

빅데이터 처리

-Big Data Processing



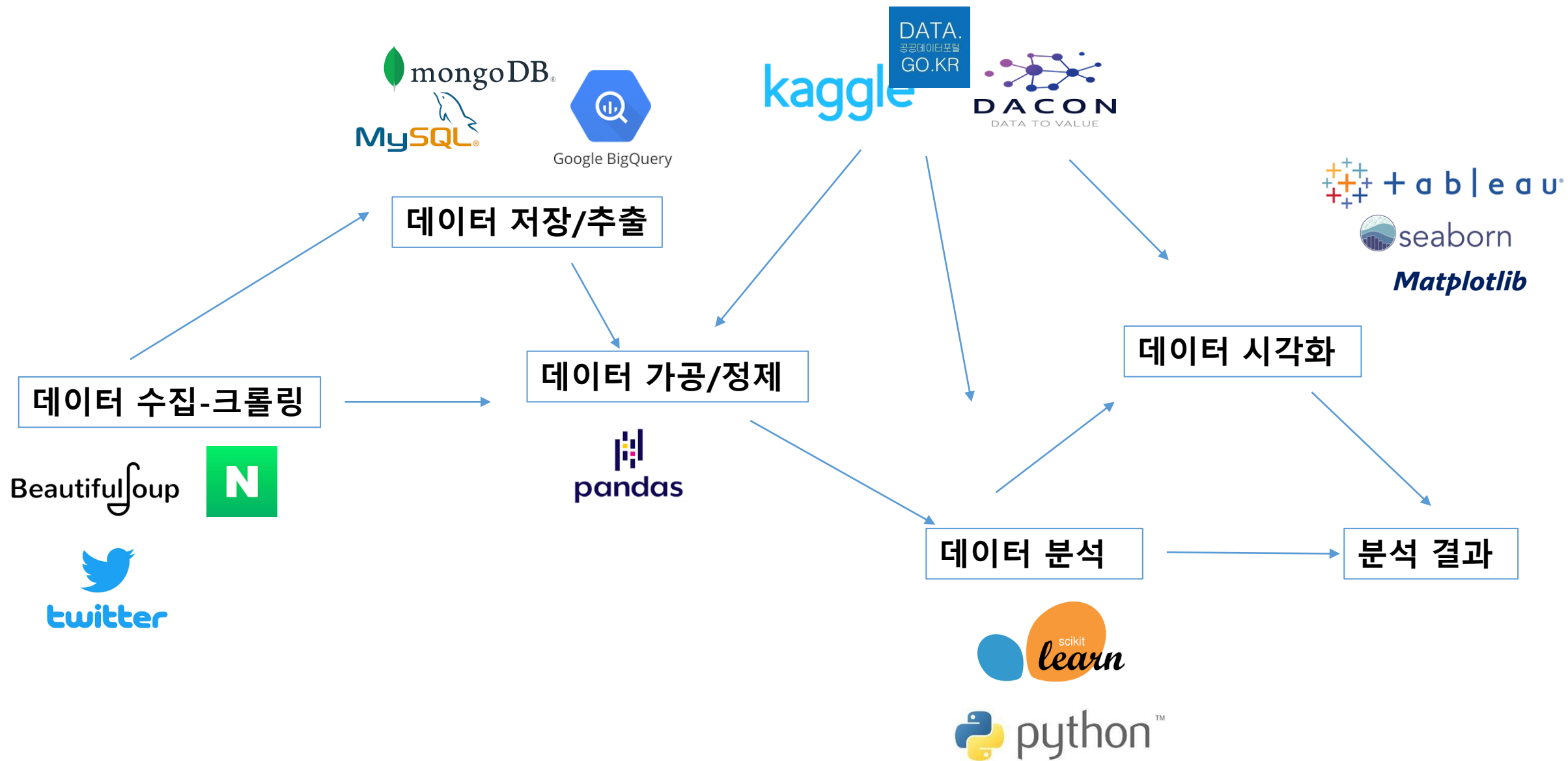
4주차

- 판다스 입문, 데이터 입출력 복습
- 판다스 데이터 살펴보기
- 판다스 데이터 사전 처리
 - 누락 데이터 처리
 - 중복 데이터 처리

- 9월 마지막 주
- PPT 1-2 장
 - 프로젝트 제목
 - 사용하는 데이터
 - 데이터 수집 방법
 - 분석 할 내용
 - 프로젝트 범위 (4p 참조)

빅데이터 처리

인하공전 컴퓨터 정보 과



-
1. 데이터과학자가 판다스를 배우는 이유
 2. 판다스 자료구조
 - 2-1. 시리즈
 - 2-2. 데이터프레임
 3. 인덱스 활용
 4. 산술 연산
 - 4-1. 시리즈 연산
 - 4-2. 데이터프레임 연산

파이썬, 머신러닝, 판다스 데이터 분석

<https://github.com/Castlegus/python-machine-learning-pandas-data-analysis>

1. 데이터과학자가 판다스를 배우는 이유

빅데이터의 시대. 데이터 과학이라는 새로운 영역의 출현.

- 클라우드 컴퓨팅의 확산. 빅데이터 저장, 분석에 필요한 컴퓨팅 자원이 매우 저렴해짐.
- 컴퓨팅 파워의 대중화는 최적의 학습환경과 연구 인프라를 제공.

데이터과학은 데이터를 연구하는 분야이고, 데이터 자체가 가장 중요한 자원

- 데이터 분석 업무의 80~90%는 데이터를 수집하고 정리하는 일이 차지.
- 나머지 10~20%는 알고리즘을 선택하고, 모델링 결과를 분석하여 데이터로부터 유용한 정보(information)을 뽑아내는 분석 프로세스의 몫.
- 데이터과학자가 하는 가장 중요한 일이 데이터를 수집하고 분석이 가능한 형태로 정리하는 것.

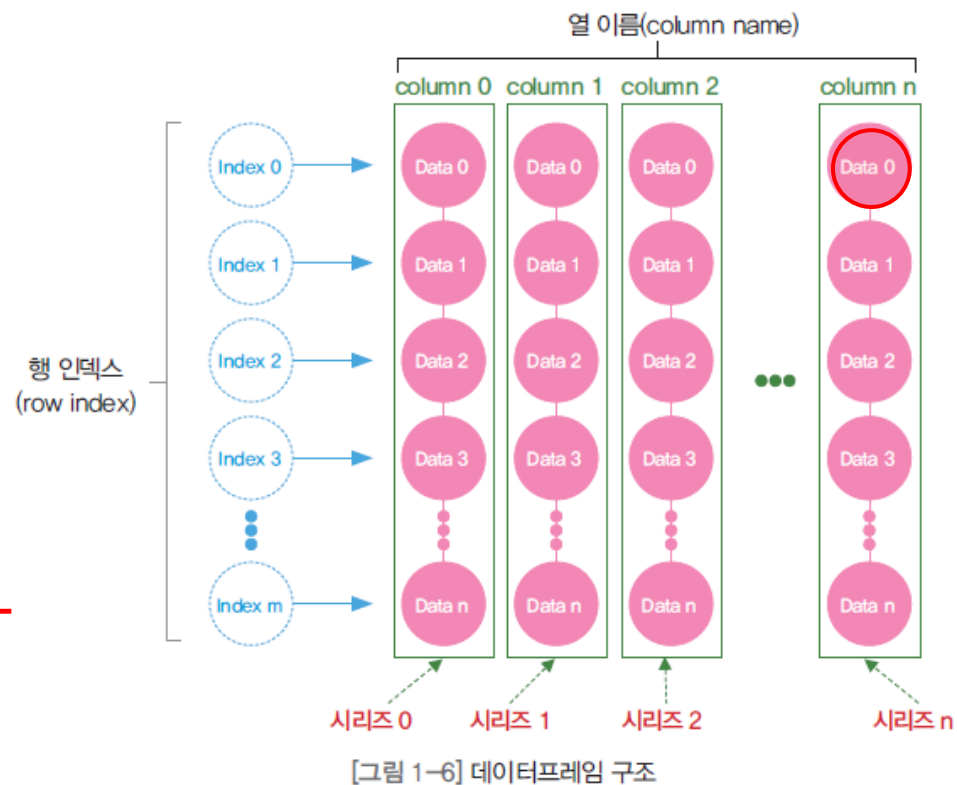
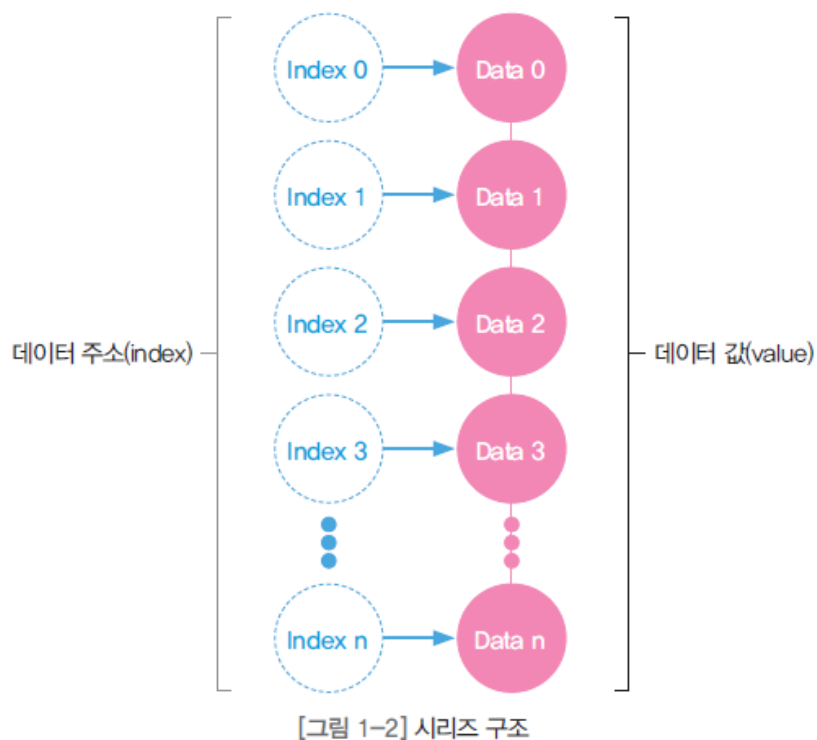
판다스는 데이터를 수집하고 정리하는데 최적화된 도구.

- 가장 배우기 쉬운 프로그래밍 언어, 파이썬(Python) 기반.
- 오픈소스(open source)로 무료로 이용 가능.



[그림 1-1] 판다스 공식 홈페이지(<http://pandas.pydata.org/>)

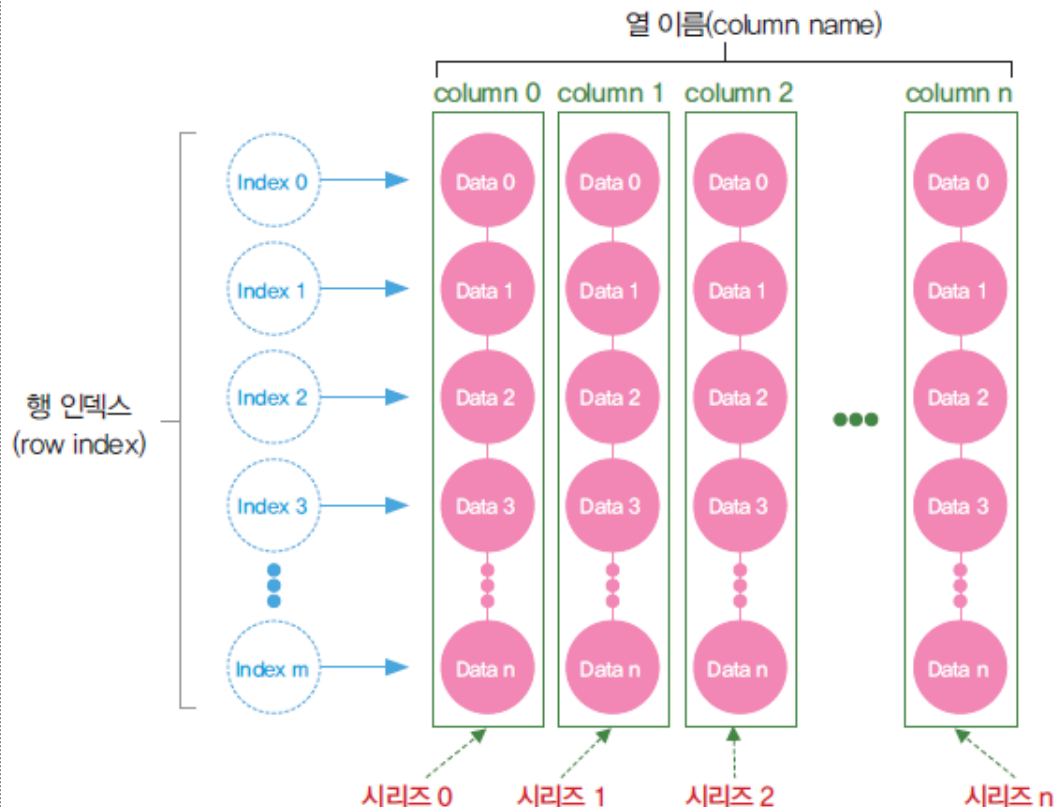
2. 판다스 자료구조



2-2. 데이터프레임

• 개요

- 1) 데이터프레임은 2차원 배열. R의 데이터프레임에서 유래.
- 2) 데이터프레임의 열은 시리즈 객체. 시리즈를 열벡터(vector)라고 하면, 데이터프레임은 여러 개의 열벡터들이 같은 행 인덱스를 기준으로 줄지어 결합된 2차원 벡터 또는 행렬(matrix).
- 3) 데이터프레임은 행과 열을 나타내기 위해 두 가지 종류의 주소를 사용. 행 인덱스(row index)와 열 이름(column name 또는 column label)으로 구분.



[그림 1-6] 데이터프레임 구조

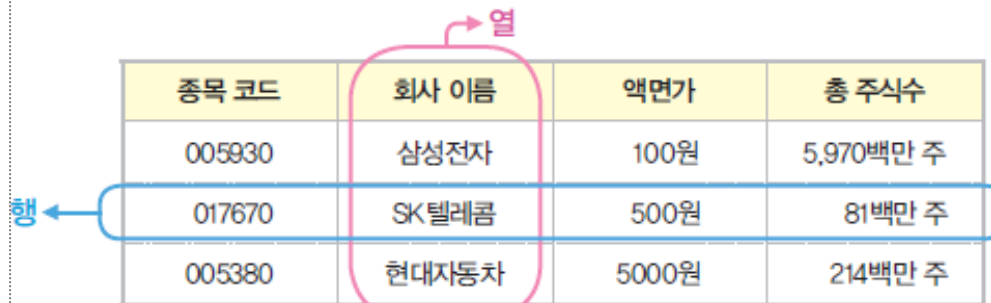
2-2. 데이터프레임

- 4) 데이터프레임의 각 열은 공통의 속성을 갖는 일련의 데이터를 나타냄.
- 5) 각 행은 개별 관측대상에 대한 다양한 속성 데이터들의 모음인 레코드(record).

[예시]

다음 주식종목 리스트에서, 각 행은 하나의 주식종목에 관한 관측값(observation)을 나타낸다.

각 열은 종목코드, 회사이름, 액면가, 총주식수 등 공통의 속성이거나 범주를 나타내는데, 보통 변수(variable)로 활용된다



종목 코드	회사 이름	액면가	총 주식수
005930	삼성전자	100원	5,970백만 주
017670	SK텔레콤	500원	81백만 주
005380	현대자동차	5000원	214백만 주

[표 1-1] 주식 종목 리스트

2-2. 데이터프레임

• 데이터프레임 만들기

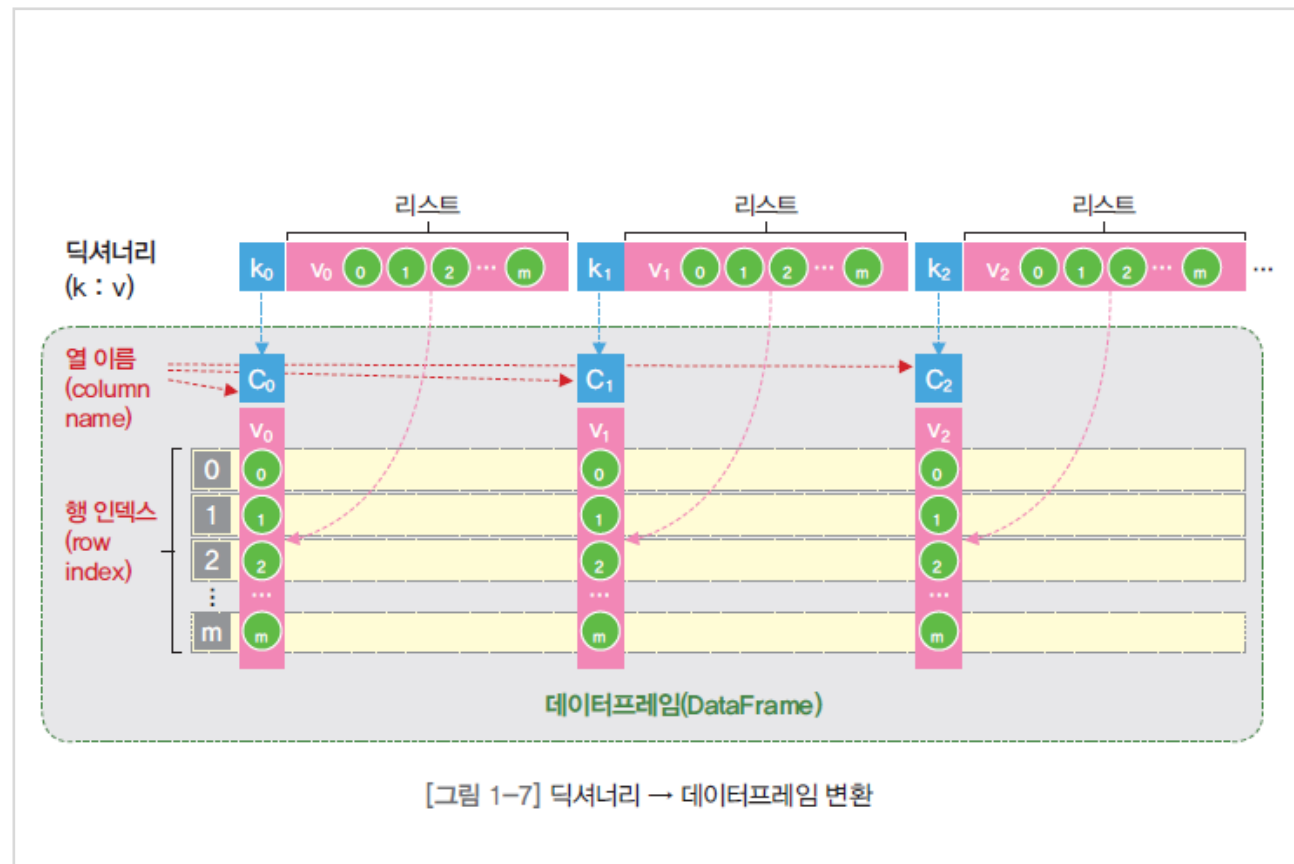
1) 같은 길이(원소의 개수가 동일한)의 배열 여러 개가 필요. 데이터프레임은 여러 개의 시리즈(열, column)를 모아 놓은 집합.

2) 판다스 DataFrame() 함수를 사용. 여러 개의 리스트를 원소로 갖는 딕셔너리를 함수에 전달하는 방식을 주로 활용.

딕셔너리 → 데이터프레임 변환: `pandas.DataFrame(딕셔너리 객체)`

3) 딕셔너리의 값(v)에 해당하는 각 리스트가 시리즈로 변환되어 데이터프레임의 각 열이 된다.

4) 딕셔너리의 키(k)는 각 시리즈의 이름으로 변환되어, 최종적으로 데이터프레임의 열 이름이 된다.



2-2. 데이터프레임

예제 1-4

원소 3개씩 담고 있는 리스트를 5개 만든다. 이들 5개의 리스트를 원소로 갖는 딕셔너리를 정의하고, 판다스 DataFrame() 함수에 전달하면 5개의 열을 갖는 데이터프레임을 만든다.

이때, 딕셔너리의 키(k)가 열 이름(c0 ~ c4)이 되고, 값(v)에 해당하는 각 리스트가 데이터프레임의 열이 된다. 행 인덱스에는 정수형 위치 인덱스(0, 1, 2)가 자동 지정된다.

〈예제 1-4〉 딕셔너리 → 데이터프레임 변환

(File: example/part1/1.4_dict_to_dataframe.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # 열이름을 key로 하고, 리스트를 value로 갖는 딕셔너리 정의 (2차원 배열)
6  dict_data = {'c0':[1,2,3], 'c1':[4,5,6], 'c2':[7,8,9], 'c3':[10,11,12], 'c4':[13,14,15]}
7
8  # 판다스 DataFrame() 함수로 딕셔너리를 데이터프레임으로 변환. 변수 df에 저장
9  df = pd.DataFrame(dict_data)
10
11 # df의 자료형 출력
12 print(type(df))
13 print('\n')
14 # 변수 df에 저장되어 있는 데이터프레임 객체를 출력
15 print(df)
```

〈실행 결과〉 코드 전부 실행

```
<class 'pandas.core.frame.DataFrame'>
```

	c0	c1	c2	c3	c4
0	1	4	7	10	13
1	2	5	8	11	14
2	3	6	9	12	15

이하공전 컴퓨터 정보 과

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

2-2. 데이터프레임

② 속성을 지정하여 변경하기

데이터프레임 df의 행 인덱스 배열을 나타내는 df.index와 열 이름 배열을 나타내는 df.columns에 새로운 배열을 할당하는 방식으로, 행 인덱스와 열 이름을 변경할 수 있다.

- 행 인덱스 변경: DataFrame 객체.index = 새로운 행 인덱스 배열
- 열 이름 변경: DataFrame 객체.columns = 새로운 열 이름 배열

〈예제 1-5〉 행 인덱스/열 이름 설정

(File: example/part1/1.5_change_df_idx_col.py(이어서 계속))

~ ~ ~ 생략 ~ ~ ~

```
18 # 행 인덱스, 열 이름 변경하기
19 df.index=['학생1', '학생2']
20 df.columns=['연령', '남녀', '소속']
21
22 print(df)          # 데이터프레임
23 print('\n')
24 print(df.index)    # 행 인덱스
25 print('\n')
26 print(df.columns) # 열 이름
```

〈실행 결과〉 코드 18~26라인을 부분 실행

	연령	남녀	소속
학생1	15	남	덕영중
학생2	17	여	수리중

```
Index(['학생1', '학생2'], dtype='object')
```

```
Index(['연령', '남녀', '소속'], dtype='object')
```

2-2. 데이터프레임

③ rename 메소드 사용

rename() 메소드를 적용하면, 행 인덱스 또는 열 이름의 일부를 선택하여 변경 가능. 단, 새로운 데이터프레임 객체를 리턴.

* 원본 객체를 변경하려면, inplace=True 옵션을 사용.

- 행 인덱스 변경: DataFrame 객체.rename(index={기존 인덱스:새 인덱스, ... })
- 열 이름 변경: DataFrame 객체.rename(columns={기존 이름:새 이름, ... })

예제 1-6

〈예제 1-6〉 행 인덱스/열 이름 변경

(File: example/part1/1.6_change_df_idx_col2.py)

```
1 # -*- coding: utf-8 -*-
2
3 import pandas as pd
4
5 # 행 인덱스/열 이름 지정하여 데이터프레임 만들기
6 df = pd.DataFrame([[15, '남', '덕영중'], [17, '여', '수리중']],
7                   index=['준서', '예은'],
8                   columns=['나이', '성별', '학교'])
9
10 # 데이터프레임 df 출력
11 print(df)
12 print("\n")
13
14 # 열 이름 중, '나이'를 '연령'으로, '성별'을 '남녀'로, '학교'를 '소속'으로 바꾸기
15 df.rename(columns={'나이': '연령', '성별': '남녀', '학교': '소속'}, inplace=True)
16
17 # df의 행 인덱스 중에서, '준서'를 '학생1'로, '예은'을 '학생2'로 바꾸기
18 df.rename(index={'준서': '학생1', '예은': '학생2'}, inplace=True)
19
20 # df 출력 (변경 후)
21 print(df)
```

〈실행 결과〉 코드 전부 실행

	나이	성별	학교
준서	15	남	덕영중
예은	17	여	수리중

	연령	남녀	소속
학생1	15	남	덕영중
학생2	17	여	수리중

2-2. 데이터프레임

• 행/열 삭제

drop() 메소드에 행을 삭제할 때는 축(axis) 옵션으로 axis=0을 입력하거나, 별도로 입력하지 않는다.

반면, 축옵션으로 axis=1을 입력하면 열을 삭제한다. 동시에 여러 개의 행 또는 열을 삭제하려면, 리스트 형태로 입력한다.

- 행 삭제: DataFrame 객체.drop(행 인덱스 또는 배열, axis=0)
- 열 삭제: DataFrame 객체.drop(열 이름 또는 배열, axis=1)

한편, drop() 메소드는 기존 객체를 변경하지 않고 새로운 객체를 반환하는 점에 유의한다. 따라서, 원본 객체를 직접 변경하기 위해서는, inplace=True 옵션을 추가한다.

2-2. 데이터프레임

inplace=True

① 행 삭제

df

	수학	영어	음악	체육
서준	90	98	85	100
우현	80	89	95	90
인아	70	95	100	90

`df.drop('우현', inplace=True)`

	수학	영어	음악	체육
서준	90	98	85	100
인아	70	95	100	90

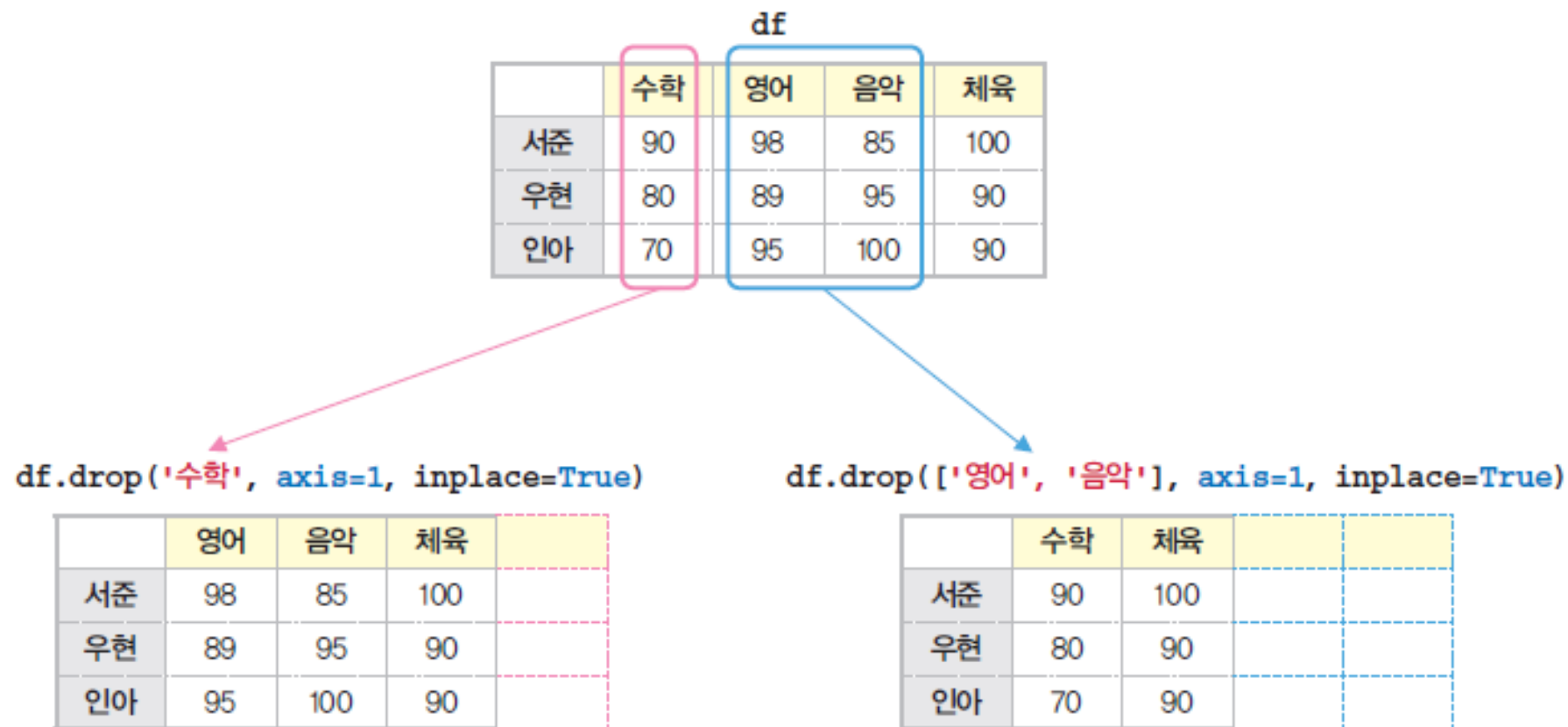
`df.drop(['우현', '인아'], axis=0, inplace=True)`

	수학	영어	음악	체육
서준	90	98	85	100

[그림 1-9] 행 삭제

2-2. 데이터프레임

② 열 삭제



[그림 1-10] 열 삭제

2-2. 데이터프레임

• 행 선택

- 1) **loc**과 **iloc** 인덱서를 사용.
- 2) 인덱스 **이름**을 기준으로 행을 선택할 때는 **loc**을 이용하고, **정수형 위치** 인덱스를 사용할 때는 **iloc**을 이용.

구분	loc	iloc
탐색 대상	인덱스 이름(index label)	정수형 위치 인덱스(integer position)
범위 지정	가능(범위의 끝 포함) 예) ['a':'c'] → 'a', 'b', 'c'	가능(범위의 끝 제외) 예) [3:7] → 3, 4, 5, 6 (* 7 제외)

[표 1-2] loc과 iloc

	수학	영어	음악	체육
서준	90	87	85	100
우현	80	89	95	90
인아	70	95	100	90

```
label1 = df.loc['서준']  
position1 = df.iloc[0]
```

2-2. 데이터프레임

• 행 선택

	수학	영어	음악	체육
서준	90	87	85	100
우현	80	89	95	90
인아	70	95	100	90

```
label2 = df.loc[['서준','우현']]  
position2 = df.iloc[[0,1]]
```

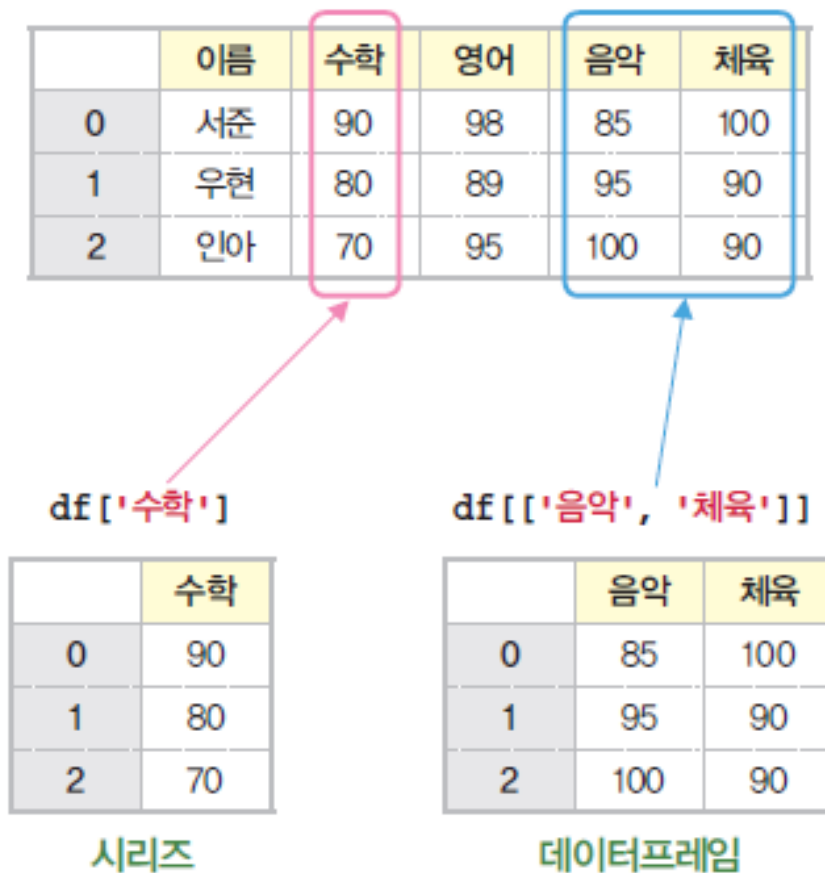
```
label3 = df.loc['서준':'우현']  
position3 = df.iloc[0:2]
```

['서준','우현'] ⇔ '서준':'우현'

[0,1] ⇔ 0:2

2-2. 데이터프레임

• 열 선택



[그림 1-11] 데이터프레임의 열 선택

Part 1. 판다스 입문

인하공전 컴퓨터 정보 과

2-2. 데이터프레임

예제 1-10

① 1개의 열 선택

• 열 선택

열 1개 선택(시리즈 생성): `DataFrame` 객체["열 이름"] 또는 `DataFrame` 객체.열 이름

`df['수학']`

`df.수학`

열 n개 선택(데이터프레임 생성): `DataFrame` 객체[[열1, 열2 , ... , 열n]]

`df [['음악' , '체육']]`

2-2. 데이터프레임

• 원소 선택

〈원소 1개 선택〉

	0 수학	1 영어	2 음악	3 체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc['서준', '음악']
df.iloc[0, 2] } 0 서준 음악 원소 85

〈원소 2개(시리즈) 선택〉

	0 수학	1 영어	2 음악	3 체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc['서준', ['음악', '체육']]
df.iloc[0, [2, 3]]
df.loc['서준', '음악' : '체육']
df.iloc[0, 2:] } 0 서준 음악 체육 시리즈 85 100

〈데이터프레임(df)의 일부분 선택〉

	0 수학	1 영어	2 음악	3 체육
0 서준	90	98	85	100
1 우현	80	89	95	90
2 인아	70	95	100	90

행, 열 좌표 { df.loc[['서준', '우현'], ['음악', '체육']]
df.iloc[[0, 1], [2, 3]]
df.loc['서준': '우현', '음악': '체육']
df.iloc[0:2, 2:] } 0 서준 음악 체육 85 100 1 우현 95 90 데이터프레임

[그림 1-12] 데이터프레임의 [행, 열] 데이터 선택

2-2. 데이터프레임

• 열 추가

- 1) 추가하려는 열 이름과 데이터 값을 입력. 마지막 열에 덧붙이듯 새로운 열을 추가.

열 추가: `DataFrame` 객체['추가하려는 열 이름'] = 데이터 값

- 2) 이때 모든 행에 동일한 값이 입력되는 점에 유의.



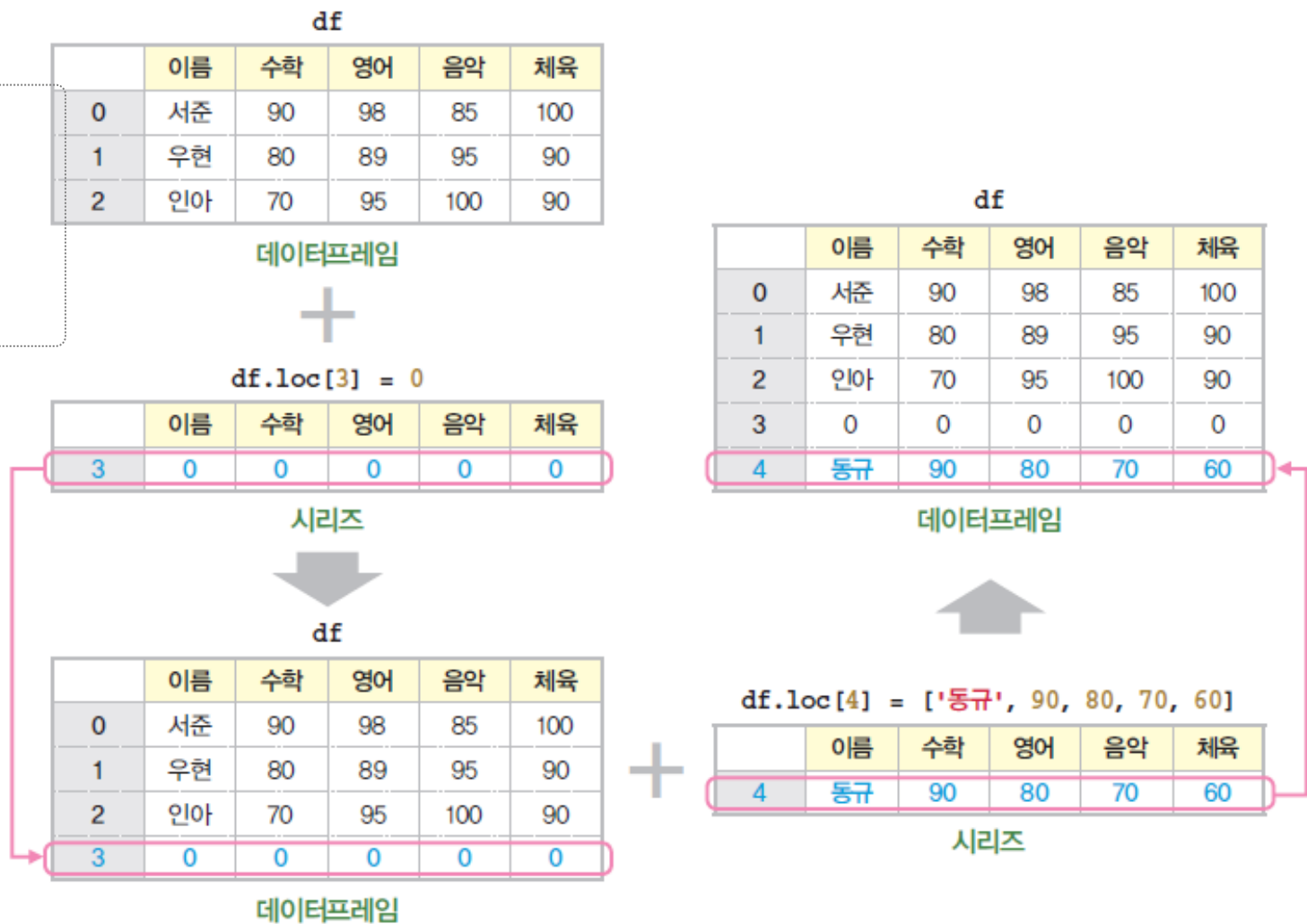
[그림 1-13] 데이터프레임의 열 추가

2-2. 데이터프레임

• 행 추가

: 행 인덱스와 데이터 값을 loc 인덱서를 사용하여 입력.

행 추가: `DataFrame.loc['새로운 행 이름'] = 데이터 값 (또는 배열)`



[그림 1-14] 데이터프레임의 행 추가

2-2. 데이터프레임

- 원소 값 변경

: 원소를 선택하고 새로운 데이터 값을 지정.

원소 값 변경: DataFrame 객체의 일부분 또는 원소를 선택 = 새로운 값

```
df.loc ['서준']['체육'] = 90
```

```
df.loc['서준',['음악','체육']] = 100, 50
```

2-2. 데이터프레임

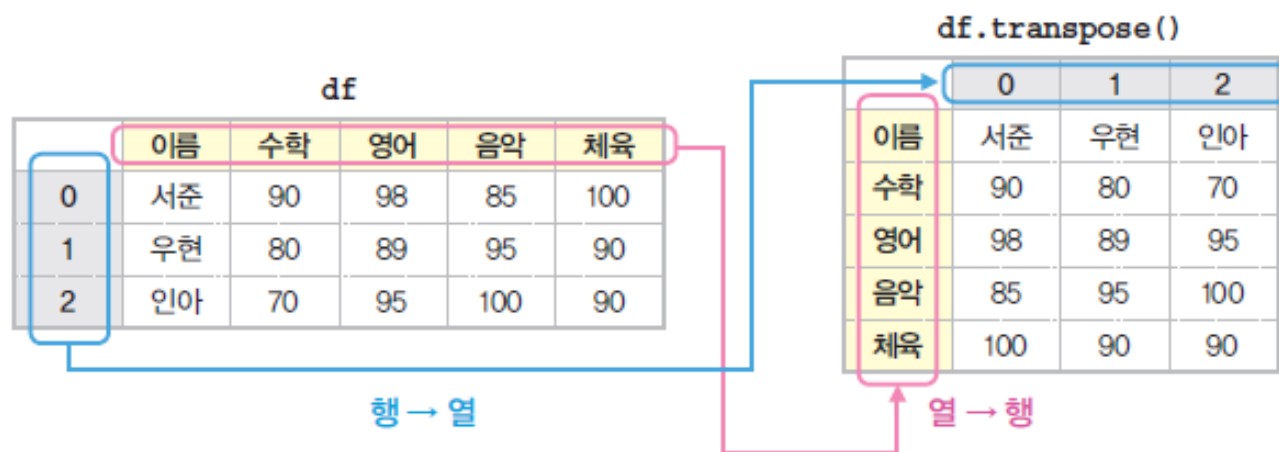
• 행, 열의 위치 바꾸기

데이터프레임의 행과 열을 서로 맞바꾸는 방법이다. 전치의 결과로 새로운 객체를 반환하므로, 기존 객체를 변경하기 위해서는 `df = df.transpose()` 또는 `df = df.T` 와 같이 입력한다.

`df.transpose()`

`df.T`

행, 열 바꾸기: `DataFrame` 객체.`transpose()` 또는 `DataFrame` 객체.`T`



[그림 1-15] 행, 열 바꾸기

3. 인덱스 활용

• 특정 열을 행 인덱스로 설정

: `set_index()` 메소드를 사용하여 데이터프레임의 특정 열을 행 인덱스로 설정한다. 새로운 객체를 반환한다.

특정 열을 행 인덱스로 설정: `DataFrame 객체.set_index(['열 이름'] 또는 '열 이름')`



3. 인덱스 활용

• 행 인덱스 재배열

reindex() 메소드를 사용하면, 데이터프레임의 행 인덱스를 새로운 배열로 재지정할 수 있다. 기존 객체를 변경하지 않고, 새로운 데이터프레임 객체를 반환한다.

새로운 배열로 행 인덱스를 재지정: `DataFrame 객체.reindex(새로운 인덱스 배열)`

기존 데이터프레임에 존재하지 않는 행 인덱스가 새롭게 추가되는 경우, 그 행의 데이터 값은 NaN 값이 입력된다.

3. 인덱스 활용

• 행 인덱스 초기화

- reset_index() 메소드로 정수형 위치 인덱스로 초기화. 기존 행 인덱스는 열로 이동. 새로운 데이터프레임 객체를 반환.

```
ndf = df.reset_index()
print(ndf)
```

	c0	c1	c2	c3	c4
r0	1	4	7	10	13
r1	2	5	8	11	14
r2	3	6	9	12	15

	index	c0	c1	c2	c3	c4
0	r0	1	4	7	10	13
1	r1	2	5	8	11	14
2	r2	3	6	9	12	15

3. 인덱스 활용

• 행 인덱스를 기준으로 데이터프레임 정렬

- `sort_index()` 메소드로 행 인덱스를 기준으로 정렬.
- `ascending` 옵션을 사용하여 오름차순 또는 내림차순 설정.
- 새롭게 정렬된 데이터프레임 객체를 반환.

```
# c1 열을 기준으로 내림차순 정렬  
ndf = df.sort_values(by='c1',ascending = False)  
print(ndf)
```

	c0	c1	c2	c3	c4
r2	3	6	9	12	15
r1	2	5	8	11	14
r0	1	4	7	10	13

1. 외부파일 읽기

1-1. CSV 파일

1-2. Excel 파일

1-3. JSON 파일

2. 웹(web)에서 가져오기

2-1. HTML 웹 페이지에서 표 속성 가져오기

2-2. 웹 스크래핑

3. API 활용하여 데이터 수집하기

4. 데이터 저장하기

4-1. CSV 파일로 저장

4-2. JSON 파일로 저장

4-3. Excel 파일로 저장

4-4. 여러 개의 데이터프레임을 하나의 Excel 파일로 저장

1. 외부파일 읽기

• 판다스 데이터 입출력 도구

판다스는 다양한 형태의 외부 파일을 읽어와서 데이터프레임으로 변환하는 함수를 제공한다. 어떤 파일이든 판다스 객체인 데이터프레임으로 변환되고 나면, 판다스의 모든 함수와 기능을 자유롭게 사용할 수 있다.

반대로, 데이터프레임을 다양한 유형의 파일로 저장할 수도 있다. [표 2-1]은 판다스 공식 사이트에서 제공하는 입출력 도구에 관한 자료를 요약한 것이다.

File Format	Reader	Writer
CSV	<code>read_csv</code>	<code>to_csv</code>
JSON	<code>read_json</code>	<code>to_json</code>
HTML	<code>read_html</code>	<code>to_html</code>
Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
MS Excel	<code>read_excel</code>	<code>to_excel</code>
HDF5 Format	<code>read_hdf</code>	<code>to_hdf</code>
SQL	<code>read_sql</code>	<code>to_sql</code>

[표 2-1] 판다스 데이터 입출력 도구(출처: <http://pandas.pydata.org>)

Part 2. 데이터 입출력

인하공전 컴퓨터 정보 과

1-1. CSV 파일

CSV 파일 → 데이터프레임: `pandas.read_csv("파일 경로(이름)")`

〈CSV 파일〉

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

* header 옵션

- '열 이름'이 되는 행을 지정
- `read_csv(file, header=?)`

❶ **header=0** (기본 값: 0행을 열 지정): `df = read_csv(file)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

	c0	c1	c2	c3
0	0	1	4	7
1	1	2	5	8
2	2	3	6	9

❷ **header=1** (1행을 열 지정): `df = read_csv(file, header=1)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

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

❸ **header=None** (행을 열 지정하지 않음): `df = read_csv(file, header=None)`

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

	0	1	2	3
0	c0	c1	c2	c3
1	0	1	4	7
2	1	2	5	8
3	2	3	6	9

1.아래와 같은 구조를 갖는 데이터 프레임에서 (원본 객체 변경:

iplace=True)

1) 이름을 index로 설정하고

2) '취미' 열을 삭제 하고

3) '나라' 행을 삭제 하고

4) '지민' 행을 이름 인덱스를 이용하여 추가 하여 '16, 중등, 효성' 데이터를 저장하고

5) column 이름 '학교'를 '교명'으로 바꾸고

6) 우석과 인아의 교명을 선경1, 한국1로 변경하고

7) 나이 기준 어린 순으로 행을 정렬 한 후

8) '국적'열을 추가 한 후 모두 '한국' 으로 저장 한 후

9) print하는 code를 작성 하시오

이름	나이	구분	취미	학교
우석	17	고등	독서	선경
인아	14	중등	산책	한국
나라	12	초등	축구	우성
민정	19	고등	독서	효성
서준	15	중등	농구	상성

2.아래와 같은 구조를 갖는 데이터 프레임에서 (원본 객체 변경: inplace=True)

- 1) '인문' 열을 추가 하여 영어와 국어 성적의 합계를 저장하고
- 2) '과학' 열을 추가 하여 물리와 화학 성적의 합계를 저장하고
- 3) 과학 성적으로 내림 차순으로 정렬하고
- 4) print하는 code를 작성하시오.

이름	국어	영어	물리	화학
우석	90	85	70	50
인아	80	90	80	70
나라	75	85	80	90
민정	90	95	90	100

Part 3. 데이터 살펴보기

1. 데이터프레임의 구조

1-1. 데이터 내용 미리보기

1-2. 데이터 요약 정보 확인하기

1-3. 데이터 개수 확인하기

2. 통계 함수 적용

2-1. 평균값

2-2. 중간값

2-3. 최대값

2-4. 최소값

2-5. 표준편차

2-6. 상관계수

1. 데이터프레임의 구조

• UCI 자동차 연비(auto mpg) 데이터셋

: 연비, 실린더 수, 배기량, 출력, 차중, 가속능력, 출시년도, 제조국, 모델명에 관한 데이터 398 개로 구성

No.	속성(attributes)		데이터 상세(범위)
1	mpg	연비	연속 값
2	cylinders	실린더 수	이산 값(예: 3, 4, 6, 8)
3	displacement	배기량	연속 값
4	horsepower	출력	연속 값
5	weight	차중	연속 값
6	acceleration	가속능력	연속 값
7	model_year	출시년도	이산 값(예: 70, 71, 80, 81)
8	origin	제조국	이산 값(예: 1(USA), 2(EU), 3(JPN))
9	name	모델명	문자열

[표 3-1] UCI 데이터셋 - "auto mpg" 상세 항목

1. 데이터프레임의 구조

1-1 데이터 내용 미리보기

`head()` 메소드 인자로 정수 `n`을 전달하면 처음 `n`개의 행을 보여준다. 반면 `tail()` 메소드에 정수 `n`을 입력하면 마지막 `n`개의 행을 보여준다. 한편 정수 `n`을 입력하지 않고 `head()`와 같이 입력하면 처음 또는 마지막 5개 행을 보여준다(디폴트값: `n=5`).

- 앞부분 미리보기: `DataFrame 객체.head(n)`
- 뒷부분 미리보기: `DataFrame 객체.tail(n)`

Part 3. 데이터 살펴보기

인하공전 컴퓨터 정보 과

G7													
	A	B	C	D	E	F	G	H	I	J	K	L	
1	18	8	307	130	3504	12	70	1	chevrolet chevelle malibu				
2	15	8	350	165	3693	11.5	70	1	buick skylark 320				
3	18	8	318	150	3436	11	70	1	plymouth satellite				
4	16	8	304	150	3433	12	70	1	amc rebel sst				
5	17	8	302	140	3449	10.5	70	1	ford torino				
6	15	8	429	198	4341	10	70	1	ford galaxie 500				
7	14	8	454	220	4354	9	70	1	chevrolet impala				
8	14	8	440	215	4312	8.5	70	1	plymouth fury iii				
9	14	8	455	225	4425	10	70	1	pontiac catalina				
10	15	8	390	190	3850	8.5	70	1	amc ambassador dpl				
11	15	8	383	170	3563	10	70	1	dodge challenger se				
12	14	8	340	160	3609	8	70	1	plymouth 'cuda 340				
13	15	8	400	150	3761	9.5	70	1	chevrolet monte carlo				
14	14	8	455	225	3086	10	70	1	buick estate wagon (sw)				
15	24	4	113	95	2372	15	70	3	toyota corona mark ii				
16	22	6	198	95	2833	15.5	70	1	plymouth duster				
17	18	6	199	97	2774	15.5	70	1	amc hornet				
18	21	6	200	85	2587	16	70	1	ford maverick				

1. 데이터프레임의 구조

```
import pandas as pd
df = pd.read_csv('/content/auto-mpg.csv', header = None)
df.columns = ['mpg','cylinders','displacement','horsepowwer','weight','acceleration','model year','origin','name']
```

```
print(df.head())
print()
print(df.tail())
```

	mpg	cylinders	displacement	horsepowwer	weight	acceleration	₩
0	18.0	8	307.0	130.0	3504.0	12.0	
1	15.0	8	350.0	165.0	3693.0	11.5	
2	18.0	8	318.0	150.0	3436.0	11.0	
3	16.0	8	304.0	150.0	3433.0	12.0	
4	17.0	8	302.0	140.0	3449.0	10.5	

	model year	origin	name
0	70	1	chevrolet chevelle malibu
1	70	1	buick skylark 320
2	70	1	plymouth satellite
3	70	1	amc rebel sst
4	70	1	ford torino

1. 데이터프레임의 구조

1-2 데이터 요약 정보 확인하기

- 데이터프레임의 크기(행, 열)

데이터프레임 클래스의 `shape` 속성은 행과 열의 개수를 튜플 형태로 보여준다. 예제에서 변수 `df`에 저장된 데이터프레임의 크기(행의 개수, 열의 개수)를 확인하려면 `df.shape`라고 입력한다.

데이터프레임의 크기 확인: `DataFrame` 객체.`.shape`

〈예제 3-1〉 데이터 살펴보기

(File: example/part3/3.1_exploratory_analysis.py(이어서 계속))

~ ~~~ 생략 ~~~

```
17 # df의 모양과 크기 확인: (행의 개수, 열의 개수)를 튜플로 반환
18 print(df.shape)
```

〈실행 결과〉 코드 17~18라인을 부분 실행

(398, 9)

1. 데이터프레임의 구조

- 데이터프레임의 기본 정보

`info()` 메소드를 데이터프레임에 적용하면 데이터프레임에 관한 기본 정보를 화면에 출력한다. 클래스 유형, 행 인덱스의 구성, 열 이름의 종류와 개수, 각 열의 자료형과 개수, 메모리 할당량에 관한 정보가 포함된다. 데이터프레임 `df`의 기본 정보를 확인하는 명령은 `df.info()`이다.

데이터프레임의 기본 정보 출력: `DataFrame` 객체.`info()`

`df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype
---  -
0   mpg             398 non-null   float64
1   cylinders       398 non-null   int64
2   displacement    398 non-null   float64
3   horsepower      398 non-null   object
4   weight          398 non-null   float64
5   acceleration    398 non-null   float64
6   model year      398 non-null   int64
7   origin          398 non-null   int64
8   name            398 non-null   object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
None
```

- 데이터프레임의 기술 통계 정보 요약

데이터프레임에 `describe()` 메소드를 적용하면, 산술(숫자) 데이터를 갖는 열에 대한 주요 기술 통계 정보(평균, 표준편차, 최대값, 최소값, 중간값 등)를 요약하여 출력한다.

데이터프레임의 기술 통계 정보 요약: `DataFrame 객체.describe()`

	mpg	cylinders	displacement	weight	acceleration	mpg
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	2970.424623	15.568090	
std	7.815984	1.701004	104.269838	846.841774	2.757689	
min	9.000000	3.000000	68.000000	1613.000000	8.000000	
25%	17.500000	4.000000	104.250000	2223.750000	13.825000	
50%	23.000000	4.000000	148.500000	2803.500000	15.500000	
75%	29.000000	8.000000	262.000000	3608.000000	17.175000	
max	46.600000	8.000000	455.000000	5140.000000	24.800000	

	model year	origin
count	398.000000	398.000000
mean	76.010050	1.572864
std	3.697627	0.802055
min	70.000000	1.000000
25%	73.000000	1.000000
50%	76.000000	1.000000
75%	79.000000	2.000000
max	82.000000	3.000000

1. 데이터프레임의 구조

1-3 데이터 개수 확인

- 각 열의 데이터 개수

〈예제 3-2〉 데이터 개수 확인

(File: example/part3/3.2_exploratory_analysis2.py)

```
1  # -*- coding: utf-8 -*-
2
3  import pandas as pd
4
5  # read_csv() 함수로 df 생성
6  df = pd.read_csv('./auto-mpg.csv', header=None)
7
8  # 열 이름 지정
9  df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
10               'acceleration', 'model year', 'origin', 'name']
11
12 # 데이터프레임 df의 각 열이 가지고 있는 원소 개수 확인
13 print(df.count())
14 print('\n')
15
16 # df.count()가 반환하는 객체 타입 출력
17 print(type(df.count()))
```

df.count()

```
mpg          398
cylinders    398
displacement 398
horsepower   398
weight       398
acceleration 398
model year   398
origin       398
name         398
dtype: int64
```

1. 데이터프레임의 구조

각 열의 고유값 개수

열 데이터의 고유값 개수: `DataFrame` 객체["열 이름"].`value_counts()`

〈예제 3-2〉 데이터 개수 확인

(File: example/part3/3.2_exploratory_analysis2.py(0이어서 계속))

```
~ ~ ~ 생략 ~ ~ ~

20 # 데이터프레임 df의 특정 열이 가지고 있는 고유값 확인
21 unique_values = df['origin'].value_counts()
22 print(unique_values)
23 print('\n')
24
25 # value_counts 메소드가 반환하는 객체 타입 출력
26 print(type(unique_values))
```

〈실행 결과〉 코드 20~26라인을 부분 실행

```
1    249
3     79
2     70
Name: origin, dtype: int64

<class 'pandas.core.series.Series'>
```

```
unique_values = df['origin'].value_counts()
```

2. 통계함수 적용

2-1 평균값

데이터프레임에 `mean()` 메소드를 적용하면, 산술 데이터를 갖는 모든 열의 평균값을 각각 계산하여 시리즈 객체로 반환한다. 데이터프레임의 특정 열을 선택하여 평균값을 계산할 수도 있다.

- 모든 열의 평균값: `DataFrame` 객체.`.mean()`
- 특정 열의 평균값: `DataFrame` 객체["열 이름"].`.mean()`

```
print(df.mean())
print()
print(df['mpg'].mean())
print(df.mpg.mean())
print()
print(df[['mpg','weight']].mean())
```

```
mpg          23.514573
cylinders     5.454774
displacement  193.425879
weight       2970.424623
acceleration  15.568090
model year    76.010050
origin        1.572864
dtype: float64
```

```
23.514572864321607
23.514572864321607
```

```
mpg          23.514573
weight       2970.424623
dtype: float64
```

2-2 중간값

데이터프레임에 `median()` 메소드를 적용하면 산술 데이터를 갖는 모든 열의 중간값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 중간값을 계산할 수도 있다.

- 모든 열의 중간값: `DataFrame 객체.median()`
- 특정 열의 중간값: `DataFrame 객체["열 이름"].median()`

2-3 최대값

데이터프레임에 `max()` 메소드를 적용하면 데이터프레임의 각 열이 갖는 데이터 값 중에서 최대값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 최대값: `DataFrame 객체.max()`
- 특정 열의 최대값: `DataFrame 객체["열 이름"].max()`

2-4 최소값

데이터프레임에 `min()` 메소드를 적용하면 데이터프레임의 각 열이 갖는 데이터 값 중에서 최소값을 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 최소값: `DataFrame 객체.min()`
- 특정 열의 최소값: `DataFrame 객체["열 이름"].min()`

2-5 표준편차

데이터프레임에 `std()` 메소드를 적용하면 산술 데이터를 갖는 열의 표준편차를 계산하여 시리즈로 반환한다. 데이터프레임의 특정 열을 선택하여 계산할 수도 있다.

- 모든 열의 표준편차: `DataFrame 객체.std()`
- 특정 열의 표준편차: `DataFrame 객체["열 이름"].std()`

Part 3. 데이터 살펴보기

인하공전 컴퓨터 정보 과

2-6 상관관계수

데이터프레임에 `corr()` 메소드를 적용하면 두 열 간의 상관관계수를 계산한다. 산술 데이터를 갖는 모든 열에 대하여 2개씩 서로 짝을 짓고, 각각의 경우에 대하여 상관관계수를 계산한다.

- 모든 열의 상관관계수: `DataFrame 객체.corr()`
- 특정 열의 상관관계수: `DataFrame 객체[열 이름의 리스트].corr()`

```
print(df.corr())  
print()  
print(df[['mpg','weight']].corr())
```

	mpg	cylinders	displacement	weight	acceleration	W
mpg	1.000000	-0.775396	-0.804203	-0.831741	0.420289	
cylinders	-0.775396	1.000000	0.950721	0.896017	-0.505419	
displacement	-0.804203	0.950721	1.000000	0.932824	-0.543684	
weight	-0.831741	0.896017	0.932824	1.000000	-0.417457	
acceleration	0.420289	-0.505419	-0.543684	-0.417457	1.000000	
model year	0.579267	-0.348746	-0.370164	-0.306564	0.288137	
origin	0.563450	-0.562543	-0.609409	-0.581024	0.205873	

	model year	origin
mpg	0.579267	0.563450
cylinders	-0.348746	-0.562543
displacement	-0.370164	-0.609409
weight	-0.306564	-0.581024
acceleration	0.288137	0.205873
model year	1.000000	0.180662
origin	0.180662	1.000000

	mpg	weight
mpg	1.000000	-0.831741
weight	-0.831741	1.000000

2-6 상관관계수

데이터프레임에 `corr()` 메소드를 적용하면 두 열 간의 상관관계수를 계산한다. 산술 데이터를 갖는 모든 열에 대하여 2개씩 서로 짝을 짓고, 각각의 경우에 대하여 상관관계수를 계산한다.

- 모든 열의 상관관계수: `DataFrame 객체.corr()`
- 특정 열의 상관관계수: `DataFrame 객체 [열 이름의 리스트].corr()`

No.	속성(attributes)		데이터 상세(범위)
1	mpg	연비	연속 값
2	cylinders	실린더 수	이산 값(예시: 3, 4, 6, 8)
3	displacement	배기량	연속 값
4	horsepower	출력	연속 값
5	weight	차중	연속 값
6	acceleration	가속능력	연속 값
7	model_year	출시년도	이산 값(예: 70, 71, 80, 81)
8	origin	제조국	이산 값(예: 1(USA), 2(EU), 3(JPN))
9	name	모델명	문자열

[표 3-1] UCI 데이터셋 - "auto mpg" 상세 항목

Part 3. 데이터 살펴보기

인하공전 컴퓨터 정보 과

과제 3

- 3.1) iris3.csv 파일을 읽고 (첫번째 행을 column 이름으로)
- 2) column 이름을 'feat1','feat2','feat3','feat4','name' 으로 변경하고
- 3) 처음 3개 행을 print 하고
- 4) 요약 정보를 print하고
- 5) 통계 정보를 print하고
- 6) 각 열 데이터의 고유값 개수를 출력하고
- 7) 'feat2' 열의 최대값 출력
- 8) 'feat4' 열의 표준 편차 출력
- 9) 'feat1'과 'feat3'의 상관계수를 출력 하는 코드를 작성 하시오

	A	B	C	D	E
1	sepal_leng	sepal_widt	petal_leng	petal_width	species
2	5.1	3.5	1.4	0.2	Iris-setosa
3	4.9	3	1.4	0.2	Iris-setosa
4	4.7	3.2	1.3	0.2	Iris-setosa
5	4.6	3.1	1.5	0.2	Iris-setosa
6	5	3.6	1.4	0.2	Iris-setosa
7	5.4	3.9	1.7	0.4	Iris-setosa
8	4.6	3.4	1.4	0.3	Iris-setosa
9	5	3.4	1.5	0.2	Iris-setosa
10	4.4	2.9	1.4	0.2	Iris-setosa
11	4.9	3.1	1.5	0.1	Iris-setosa
12	5.4	3.7	1.5	0.2	Iris-setosa
13	4.8	3.4	1.6	0.2	Iris-setosa
14	4.8	3	1.4	0.1	Iris-setosa
15	4.3	3	1.1	0.1	Iris-setosa

Part 5. 데이터 사전처리

1. 누락 데이터 처리
2. 중복 데이터 처리
3. 데이터 표준화
 - 3-1. 단위 환산
 - 3-2. 자료형 변환
4. 범주형(카테고리) 데이터 처리
 - 4-1. 구간 분할
 - 4-2. 더미 변수
5. 정규화
6. 시계열 데이터
 - 6-1. 다른 자료형을 시계열 객체로 변환
 - 6-2. 시계열 데이터 만들기
 - 6-3. 시계열 데이터 활용

1. 누락 데이터 처리

데이터 프레임에는 원소 데이터 값이 누락 되는 경우가 있음. 이렇게 존재하는 않은 데이터를 **NaN (Not A Number)** 머신 러닝 분석 모형에 데이터를 입력 하기 전에 반드시 누락 데이터를 **제거 하거나** 다른 **적절한 값으로 대체** 하는 과정이 필요함.

```
import seaborn as sns
df = sns.load_dataset('titanic')
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

1. 누락 데이터 처리

df.info()

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 15 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   survived    891 non-null    int64  
1   pclass      891 non-null    int64  
2   sex         891 non-null    object  
3   age        714 non-null    float64  
4   sibsp       891 non-null    int64  
5   parch       891 non-null    int64  
6   fare        891 non-null    float64  
7   embarked    889 non-null    object  
8   class       891 non-null    category  
9   who         891 non-null    object  
10  adult_male   891 non-null    bool  
11  deck        203 non-null    category  
12  embark_town  889 non-null    object  
13  alive       891 non-null    object  
14  alone       891 non-null    bool  
dtypes: bool(2), category(2), float64(2), int64(4), object(5)  
memory usage: 80.7+ KB
```



$891 - 203 = 688$ (NaN 데이터)

1. 누락 데이터 처리

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
#   Column      Non-Null Count  Dtype
---  -
0   survived    891 non-null    int64
1   pclass      891 non-null    int64
2   sex         891 non-null    object
3   age         714 non-null    float64
4   sibsp       891 non-null    int64
5   parch       891 non-null    int64
6   fare        891 non-null    float64
7   embarked    889 non-null    object
8   class       891 non-null    category
9   who         891 non-null    object
10  adult_male   891 non-null    bool
11  deck        203 non-null    category
12  embark_town  889 non-null    object
13  alive       891 non-null    object
14  alone       891 non-null    bool
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```



$891 - 203 = 688$ (NaN 데이터)

1. 결측치 처리하기 : 드롭과 채우기

- 데이터를 삭제하거나 데이터를 채움
 - 데이터가 없으면 해당 행이나 열을 삭제
 - 평균값, 최빈값, 중간값 등으로 데이터를 채움

```
In [1]: import pandas as pd
import numpy as np

raw_data = {'first_name': ['Jason', np.nan, 'Tina', 'Jake', 'Amy'],
            'last_name': ['Miller', np.nan, 'Ali', 'Milner', 'Cooze'],
            'age': [42, np.nan, 36, 24, 73],
            'sex': ['m', np.nan, 'f', 'm', 'f'],
            'preTestScore': [4, np.nan, np.nan, 2, 3],
            'postTestScore': [25, np.nan, np.nan, 62, 70]}

df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age',
'sex', 'preTestScore', 'postTestScore'])
df
```

Out [1]:

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

1.1 드롭

- 드롭(drop) : 결측치가 나온 열이나 행을 삭제
- dropna 사용하여 NaN이 있는 모든 데이터의 행을 없앴

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

In [3]:	df.dropna()																																	
Out [3]:	<table><tr><th></th><th>first_name</th><th>last_name</th><th>age</th><th>sex</th><th>preTestScore</th><th>postTestScore</th></tr><tr><td>0</td><td>Jason</td><td>Miller</td><td>42.0</td><td>m</td><td>4.0</td><td>25.0</td></tr><tr><td>3</td><td>Jake</td><td>Milner</td><td>24.0</td><td>m</td><td>2.0</td><td>62.0</td></tr><tr><td>4</td><td>Amy</td><td>Cooze</td><td>73.0</td><td>f</td><td>3.0</td><td>70.0</td></tr></table>							first_name	last_name	age	sex	preTestScore	postTestScore	0	Jason	Miller	42.0	m	4.0	25.0	3	Jake	Milner	24.0	m	2.0	62.0	4	Amy	Cooze	73.0	f	3.0	70.0
	first_name	last_name	age	sex	preTestScore	postTestScore																												
0	Jason	Miller	42.0	m	4.0	25.0																												
3	Jake	Milner	24.0	m	2.0	62.0																												
4	Amy	Cooze	73.0	f	3.0	70.0																												

02 데이터 전처리의 전략

- 매개변수 how로 조건에 따라 결측치를 지움
 - how에는 매개변수 'all'과 'any' 사용
 - 'all'은 행에 있는 모든 값이 NaN일 때 해당 행을 삭제
 - 'any'는 하나의 NaN만 있어도 삭제
- dropna의 기본 설정은 'any'라서 모든 결측치를 지움

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

In [5]:	df_cleaned = df.dropna(how='all') df_cleaned																																									
Out [5]:	<table><tr><th></th><th>first_name</th><th>last_name</th><th>age</th><th>sex</th><th>preTestScore</th><th>postTestScore</th></tr><tr><td>0</td><td>Jason</td><td>Miller</td><td>42.0</td><td>m</td><td>4.0</td><td>25.0</td></tr><tr><td>2</td><td>Tina</td><td>Ali</td><td>36.0</td><td>f</td><td>NaN</td><td>NaN</td></tr><tr><td>3</td><td>Jake</td><td>Milner</td><td>24.0</td><td>m</td><td>2.0</td><td>62.0</td></tr><tr><td>4</td><td>Amy</td><td>Cooze</td><td>73.0</td><td>f</td><td>3.0</td><td>70.0</td></tr></table>								first_name	last_name	age	sex	preTestScore	postTestScore	0	Jason	Miller	42.0	m	4.0	25.0	2	Tina	Ali	36.0	f	NaN	NaN	3	Jake	Milner	24.0	m	2.0	62.0	4	Amy	Cooze	73.0	f	3.0	70.0
		first_name	last_name	age	sex	preTestScore	postTestScore																																			
	0	Jason	Miller	42.0	m	4.0	25.0																																			
	2	Tina	Ali	36.0	f	NaN	NaN																																			
	3	Jake	Milner	24.0	m	2.0	62.0																																			
	4	Amy	Cooze	73.0	f	3.0	70.0																																			

02 데이터 전처리의 전략

- 매개변수 threshfh 데이터의 개수를 기준으로 삭제
 - thresh=1 지정하면 데이터가 한 개라도 존재하는 행은 남김
 - thresh=5 지정하면 데이터가 다섯 개 이상 있어야 남김

	first_name	last_name	age	sex	preTestScore	postTestScore
0	Jason	Miller	42.0	m	4.0	25.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Tina	Ali	36.0	f	NaN	NaN
3	Jake	Milner	24.0	m	2.0	62.0
4	Amy	Cooze	73.0	f	3.0	70.0

In [7]:	df.dropna(axis=0, thresh=1)																																								
Out [7]:	<table><tr><th></th><th>first_name</th><th>last_name</th><th>age</th><th>sex</th><th>preTestScore</th><th>postTestScore</th><th>location</th></tr><tr><td>0</td><td>Jason</td><td>Miller</td><td>42.0</td><td>m</td><td>4.0</td><td>25.0</td><td>NaN</td></tr><tr><td>2</td><td>Tina</td><td>Ali</td><td>36.0</td><td>f</td><td>NaN</td><td>NaN</td><td>NaN</td></tr><tr><td>3</td><td>Jake</td><td>Milner</td><td>24.0</td><td>m</td><td>2.0</td><td>62.0</td><td>NaN</td></tr><tr><td>4</td><td>Amy</td><td>Cooze</td><td>73.0</td><td>f</td><td>3.0</td><td>70.0</td><td>NaN</td></tr></table>		first_name	last_name	age	sex	preTestScore	postTestScore	location	0	Jason	Miller	42.0	m	4.0	25.0	NaN	2	Tina	Ali	36.0	f	NaN	NaN	NaN	3	Jake	Milner	24.0	m	2.0	62.0	NaN	4	Amy	Cooze	73.0	f	3.0	70.0	NaN
	first_name	last_name	age	sex	preTestScore	postTestScore	location																																		
0	Jason	Miller	42.0	m	4.0	25.0	NaN																																		
2	Tina	Ali	36.0	f	NaN	NaN	NaN																																		
3	Jake	Milner	24.0	m	2.0	62.0	NaN																																		
4	Amy	Cooze	73.0	f	3.0	70.0	NaN																																		
In [8]:	df.dropna(thresh=5)																																								
Out [8]:	<table><tr><th></th><th>first_name</th><th>last_name</th><th>age</th><th>sex</th><th>preTestScore</th><th>postTestScore</th><th>location</th></tr><tr><td>0</td><td>Jason</td><td>Miller</td><td>42.0</td><td>m</td><td>4.0</td><td>25.0</td><td>NaN</td></tr><tr><td>3</td><td>Jake</td><td>Milner</td><td>24.0</td><td>m</td><td>2.0</td><td>62.0</td><td>NaN</td></tr><tr><td>4</td><td>Amy</td><td>Cooze</td><td>73.0</td><td>f</td><td>3.0</td><td>70.0</td><td>NaN</td></tr></table>		first_name	last_name	age	sex	preTestScore	postTestScore	location	0	Jason	Miller	42.0	m	4.0	25.0	NaN	3	Jake	Milner	24.0	m	2.0	62.0	NaN	4	Amy	Cooze	73.0	f	3.0	70.0	NaN								
	first_name	last_name	age	sex	preTestScore	postTestScore	location																																		
0	Jason	Miller	42.0	m	4.0	25.0	NaN																																		
3	Jake	Milner	24.0	m	2.0	62.0	NaN																																		
4	Amy	Cooze	73.0	f	3.0	70.0	NaN																																		

1.2 채우기

- 채우기(fill) : 비어있는 값을 채움
- 일반적으로 드롭한 후에 남은 값들을 채우기 처리
- 평균, 최빈값 등 데이터의 분포를 고려해서 채움
- 함수 fillna 사용

In [9]:

df.fillna(0)

Out [9]:

	first_name	last_name	age	sex	preTestScore	postTestScore	location
0	Jason	Miller	42.0	m	4.0	25.0	0.0
1	0	0	0.0	0	0.0	0.0	0.0
2	Tina	Ali	36.0	f	0.0	0.0	0.0
3	Jake	Milner	24.0	m	2.0	62.0	0.0
4	Amy	Cooze	73.0	f	3.0	70.0	0.0

02 데이터 전처리의 전략

- 빈 값에 평균값을 채우려면 열 단위의 평균값을 계산하여 해당 열에만 값을 채움
 - 매개변수 inplace는 변경된 값을 리턴시키는 것이 아니고 해당 변수 자체의 값을 변경

In [10]:	<pre>df["preTestScore"].fillna(df["preTestScore"].mean(), inplace=True)</pre> <pre>df</pre>																																																						
Out [10]:	<table><tr><th></th><th>first_name</th><th>last_name</th><th>age</th><th>sex</th><th>preTestScore</th><th>postTestScore</th><th>location</th></tr><tr><td>0</td><td>Jason</td><td>Miller</td><td>42.0</td><td>m</td><td>4.0</td><td>25.0</td><td>NaN</td></tr><tr><td>1</td><td>NaN</td><td>NaN</td><td>NaN</td><td>NaN</td><td>3.0</td><td>NaN</td><td>NaN</td></tr><tr><td>2</td><td>Tina</td><td>Ali</td><td>36.0</td><td>f</td><td>3.0</td><td>NaN</td><td>NaN</td></tr><tr><td>3</td><td>Jake</td><td>Milner</td><td>24.0</td><td>m</td><td>2.0</td><td>62.0</td><td>NaN</td></tr><tr><td>4</td><td>Amy</td><td>Cooze</td><td>73.0</td><td>f</td><td>3.0</td><td>70.0</td><td>NaN</td></tr></table>								first_name	last_name	age	sex	preTestScore	postTestScore	location	0	Jason	Miller	42.0	m	4.0	25.0	NaN	1	NaN	NaN	NaN	NaN	3.0	NaN	NaN	2	Tina	Ali	36.0	f	3.0	NaN	NaN	3	Jake	Milner	24.0	m	2.0	62.0	NaN	4	Amy	Cooze	73.0	f	3.0	70.0	NaN
	first_name	last_name	age	sex	preTestScore	postTestScore	location																																																
0	Jason	Miller	42.0	m	4.0	25.0	NaN																																																
1	NaN	NaN	NaN	NaN	3.0	NaN	NaN																																																
2	Tina	Ali	36.0	f	3.0	NaN	NaN																																																
3	Jake	Milner	24.0	m	2.0	62.0	NaN																																																
4	Amy	Cooze	73.0	f	3.0	70.0	NaN																																																

1. 누락 데이터 처리

- `isnull()` : 누락 데이터면 `True`를 반환하고, 유효한 데이터가 존재하면 `False`를 반환한다.
- `notnull()` : 유효한 데이터가 존재하면 `True`를 반환하고, 누락 데이터면 `False`를 반환한다.

```
# 누락 데이터 찾기  
df.head().isnull()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	True	False	False	False

```
[ ] df.head().notnull()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	True	True	True	True	True	True	True	True	True	True	True	False	True	True	True
1	True	True	True	True	True	True	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True	True	True	False	True	True	True
3	True	True	True	True	True	True	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True	True	True	False	True	True	True

1. 누락 데이터 처리- 누락 데이터 확인

```
missing_df = df.isnull()
for col in missing_df.columns:
    missing_count = missing_df[col].value_counts() # 각 열의 NaN 개수 파악

    try:
        print(col, ': ', missing_count[True]) # NaN 값이 있으면 개수 출력
    except:
        print(col, ': ', 0) # NaN 값이 없으면 0개 출력

# try & except: try 하위 명령에서 오류가 발생할 경우, except 실행
```



```
survived : 0
pclass : 0
sex : 0
age : 177
sibsp : 0
parch : 0
fare : 0
embarked : 2
class : 0
who : 0
adult_male : 0
deck : 688
embark_town : 2
alive : 0
alone : 0
```


1. 누락 데이터 처리-누락 데이터 제거

df.dropna()

age 열에 나이 데이터가 없는 모든 행 삭제

```
df_age = df.dropna(axis = 0, subset = ['age'], how = 'any') # subset : 열을 list 형태로 입력  
# how = 'any': NaN값이 하나라도 존재하면 삭제  
# how = 'all': 모든 데이터가 NaN값일 때 삭제
```

```
print(len(df_age))
```

714

1. 누락 데이터 치환

`df.fillna()`

NaN 값을 평균값으로 치환하기

```
print(df['age'].head(10))
```

```
print('\n')
```

```
mean_age = df['age'].mean()
```

```
df['age'].fillna(mean_age, inplace = True)
```

```
print(df['age'].head(10))
```

```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: age, dtype: float64
```

```
0    22.000000
1    38.000000
2    26.000000
3    35.000000
4    35.000000
5    29.699118
6    54.000000
7     2.000000
8    27.000000
9    14.000000
Name: age, dtype: float64
```

1. 누락 데이터 치환

df.fillna()

.fillna(method = 'ffill', inplace = True) 바로 앞에 있는 값으로 치환
fillna(method = 'bfill', inplace = True) 바로 뒤에 있는 값으로 치환

이웃하고 있는 값으로 바꾸기
import seaborn as sns
df = sns.load_dataset('titanic')

```
print(df['embark_town'][825:830])  
print('\n')
```

```
# embark_town 열의 NaN 값을 바로 앞에 있는 828행의 값으로 변경하기  
df['embark_town'].fillna(method = 'ffill', inplace = True)  
print(df['embark_town'][825:830])
```

```
# method = 'bfill'은 NaN 행의 다음 행 값으로 바꾼다.
```

```
825    Queenstown  
826    Southampton  
827     Cherbourg  
828    Queenstown  
829           NaN  
Name: embark_town, dtype: object
```

```
825    Queenstown  
826    Southampton  
827     Cherbourg  
828    Queenstown  
829    Queenstown  
Name: embark_town, dtype: object
```

2. 중복 데이터 처리

중복 데이터 확인

`deduplicated()` : 전에 나온 행들과 비교하여 중복되는 행이면 `true`를 반환.
처음 나오는 행에 대해서는 `false`를 반환.

2. 중복 데이터 처리

중복 데이터 확인

```
import pandas as pd
df = pd.DataFrame({'c1':['a','a','b','a','b'],
                  'c2':[1,1,1,2,2],
                  'c3':[1,1,2,2,2]})

print(df)
print('\n')

# 중복 데이터 확인
df_dup = df.duplicated() # 전에 나온 행들과 비교하여 중복되는 행이면 True를 반환
                        # 처음 나오는 행에 대해서는 False 반환
print(df_dup)
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

0	False
1	True
2	False
3	False
4	False

dtype: bool

2. 중복 데이터 처리

중복 데이터 확인

```
col_dup = df['c2'].duplicated()  
print(col_dup)
```

```
0    False
```

```
1     True
```

```
2     True
```

```
3    False
```

```
4     True
```

```
Name: c2, dtype: bool
```

2. 중복 데이터 처리

중복 데이터 제거

```
# 중복 데이터 제거  
print(df)  
print('\n')
```

```
df2 = df.drop_duplicates() # 중복되는 행을 제거하고 고유한 관측값을 가진 행들  
만 남긴다.  
print(df2)
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

2. 중복 데이터 처리

중복 데이터 제거

```
df3 = df.drop_duplicates(subset = ['c2','c3']) # subset 옵션에 '열 이름 리스트' 전달  
# 'c2','c3' 열을 기준으로 판별  
print(df3)
```

	c1	c2	c3
0	a	1	1
1	a	1	1
2	b	1	2
3	a	2	2
4	b	2	2

	c1	c2	c3
0	a	1	1
2	b	1	2
3	a	2	2

4.code에서 정의한 데이터에서

- 1) 하나라도 NaN이 있는 행은 삭제하고 print하는 code (df1에 저장 후 print)
- 2) 모든 값이 NaN인 행을 삭제하고 print하는 code (df2에 저장 후 print)
- 3) NaN이 포함된 행을 삭제할때 데이터가 4개 이상 있어야 남기도록 하고 (df3) print하는 code를 작성 하시오

	name	age	math	physics	korean
0	Jason	42.0	4.0	25.0	25.0
1	NaN	NaN	NaN	NaN	NaN
2	Tina	36.0	NaN	NaN	NaN
3	Jake	NaN	2.0	62.0	62.0
4	Amy	73.0	3.0	70.0	70.0

- 5.1) 'exam1.csv' 파일을 읽고 (첫번째 행이 열이름이 되도록)
- 2) english의 결측치는 0으로 치환하고
- 3) math의 결측치는 math의 평균 값으로 치환하고
- 4) nclass의 결측치는 앞에 값으로 치환하고
- 5) print하는 code를 작성하시오
(inplace=True 사용)

	A	B	C	D	E	F
1	id	nclass	math	english	science	
2	1	1	50	98	50	
3	2	1	60	97	60	
4	3	1	45	86	78	
5	4	1	30	98	58	
6	5	2	25	80	65	
7	6	2		89	98	
8	7		80	90	45	
9	8	2	90	78	25	
10	9	3	20	98	15	
11	10	3	50	98	45	
12	11	3	65	65	65	
13	12	3	45		32	
14	13	4	46	98	65	
15	14		48	87	12	
16	15	4	75	56	78	
17	16	4	58	98		
18	17	5	65	68	98	
19	18	5	80	78	90	

6. 아래 code에서 정의하는 데이터 프레임에서

1) 중복 행을 제거 하고 print하는 code(df1)

2) c1,c3을 기준으로 중복행을 제거 하고 print하는 code(df2)
를 작성 하시오.

	c1	c2	c3
0	a	2	1
1	a	1	1
2	b	2	2
3	a	1	2
4	b	2	2

- 7.1) wine1.cse 파일을 읽고 (첫행부터 데이터로 읽음)
- 2) column 이름을 'country', 'year','feat1','feat2','feat3','feat4'로 지정하고
- 3) country의 누락 데이터는 france로 저장하고
- 4) feat1, feat2, feat3, feat4의 누락 데이터는 각 열의 평균으로 저장하고
- 5) 'country','year' 기준으로 중복데이터를 제거 하고
- 6) 'feat1-2' 열을 추가 하여 feat1과 feat2의 합의 값을 저장하고
- 7) 'year'를 기준으로 오름 차순으로 출력하는 code를 작성 하시오

	A	B	C	D	E	F	G
1	france	1970	7.4	0.7	0	1.9	
2	france	1990	7.8	0.88	0	2.6	
3	italy	2010		0.76	0.04	2.3	
4	chile	2000	11.2	0.28	0.56	1.9	
5	usa	1990	7.4	0.7	0		
6	france	1995	7.4	0.66	0	1.8	
7	italy	1980	7.9	0.6	0.06	1.6	
8		1990	7.3		0	1.2	
9	usa	2000	7.8	0.58	0.02	2	
10	usa	1995	7.5	0.5	0.36	6.1	
11	france	1995	7.4	0.66	0	1.8	
12							
13							

수고하셨습니다

jhmin@inhatec.ac.kr