

# 데이터프로그래밍 기초 7일차

---

2026-1 DS Bootcamp

부산대학교  
데이터사이언스전문대학원  
석사과정 박민서

---

# CONTENTS

---

1 NumPy

2 Pandas

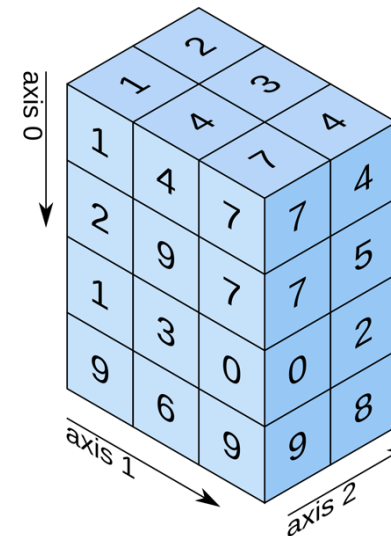
3 Basic Data Analysis

## ■ 넘파이 (NumPy)?

- Python에서 행렬이나 일반적으로 대규모 다차원 배열을 쉽게 처리할 수 있도록 지원하는 라이브러리
- <https://numpy.org/doc/stable/user/quickstart.html>

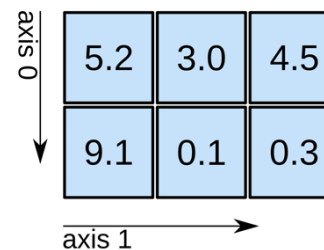


3D array



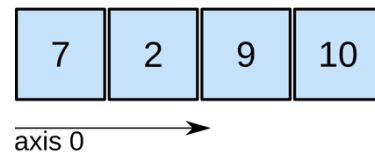
shape: (4, 3, 2)

2D array



shape: (2, 3)

1D array



shape: (4,)

## ■ Install NumPy

- (새 프로젝트에서 설치 시) 새 가상환경을 만든 뒤, numpy 라이브러리 설치
- 가상환경 만드는 법: **conda create -n [가상환경 이름] [python=버전, 선택]**
- numpy 설치 방법: **pip install numpy**

```
Microsoft Windows [Version 10.0.26200.7705]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\minsu\OneDrive\바탕 화면\26-1 BC>conda create -n my_env python=3.13
```

```
C:\Users\minsu\OneDrive\바탕 화면\26-1 BC>conda activate my_env
```

```
(my_env) C:\Users\minsu\OneDrive\바탕 화면\26-1 BC>pip install numpy
```

## ▪ Array in NumPy (ndarray)

- 배열 (array): NumPy에서의 빠른 연산을 위한 자료형

```
import numpy as np # np로 줄여서 많이 사용

my_list = list(range(10))
# my_list = [i for i in range(10)]
my_array = np.arange(10)
# my_array = np.array([i for i in range(10)])

print(my_list)
print(type(my_list))
print(my_array)
print(type(my_array))
```

✓ 0.0s

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
[0 1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
```

## ▪ Array in NumPy (ndarray)

- 배열 (array): NumPy에서의 빠른 연산을 위한 자료형

```
import numpy as np # np로 줄여서 많이 사용

my_list = list(range(10))
# my_list = [i for i in range(10)]
my_array = np.arange(10)
# my_array = np.array([i for i in range(10)])

print(my_list)
print(type(my_list))
print(my_array)
print(type(my_array))
```

✓ 0.0s

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
<class 'list'>
[0 1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
```

## ▪ Array in NumPy (ndarray)

- 브로드 캐스트 (broadcast): Array의 요소마다 연산 적용 → NumPy array와 list의 가장 큰 차이점

```
import numpy as np

my_list = list(range(10))
my_array = np.arange(10)
# 모든 요소에 1을 더하려고 할 때
my_list2 = [i + 1 for i in my_list]
print(f"my_list={my_list}, my_list2={my_list2}")

my_array2 = my_array + 1    # broadcast
print(f"my_array={my_array}, my_array2={my_array2}")

✓ 0.0s

my_list=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], my_list2=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_array=[0 1 2 3 4 5 6 7 8 9], my_array2=[ 1  2  3  4  5  6  7  8  9 10]
```

## ▪ Array in NumPy (ndarray)

- 브로드 캐스트 (broadcast): Array의 요소마다 연산 적용 → NumPy array와 list의 가장 큰 차이점

```
import numpy as np

my_list = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
my_array = np.array([[0, 1, 2], [1, 2, 3], [2, 3, 4]])
# 모든 요소에 1을 더하려고 할 때
my_list2 = [[i + 1 for i in my_list[j]] for j in range(len(my_list))]
print(f"my_list={my_list}\nmy_list2={my_list2}")

my_array2 = my_array + 1    # broadcast
print(f"my_array={my_array}\nmy_array2={my_array2}")
```

✓ 0.0s

```
my_list=[[0, 1, 2], [1, 2, 3], [2, 3, 4]]
my_list2=[[1, 2, 3], [2, 3, 4], [3, 4, 5]]
my_array=[[0 1 2]
 [1 2 3]
 [2 3 4]]
my_array2=[[1 2 3]
 [2 3 4]
 [3 4 5]]
```

## ▪ Array in NumPy (ndarray)

- 브로드 캐스트 (broadcast): Array의 요소마다 연산 적용 → NumPy array와 list의 가장 큰 차이점

```
import numpy as np
# 두 배열(or list)의 연산
# list
my_list = [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
my_list2 = [[1, 3, 5], [2, 4, 6], [3, 6, 9]]
# 결과 저장용
new_list = [[0 for _ in range(len(my_list[0]))] for _ in range(len(my_list))]
for i in range(len(my_list)):
    for j in range(len(my_list[0])):
        new_list[i][j] = my_list[i][j] + my_list2[i][j]

print(new_list)

# np.array
my_array = np.array([[0, 1, 2], [1, 2, 3], [2, 3, 4]])
my_array2 = np.array([[1, 3, 5], [2, 4, 6], [3, 6, 9]])
new_array = my_array + my_array2
print(new_array)
```

✓ 0.0s

```
[[1, 4, 7], [3, 6, 9], [5, 9, 13]]
[[ 1  4  7]
 [ 3  6  9]
 [ 5  9 13]]
```

## ▪ ndarray 기능

- dtype: ndarray에 들어있는 자료들의 type을 결정

```
my_array = np.array([1, 2, 3, 4, 5], dtype=np.int64)    # 64비트 정수형
my_array2 = np.array([1, 2, 3, 4, 5], dtype=np.float64) # 64비트 실수형
print(my_array)
print(my_array2)
```

✓ 0.0s

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

## ▪ ndarray 기능

- Indexing: Array의 요소에 접근 = List와 동일

```
my_array = np.array([1, 2, 3, 4, 5])  
print(my_array[2]) # 3  
print(my_array[1:4]) # [2 3 4]
```

✓ 0.0s

3

[2 3 4]

## ▪ ndarray 기능

- Indexing: Array의 요소에 접근 = List와 동일

```
my_array = np.array([1, 2, 3, 4, 5])  
print(my_array[2]) # 3  
print(my_array[1:4]) # [2 3 4]
```

✓ 0.0s

3

[2 3 4]

## ▪ ndarray 기능

- Boolean Indexing: True / False를 반환하는 조건식을 활용한 특정 요소 추출

```
scores = np.array([55, 78, 82, 90, 67, 88])
mask = (scores >= 80) # [False False True True False True]
print(mask)

high_scores = scores[scores >= 80]
print(high_scores) # [82 90 88]
```

✓ 0.0s

```
[False False  True  True False  True]
[82 90 88]
```

## ▪ ndarray 기능

- Boolean Indexing: True / False를 반환하는 조건식을 활용한 특정 요소 추출

```
A = np.array([
    [60, 75, 90],
    [85, 70, 65],
    [95, 88, 72]
])

print(A[A >= 80]) # numpy에서는 무조건 1차원 배열로 반환
✓ 0.0s

[90 85 95 88]
```

## ▪ ndarray 기능

- `ndim`: Array가 가진 축 개수      `size`: Array의 요소 개수      `shape`: Array의 차원을 나타낸 tuple을 반환

```
my_array = np.array([1, 2, 3, 4, 5])
print(my_array.ndim)      # 1
print(my_array.size)      # 5
print(my_array.shape)     # (5,)
```

✓ 0.0s

```
1
5
(5,)
```

## ▪ ndarray 기능

- `ndim`: Array가 가진 축 개수      `size`: Array의 요소 개수      `shape`: Array의 차원을 나타낸 tuple을 반환

```
my_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(my_array.ndim)    # 2  
print(my_array.size)    # 9  
print(my_array.shape)   # (3, 3)
```

✓ 0.0s

2

9

(3, 3)

## ▪ ndarray 기능

- dot(연산자 @): 두 Array간의 행렬곱 연산

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
B = np.array([[1, 3], [2, 4], [3, 5]])  
print(np.dot(A, B))  
print()  
print(A @ B)
```

✓ 0.0s

```
[[14 26]  
 [32 62]  
 [50 98]]
```

```
[[14 26]  
 [32 62]  
 [50 98]]
```

## ▪ ndarray 기능

- transpose: 수학에서의 전치 행렬(Transposed Matrix) → Array의 행과 열의 값들을 바꿈

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

array_T = my_array.T
print(array_T)
print(array_T.shape)
```

✓ 0.0s

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

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

## ▪ ndarray 기능

- transpose: 수학에서의 전치 행렬(Transposed Matrix)  $\rightarrow$  Array의 행과 열의 값들을 바꿈

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[1, 3, 5], [2, 4, 6]])
print(np.dot(A, B.T))    # 전치를 통한 행렬 곱 연산 가능
```

✓ 0.0s

```
[[ 22  28]
 [ 49  64]
 [ 76 100]]
```

## ▪ ndarray 기능

- zeros, ones: Array의 요소가 모두 0 (혹은 1)로 이루어진 Array 생성

```
print(np.zeros((3, 5)))  
print()  
print(np.ones((3, 5)))
```

✓ 0.0s

```
[[0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0.]]
```

```
[[1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1.]]
```

## ▪ ndarray 기능

- eye: Array의 대각 성분 (행, 열의 위치가 같은 부분)의 값이 1, 나머지는 0인 Array 생성

```
print(np.eye(3, 3)) # np.eye(행, 열)
print()
print(np.eye(3, 5))
print()
print(np.eye(5, 3))
```

✓ 0.0s

```
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]]
```

```
[[1.  0.  0.  0.  0.]
 [0.  1.  0.  0.  0.]
 [0.  0.  1.  0.  0.]
```

```
[[1.  0.  0.]
 [0.  1.  0.]
 [0.  0.  1.]
 [0.  0.  0.]
 [0.  0.  0.]]
```

## ▪ ndarray 기능

- arange: Array에서의 범위 ≡ range( ) 함수와 비슷한 기능

```
print(np.arange(10))      # [0 1 2 3 4 5 6 7 8 9]
print(np.arange(5, 10))   # [5 6 7 8 9]
print(np.arange(1, 10, 2)) # [1 3 5 7 9]
```

✓ 0.0s

```
[0 1 2 3 4 5 6 7 8 9]
[5 6 7 8 9]
[1 3 5 7 9]
```

## ▪ ndarray 기능

- linspace: 시작값과 끝값을 지정한 개수만큼 일정한 간격으로 자른 Array 생성

```
print(np.linspace(1, 10, 5))    # [1.  3.25  5.5  7.75 10.]
print(np.linspace(1, 100, 50))

✓ 0.0s
```

```
[ 1.    3.25  5.5   7.75 10. ]
[  1.          3.02040816  5.04081633  7.06122449  9.08163265
 11.10204082 13.12244898 15.14285714 17.16326531 19.18367347
 21.20408163 23.2244898  25.24489796 27.26530612 29.28571429
 31.30612245 33.32653061 35.34693878 37.36734694 39.3877551
 41.40816327 43.42857143 45.44897959 47.46938776 49.48979592
 51.51020408 53.53061224 55.55102041 57.57142857 59.59183673
 61.6122449  63.63265306 65.65306122 67.67346939 69.69387755
 71.71428571 73.73469388 75.75510204 77.7755102  79.79591837
 81.81632653 83.83673469 85.85714286 87.87755102 89.89795918
 91.91836735 93.93877551 95.95918367 97.97959184 100.          ]
```

## ▪ ndarray 기능

- reshape: Array 차원을 다시 재설정

```
my_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(my_array.shape)
print()
# my_array의 shape을 (2, 6)으로 변경
new_array1 = my_array.reshape((2, 6))
print(new_array1)
print(new_array1.shape)
print()
# 변경할 수 없는 shape을 입력하면 오류 발생
new_array2 = my_array.reshape((5, 3)) # size가 15개인 array였으면
print(new_array2)
```

⊗ 0.0s

```
(4, 3)

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

-----

```
ValueError                                Traceback (most recent call last)
Cell In[32], line 10
      8 print()
      9 # 변경할 수 없는 shape을 입력하면 오류 발생
--> 10 new_array2 = my_array.reshape((5, 3)) # size가 15개인 array였으면
     11 print(new_array2)
```

ValueError: cannot reshape array of size 12 into shape (5,3)

## ▪ ndarray 기능

- reshape: Array 차원을 다시 재설정

```
my_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print(my_array.shape)
print()
# reshape할 차원에 -1 입력 => 차원 수 자동 계산 후 처리
new_array3 = my_array.reshape((3, -1)) # 행은 3개, 열은 자동 계산 (4)
print(new_array3)
# 자동계산이 안 되는 경우 오류 발생
new_array4 = my_array.reshape((5, -1)) # 행은 3개, 열은 자동 계산 (12/5 ?)
print(new_array3)
```

⊗ 0.0s

```
(4, 3)

[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

-----

```
ValueError                                Traceback (most recent call last)
Cell In[34], line 8
      6 print(new_array3)
      7 # 자동계산이 안 되는 경우 오류 발생
----> 8 new_array4 = my_array.reshape((5, -1)) # 행은 3개, 열은 자동 계산 (12/5 ?)
      9 print(new_array3)
```

ValueError: cannot reshape array of size 12 into shape (5,newaxis)

## ▪ ndarray 기능

- flatten: 다차원 Array를 1차원 Array로 재배열

```
scores = np.array([
    [
        [80, 82, 85, 88],
        [70, 75, 78, 80],
        [90, 92, 94, 96],
    ],
    [
        [60, 65, 68, 70], # 학생2 - 과목1
        [88, 85, 83, 82], # 학생2 - 과목2
        [77, 79, 81, 84], # 학생2 - 과목3
    ]
])
```

```
print(scores.shape) # (2, 3, 4)
flat = scores.flatten()
print(flat)
print(flat.shape) # (24,)
```

✓ 0.0s

```
(2, 3, 4)
[80 82 85 88 70 75 78 80 90 92 94 96 60 65 68 70 88 85 83 82 77 79 81 84]
(24,)
```

## ▪ ndarray 기능

- concatenate: Array들을 특정 축을 기준으로 붙일 때, 사용

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

B = np.array([
    [10, 20],
    [30, 40]
])

# 기본은 행 방향으로 concat
C = np.concatenate([A, B]) # [[1 2] [3 4] [10 20] [30 40]]
print(C)
print()

# axis=1 (열) 기준으로 concat
C2 = np.concatenate([A, B], axis = 1) # [[1 2 10 20] [3 4 30 40]]
print(C2)
```

✓ 0.0s

```
[[ 1  2]
 [ 3  4]
 [10 20]
 [30 40]]

[[ 1  2 10 20]
 [ 3  4 30 40]]
```

## ▪ NumPy에서의 무작위성 (Randomness)

- `numpy.random`: NumPy에서의 확률 분포 속 무작위 수를 다루는 모듈

```
# 매 실행마다 결과가 바뀜 (무작위성)
print(np.random.normal(size=5)) # 정규분포에서 5개의 난수 추출
print(np.random.uniform(0, 1, size=5)) # 0~1의 균등분포에서 5개 난수 추출
print(np.random.binomial(5, 0.5, size=5)) # n=5, p=0.5의 이항분포에서 5개 난수 추출
✓ 0.0s

[ 1.97261187 -0.41454827  0.32252423  0.758939    0.92588684]
[0.04802104 0.50826484 0.50993214 0.21245778 0.04853914]
[1 2 3 2 2]
```

## ▪ NumPy에서의 무작위성 (Randomness)

- `numpy.random`: NumPy에서의 확률 분포 속 무작위 수를 다루는 모듈

```
# 시드 고정 (재현성 확보)
np.random.seed(42)
# 이후의 numpy 무작위성은 고정이 됨
print(np.random.normal(size=5))
print(np.random.uniform(0, 1, size=5))
print(np.random.binomial(5, 0.5, size=5))

✓ 0.0s

[ 0.49671415 -0.1382643  0.64768854  1.52302986 -0.23415337]
[0.05808361 0.86617615 0.60111501 0.70807258 0.02058449]
[5 4 2 1 1]
```

---

# Q&A

---

## ■ 판다스 (Pandas)

- Python에서 데이터 분석을 위해 만들어진 라이브러리
- .csv, .xlsx, .hdf 등의 파일들을 처리 후 분석 가능
- 주로 정형 데이터 (Tabular Data)를 처리할 때 사용
- [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/index.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html)



```
C:\Users\minsu\OneDrive\바탕 화면\26-1 BC>conda activate my_env  
(my_env) C:\Users\minsu\OneDrive\바탕 화면\26-1 BC>pip install pandas
```

## ■ 판다스 (Pandas)

- Python에서 데이터 분석을 위해 만들어진 라이브러리
- .csv, .xlsx, .hdf 등의 파일들을 처리 후 분석 가능
- 주로 정형 데이터 (Tabular Data)를 처리할 때 사용



```
!pip install pandas
✓ 15.0s Python

Collecting pandas
  Downloading pandas-3.0.0-cp313-cp313-win_amd64.whl.metadata (19 kB)
Requirement already satisfied: numpy>=1.26.0 in c:\users\minsu\anaconda3\envs\my_env\lib\site-packages
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\minsu\anaconda3\envs\my_env\lib\site
Collecting tzdata (from pandas)
  Downloading tzdata-2025.3-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in c:\users\minsu\anaconda3\envs\my_env\lib\site-packages (fro
Downloading pandas-3.0.0-cp313-cp313-win_amd64.whl (9.7 MB)
----- 0.0/9.7 MB ? eta -:-:--
----- 2.1/9.7 MB 11.8 MB/s eta 0:00:01
----- 4.7/9.7 MB 11.8 MB/s eta 0:00:01
----- 7.1/9.7 MB 11.8 MB/s eta 0:00:01
----- 9.4/9.7 MB 11.8 MB/s eta 0:00:01
----- 9.7/9.7 MB 11.3 MB/s 0:00:00
Downloading tzdata-2025.3-py2.py3-none-any.whl (348 kB)
Installing collected packages: tzdata, pandas
----- 0/2 [tzdata]
----- 0/2 [tzdata]
----- 0/2 [tzdata]
----- 1/2 [pandas]
----- 1/2 [pandas]
----- 1/2 [pandas]
----- 1/2 [pandas]
----- 1/2 [pandas]
...
----- 1/2 [pandas]
----- 2/2 [pandas]

Successfully installed pandas-3.0.0 tzdata-2025.3
```

## ■ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
import numpy as np
import pandas as pd

# ndarray
arr = np.arange(100, 105)
print(f"ndarray: {arr}")
print()
# Series
s = pd.Series(arr)
print("[Series]")
print(s)
```

✓ 0.0s

ndarray: [100 101 102 103 104]

[Series]

0	100
1	101
2	102
3	103
4	104

dtype: int64

## ▪ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
# 다양한 타입의 데이터를 Series로 생성 -> object type으로 저장  
s = pd.Series([91, 2.5, '스포츠', 4, 5.16])  
print(s)
```

✓ 0.0s

```
0    91  
1    2.5  
2   스포츠  
3     4  
4   5.16  
dtype: object
```

## ▪ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
scores = pd.Series(  
    [78, 85, 92, 67, 88],  
    index=["A", "B", "C", "D", "E"] # 원하는 Index로도 지정 가능  
)  
print(scores)
```

✓ 0.0s

A	78
B	85
C	92
D	67
E	88

dtype: int64

## ▪ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
scores = pd.Series(  
    [78, 85, 92, 67, 88],  
    index=["A", "B", "C", "D", "E"],  
    dtype=np.float64    # 원하는 자료형으로도 지정 가능  
)  
print(scores)
```

✓ 0.0s

A	78.0
B	85.0
C	92.0
D	67.0
E	88.0

dtype: float64

## ■ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
scores = pd.Series(  
    [78, 85, 92, 67, 88],  
    index=["A", "B", "C", "D", "E"],  
    dtype=np.float64    # 원하는 자료형으로도 지정 가능  
)  
print(scores)  
print(scores.values) # 값만 따로 추출 가능  
print(type(scores.values))  
✓ 0.0s  
A    78.0  
B    85.0  
C    92.0  
D    67.0  
E    88.0  
dtype: float64  
[78. 85. 92. 67. 88.]  
<class 'numpy.ndarray'>
```

## ■ Pandas 기초

- Series: Array + Index + dtype의 자료구조

```
# Boolean Indexing
scores = pd.Series(
    [78, 85, 92, 67, 88],
    index=["A", "B", "C", "D", "E"],
    dtype=np.float64    # 원하는 자료형으로도 지정 가능
)
mask = (scores >= 80)
print(mask)

print(scores[mask])
```

✓ 0.0s

A	False
B	True
C	True
D	False
E	True

dtype: bool

B	85.0
C	92.0
E	88.0

dtype: float64

## ▪ Pandas 기초

- DataFrame: 2차원 데이터 구조 (행, 열이 존재 / Excel 데이터와 똑같은 구조)

```
df = pd.DataFrame(  
    [[1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9]]  
)  
print(df)  
print(type(df))  
df # Jupyter Notebook에서 간단한 출력 가능
```

✓ 0.0s

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

<class 'pandas.DataFrame'>

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

## ▪ Pandas 기초

- DataFrame: 2차원 데이터 구조 (행, 열이 존재 / Excel 데이터와 똑같은 구조)

```
df = pd.DataFrame(  
    [[1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9]]  
    , columns = ["col1", "col2", "col3"] # column 이름 지정 가능  
)
```

df

✓ 0.0s

	col1	col2	col3
0	1	2	3
1	4	5	6
2	7	8	9

## ▪ Pandas 기초

- DataFrame: 2차원 데이터 구조 (행, 열이 존재 / Excel 데이터와 똑같은 구조)

```
df = pd.DataFrame(  
    [[1, 2, 3],  
     [4, 5, 6],  
     [7, 8, 9]]  
    , columns = ["col1", "col2", "col3"]  
    , index = ["idx1", "idx2", "idx3"] # 인덱스 custom 가능  
)  
  
df  
✓ 0.0s
```

	col1	col2	col3
idx1	1	2	3
idx2	4	5	6
idx3	7	8	9

## ▪ Pandas 기초

- DataFrame: 2차원 데이터 구조 (행, 열이 존재 / Excel 데이터와 똑같은 구조)

```
# 딕셔너리 자료도 DataFrame으로 변환 가능
data = {
    'name': ['Kim', 'Lee', 'Park'],
    'age': [24, 27, 34],
    'children': [2, 1, 3]
}

df = pd.DataFrame(data)
df
```

✓ 0.0s

	name	age	children
0	Kim	24	2
1	Lee	27	1
2	Park	34	3

## ▪ Pandas 기초

- DataFrame: 2차원 데이터 구조 (행, 열이 존재 / Excel 데이터와 똑같은 구조)

```
# 딕셔너리 자료도 DataFrame으로 변환 가능
data = {
    'name': ['Kim', 'Lee', 'Park'],
    'age': [24, 27, 34],
    'children': [2, 1, 3]
}

df = pd.DataFrame(data)
df
```

✓ 0.0s

	name	age	children
0	Kim	24	2
1	Lee	27	1
2	Park	34	3

## ■ 데이터 분석 기초 with Pandas

- 데이터 불러오기: .csv, .xlsx 등을 open( ) 대신 read\_\*\*\*() 함수로 불러옴
- DataFrame이란 객체로 저장

```
# Iris 데이터셋 불러오기
df = pd.read_csv("Iris.csv") # 상대경로 "./Iris.csv"도 가능
df
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns



## ■ 데이터 분석 기초 with Pandas

- 열 이름 추출

```
# Column 이름 추출
print(df.columns)
df
```

✓ 0.0s

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
      'Species'],
      dtype='str')
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

## ■ 데이터 분석 기초 with Pandas

- 특정 열 추출

```
# 단일 열 추출 -> Series 반환
df["SepalLengthCm"]
✓ 0.0s
```

0	5.1
1	4.9
2	4.7
3	4.6
4	5.0
...	...
145	6.7
146	6.3
147	6.5
148	6.2
149	5.9

Name: SepalLengthCm, Length: 150, dtype: float64

```
# 단일 열 추출 -> Series 반환
df[["SepalLengthCm"]]
✓ 0.0s
```

	SepalLengthCm
0	5.1
1	4.9
2	4.7
3	4.6
4	5.0
...	...
145	6.7
146	6.3
147	6.5
148	6.2
149	5.9

150 rows × 1 columns

```
# 복수 열 추출 -> DataFrame
df[["SepalLengthCm", "SepalWidthCm"]]
✓ 0.0s
```

	SepalLengthCm	SepalWidthCm
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6
...	...	...
145	6.7	3.0
146	6.3	2.5
147	6.5	3.0
148	6.2	3.4
149	5.9	3.0

150 rows × 2 columns

## ■ 데이터 분석 기초 with Pandas

- `head()`, `tail()`: DataFrame의 맨 앞, 뒤의 지정된 개수의 행만 출력 (기본 5개)
- 자료가 어떻게 생겼는지 빠르게 확인하고 싶을 때 사용

```
df.head() # df의 맨 앞 5개의 행 출력
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `head()`, `tail()`: DataFrame의 맨 앞, 뒤의 지정된 개수의 행만 출력 (기본 5개)
- 자료가 어떻게 생겼는지 빠르게 확인하고 싶을 때 사용

```
df.tail() # df의 맨 뒤 5개의 행 출력
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

## ■ 데이터 분석 기초 with Pandas

- `head()`, `tail()`: DataFrame의 맨 앞, 뒤의 지정된 개수의 행만 출력 (기본 5개)
- 자료가 어떻게 생겼는지 빠르게 확인하고 싶을 때 사용

```
df.head(3) # df의 맨 앞 3개의 행 출력
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `head()`, `tail()`: DataFrame의 맨 앞, 뒤의 지정된 개수의 행만 출력 (기본 5개)
- 자료가 어떻게 생겼는지 빠르게 확인하고 싶을 때 사용

```
df.head(3) # df의 맨 앞 3개의 행 출력
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `head()`, `tail()`: DataFrame의 맨 앞, 뒤의 지정된 개수의 행만 출력 (기본 5개)
- 자료가 어떻게 생겼는지 빠르게 확인하고 싶을 때 사용

```
df.head(3) # df의 맨 앞 3개의 행 출력
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `info()`: DataFrame의 컬럼별 정보 출력
- 데이터의 개수, 데이터 타입, 결측치(Missing Value, NaN Value) 확인 시 사용

```
<class 'pandas.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Id              150 non-null   int64
1   SepalLengthCm   150 non-null   float64
2   SepalWidthCm    150 non-null   float64
3   PetalLengthCm   150 non-null   float64
4   PetalWidthCm    150 non-null   float64
5   Species         150 non-null   str
dtypes: float64(4), int64(1), str(1)
memory usage: 7.2 KB
```

## ■ 데이터 분석 기초 with Pandas

- `value_counts()`: DataFrame의 Column별 값의 분포 확인 시 사용
- Labeling이 된, 범주형 자료의 분포를 확인할 때 주로 사용

```
# Species의 분포 확인  
df["Species"].value_counts()
```

✓ 0.0s

```
Species  
Iris-setosa      50  
Iris-versicolor  50  
Iris-virginica   50  
Name: count, dtype: int64
```

## ■ 데이터 분석 기초 with Pandas

- DataFrame의 dtype: int, float, (str), object, category
- `astype()`: DataFrame의 열의 자료형을 바꿀 때 사용

```
# df의 "Id"열  
df["Id"]  
✓ 0.0s
```

0	1
1	2
2	3
3	4
4	5
...	
145	146
146	147
147	148
148	149
149	150

Name: Id, Length: 150, dtype: int64

```
# ID를 float로 변환  
df["Id"].astype(float)  
# 변환을 적용하려면  
# df["Id"] = df["Id"].astype(float)  
✓ 0.0s
```

0	1.0
1	2.0
2	3.0
3	4.0
4	5.0
...	
145	146.0
146	147.0
147	148.0
148	149.0
149	150.0

Name: Id, Length: 150, dtype: float64

## ■ 데이터 분석 기초 with Pandas

- DataFrame의 dtype: int, float, (str), object, category
- `astype()`: DataFrame의 열의 자료형을 바꿀 때 사용

```
# df의 "Id"열
df["Id"]
✓ 0.0s
```

0	1
1	2
2	3
3	4
4	5
...	
145	146
146	147
147	148
148	149
149	150

Name: Id, Length: 150, dtype: int64

```
# ID를 object로 변환
df["Id"].astype(object)
# 변환을 적용하려면
# df["Id"] = df["Id"].astype(object)
✓ 0.0s
```

0	1
1	2
2	3
3	4
4	5
...	
145	146
146	147
147	148
148	149
149	150

Name: Id, Length: 150, dtype: object

## ■ 데이터 분석 기초 with Pandas

- DataFrame의 dtype: int, float, (str), object, category
- astype(): DataFrame의 열의 자료형을 바꿀 때 사용

```
# df의 "Id"열
df["Id"]
✓ 0.0s
```

0	1
1	2
2	3
3	4
4	5
...	
145	146
146	147
147	148
148	149
149	150

Name: Id, Length: 150, dtype: int64

```
# ID를 Category로 변환
df["Id"].astype('category')
# 변환을 적용하려면
# df["Id"] = df["Id"].astype('category')
```

```
✓ 0.0s
```

0	1
1	2
2	3
3	4
4	5
...	
145	146
146	147
147	148
148	149
149	150

Name: Id, Length: 150, dtype: category  
Categories (150, int64): [1, 2, 3, 4, ..., 147, 148, 149, 150]

## ■ 데이터 분석 기초 with Pandas

- `sort_index( )`: Index 기준으로 데이터 행 정렬
- `sort_values( )`: 특정 값 기준으로 데이터 행 정렬

```
# Index 기준 내림차순으로 정렬 -> 5개 행 출력  
df.sort_index(ascending=False).head(5)
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
149	150	5.9	3.0	5.1	1.8	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica

## ■ 데이터 분석 기초 with Pandas

- `sort_index()`: Index 기준으로 데이터 행 정렬
- `sort_values()`: 특정 값 기준으로 데이터 행 정렬

```
# SepalLengthCm 열의 값 기준으로 정렬  
df.sort_values(by='SepalLengthCm').head()  
✓ 0.0s
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `sort_index()`: Index 기준으로 데이터 행 정렬
- `sort_values()`: 특정 값 기준으로 데이터 행 정렬

```
# SepalLengthCm 열의 값 + SepalWidthCm 열의 값 기준으로 정렬
df.sort_values(by=['SepalLengthCm', 'SepalWidthCm']).head()
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa

## ■ 데이터 분석 기초 with Pandas

- `.loc`['인덱스 값', '열 이름']: 인덱스의 값, 열 이름을 통한 특정 값 접근
- `.iloc`['행 인덱스', '열 인덱스']: 자료의 행, 열의 인덱스를 통한 특정 값 접근

```
# .loc을 통한 접근
df.loc[23, 'SepalLengthCm'] # Index 23, SepalLengthCm열에 저장된 값
✓ 0.0s
np.float64(5.1)
```

```
#.iloc을 통한 접근
df.iloc[23, 1] # 23번째 행, 1번째 열에 저장된 값
✓ 0.0s
np.float64(5.1)
```

## ■ 데이터 분석 기초 with Pandas

- 조건부 인덱싱: 특정 조건에 맞는 값, DataFrame 반환

```
# df[조건 넣는 방식]
df[df["SepalLengthCm"] < 5]
✓ 0.0s
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
22	23	4.6	3.6	1.0	0.2	Iris-setosa
24	25	4.8	3.4	1.9	0.2	Iris-setosa
29	30	4.7	3.2	1.6	0.2	Iris-setosa
30	31	4.8	3.1	1.6	0.2	Iris-setosa
34	35	4.9	3.1	1.5	0.1	Iris-setosa
37	38	4.9	3.1	1.5	0.1	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
45	46	4.8	3.0	1.4	0.3	Iris-setosa
47	48	4.6	3.2	1.4	0.2	Iris-setosa
57	58	4.9	2.4	3.3	1.0	Iris-versicolor
106	107	4.9	2.5	4.5	1.7	Iris-virginica

## ■ 데이터 분석 기초 with Pandas

- 조건부 인덱싱: 특정 조건에 맞는 값, DataFrame 반환
- `.isin( )`: 특정 값의 포함 여부를 반환

```
df[df["Species"].isin(["Iris-setosa", "Iris-versicolor"])]
```

✓ 0.0s

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
95	96	5.7	3.0	4.2	1.2	Iris-versicolor
96	97	5.7	2.9	4.2	1.3	Iris-versicolor
97	98	6.2	2.9	4.3	1.3	Iris-versicolor
98	99	5.1	2.5	3.0	1.1	Iris-versicolor
99	100	5.7	2.8	4.1	1.3	Iris-versicolor

100 rows × 6 columns

## ■ 데이터 분석 기초 with Pandas

- `.describe()`: DataFrame의 수치 데이터에 대한 기초 통계량 제시

```
# 기초통계량 요약 제시
df.describe()
✓ 0.0s
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

---

# Q&A

---