

9. Pandas

1. Pandas 개념

Pandas는 금융 데이터 분석을 목적으로 개발

구조화된 데이터를 쉽고 빠르게 가공할 수 있는 자료형과 함수를 제공

Pandas 이름은 계량 경제학에서 동일한 조사 대상으로부터 여러 시점에 걸쳐 반복적으로 수집한 데이터를 지칭하는 패널 데이터와 파이썬 데이터 분석에서 가져옴

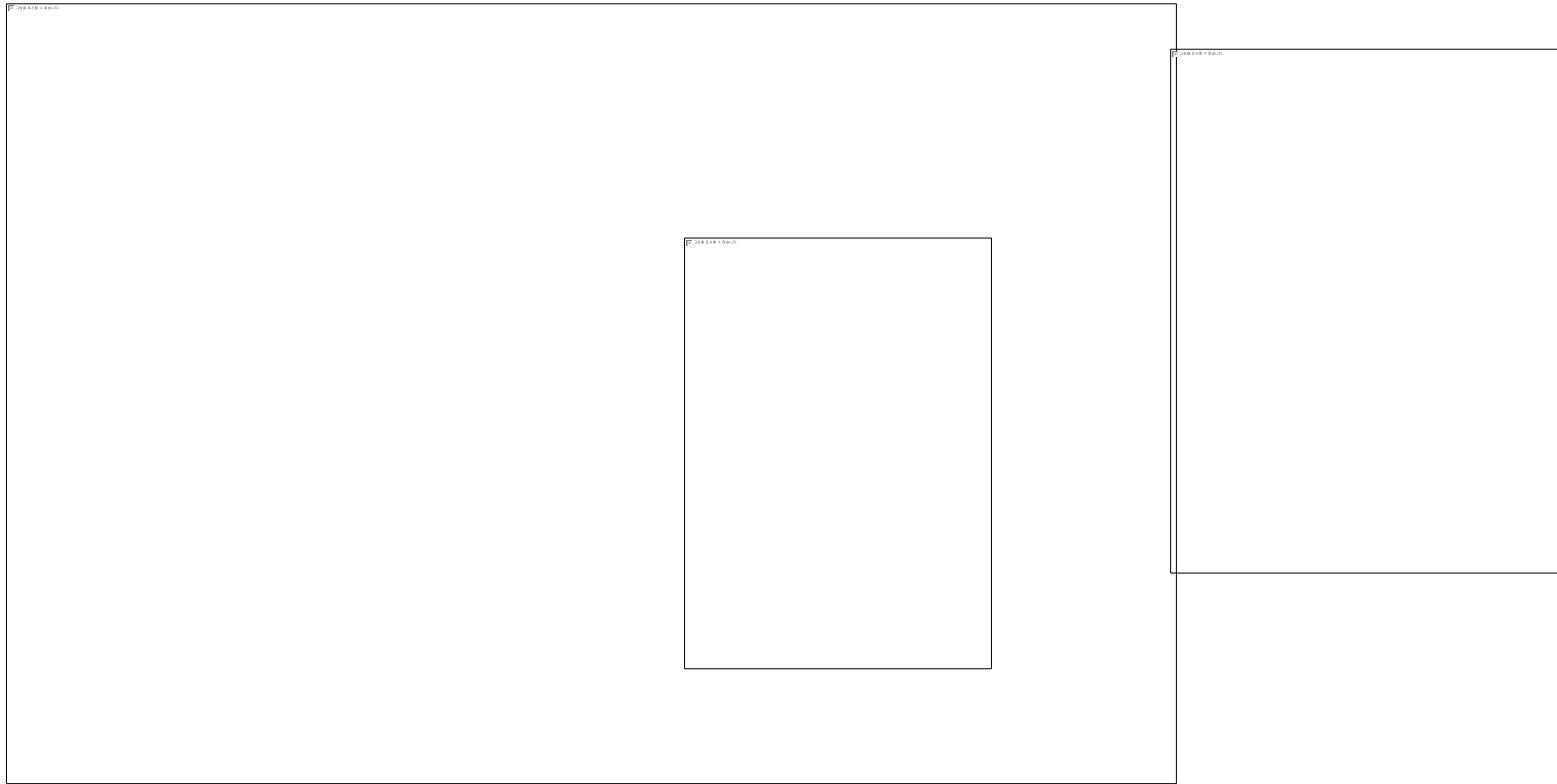
Pandas는 Numpy를 기반으로 구현했기 때문에 대부분의 함수가 Numpy와 유사.

파이썬 기반 데이터 시각화 라이브러리인 파이플롯과도 쉽게 호환되기 때문에 데이터 과학용 기본 라이브러리로 널리 활용됨.

Pandas의 개발자 웨스 매키니는 설계 당시부터 R언어의 `data.frame` 객체를 고려했기 때문에, Pandas도 시리즈와 데이터 프레임 자료형 객체를 제공

1. Pandas 개념

DataFrame , Series



1. Pandas 개념

빅데이터의 시대. 데이터 과학이라는 새로운 영역의 출현.

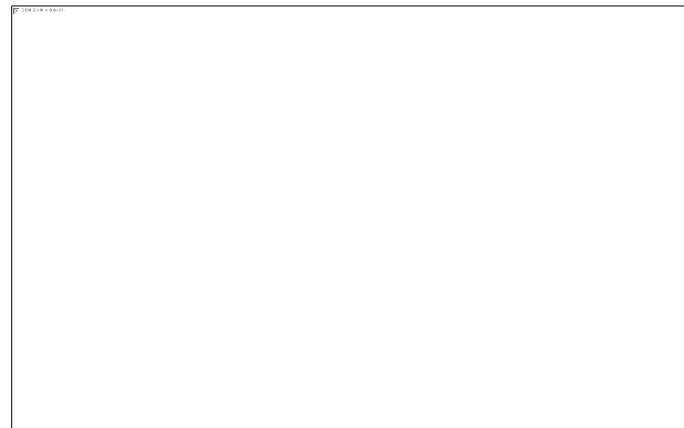
- 클라우드 컴퓨팅의 확산. 빅데이터 저장, 분석에 필요한 컴퓨팅 자원이 매우 저렴해짐.
- 컴퓨팅 파워의 대중화는 최적의 학습환경과 연구 인프라를 제공.

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.



2. 판다스 자료구조

- 목적

분석을 위해 다양한 소스(source)로부터 수집하는 데이터는 형태나 속성이 매우 다양하다. 특히, 서로 다른 형식을 갖는 여러 종류의 데이터를 컴퓨터가 이해할 수 있도록 동일한 형식을 갖는 구조로 통합할 필요가 있다. 판다스의 일차적인 목적은 **형식적으로 서로 다른 여러 가지 유형의 데이터를 공통의 포맷으로 정리**하는 것이다.

- 종류

판다스는 **시리즈(Series)**와 **데이터프레임(DataFrame)**이라는 **구조화된 데이터 형식**을 제공한다. 서로 다른 종류의 데이터를 한곳에 담는 그릇(컨테이너)이 된다. 다만, 시리즈는 1차원 배열이고, 데이터프레임이 2차원 배열이라는 점에서 차이가 있다. 특히, 행과 열로 이루어진 2차원 구조의 데이터프레임은 데이터 분석 실무에서 자주 사용된다.



2. 판다스 자료구조

2-1. 시리즈

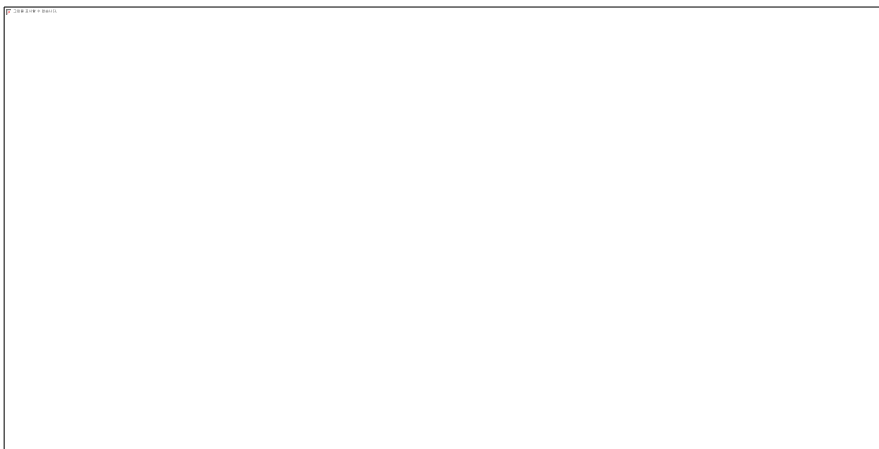
- 시리즈 만들기

1) 딕셔너리와 시리즈의 구조가 비슷하기 때문에, **딕셔너리를 시리즈로 변환**하는 방법을 많이 사용.

2) 판다스 내장 함수인 Series()를 이용하고, 딕셔너리를 함수의 매개변수(인자)로 전달.



3) 딕셔너리의 키(k)는 시리즈의 인덱스에 대응하고, 딕셔너리의 각 키에 매칭되는 값(v)이 시리즈의 데이터 값(원소)로 변환.



2. 판다스 자료구조

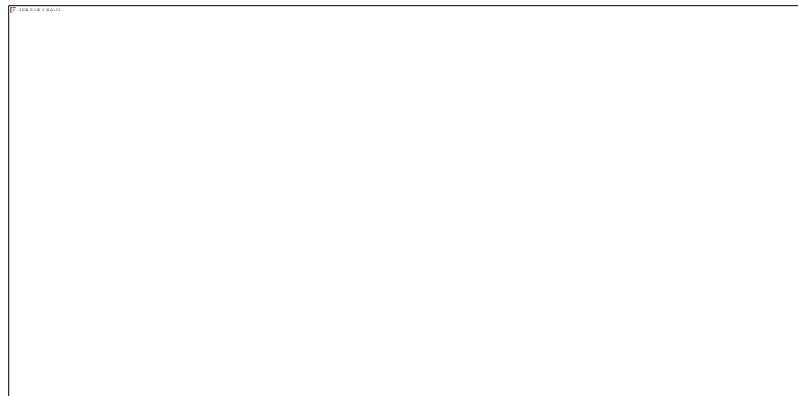
2-1. 시리즈

- 인덱스 구조

- 1) 인덱스는 자기와 짝을 이루는 **원소의 순서와 주소를 저장**.
- 2) 인덱스를 잘 활용하면 데이터 값의 탐색, 정렬, 선택, 결합 등 데이터 조작을 쉽게 할 수 있다.
- 3) **인덱스의 종류 (2가지)**
 - ① 정수형 위치 인덱스(integer position)
 - ② 인덱스 이름(index name) 또는 인덱스 라벨(index label)

파이썬 리스트를 시리즈로 변환해 본다. 단, 딕셔너리의 키처럼 인덱스로 변환될 값이 없다. 따라서, 인덱스를 별도로 정의하지 않으면, 디폴트로 정수형 위치 인덱스(0, 1, 2, ...)가 자동 지정된다. 다음 예제에서는 0 ~ 4 범위의 정수값이 인덱스로 지정된다.

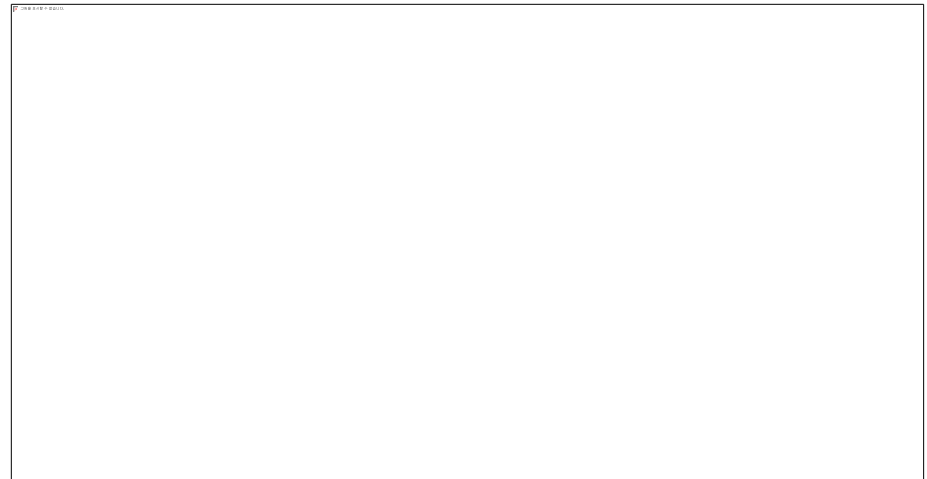
① 시리즈 만들기



2-1 시리즈

• 원소 선택

- 1) 인덱스를 이용하여, 시리즈의 원소를 선택.
- 2) 하나의 원소를 선택하거나, 여러 원소를 한꺼번에 선택 가능.
- 3) 인덱스 범위를 지정하여 여러 개의 원소 선택 가능.
- 4) 인덱스의 유형에 따라 사용법이 조금 다르다.
 - 정수형 인덱스: 대괄호([]) 안에 숫자 입력. (0부터 시작)
 - 인덱스 이름 (라벨): 대괄호([]) 안에 이름과 함께 따옴표를 입력. 큰 따옴표(" "), 작은 따옴표(' ') 모두 사용.



2-1 시리즈

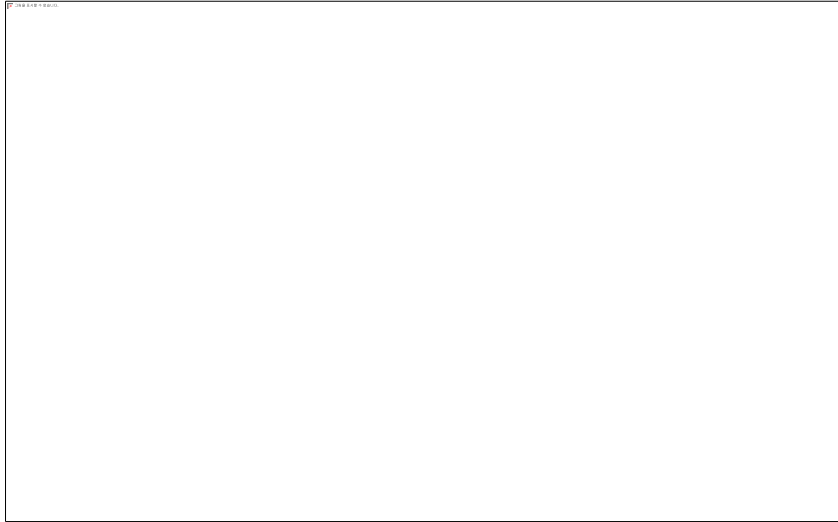
• 원소 선택 (계속)

③ 여러 개의 원소를 선택(인덱스 리스트 활용)

인덱스 리스트를 활용하는 방법이다.

여러 개의 인덱스를 리스트 형태로 대괄호([]) 안에 입력하면, 짝을 이루는 원소 데이터를 모두 반환한다.

정수형 위치 인덱스는 0부터 시작하기 때문에 2번째 인덱스 이름인 '생년월일'은 정수형 인덱스 1을 사용하고, 3번째 인덱스 이름인 '성별'은 정수형 인덱스 2를 사용한다.

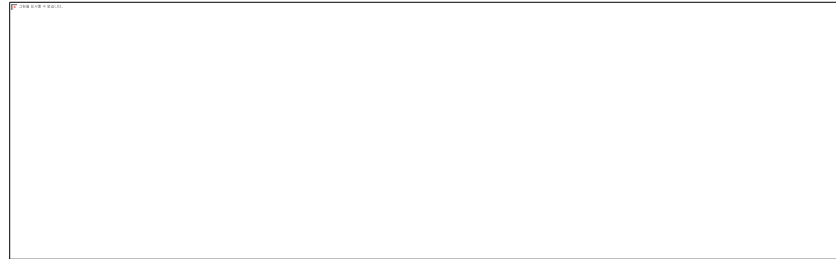
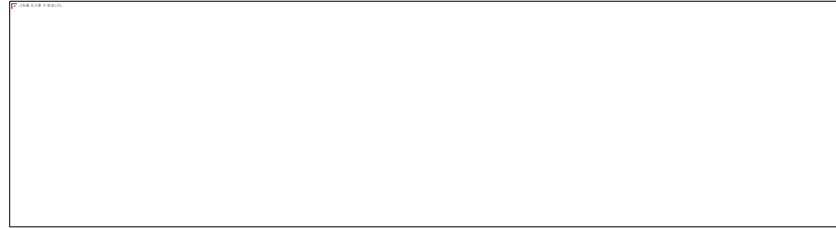


④ 여러 개의 원소를 선택(인덱스 범위 지정)

인덱스 범위를 지정하여 선택하는 방법이다.

22번 라인의 `sr[1 : 2]` 에서 정수형 위치 인덱스를 사용할 때는, 범위의 끝(2)이 포함되지 않는다('성별' 불포함).

그러나, 21번 라인의 `sr['생년월일' : '성별']` 과 같이, 인덱스 이름을 사용하면 범위의 끝('성별')이 포함된다.



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.

- 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 속성이나 범주를 나타내는데, 보통 변수(variable)로 활용된다

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.

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.

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.

2-2 데이터프레임

• 행 인덱스/열 이름 설정

: 데이터프레임의 행 인덱스와 열 이름을 사용자가 지정 가능.

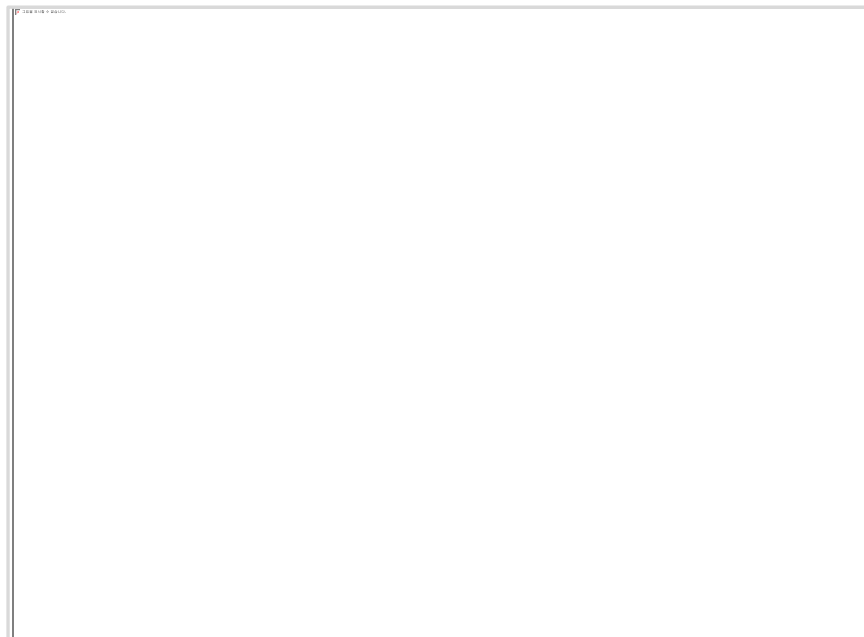
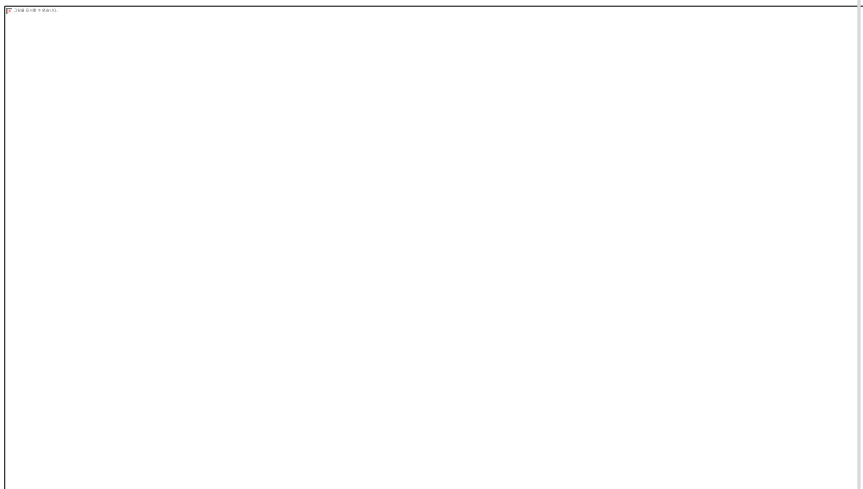


예제 1-5

① 데이터프레임을 만들 때 설정

'3개의 원소를 갖는 리스트' 2개를 원소로 갖는 리스트(2차원 배열)로 데이터프레임을 만든다. 이때, 각 리스트가 행으로 변환되는 점에 유의한다.

- index 옵션: ['준서', '예은'] 배열을 지정
- columns 옵션: ['나이', '성별', '학교'] 배열을 지정



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.



2-2 데이터프레임

• 행 인덱스/열 이름 설정 (계속)

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.

예제 1-6

2-2 데이터프레임

• 행/열 삭제

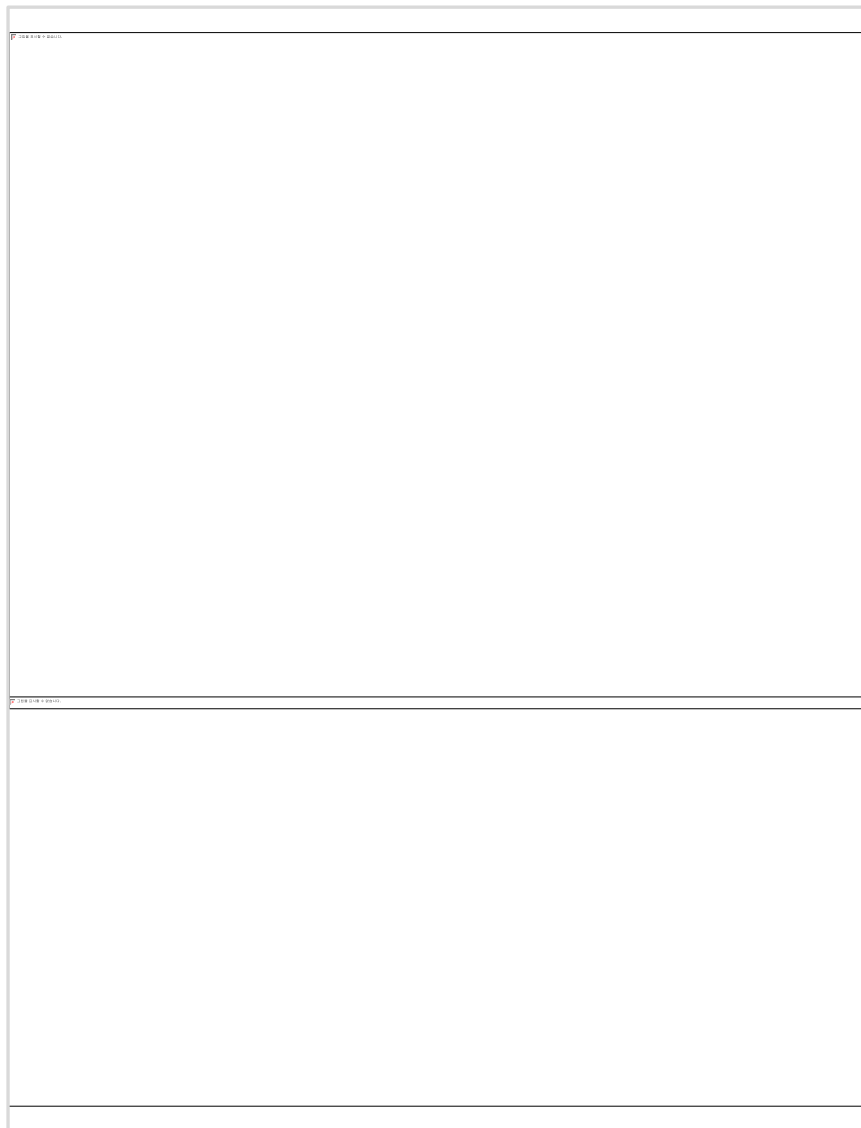
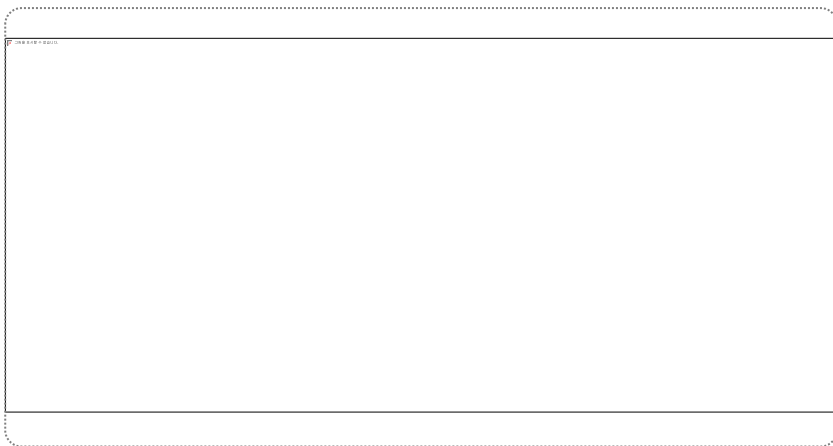
drop() 메소드에 행을 삭제할 때는 축(axis) 옵션으로 axis=0을 입력하거나, 별도로 입력하지 않는다.

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



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

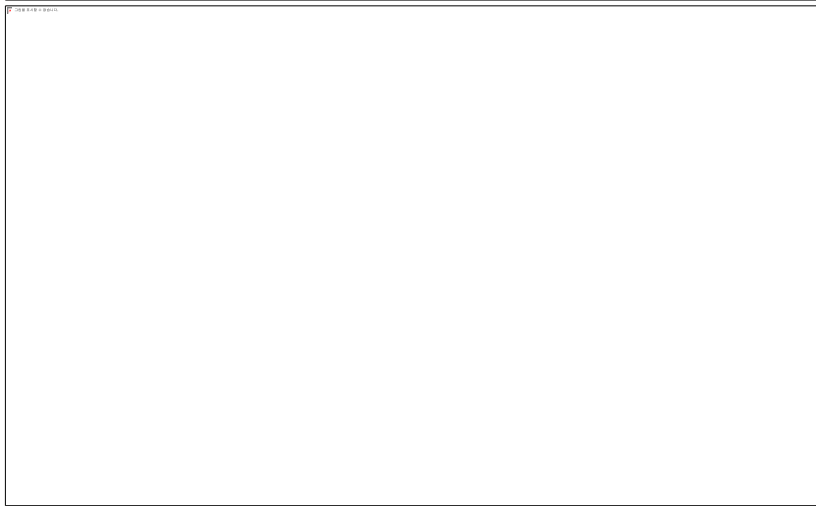
① 행 삭제



2-2 데이터프레임

- 행/열 삭제 (계속)

- ② 열 삭제



2-2 데이터프레임

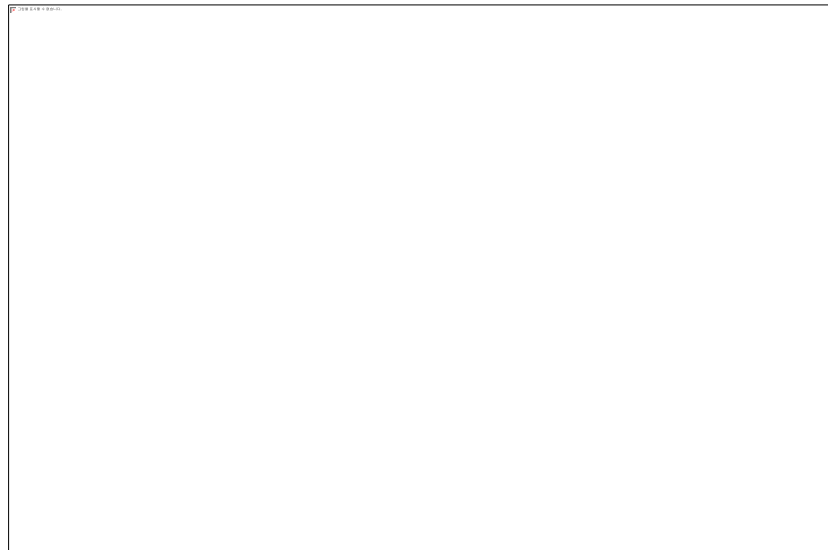
• 행 선택

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



예제 1-9

① 1개의 행 선택



데이터프레임의 첫 번째 행에는 '서준' 학생의 과목별 점수 데이터가 입력되어 있다. '서준' 학생의 과목별 점수 데이터를 행으로 추출하면 시리즈 객체가 반환된다.

`loc` 인덱서를 이용하려면 '서준'이라는 인덱스 이름을 직접 입력하고, `iloc`을 이용할 때는 첫 번째 정수형 위치를 나타내는 0을 입력한다. 각각 반환되는 값을 `label1` 변수와 `position1` 변수에 저장, 출력하면 같은 결과를 갖는다.

2-2 데이터프레임

• 행 선택 (계속)

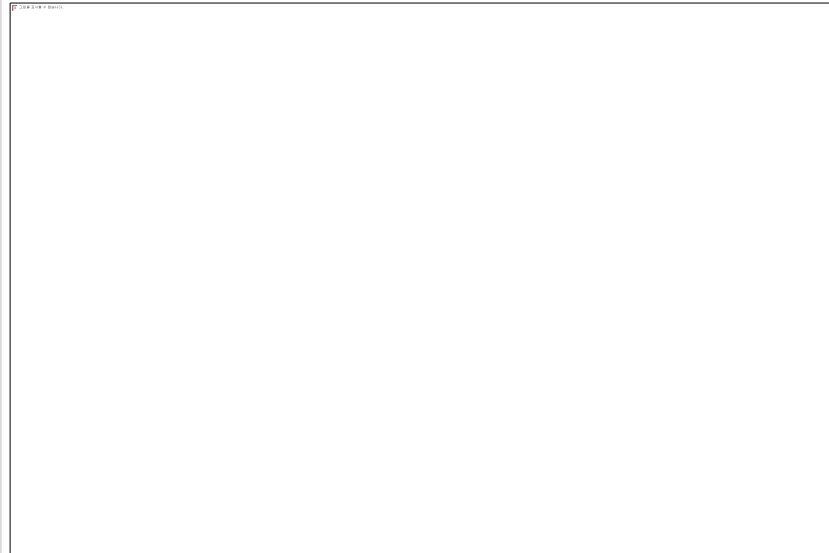
② 여러 개의 행을 선택(인덱스 리스트 활용)



2개 이상의 행 인덱스를 배열로 입력하면, 매칭되는 모든 행 데이터를 동시에 추출한다.

데이터프레임 df의 첫번째와 두번째 행에 있는 '서준', '우현' 학생을 인덱싱으로 선택해 본다. loc 인덱서는 ['서준', '우현'] 과 같이 인덱스 이름을 배열로 전달하고, iloc을 이용할 때는 [0, 1] 과 같이 정수형 위치를 전달한다. 이때, label2 변수와 position2 변수에 저장된 값은 같다.

③ 여러 개의 원소를 선택(인덱스 범위 지정)

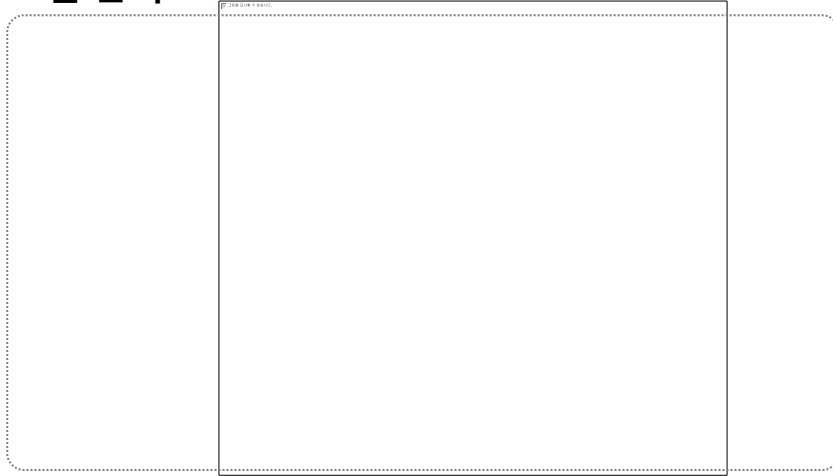


행 인덱스의 범위를 지정하여 여러 개의 행을 동시에 선택하는 슬라이싱 기법을 사용한다.

단, 인덱스 이름을 범위로 지정한 label3의 경우에는 범위의 마지막 값인 '우현' 학생의 점수가 포함되지만, 정수형 위치 인덱스를 사용한 position3에는 범위의 마지막 값인 '우현' 학생의 점수가 제외된다.

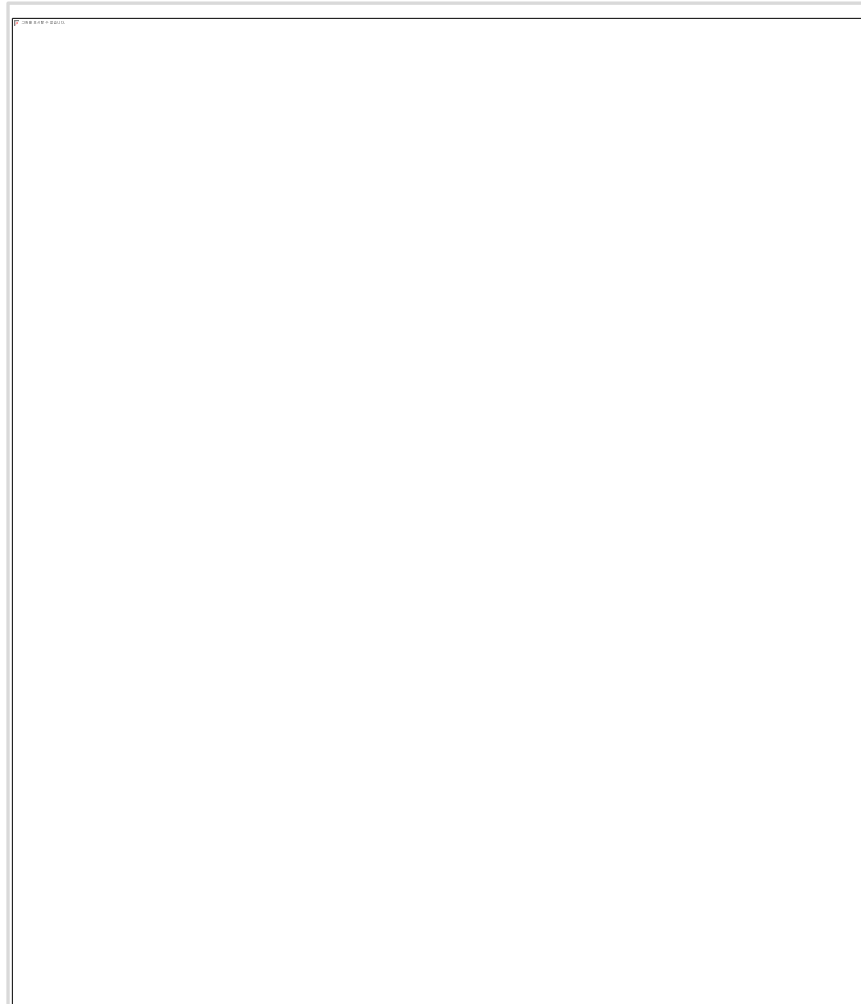
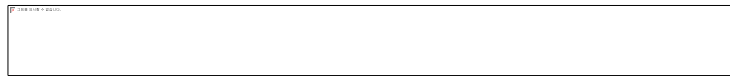
2-2 데이터프레임

• 열 선택



예제 1-10

① 1개의 열 선택



`type()` 함수를 사용하여, 데이터프레임에서 1개의 열을 선택할 때 반환되는 객체의 자료형을 확인하면 시리즈이다.

2-2 데이터프레임

- 열 선택 (계속)

- ② n개의 열 선택 (리스트 입력)

The diagram shows a large rectangular box representing a data frame. At the top, there is a smaller rectangular box representing a list of column indices or names. An arrow points from this list box to the main data frame box, indicating the selection of columns. The main box is empty, representing the data frame after column selection.

이 때, 반환되는 객체의 자료형을 확인하면 데이터프레임이다.

- 원소 선택

The diagram shows a large rectangular box representing a data frame. It is empty, representing the data frame before element selection.

예제 1-11

The diagram shows a large rectangular box representing a data frame. It is empty, representing the data frame before element selection.

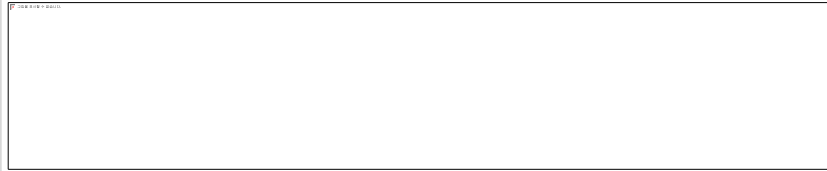

2-2 데이터프레임

• 원소 선택 (계속)




① 1개의 원소를 선택



② 2개 이상의 원소를 선택 (시리즈)



③ 2개 이상의 원소를 선택 (데이터프레임)



2-2 데이터프레임

• 열 추가

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

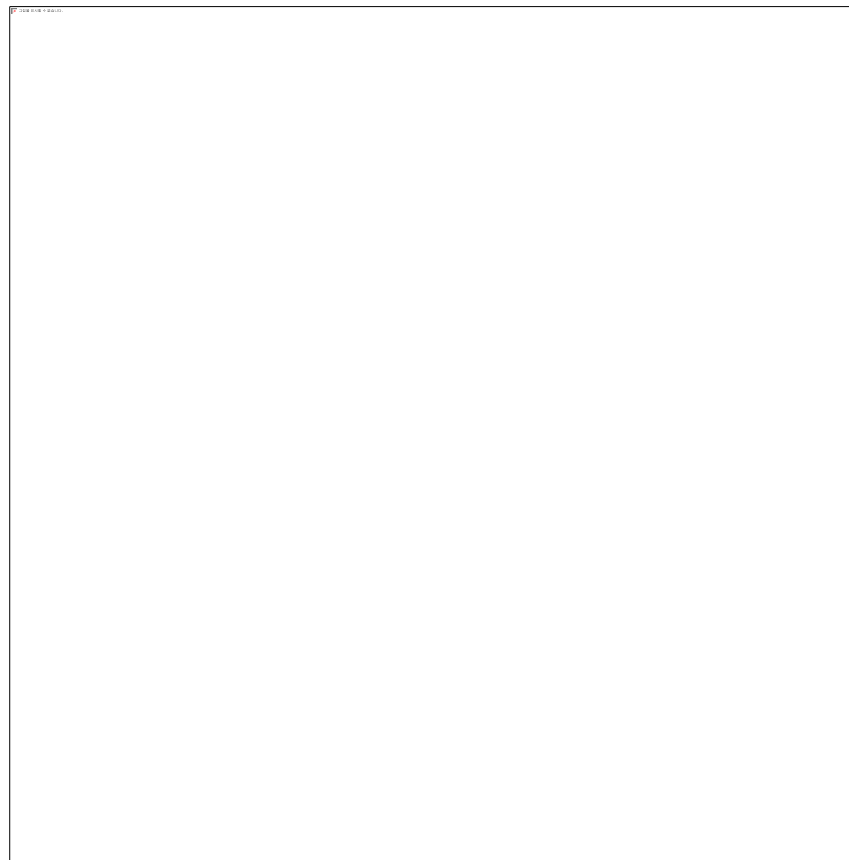


- 2) 이때 모든 행에 동일한 값이 입력되는 점에 유의.



예제 1-12

다음 예제에서 '국어' 열을 새로 추가하는데, 모든 학생들의 국어 점수가 동일하게 80점으로 입력되는 과정을 보여준다.



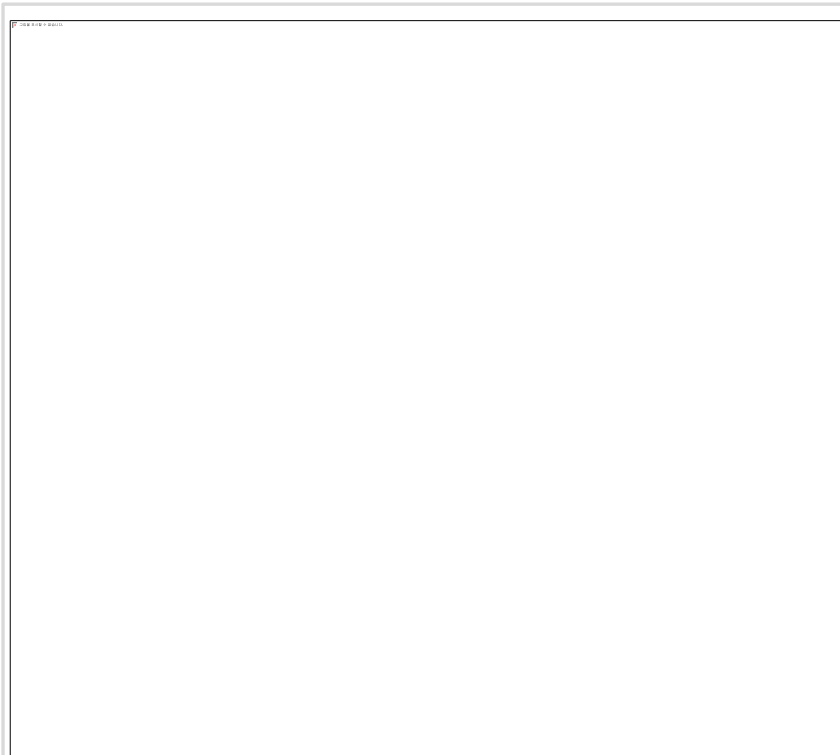
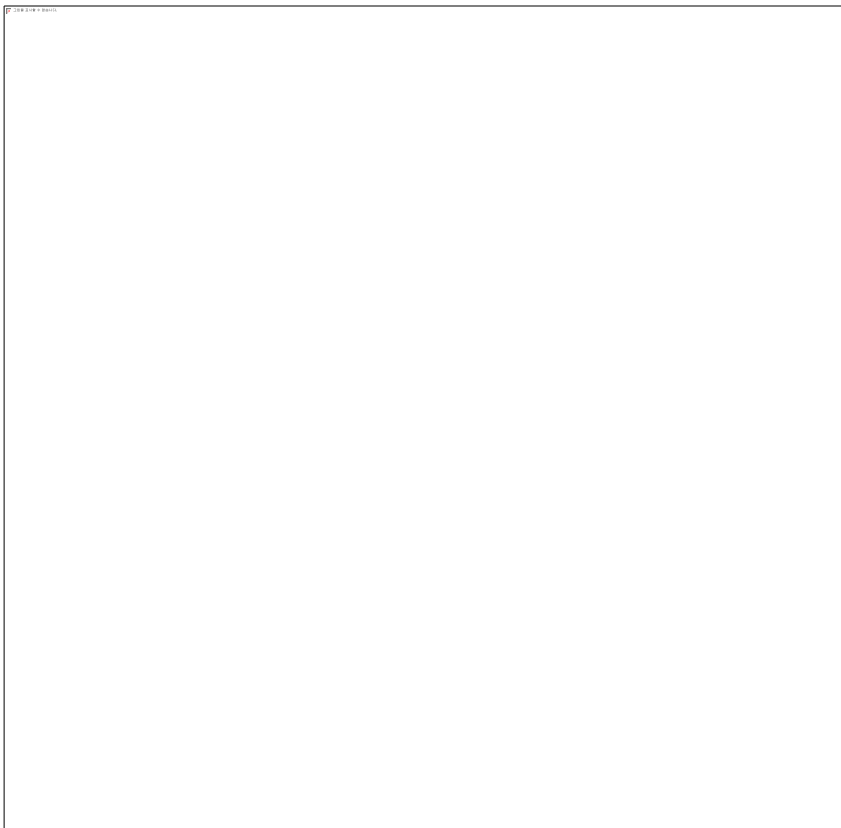
2-2 데이터프레임

2-2 데이터프레임

- 원소 값 변경

: 원소를 선택하고 새로운 데이터 값을 지정.

① 1개의 원소를 변경



앞의 예제에서 '서준' 학생의 '체육' 점수를 선택하는 여러 방법을 시도하였다. 각 방법을 비교하기 위해, 각기 다른 점수를 새로운 값으로 입력하여 원소를 변경하였다. (변경된 값을 원으로 표시)

2-2 데이터프레임

- 원소값 변경 (계속)

- ② 1개 이상의 원소를 변경

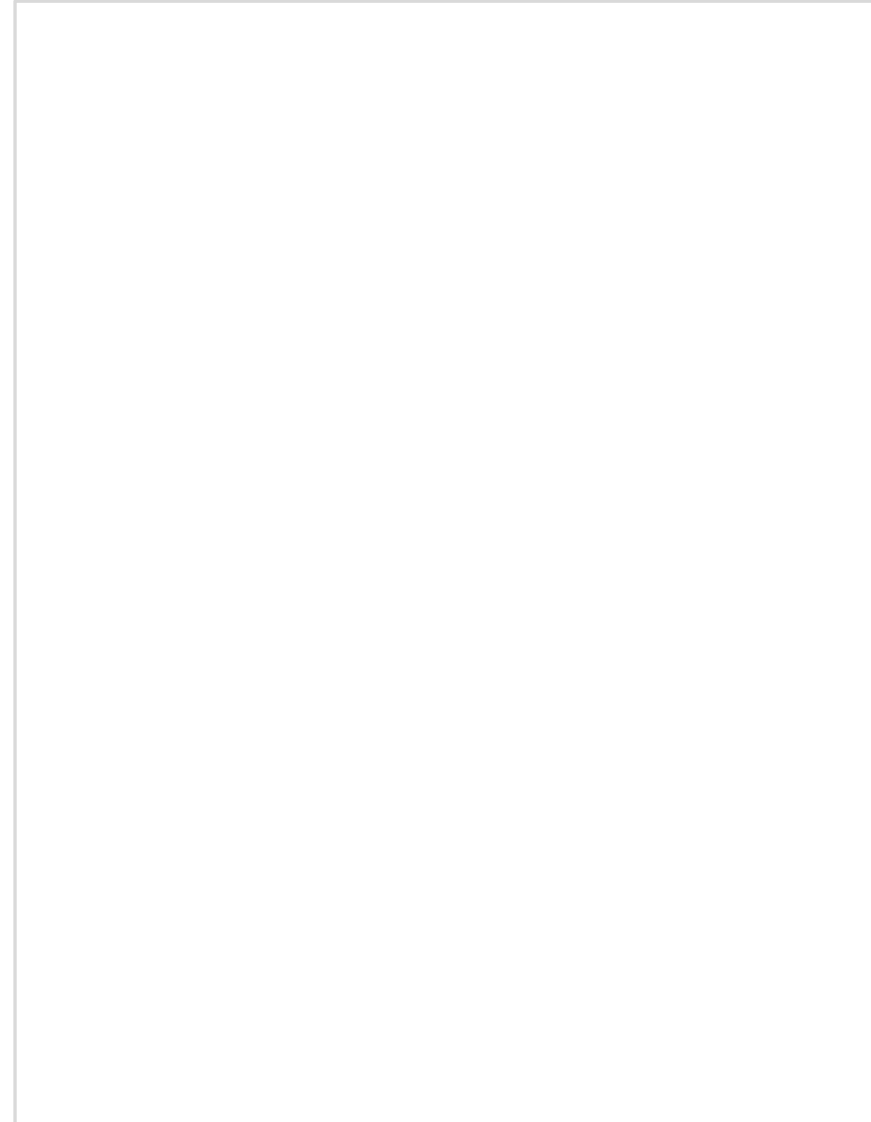
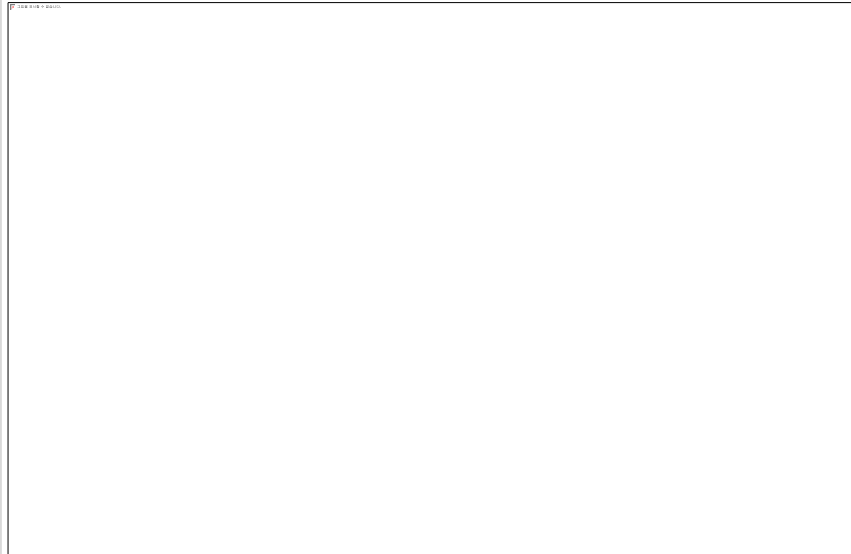
- 행, 열의 위치 바꾸기

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.

예제 1-15

2-2 데이터프레임

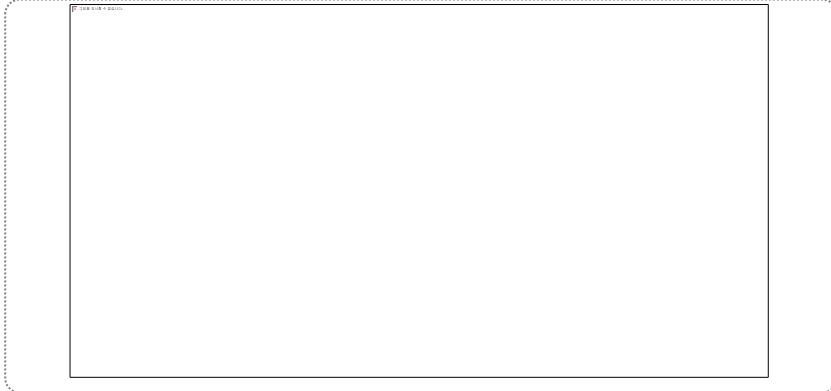
- 행, 열의 위치 바꾸기 (계속)



3. 인덱스 활용

- 특정 열을 행 인덱스로 설정

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



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.

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

예제의 12행에서 새롭게 추가된 'r3', 'r4' 인덱스에 해당하는 모든 열에 대해 NaN 값이 입력된다. 이럴 경우, 데이터가 존재하지 않는다는 뜻의 NaN 대신 유효한 값으로 채우려면 예제의 21행과 같이 `fill_value` 옵션에 원하는 값(0)을 입력한다.

NaN은 "Not a Number" 라는 뜻이다. 유효한 값이 존재하지 않는 누락 데이터를 말한다.

3. 인덱스 활용

• 행 인덱스 초기화

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

IN Jupyter Notebook

IN Jupyter Notebook

• 행 인덱스를 기준으로 데이터프레임 정렬

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

IN Jupyter Notebook

IN Jupyter Notebook

IN Jupyter Notebook

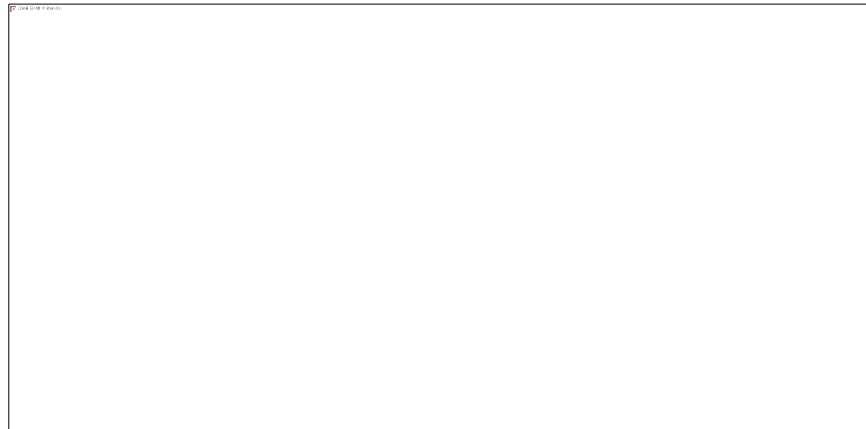
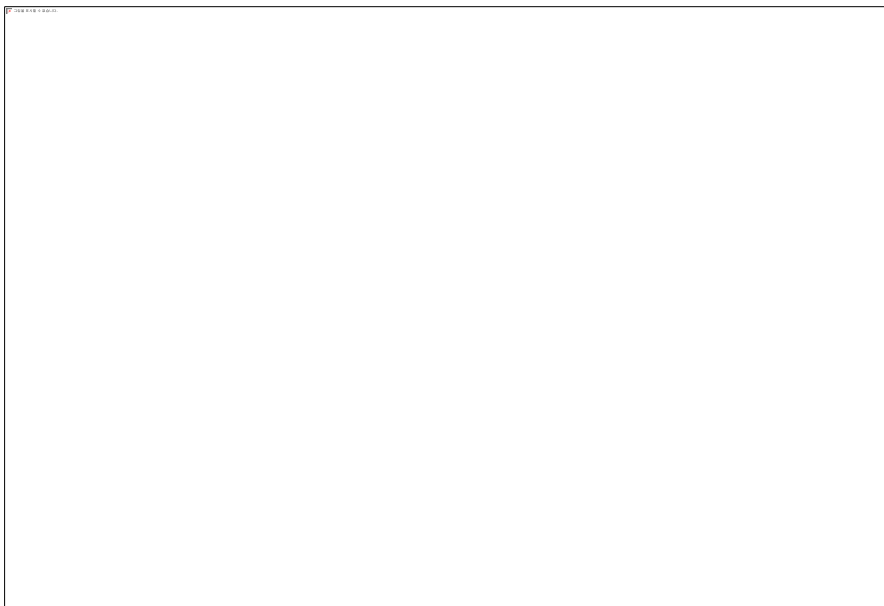
4. 산술연산

- 판다스 객체의 산술연산은 내부적으로 **3단계 프로세스**를 거친다.

- ① 행/열 인덱스를 기준으로 모든 원소를 정렬한다.
- ② 동일한 위치에 있는 원소끼리 일대일로 대응시킨다.
- ③ 일대일 대응이 되는 원소끼리 연산을 처리한다. 이때, 대응되는 원소가 없으면 NaN으로 처리한다.

4-1. 시리즈 연산

- 시리즈 vs. 숫자

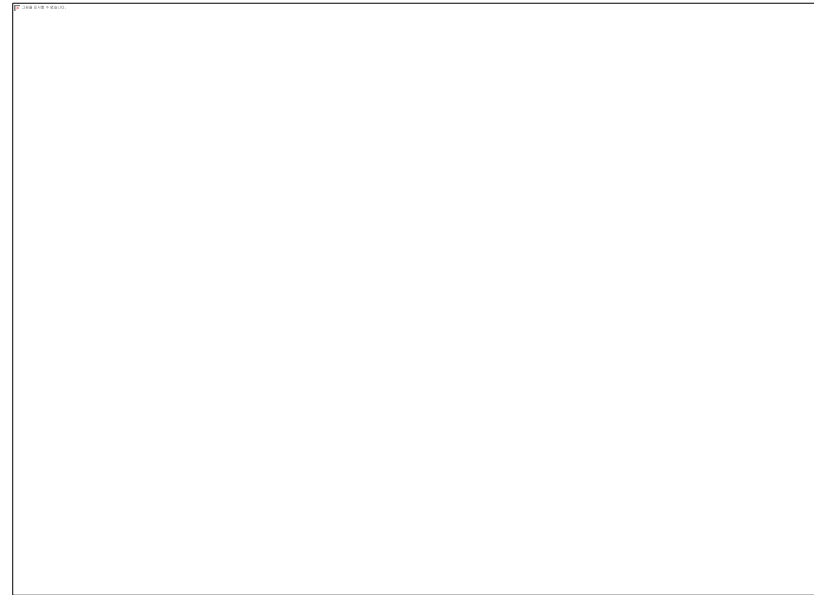
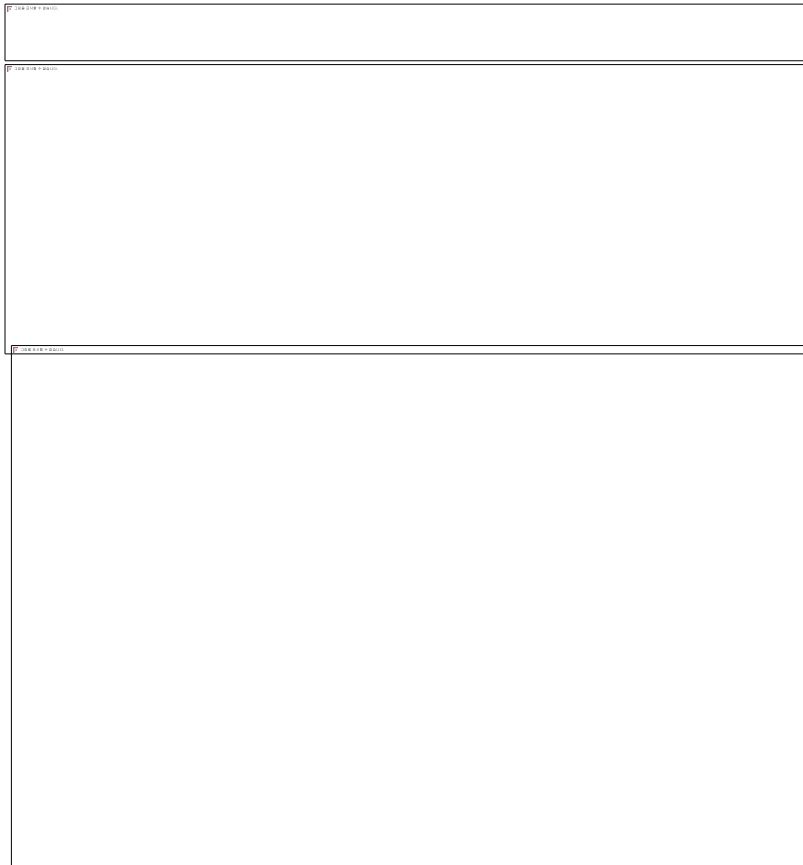


시리즈 객체에 어떤 숫자를 더하면, 시리즈의 개별 원소에 각각 숫자를 더하고 계산한 결과를 시리즈 객체로 반환한다. 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능하다. 앞의 예제에서는 시리즈 객체의 각 원소를 200으로 나누는 과정을 살펴본다.

4-1 시리즈 연산

• 시리즈 vs. 시리즈

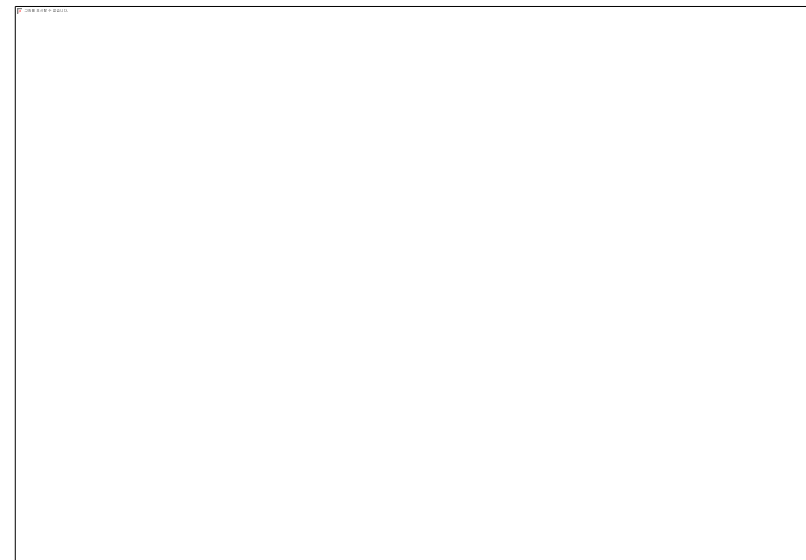
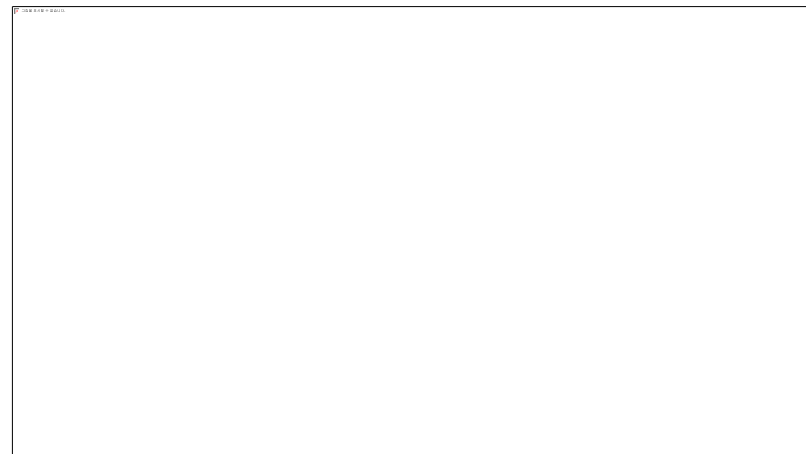
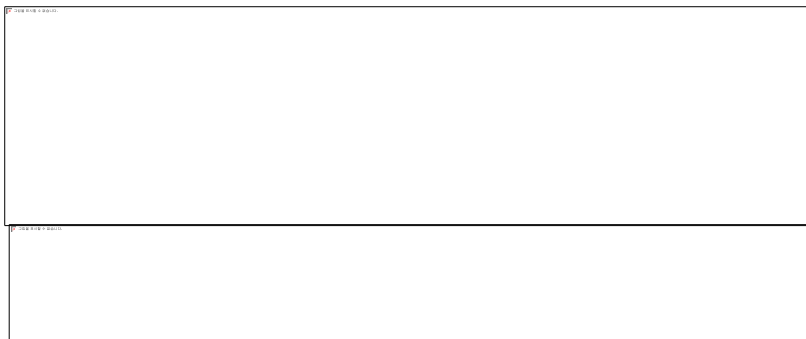
- 시리즈와 시리즈 사이에 사칙연산 처리.
- 같은 인덱스를 가진 원소끼리 계산.
- 해당 인덱스에 연산 결과를 매칭하여 새 시리즈를 반환.



4-1 시리즈 연산

• 시리즈 vs. 시리즈 (계속)

- 두 시리즈의 원소 개수가 다르거나, 혹은 시리즈의 크기가 같더라도 인덱스 값이 다를 수 있다. 이런 경우, 유효한 값이 존재하지 않는다는 의미로 NaN으로 처리한다.
- 한편, 같은 인덱스가 양쪽에 모두 존재하여 서로 대응되어도 어느 한 쪽의 데이터 값이 NaN인 경우가 있다. 이때, 연산의 대상인 데이터가 존재하지 않기 때문에, NaN으로 처리한다.

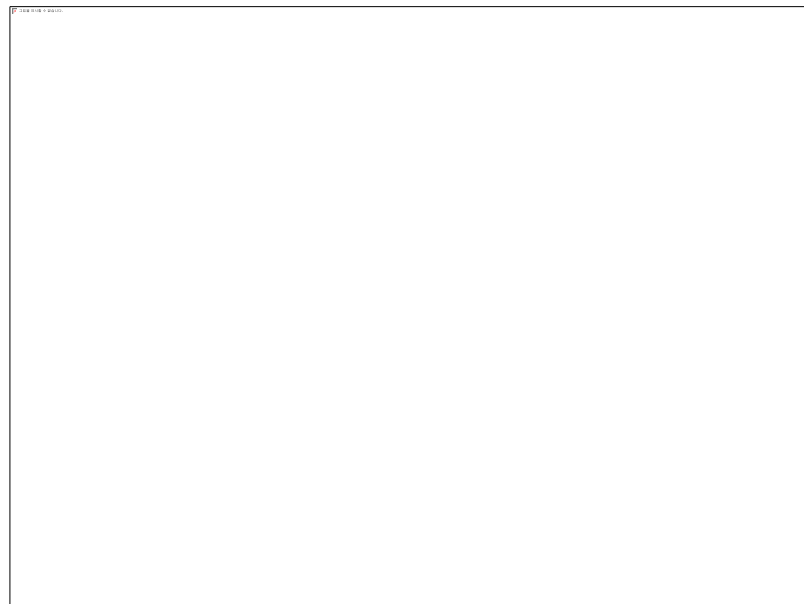
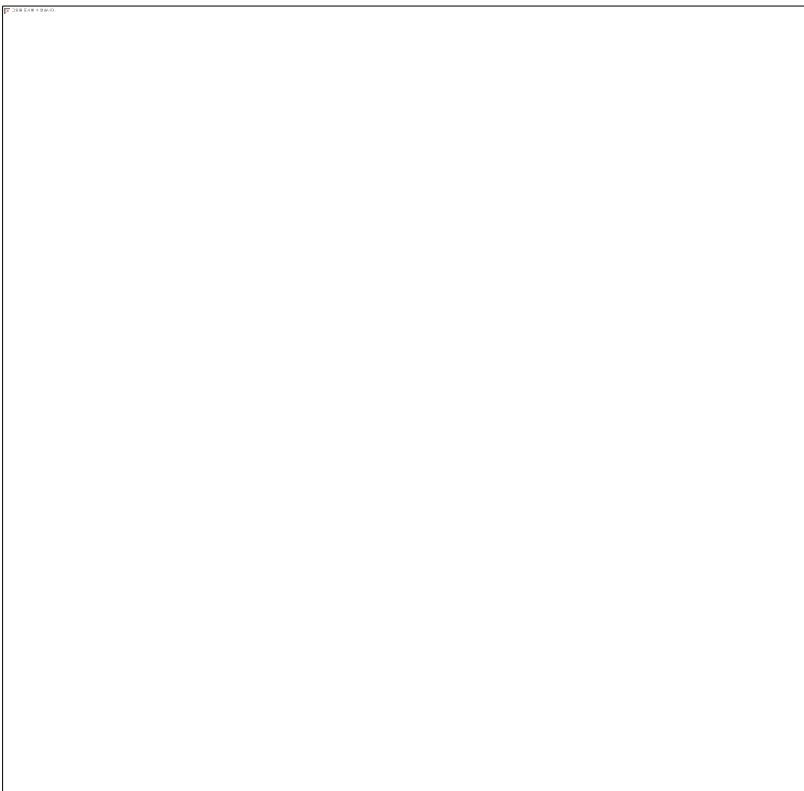


4-1 시리즈 연산

• 연산 메소드 활용

- 객체 사이에 공통 인덱스가 없는 경우에 NaN으로 반환.
- 이런 상황을 피하려면 연산 메소드에 fill_value 옵션을 설정.

※ 다음 예제는 누락 데이터 NaN 대신 숫자 0을 입력하는 것을 예시로 설명.

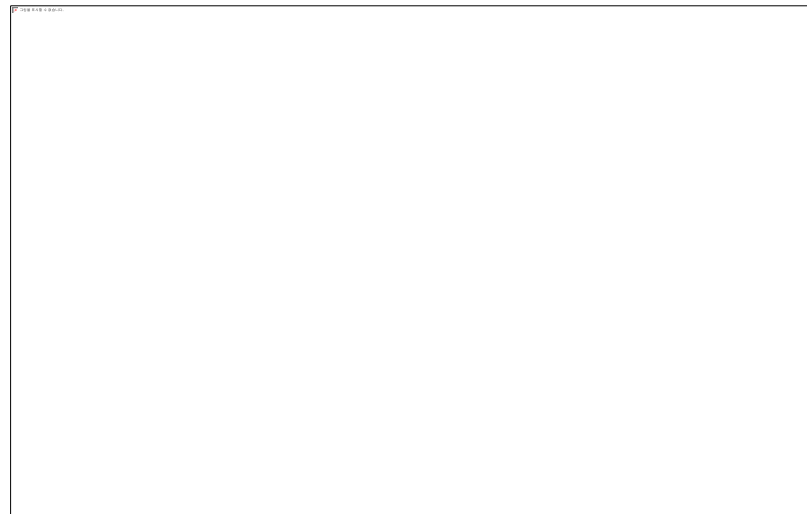


4-2 데이터 프레임 연산

데이터프레임은 여러 시리즈가 한데 모인 것이므로, 시리즈 연산을 확장하는 개념으로 이해한다. 먼저 행/열 인덱스를 기준으로 정렬하고, 일대일 대응되는 원소끼리 연산한다.

• 데이터프레임 vs. 숫자

- 데이터프레임에 어떤 숫자를 더하면, 모든 원소에 숫자를 더한다. 덧셈, 뺄셈, 곱셈, 나눗셈 모두 가능하다.
- 기존 데이터프레임의 형태를 그대로 유지한 채, 원소 값만 새로운 값으로 바뀐다.



※ 타이타닉('titanic') 데이터셋

Seaborn 라이브러리에서 제공하는 데이터셋으로, 타이타닉호 탑승자에 대한 인적사항과 구조 여부 등을 정리한 자료이다.

`load_dataset()` 함수로 불러온다.

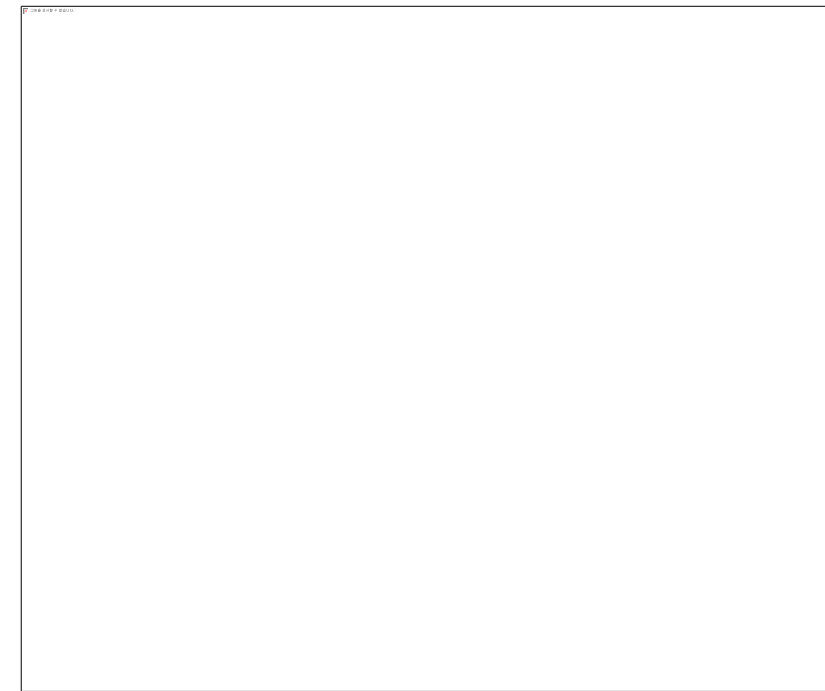
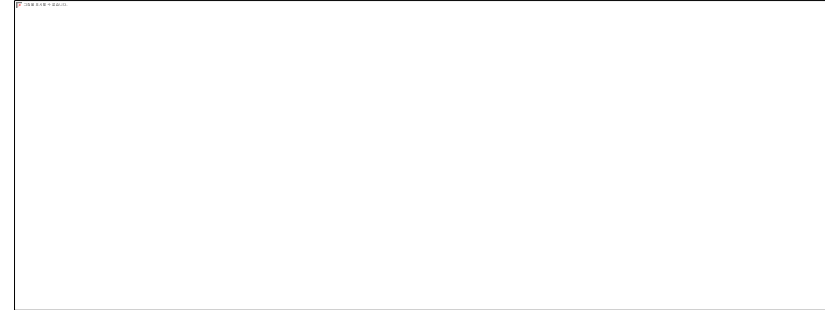
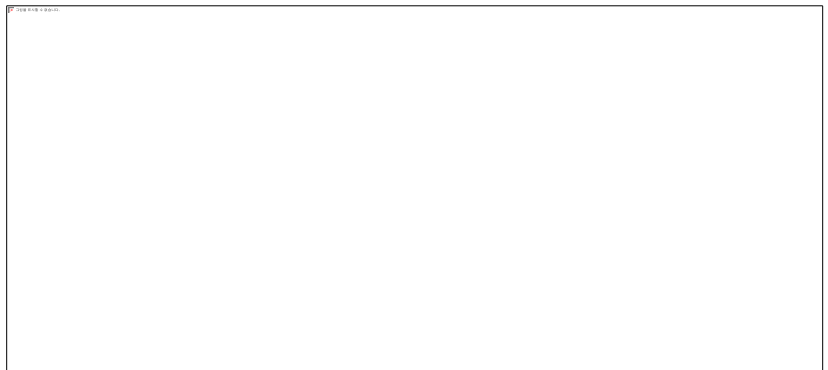
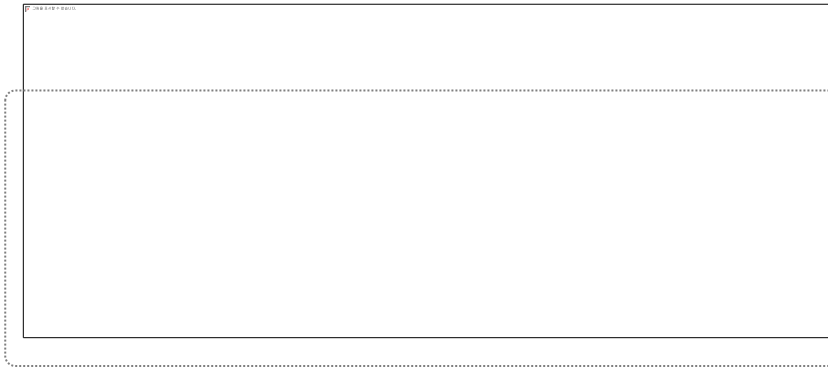
[Seaborn 내장 데이터셋의 종류]

'anscombe', 'attention', 'brain_networks', 'car_crashes',
'diamonds', 'dots', 'exercise', 'flights', 'fmri', 'gammas', 'iris',
'mpg', 'planets', 'tips', 'titanic'

4-2 데이터 프레임 연산

• 데이터프레임 vs. 데이터프레임

- 각 데이터프레임의 같은 행, 같은 열 위치에 있는 원소끼리 계산한다. 이처럼 동일한 위치의 원소끼리 계산한 결과값을 원래 위치에 다시 입력하여 데이터프레임을 만든다.



1. Pandas 개념

DataFrame

```
df = pd.DataFrame({  
    "Name": ["Braund, Mr.Owen Harris",  
            "Allen, Mr. William Henry",  
            "Bonnell, Miss. Elizabeth"],  
    "Age": [22, 35, 58],  
    "Sex": ["male", "male", "female"]})
```

1. Pandas 개념



1. Pandas 개념

csv 파일 읽어 오기

```
Titanic = pd.read_csv('../data/titanic.csv', dtype='unicode')
```

Excel 파일 읽어오기

```
titanic = pd.read_excel('../data/titanic.xlsx', sheet_name='passengers')
```

2. csv 파일

command 에서 파일 명 입력받아 파일 복사하기

```
import sys #command 라인에서 입력 받기.
input_file = sys.argv[1] #jeju1.csv
output_file = sys.argv[2] #jeju1_bak.csv
with open(input_file, 'r', newline="", encoding="utf8") as filereader :
    with open(output_file, 'w', newline="", encoding="utf8") as filewriter :
        header = filereader.readline() #한줄 읽기
        print(type(header))
        print(header)
        header = header.strip() # 공백제거.
        print(header)
        header_list = header.split(",") #, 기준으로 문자열 분리. csv파일은 , 기준으로 셀로 분리
        print(header_list)
        #map(str, header_list) : header_list 리스트의 요소들을 문자열로 변환
        filewriter.write(",".join(map(str, header_list))+ "\r\n")
        for row_list in filereader : #입력 스트림에서 파일을 한줄씩 읽기
            filewriter.write(row_list)
```

2. csv 파일

파일을 읽기 위한 모듈 codecs 사용

```
import codecs
filename = "jeju1.csv"
csv = codecs.open(filename, "r", "utf8").read()
data = [] #한줄을 ,분리한 내용을 저장.
rows = csv.split("\r\n") #line 별로 나눠서 리스트로 생성
for row in rows :
    if row == "" : continue
    cells = row.split(",")
    data.append(cells)
outfp = open("jeju1.txt", "w", encoding='UTF8')
for c in data:
    print(c[0], c[1], c[2])
    outfp.write(" ".join(map(str,c)) + "\n")
outfp.flush()
outfp.close()
```

2. csv 파일

pandas를 이용하여 csv 파일 읽기

```
import pandas as pd

infile = "jeju1.csv"
df = pd.read_csv(infile)
print(df)
print(type(df))
#경도만 출력
print(df["LON"])
#위도만 출력
print(df["LAT"])
#장소만 출력
print(df["장소"])

#3행만 출력
print(df.iloc[3])
```


3. Pandas 이해

행 선택

```
import pandas as pd

df = pd.DataFrame({"A": [1, 4, 7], "B": [2, 5, 8], "C": [3, 6, 9]})
print(df)
print(df["A"])
#행 선택
# iloc[인덱스] : 인덱스 기준으로 조회
# loc[이름기준] : 행의 이름 기준으로 조회
print("행의 인덱스값으로 조회하기")
print("df.iloc[0]=", df.iloc[0])
print("df.iloc[1]=", df.iloc[1])
print("df.iloc[2]=", df.iloc[2])

print("df.loc[0]=", df.loc[0])
print("df.loc[1]=", df.loc[1])
print("df.loc[2]=", df.loc[2])
```

```
df = pd.DataFrame(data=([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),
                  index=[2, "A", 4], columns=[51, 52, 54])
print(df)
print(df.iloc[2])
print(df.loc[2])
```

3. Pandas 이해

supplier_data.csv 파일을 pandas를 이용하여 읽기

```
import pandas as pd

infile = "supplier_data.csv"
df = pd.read_csv(infile)
print(df);

importdate = ["1/20/14","1/30/14"]
#isin : 주어진 값이 맞는지? boolean값으로 리턴.
print(df["Purchase Date"].isin(importdate))
df_inset = df.loc[df["Purchase Date"].isin(importdate),:]
print(df_inset)
print(df["Invoice Number"].str.startswith("920-"))
df_inset = df.loc[df["Invoice Number"].str.startswith("920-"),:]
print(df_inset)
print(type(df_inset))
#pandas 데이터를 csv 파일로 생성.
df_inset.to_csv("pandas_out3.csv",index=False)
```

3. Pandas 이해

pandas 파일의 행과열 선택

```
import pandas as pd
infile = "supplier_data.csv"
df = pd.read_csv(infile)

df_col = df.iloc[:,[0,3]] #모든행, 0번,3번열 조회
print("df.iloc[:,[0,3]] =>")
print(df_col)

df_col = df.iloc[0:4,0:3] #0~3인덱스행, 0~2인덱스
print("df.iloc[0:4,0:3] =>")
print(df_col)
```

```
#Invoice Number값이 920- 시작하는 행의
# Invoice Number, Cost 컬럼만 조회하기
# 조건을 만족하는 데이터를 pandas_out4.csv 파일로
저장하기

df_col= df.loc[df["Invoice Number"].str.startswith("920-"),
               ["Invoice Number", "Cost"]]
print(df_col)
df_col.to_csv("pandas_out4.csv", index=False)
```

3. excel 파일

xlsx 파일 읽기

```
import openpyxl #pip install openpyxl
filename="sales_2015.xlsx"
book = openpyxl.load_workbook(filename)
sheet = book.worksheets[0]
data=[]
for row in sheet.rows :
    line = []
    for l,d in enumerate(row) :
        line.append(d.value)
# print(line)
    data.append(line)
#print(data)
# enumerate : 리스트에서 데이터와 index값을 제공함.
for i in range(0,len(data)) :
    print(i+1,":",data[i])
print("enumerate 함수 사용")
for i,d in enumerate(data) :
    print(i+1,":",d)
```

3. excel 파일

xlsx 파일의 모든 sheet 읽기

```
import openpyxl
filename="sales_2015.xlsx"
book = openpyxl.load_workbook(filename)
#book.worksheets : excel 파일의 모든 sheet를 리턴.
for i, sheet in enumerate(book.worksheets):
    print(book.sheetnames[i]) #sheet 이름
    data=[]
    for r,row in enumerate(sheet.rows): #해당 sheet의 행값 들.
        line=[]
        for i,c in enumerate(row):
            line.append(c.value)
        print(r+1,":",line)
        data.append(line)
```

3. excel 파일

xls 파일 읽기

```
from xlrd import open_workbook
infile = "ssec1804.xls"
workbook = open_workbook(infile) #xls 파일
print("sheet 의 갯수",workbook.nsheets)
#workbook.sheets() : sheet들 정보 저장
#worksheet : 한개의 sheet 데이터 저장
for worksheet in workbook.sheets() :
    print("worksheet 이름:",worksheet.name)
    print("행의 수:",worksheet.nrows)
    print("컬럼의 수:",worksheet.ncols)
    #해당 sheet의 데이터 출력
    for row_index in range(worksheet.nrows) :
        for column_index in range(worksheet.ncols) :
            #worksheet.cell_value(행인덱스,열인덱스) : excel 파일의 셀 데이터
            print(worksheet.cell_value(row_index,column_index),"",end="")
        print()
    #    pass
```

3. excel 파일

xls 파일을 읽고 쓰기

```
from xlrd import open_workbook #excel 파일 읽기 위한 모듈
from xlwt import Workbook     #excel 파일을 쓰기 위한 모듈 pip install xlwt

infile = "ssec1804.xls" #원본데이터
outfile = "ssec1804out.xls" #복사데이터 (하나의 sheet만 저장)

outworkbook = Workbook() #비어있는 xls 파일

out_sheet = outworkbook.add_sheet("전체증감") #출력한 xls파일에 sheet 추가. 이름설정

#open_workbook(infile) : 원본데이터 xls파
with open_workbook(infile) as workbook :
    #worksheet : 원본 xls파일의 sheet의 이름이 "1.전체증감" sheet 데이터
    worksheet = workbook.sheet_by_name("1.전체증감")
    for rindex in range(worksheet.nrows) :
```

3. excel 파일

xls 파일을 읽고 쓰기

```
#out_sheet.write(rindex,cindex,원본cell) : 원본cell의 데이터를 rindex,cindex 의 데이터로 저장
for cindex in range(worksheet.ncols) :
    out_sheet.write(rindex,cindex,
                    worksheet.cell_value(rindex,cindex))
    print(worksheet.cell_value(rindex,cindex))
```

```
#outworkbook의 데이터를 ssec1804out.xls파일로 저
outworkbook.save(outfile)
```


3. excel 파일

pandas를 이용하여 excel 파일 읽고 쓰기

```
import pandas as pd

infile = "sales_2015.xlsx"
outfile = "sales_2015_pd.xlsx"
#read_excel(file이름,sheet이름,인덱스컬럼여부)
df = pd.read_excel(infile,"january_2015",index_col=None)
print(df)
#df_value : Sale Amount 컬럼의 값이 500.0보다 큰 값을 가지는 행을 조회.
df_value = df[df["Sale Amount"].astype(float) > 500.0]
#xlsx 파일로 저장
#writer : sales_2015_pd.xlsx 파일을 출력파일로 설정. openpyxl=> xlsx형태
writer = pd.ExcelWriter(outfile,engine="openpyxl")
#df_value 의 데이터를 writer파일에 jan_15_out sheet로 저장. 인덱스값 없음
df_value.to_excel(writer,sheet_name="jan_15_out",index=False)
#파일로 생성.
writer.save()
```

3. excel 파일

pandas를 이용하여 xlsx 파일의 모든 sheet를 읽기

```
import pandas as pd

infile = "sales_2015.xlsx"
df = pd.read_excel(infile, sheet_name=None, index_col=None)
row_output = [] #모든 sheet의 데이터 저장
#df.items() : sheet의 정보들.
# worksheet_name : sheet 이름
# data : sheet의 데이터
for worksheet_name, data in df.items():
    print("===", worksheet_name, "===")
    row_output.append(data[data["Sale Amount"].replace("$", "").
                          replace(",", "").astype(float) > 200.0])
```

3. excel 파일

pandas를 이용하여 xlsx 파일의 모든 sheet를 읽기

```
"""
```

```
pd.concat : row_output 를 여러개의 데이터를 연결하기
```

```
axis=0 : row에 연결
```

```
axis=1 : column에 연결
```

```
"""
```

```
filtered_row = pd.concat(row_output,axis=0,ignore_index=True)
```

```
#filtered_row : 모든 sheet의에서 주어진 조건에 맞는 데이터 저
```

```
writer = pd.ExcelWriter("sales_all_2015.xlsx",engine="openpyxl")
```

```
filtered_row.to_excel(writer,sheet_name="sale_2015",index=False)
```

```
writer.save()
```

3. excel 파일

pandas를 이용하여 xls 파일 읽기-출력되는 excel파일의 sheet를 여러개로 저장

```
import pandas as pd
infile = "ssec1804.xls"
outfile = "ssec1804_bak.xls"
writer = pd.ExcelWriter(outfile)
df = pd.read_excel(infile, sheet_name=None, index_col=None)
for worksheet_name, data in df.items():
    print("===", worksheet_name, "===")
    print(data)
    data.to_excel(writer, sheet_name=worksheet_name, index=False, header=False)
writer.save()
```

3. excel 파일

폴더에 속한 excel 파일들을 모두 읽어서 하나의 excel 파일로 저장

```
import pandas as pd
import glob #경로명 표현시 사용되는 모듈
import os   #시스템 관련된 모듈
inpath="C:/kms/20200914/python/workspace/excel/"
outfile="sales_all.xlsx"
#inpath : 조회되는 폴더
#*.xls* : inpath 폴더 내부의 파일 중 xls, xlsx 확장자를 가진 파일 전부
excels = glob.glob(os.path.join(inpath,"*.xls*"))
print(type(excels)) #리스트 excel 파일의 목록
rows = []
for excel in excels :
    print(excel) #조회된 파일의 이름
    #excel 파일 읽기.
    dfs = pd.read_excel(excel,sheet_name=None,index_col=None)
    for sheet_name,df in dfs.items() :
        rows.append(df)
```

3. excel 파일

폴더에 속한 excel 파일들을 모두 읽어서 하나의 excel 파일로 저장

```
df_concat = pd.concat(rows,sort=False,axis=0,ignore_index=True)
writer = pd.ExcelWriter(outfile)
df_concat.to_excel(writer,sheet_name="all_data_all_worksheet",index=False)
writer.save()
```