므로 형태는 2차원 배열과 비슷.
  - 데이터프레임은 행 인덱스, 열 이름(또는 열 인덱스), 값으로 구성.
    - » **행 인덱스:** 가로줄인 행(Row)을 구분하는 고유한 인덱스.
    - » **열 이름:** 세로줄인 열(Column)을 열 이름으로 구분
    - » **값(Value):** 행과 열이 교차하는 곳에 저장되는 데이터

	이름	성별	나이	키
0	한빛	남자	20	180
1	한결	남자	21	177
2	한라	여자	22	160

그림 6-3 데이터프레임

III. 넘파이와 판다스 비교

- 넘파이의 특징
  - **다차원 배열 객체:** 객체 ndarray를 사용하여 다차원 배열을 생성하고 관리하는 기능이 넘파이의 핵심.
  - **정교한 브로드캐스팅:** 브로드캐스팅이란 넘파이에서 모양(Shape)이 서로 다른 배열끼리 연산하는 방식.
  - **C, C++, 포트란 코드를 통합:** 넘파이 내부의 복잡한 알고리즘이나 고성능이 필요한 부분은 C언어나 C++, 포트란(Fortran) 코드로 구현되어 있음.
  - **수학적 알고리즘 제공:** 넘파이에서는 선형대수의 함수, 푸리에 변환, 난수 기능 등 수학의 다양한 알고리즘을 제공.

III. 넘파이와 판다스 비교

- 판다스의 특징
  - **대용량 데이터 처리:** 판다스를 활용하면 시리즈와 데이터프레임 자료구조로 대용량 데이터를 빠르게 처리.
  - **시각적으로 알아보기 편리한 표 형태:** 데이터 구조가 표 형태이기 때문에 사용자가 데이터를 알아보기에 편리.
  - **데이터 분석 도구 제공:** 판다스에서 제공하는 기능 중 데이터 분석에 자주 사용하는 기능은 결측치 처리, 관계 연산, 시계열.

02

넘파이 활용

02. 넘파이 활용

I. 넘파이의 배열 생성

– ndarray를 만들 때 넘파이 라이브러리에 정의된 array( ) 함수를 사용.

[코드 6-1] 배열 생성

```
import numpy as np

#리스트를 생성하고 배열로 변환하기
list1 = [1, 2, 3, 4]
a = np.array(list1)
print('a.shape: ', a.shape)
print('a[0]: ', a[0])

#2차원 배열 생성하기
b = np.array([[1, 2, 3], [4, 5, 6]])
print('b.shape: ', b.shape)
print('b[0,0]: ', b[0,0])
print('b[0]: ', b[0])
```

```
a.shape: (4,)
a[0]: 1
b.shape: (2,3)
b[0,0]: 1
b[0]: [1, 2, 3]
```

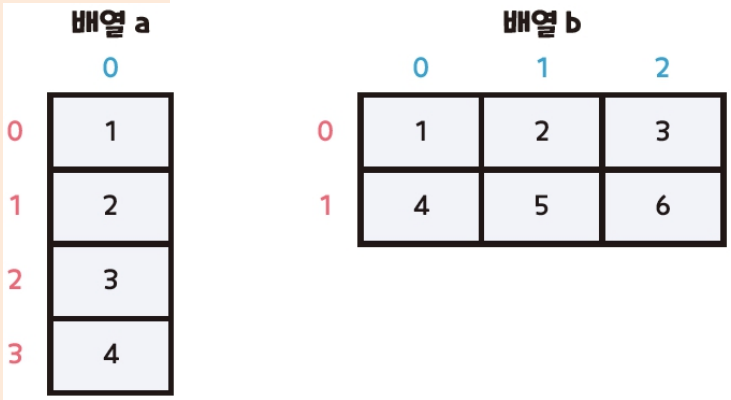


그림 6-4 넘파이 배열의 예

– 리스트 list1을 배열로 변환하여 변수 a에 할당, 배열 b는 요소를 바로 array( ) 함수의 인자로 작성하여 생성.

02. 넘파이 활용

I. 넘파이의 배열 생성

- 배열 생성 함수

표 6-3 넘파이 배열을 생성하는 함수

함수	설명
array()	리스트를 배열로 변환
arange()	일정한 간격의 수를 ndarray 배열로 반환(파이썬의 range 함수와 유사함)
ones()	1로 채운 n차원 배열을 생성
zeros()	0으로 채운 n차원 배열을 생성
empty()	초기화하지 않은 빈 n차원 배열을 생성
eye() 또는 identity()	대각선 요소에만 1을 채우고 그 외에는 0으로 채워 2차원 배열을 생성
linespace()	초깃값부터 최종값까지 지정한 간격의 수를 채워 배열을 생성
full()	지정한 모양에 지정한 값으로 채운 배열을 생성

02. 넘파이 활용

I. 넘파이의 배열 생성

- 배열 생성 함수

[코드 6-2] 배열 생성 함수

```
a = np.zeros(2)
print('a\n', a)
b = np.zeros((2,2))
print('b\n', b)
c = np.ones((2,3))
print('c\n', c)
d = np.full((2,3), 5)
print('d\n', d)
e = np.eye(3)
print('e\n', e)
```

```
a
 [0. 0.]
b
 [[0. 0.]
 [0. 0.]]
c
 [[1. 1. 1.]
 [1. 1. 1.]]
d
 [[5 5 5]
 [5 5 5]]
e
 [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

02. 넘파이 활용

I. 넘파이의 배열 생성

- 넘파이 자료형
  - array( ) 함수로 배열을 생성할 때 dtype 옵션을 추가하여 배열의 넘파이 자료형을 지정할 수 있음.
  - 배열 객체에 dtype 속성을 사용하면 넘파이 자료형을 확인할 수 있음.

[코드 6-2] 배열 생성 함수

```
#실수형 배열 생성하기
a = np.array([1, 2], dtype=np.float64)
print(a.dtype)

#정수형 배열로 변환하기
a_i8 = a.astype(np.int8)
print(a_i8.dtype)
```

```
float64
int8
```

  - 배열을 생성하면서 dtype으로 자료형을 지정.

02. 넘파이 활용

I. 넘파이의 배열 생성

- 넘파이 배열의 속성
  - 다차원 배열 객체 ndarray는 [그림 6-5]와 같은 속성을 가짐.

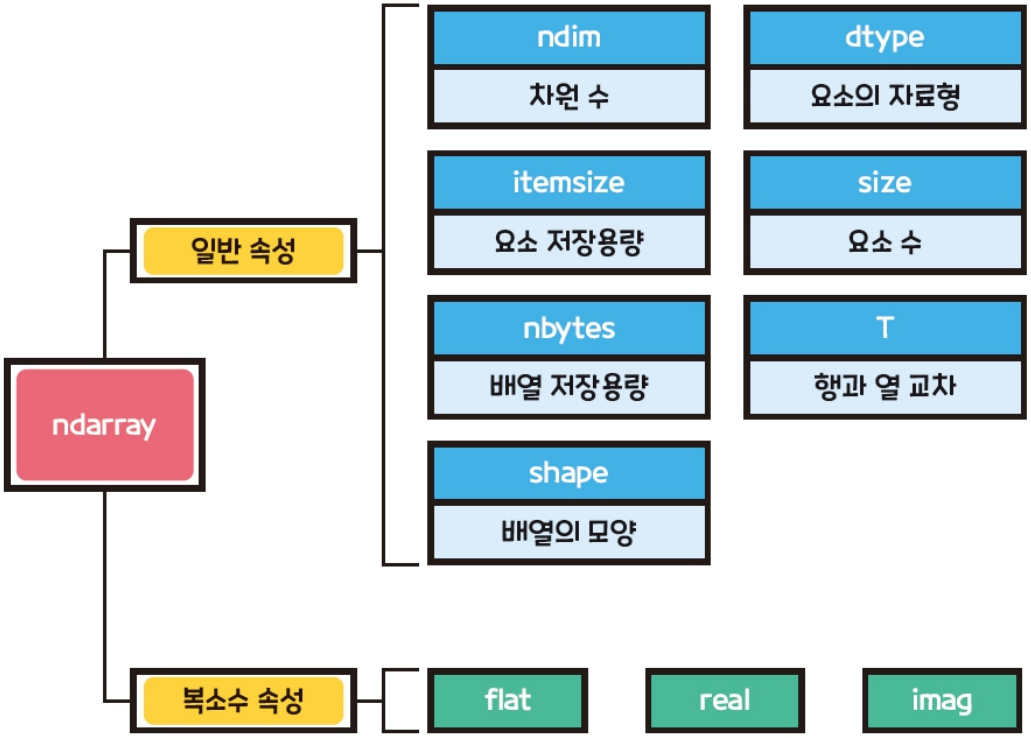


그림 6-5 ndarray의 속성

02. 넘파이 활용

I. 넘파이의 배열 생성

- 넘파이 배열의 속성
  - ndim 속성: 'number of dimensions'의 약자로, 배열의 차원을 나타냄.
  - itemsize 속성: 요소의 바이트 수

    size 속성: 요소의 개수

    nbytes 속성: 배열 전체의 바이트 수.
  - Shape는 배열의 모양을 나타내는 속성.
  - 2차원 배열의 T 속성은 행과 열을 바꾼 교체 배열.
  - 속성을 호출하려면 객체 뒤에 속성 이름을 점으로 연결.

```
객체.속성
```

02. 넘파이 활용

I. 넘파이의 배열 생성

- 넘파이 배열의 속성

[코드 6-4] 배열의 속성 호출

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


print('type(arr):',type(arr))
print('arr.ndim:',arr.ndim)
print('arr.dtype:',arr.dtype)
print('arr.itemsize:',arr.itemsize)
print('arr.size:',arr.size)
print('arr.nbytes:',arr.nbytes)
print('arr.T:\n',arr.T)
print('arr.shape:',arr.shape)
```

```
type(arr): <class
'numpy.ndarray'>
arr.ndim: 2
arr.dtype: int64
arr.itemsize: 8
arr.size: 6
arr.nbytes: 48
arr.T:
 [[0 3]
 [1 4]
 [2 5]]
arr.shape: (2, 3)
```

02. 넘파이 활용

I. 넘파이의 배열 생성

- 배열의 모양
  - 넘파이 배열의 전체 요소 개수를 유지하면서 모양을 변경할 수 있음.

표 6-4 배열 모양을 변경하는 함수

함수	설명
flatten()	1차원 배열로 변경
resize(i, j)	배열의 모양을 i × j 로 변경
transpose()	열과 행을 교차

  - 배열의 shape 속성에 튜플을 할당하여 모양을 지정할 수 있음.

```
객체.shape = (행 크기, 열 크기)
```

02. 넘파이 활용

I. 넘파이의 배열 생성

- 배열의 모양
  - flatten( ) 함수: 다차원 배열을 1차원 배열로 변경.

    resize( ) 함수: 튜플을 입력하면 배열의 모양을 변경.

[코드 6-5] 배열 모양 변경

```
#1차원 배열 생성하기
a = np.arange(8)
print('a\n', a)

#다차원 배열로 변경하기
a.shape = (2,4)
print('shape\n', a)

#1차원 배열로 변경하기
print('flatten\n', a.flatten( ))

#resize 함수로 모양 변경하기
a.resize((4,2))
print('resize\n', a)
```

```
a
 [0 1 2 3 4 5 6 7]
shape
 [[0 1 2 3]
 [4 5 6 7]]
flatten
 [0 1 2 3 4 5 6 7]
resize
 [[0 1]
 [2 3]
 [4 5]
 [6 7]]
```

I. 넘파이의 배열 생성

- 배열의 모양
  - transpose( ) 함수는 속성 T와 마찬가지로 행렬의 행과 열을 교차

[코드 6-6] 행렬의 행과 열 교차

```
a = np.array([[0, 1, 2], [3, 4, 5]])
print('a\n', a)

b = a.transpose( )
print('b\n', b)
c = a.T
print('c\n', c)
```

```
a
 [[0 1 2]
 [3 4 5]]
b
 [[0 3]
 [1 4]
 [2 5]]
c
 [[0 3]
 [1 4]
 [2 5]]
```

II. 넘파이 배열 다루기

- 마스킹
  - 논리값 인덱싱(Boolean indexing) 또는 마스킹(Masking): 조건에 맞는 값을 출력.

[코드 6-7] 배열로 마스킹

```
mask = np.array([0, 1, 1, 0], dtype=bool)
print(mask)

data = np.random.randn(4,2)
print('\ndata 출력\n',data) #랜덤 데이터 배열 data는 책과 다르게 나타남

print('\n마스킹된 데이터 출력\n',data[mask])
print('\n마스킹 역전된 데이터 출력\n',data[~mask])
```

```
[False True True False]

data 출력
[[-1.26749965 1.28790902]
 [ 0.38759648 -2.63253386]
 [-0.15212489 -0.98963632]
 [ 0.4737733 -1.69444807]]

마스킹된 데이터 출력
[[ 0.38759648 -2.63253386]
 [ -0.15212489 -0.98963632]]

마스킹 역전된 데이터 출력
[[-1.26749965 1.28790902]
 [ 0.4737733 -1.69444807]]
```

  - True와 False 대신 1과 0로 마스크 배열 mask를 만듦.
  - 넘파이의 random.randn( ) 함수는 평균이 0이고 표준편차가 1인 난수를 생성.

II. 넘파이 배열 다루기

- 마스킹
  - 조건식 마스킹으로 데이터에서 기준보다 큰 값이나 작은 값을 쉽게 찾음.

[코드 6-8] 조건식으로 마스킹

```
posit = data[data > 0]
print('양수 데이터 출력', '\n',posit)

#다중 조건
over1 = data[1][data[1] > 0]
print('두 번째 행의 양수 데이터 출력','\n',over1)
```

[코드 6-8] 실행 결과

```
양수 데이터 출력
[1.28790902 0.38759648 0.4737733]
두 번째 행의 양수 데이터 출력
[0.38759648]
```

II. 넘파이 배열 다루기

- 유니버설 함수와 브로드캐스팅
  - 유니버설 함수: 배열 요소끼리의 기본 연산을 수행하는 함수.

표 6-5 넘파이 연산 함수

함수	설명	사용법
abs() fabs()	원소의 절댓값을 반환 실수 원소의 절댓값을 반환	numpy.abs(arr) numpy.fabs(arr)
sqrt()	원소의 제곱근을 반환	numpy.sqrt(arr)
square()	원소의 제곱을 반환	numpy.square(arr)
exp()	원소의 지수를 반환	numpy.exp(arr)
log()	원소에 밑이 e인 로그를 취하여 반환	numpy.log(arr)
add()	두 배열 원소의 합을 반환	numpy.add(arr1, arr2)
subtract()	두 배열 원소의 차를 반환	numpy.subtract(arr1, arr2)
multiply()	두 배열 원소의 곱을 반환	numpy.multiply(arr1, arr2)
divide() floor_divide()	두 배열 원소의 나눗셈 결과를 반환 나눗셈의 정수 몫을 반환	numpy.divide(arr1, arr2) numpy.floor_divide(arr1, arr2)
mod()	두 배열 원소 나눗셈의 정수 나머지를 반환	numpy.mod(arr1, arr2)

II. 넘파이 배열 다루기

- 유니버설 함수와 브로드캐스팅
  - 서로 다른 형태의 배열을 유니버설 함수로 연산할 때 브로드캐스팅이 일어남.
  - 브로드캐스팅(Broadcasting): 둘 중 작은 차원인 배열을 변형하여 큰 차원의 배열에 맞춘 다음 두 배열을 요소별로 연산하는 동작.

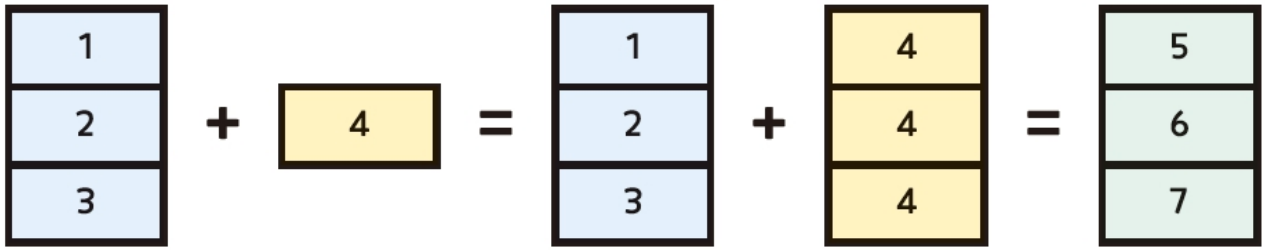


그림 6-6 브로드캐스팅

II. 넘파이 배열 다루기

- 배열 복사
  - 얕은 복사: 등호(=)로 복제. 원본 데이터의 주소가 복사됨.
    깊은 복사: copy( ) 함수를 사용. 배열 복사본을 생성함. 원본을 수정해도 복사본이 바뀌지 않음.

[코드 6-9] 얕은 복사

```
a = np.arange(6)
b = a
print(a)
print(b is a)

b[0] = 10
print(a)
```

```
[0 1 2 3 4 5]
True
[10 1 2 3 4 5]
```

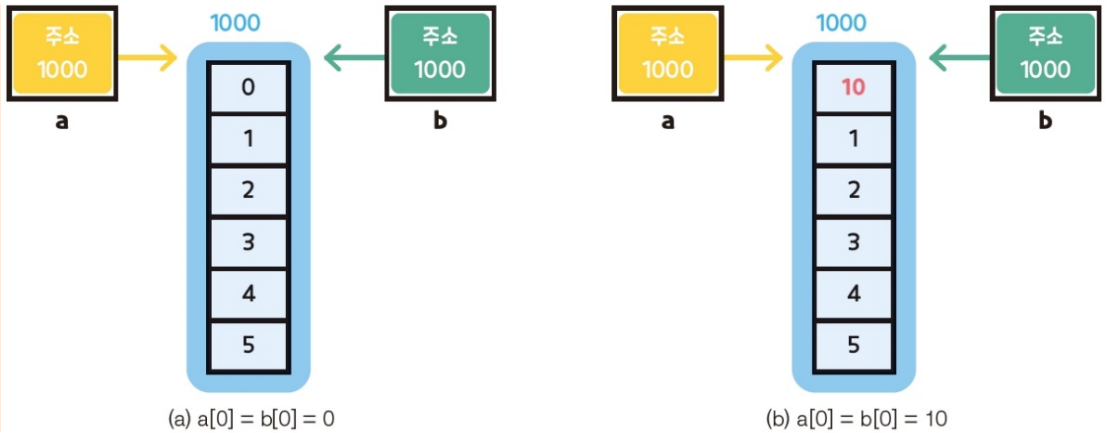


그림 6-7 얕은 복사

  - 복사본인 배열 b의 값만 바꾸어도 원본 배열 a의 요소가 함께 변경됨.

02. 넘파이 활용

II. 넘파이 배열 다루기

- 배열 복사

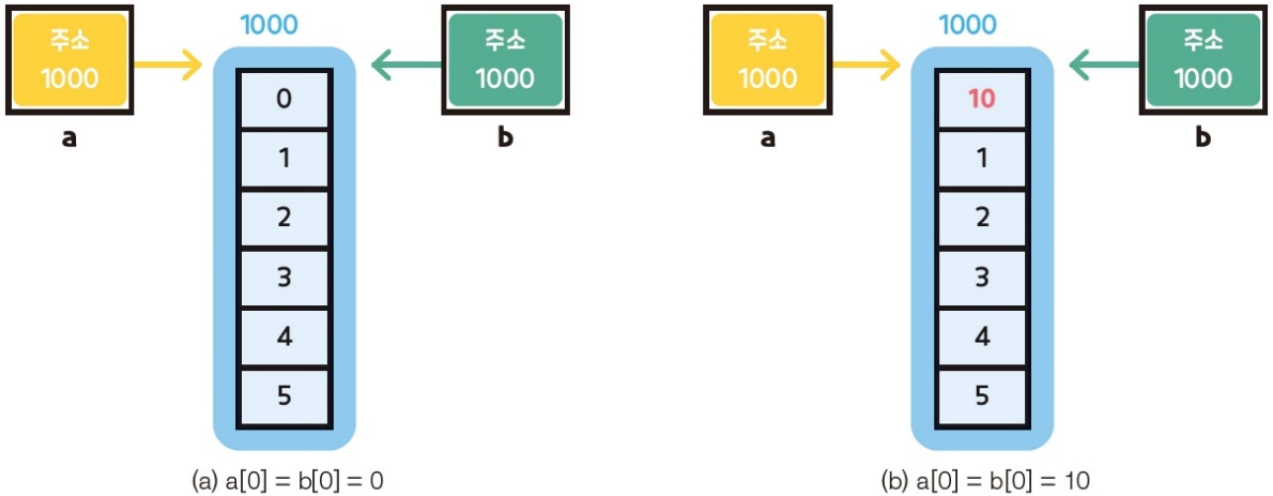


그림 6-7 얕은 복사

- a[0]은 변수 a가 가리키는 배열 중 첫 번째 요소를 의미함.
- a의 값을 b에 얕은 복사로 저장했으므로 b도 같은 배열을 가리키게 됨.
- b[0]에 10을 할당하여 배열 원소를 변경하면 같은 주소를 가리키는 a[0]의 값도 10으로 변경됨.

II. 넘파이 배열 다루기

- 배열 복사

[코드 6-10] 깊은 복사

```
a = np.arange(6)
c = a.copy( )

c[0] = 20
print('A: ', a)
print('C: ', c)
```

[코드 6-10] 실행 결과

```
A: [0 1 2 3 4 5]
C: [20 1 2 3 4 5]
```

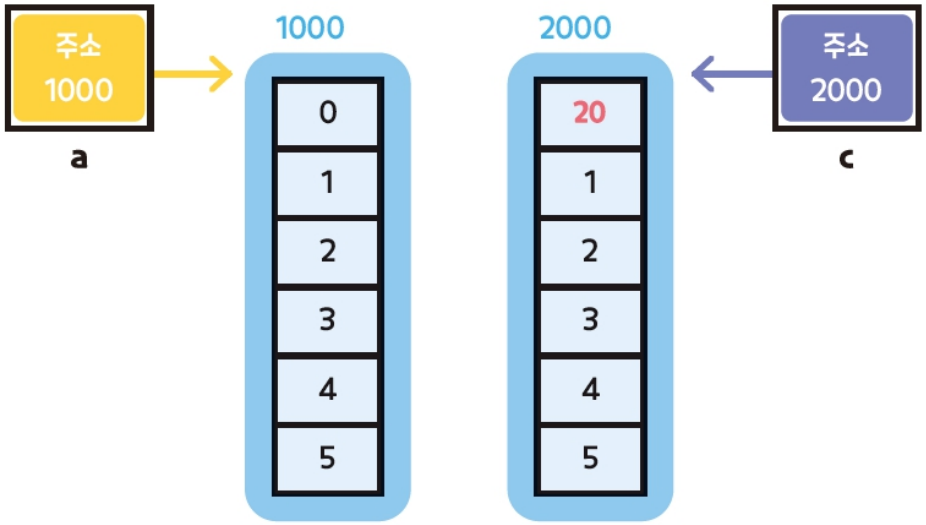


그림 6-8 깊은 복사

– copy( ) 함수를 사용하면 복사본인 배열 c의 값을 수정해도 원본 a에 영향이 없음.

II. 넘파이 배열 다루기

- 배열 복사

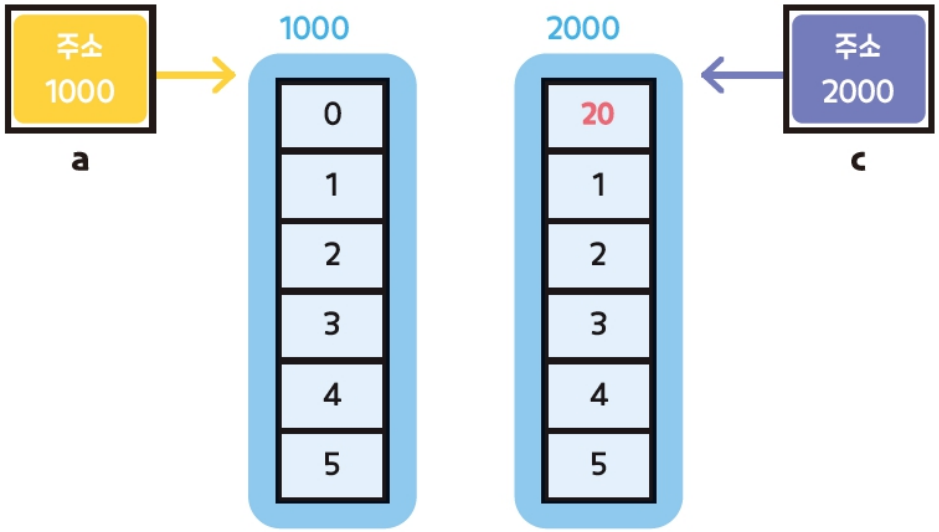


그림 6-8 깊은 복사

– c[0]에 20을 할당하더라도 a가 가리키는 배열과는 별개이므로 a[0]은 그대로 0.

II. 넘파이 배열 다루기

- 배열 정렬

표 6-6 정렬 함수

함수	설명
np.sort(배열)	배열을 정렬해서 반환, 원본을 유지함
배열.sort( )	배열 정렬, 원본을 정렬함
np.argsort(배열)	정렬된 배열의 원래 인덱스를 반환

[코드 6-11] 배열 정렬

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

print('원본\n',a)
print('정렬 후\n',np.sort(a))

print('원본\n',a)
print('정렬한 인덱스\n',np.argsort(a))

a.sort( )
print('원본\n',a)
```

```
원본
[3 2 5 1 4]
정렬 후
[1 2 3 4 5]
원본
[3 2 5 1 4]
정렬한 인덱스
[3 1 0 4 2]
원본
[1 2 3 4 5]
```

– np.sort( ) 함수는 원본 배열 a의 요소 순서가 유지되고, sort( ) 함수는 원본을 정렬

03

판다스 활용

I. 시리즈와 데이터프레임 생성

– 판다스의 Series( ) 함수에 리스트를 입력하여 리스트 데이터로 시리즈를 생성.

[코드 6-12] 리스트로 시리즈 생성

```
import pandas as pd

a = pd.Series([1, 2, 3, 4])
print(a)
b = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
print(b)
```

```
0 1
1 2
2 3
3 4
dtype: int64
a 1
b 2
c 3
dtype: int64
```

– 시리즈 a는 인덱스를 지정하지 않았으므로 기본 인덱스로 숫자 0, 1, 2, 3이 지정됨.
시리즈 b는 값 1, 2, 3에 인덱스로 문자열 'a', 'b', 'c'를 지정.

02. 판다스 활용

I. 시리즈와 데이터프레임 생성

- 이번에는 판다스의 DataFrame( ) 함수로 데이터프레임을 만들기.
- 파이썬의 리스트, 파이썬의 딕셔너리, 넘파이의 배열을 활용.
- 리스트를 만들어 데이터프레임의 데이터로 한 행씩 지정할 수 있음.

```
pd.DataFrame(값, columns=열 이름 리스트)
```

[코드 6-13] 리스트로 데이터프레임 생성

```
list1 = list([['한빛', '남자', '20', '180'],
              ['한결', '남자', '21', '177'],
              ['한라', '여자', '20', '160']])
col_names = ['이름', '성별', '나이', '키']
pd.DataFrame(list1, columns=col_names)
```

	이름	성별	나이	키
0	한빛	남자	20	180
1	한결	남자	21	177
2	한라	여자	22	160

I. 시리즈와 데이터프레임 생성

– 데이터프레임을 생성할 때 딕셔너리를 사용하면
키는 열 이름이 되고 값은 열에 대한 각 행의 데이터이므로 리스트 형식.

```
pd.DataFrame({키1:{인덱스0:값1, 인덱스1:값2}, 키2:{인덱스0:값3, 인덱스1:값4}})
```

[코드 6-14] 딕셔너리로 데이터프레임 생성

```
dict1 = {'이름':{0:'한빛', 1:'한결', 2:'한라'},
         '성별':{0:'남자', 1:'남자', 2:'여자'},
         '나이':{0:'20', 1:'21', 2:'20'},
         '키':{0:'180', 1:'177', 2:'160'}}
pd.DataFrame(dict1)
```

– 실행 결과로 생성되는 데이터프레임은 [코드 6-13]의 결과와 동일함.

I. 시리즈와 데이터프레임 생성

- 실행 결과로 생성되는 데이터프레임은 [코드 6-13]의 결과와 동일
  - 열 이름 리스트를 만들고 columns 옵션으로 데이터프레임의 열 이름으로 지정.

[코드 6-15] 배열로 데이터프레임 생성

```
import numpy as np

#배열
arr1 = np.array([['한빛','남자', '20', '180'],
                 ['한결','남자', '21', '177'],
                 ['한라','여자', '20', '160']])

#열 이름 리스트
col_names = ['이름','성별','나이','키']

#데이터프레임 생성하기
pd.DataFrame(arr1, columns=col_names)
```

  - 실행 결과로 생성되는 데이터프레임은 [코드 6-13]과 동일함.

I. 시리즈와 데이터프레임 생성

- CSV 파일로 데이터프레임 생성
  - 데이터프레임을 CSV로 저장하거나 CSV 파일을 데이터프레임 객체로 읽어옴.

[코드 6-16] CSV 파일로 데이터프레임 생성

```
list1 = list([['허준호','남자','30','183'],
              ['이가원','여자','24','162'],
              ['배규민','남자','23','179'],
              ['고고림','남자','21','182'],
              ['이새봄','여자','28','160'],
              ['이보람','여자','26','163'],
              ['이루리','여자','24','157'],
              ['오다현','여자','24','172']])
col_names = ['이름','성별','나이','키']
df = pd.DataFrame(list1, columns=col_names)
df
```

	이름	성별	나이	키
0	허준호	남자	30	183
1	이가원	여자	24	162
2	배규민	남자	23	179
3	고고림	남자	21	182
4	이새봄	여자	28	160
5	이보람	여자	26	163
6	이루리	여자	24	157
7	오다현	여자	24	172

I. 시리즈와 데이터프레임 생성

- CSV 파일로 데이터프레임 생성

[코드 6-17] 데이터프레임을 CSV 파일로 저장

```
#디렉토리에 csv 파일로 저장하기
df.to_csv('./file.csv', header=True, index=False,
encoding='utf-8')
#CSV 파일 읽기
df2 = pd.read_csv('./file.csv', sep=',')

df2
```

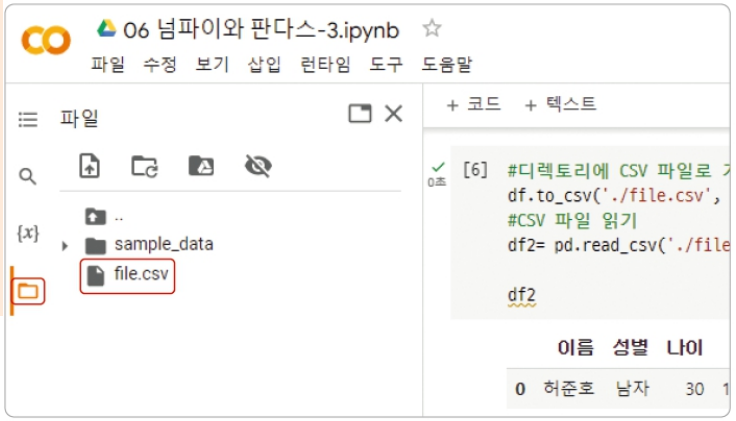


그림 6-9 디렉토리에서 CSV 파일 확인

- Colab 디렉토리를 새로고침하면 'file.csv' 파일이 생성됨.
- 실행 결과로 나타나는 데이터프레임 df2는 [코드 6-16]의 데이터프레임 df와 동일.

I. 시리즈와 데이터프레임 생성

- 열 이름 조회

[코드 6-18] 모든 열 이름 조회

```
df.columns
```

[코드 6-18] 실행 결과

```
Index(['이름', '성별', '나이', '키'], dtype='object')
```

I. 시리즈와 데이터프레임 생성

- 열 이름 조회
  - describe( ) 함수로 열별 값의 개수, 빈도 수와 같은 통계를 확인.

[코드 6-19] 데이터프레임 값 개수와 빈도수 확인

```
df.describe( )
```

[코드 6-19] 실행 결과

	이름	성별	나이	키
count	8	8	8	8
unique	8	2	6	8
top	허준호	여자	24	183
freq	1	5	3	1

  - count는 값의 개수, unique는 유일한 값의 개수, top은 제일 개수가 많은 값이고, freq는 빈도수(frequency)이므로 그 개수를 나타냄.

I. 시리즈와 데이터프레임 생성

- 데이터 미리보기
  - 데이터 미리보기 기능은 데이터의 일부만 보고 열이 어떻게 구성되어 있는지, 값의 범위는 어느 정도인지 확인할 때 사용.
  - head( ) 함수와 tail( ) 함수에 정수 n을 입력하면 각각 데이터프레임의 처음 n개 행, 마지막 n개 행을 반환.

[코드 6-20] 데이터프레임의 처음 세 행 검색

```
df.head(3)
```

	이름	성별	나이	키
0	허준호	남자	30	183
1	이가원	여자	24	162
2	배규민	남자	23	179

I. 시리즈와 데이터프레임 생성

- 데이터 미리보기

[코드 6-21] 데이터프레임의 마지막 다섯 행 검색

```
df.tail( )
```

	이름	성별	나이	키
3	고고림	남자	21	182
4	이새봄	여자	28	160
5	이보람	여자	26	163
6	이루리	여자	24	157
7	오다현	여자	24	172

II. 데이터프레임 데이터 분석

- 행 정렬
  - 인덱스나 특정 열의 값을 기준으로 행을 정렬할 수 있음.
  - 기본 정렬 방향은 오름차순(ascending)이며 내림차순(descending) 정렬은 ascending 인자를 False로 지정.

[코드 6-22] 인덱스 기준으로 정렬

```
df.sort_index(axis=0).head( )
```

[코드 6-22] 실행 결과

	이름	성별	나이	키
0	허준호	남자	30	183
1	이가원	여자	24	162
2	배규민	남자	23	179
3	고고림	남자	21	182
4	이새봄	여자	28	160

II. 데이터프레임 데이터 분석

- 행 정렬
  - sort_values( ) 함수를 이용하여 특정 열을 기준으로 정렬.

[코드 6-23] 열 기준으로 정렬

```
#나이 열과 키 열을 기준으로 행을 정렬하기
df.sort_values(by=['나이', '키'], ascending=False).head( )
```

[코드 6-23] 실행 결과

	이름	성별	나이	키
0	허준호	남자	30	183
4	이새롬	여자	28	160
5	이보람	여자	26	163
7	오다현	여자	24	172
1	이가원	여자	24	162

II. 데이터프레임 데이터 분석

- 데이터 조회
  - 데이터를 열이나 조건식을 기준으로 조회할 수 있음.

```
데이터프레임[조건식]          #시리즈로 출력하기
데이터프레임[[조건식]]        #데이터프레임으로 출력하기
```

  - 조건식 자리에 리스트 형식으로 원하는 열 이름을 적기.

[코드 6-24] 열 이름으로 데이터 조회

```
df[['이름', '키']].head( )
```

	이름	키
0	허준호	183
1	이가원	162
2	배규민	179
3	고고림	182
4	이새봄	160

II. 데이터프레임 데이터 분석

- 데이터 조회
  - iloc 함수를 사용하여 인덱스로 데이터를 조회할 수 있음.

```
df.iloc[행 인덱스, 열 이름]
```

[코드 6-25] 인덱스로 데이터 조회

```
df.iloc[1:4, 0:3]
```

[코드 6-25] 실행 결과

	이름	성별	나이
1	이가원	여자	24
2	배규민	남자	23
3	고고림	남자	21

  - 두 번째 행부터 네 번째 행의 첫 번째 열부터 세 번째 열을 조회.

II. 데이터프레임 데이터 분석

- 데이터 조회
  - 키가 180보다 큰 사람만 조회.

[코드 6-26] 조건식을 만족하는 데이터 조회(1)

```
df[df['키'] > 180].head( )
```

[코드 6-26] 실행 결과

```
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-16-c264ba620ede> in <module>
----> 1 df[df['키'] > 180]
TypeError: '>' not supported between instances of 'str' and 'int'
```

  - 오류는 키 열의 데이터 자료형이 문자열이기 때문.

II. 데이터프레임 데이터 분석

- 데이터 조회

[코드 6-27] 조건식을 만족하는 데이터 조회(2)

```
list1 = list([['허준호', '남자', 30, 183],
              ['이가원', '여자', 24, 162],
              ['배규민', '남자', 23, 179],
              ['고고림', '남자', 21, 182],
              ['이새봄', '여자', 28, 160],
              ['이보람', '여자', 26, 163],
              ['이루리', '여자', 24, 157],
              ['오다현', '여자', 24, 172]])

col_names = ['이름', '성별', '나이', '키']
df = pd.DataFrame(list1, columns=col_names)

df[df['키'] > 180]
```

	이름	성별	나이	키
0	허준호	남자	30	183
3	고고림	남자	21	182

– 키 열의 데이터를 숫자형으로 입력하면 조건식으로 데이터를 조회할 수 있음.

II. 데이터프레임 데이터 분석

- 데이터 조회
  - 조건식에 isin( ) 함수, and 기호(&), or 기호(|), str.contains( ) 함수를 사용하여 더 복잡한 조건으로 조회할 수도 있음.

[코드 6-28] 리스트 요소와 일치하는 데이터 조회

```
df[df['나이'].isin([21, 23])]
```

[코드 6-28] 실행 결과

	이름	성별	나이	키
2	배규민	남자	23	179
3	고고림	남자	21	182

  - 나이가 21세 또는 23세인 사람을 조회.

II. 데이터프레임 데이터 분석

- 데이터 조회

[코드 6-29] 두 조건식을 동시에 만족하는 데이터 조회

```
df[(df['성별'] == '여자') & (df['키'] > 160)]
```

[코드 6-29] 실행 결과

	이름	성별	나이	키
1	이가원	여자	24	162
5	이보람	여자	26	163
7	오다현	여자	24	172

– '여자'이면서 키가 160보다 큰 사람을 조회.

II. 데이터프레임 데이터 분석

- 데이터 조회

[코드 6-30] 두 조건식 중 하나 이상 만족하는 데이터 조회

```
df[(df['나이'] >= 28) | (df['성별'] == '남자')]
```

[코드 6-30] 실행 결과

	이름	성별	나이	키
0	허준호	남자	30	183
2	배규민	남자	23	179
3	고고림	남자	21	182
4	이새봄	여자	28	160

– 28세 이상이거나 '남자'인 사람을 조회.

II. 데이터프레임 데이터 분석

- 데이터 조회

[코드 6-31] 특정 문자열을 포함하는 문자열 데이터 조회

```
df[df['이름'].str.contains('봄')]
```

[코드 6-31] 실행 결과

	이름	성별	나이	키
4	이새봄	여자	28	160

– 이름에 '봄'이 들어간 사람을 조회.

II. 데이터프레임 데이터 분석

- 데이터프레임 통계
  - 나이와 키 데이터가 숫자형인 데이터프레임 df에 describe( ) 함수를 사용.

[코드 6-32] 데이터프레임 통계 확인

```
df.describe( )
```

[코드 6-32] 실행 결과

	나이	키
count	8.000000	8.000000
mean	25.000000	169.750000
std	2.878492	10.552589
min	21.000000	157.000000
25%	23.750000	161.500000
50%	24.000000	167.500000
75%	26.500000	179.750000
max	30.000000	183.000000

II. 데이터프레임 데이터 분석

- 데이터 갱신
  - 행 인덱스나 열 이름으로 데이터프레임 데이터를 조회하고 수정할 수 있음.

[코드 6-33] 인덱스로 조회한 데이터를 수정

```
df.loc[4,'키'] = df.loc[4,'키'] + 5
df.loc[[4]]
```

[코드 6-33] 실행 결과

	이름	성별	나이	키
4	이새봄	여자	28	165

  - 인덱스 4번 행의 키 열 값을 5만큼 증가 시킨다.

II. 데이터프레임 데이터 분석

- 데이터 갱신
  - 반복 연산자(*)를 사용하여 명령을 반복 실행할 수 있음.

[코드 6-34] 반복 연산자 사용

```
df.loc[1:3,'키'] = ['모름'] * 3
df
```

[코드 6-34] 실행 결과

	이름	성별	나이	키
0	허준호	남자	30	183
1	이가원	여자	24	모름
2	배규민	남자	23	모름
3	고고림	남자	21	모름
4	이새봄	여자	28	160
5	이보람	여자	26	163
6	이루리	여자	24	157
7	오다현	여자	24	172

  - 인덱스 1번 행부터 3번 행까지 키 데이터를 '모름'으로 한꺼번에 변경.

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - set_index( ) 함수로 중복 데이터가 없는 이름 열을 행 인덱스로 지정.

[코드 6-35] 인덱스 변경(1)

```
df.set_index('이름', inplace=True)
df.head(3)
```

[코드 6-35] 실행 결과

	성별	나이	키
이름			
허준호	남자	30	183
이가원	여자	24	162
배규민	남자	23	179

  - inplace=True를 지정하면 작업을 데이터프레임 원본에 적용.

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 나이에 10000을 곱한 만큼 보너스.

[코드 6-36] 열 생성

```
df['보너스'] = df['나이'] * 10000
df.head(3)
```

[코드 6-36] 실행 결과

이름	성별	나이	키	보너스
허준호	남자	30	183	300000
이가원	여자	24	162	240000
배규민	남자	23	179	230000

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 추가한 보너스 열을 drop( ) 함수로 삭제.

[코드 6-37] 열 삭제

```
df.drop('보너스', axis=1, inplace=True)
df.head(3)
```

[코드 6-37] 실행 결과

이름	성별	나이	키
허준호	남자	30	183
이가원	여자	24	162
배규민	남자	23	179

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 인덱스를 기본 인덱스로 복구하고 싶다면 reset_index( ) 함수.

[코드 6-38] 인덱스 변경(2)

```
df.reset_index(inplace=True)
df.head(3)
```

[코드 6-38] 실행 결과

	이름	성별	나이	키
0	허준호	남자	30	183
1	이가원	여자	24	162
2	배규민	남자	23	179

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 데이터를 다른 값으로 치환할 때 replace( ) 함수.

[코드 6-39] 데이터 치환

```
rep_cond = {'성별':{'남자':1, '여자':0}}
df2 = df.replace(rep_cond)
df2.head(3)
```

[코드 6-39] 실행 결과

	이름	성별	나이	키
0	허준호	1	30	183
1	이가원	0	24	162
2	배규민	1	23	179

  - 성별이 남자일 때 1, 여자일 때 0으로 치환한 데이터프레임을 df2로 저장.

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 성별을 기준으로 조회하여 키의 평균과 표준편차를 계산.

[코드 6-40] 데이터프레임의 값 연산

```
#성별 평균 키
mean_by_gender = df.groupby(by=['성별'], as_index=False)['키'].mean( )
mean_by_gender.rename(columns={'키':'평균 키'}, inplace=True)

#성별 키의 표준편차
std_by_gender = df.groupby(by=['성별'], as_index=False)['키'].std( )
std_by_gender.rename(columns = {'키':'키의 표준편차'}, inplace=True)

new_df = pd.merge(mean_by_gender, std_by_gender)
new_df
```

	성별	평균 키	키의 표준편차
0	남자	181.333333	2.081666
1	여자	162.800000	5.630275

II. 데이터프레임 데이터 분석

- 데이터프레임 구조 수정
  - 성별을 기준으로 그룹화하고 그룹별 키의 평균을 계산, 계산 결과는 새로운 데이터프레임 mean_by_gender에 할당.
  - rename( ) 함수로 mean_by_gender 데이터프레임에서 키 열의 열 이름을 '평균 키'로 변경.
  - 성별을 기준으로 그룹화하고 그룹별 키의 표준편차를 계산합니다. 계산 결과는 std_by_gender에 할당.
  - 데이터프레임 std_by_gender에서 키 열의 열 이름을 '키의 표준편차'로 변경.
  - 두 데이터프레임 mean_by_gender와 std_by_gender를 병합

LAB. 우수 초콜릿 분석

시중에 판매되는 초콜릿 중에 우수 상품의 특징을 알고 싶습니다. 초콜릿에 1점부터 5점 사이의 평점을 매긴 데이터를 분석해 봅시다.

1. CSV를 읽어 배열로 반환하는 넘파이의 loadtxt( ) 함수를 사용하고 delimiter 인자로 데이터를 읽어올 때 구분 기호를 콤마(,)로 설정.
2. 모든 초콜릿의 평균 평점을 알아보기. 슬라이싱한 넘파이 배열에 통계함수 mean( )을 사용
3. 평점이 4 이상인 우수 초콜릿을 high_level로 저장. 우수 초콜릿의 번호를 정수로 변환
4. 우수 초콜릿의 카카오 함유량 빈도를 분석합니다. 넘파이의 unique( ) 함수는 배열에서 중복 값을 제거하여 남은 값의 배열과 빈도수를 반환
5. 우수 초콜릿 중 가장 빈도수가 큰 카카오 함유량을 구함

LAB. 강의 시간표 분석

다음 표의 데이터로 판다스 데이터프레임을 생성하고 시간표 데이터를 분석해 봅시다.

1. 열 이름과 시간표 데이터를 리스트로 저장.
2. 시간표 데이터를 데이터프레임 객체 df로 변환하여 CSV 파일로 저장
3. 'timetable.csv' 파일을 데이터프레임 객체 df2로 읽고, 열 이름이 '교수'인 열을 추가. 값으로 '김예희', '오정현', '인세훈', '이새봄', '배유진', '이가원'을 저장.
4. 강의실을 기준으로 그룹화하여 max( ) 함수로 최대 시간 수를 구하기.

과목번호	과목명	강의실	시간수
C1	인공지능개론	R1	3
C2	웃음치료	R2	2
C3	경영학	R3	3
C4	3D디자인	R4	4
C5	스포츠경영	R2	2
C6	예술의 세계	R3	1

THANK YOU

Q&A?