Pandas

1. Pandas의 특징

- Pandas는 금융 데이터 분석을 목적으로 개발
- 구조화된 데이터 를 쉽고 빠르게 가공할 수 있는 자료형과 함수를 제공
- Pandas 이름 은 계량 경제학에서 동일한 조사 대상으로부터 여러 시점에 걸쳐 반복적 으로 수집한 데이터를 지칭하는 패널 데이터와 파이썬 데이터 분석에서 가져옴
- Pandas는 Numpy를 기반으로 구현했기 때문에 대부분의 함수가 Numpy 와 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 고려하여 설계

1. Pandas의 특징

빅데이터의 시대. 데이터 과학이라는 새로운 영역의 출현.

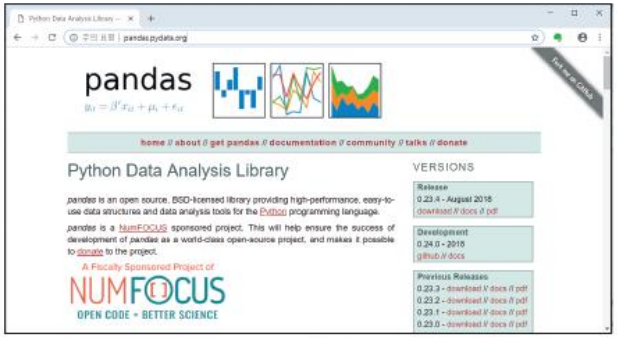
- 클라우드 컴퓨팅의 확산. 빅데이터 저장, 분석에 필요한 컴퓨팅 자원이 매우 저렴해짐.
- 컴퓨팅 파워의 대중화는 최적의 학습환경과 연구 인프라를 제공.

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/)

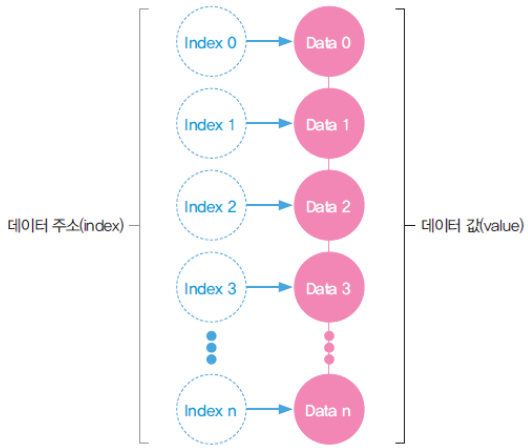
2. 판다스 자료구조

- **목적**

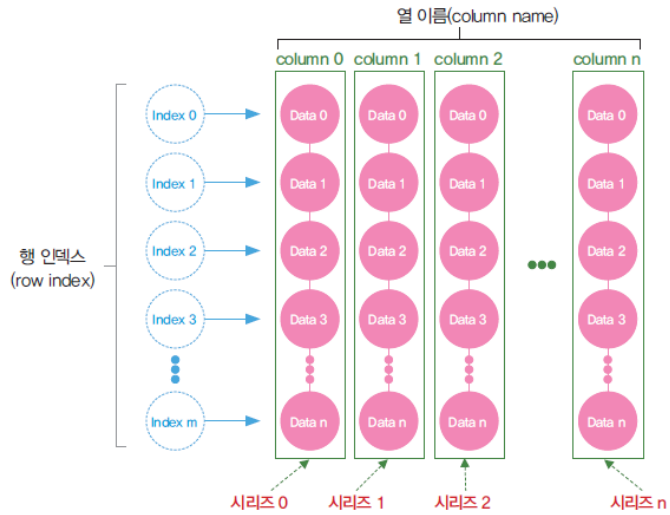
분석을 위해 다양한 소스(source)로부터 수집하는 데이터는 형태나 속성이 매우 다양하다. 특히, 서로 다른 형식을 갖는 여러 종류의 데이터를 컴퓨터가 이해할 수 있도록 동일한 형식을 갖는 구조로 통합할 필요가 있다. 판다스의 일차적인 목적은 **형식적으로 서로 다른 여러 가지 유형의 데이터를 공통의 포맷으로 정리**하는 것이다.

- **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.



[그림 1-2] 시리즈 구조



[그림 1-6] 데이터프레임 구조

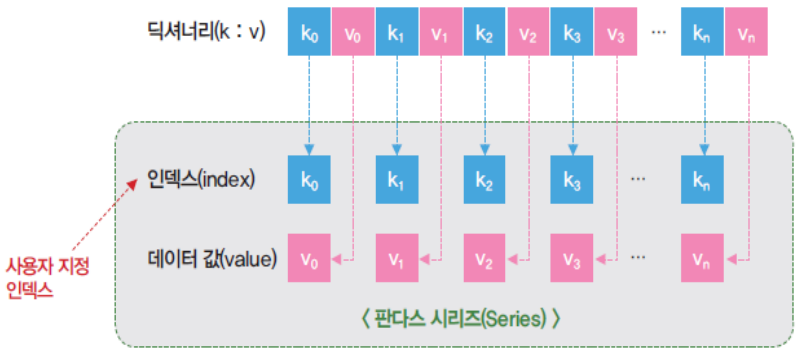
• 시리즈 만들기

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.

```
딕셔너리 → 시리즈 변환: pandas.Series( 딕셔너리 )
```

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



[그림 1-3] 딕셔너리 → 시리즈 변환

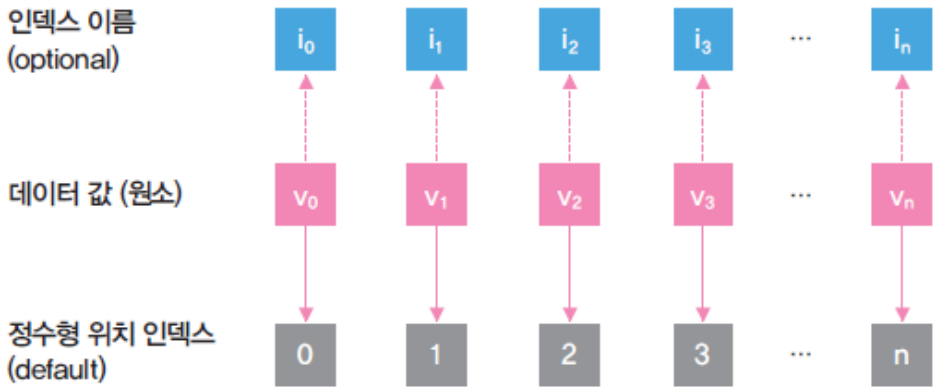
• 인덱스 구조

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)**

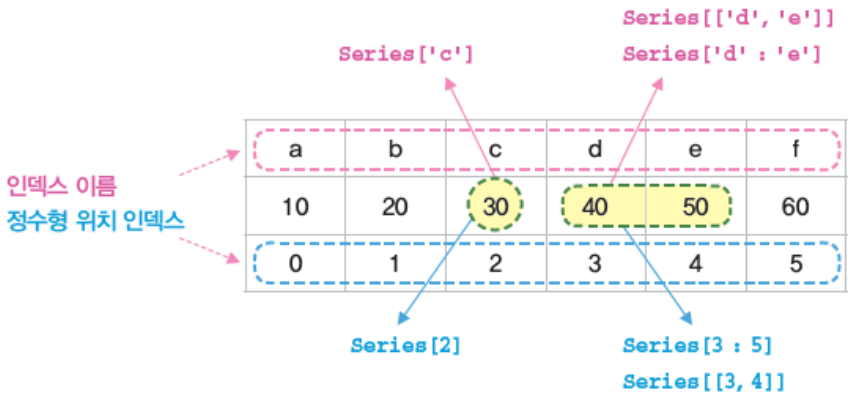
①정수형 위치 인덱스(integer position)
②인덱스 이름(index name) 또는 인덱스 라벨(index label)



[그림 1-4] 시리즈 인덱스의 유형

• 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.



[그림 1-5] 시리즈 원소 선택

- **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 않다('성별' 불포함).

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

2-2. 데이터프레임

- **개요**

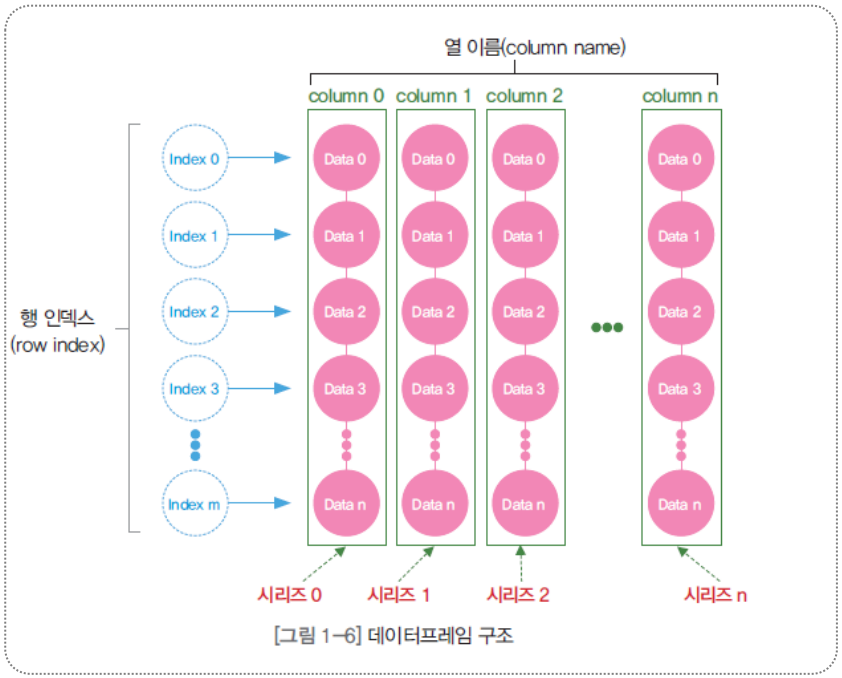
1) 데이터프레임은 2차원 배열. R의 데이터프레임에서 유래.

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).



[그림 1-6] 데이터프레임 구조

[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

열

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

행

[표 1-1] 주식 종목 리스트

2-2. 데이터프레임

- **데이터프레임 만들기**

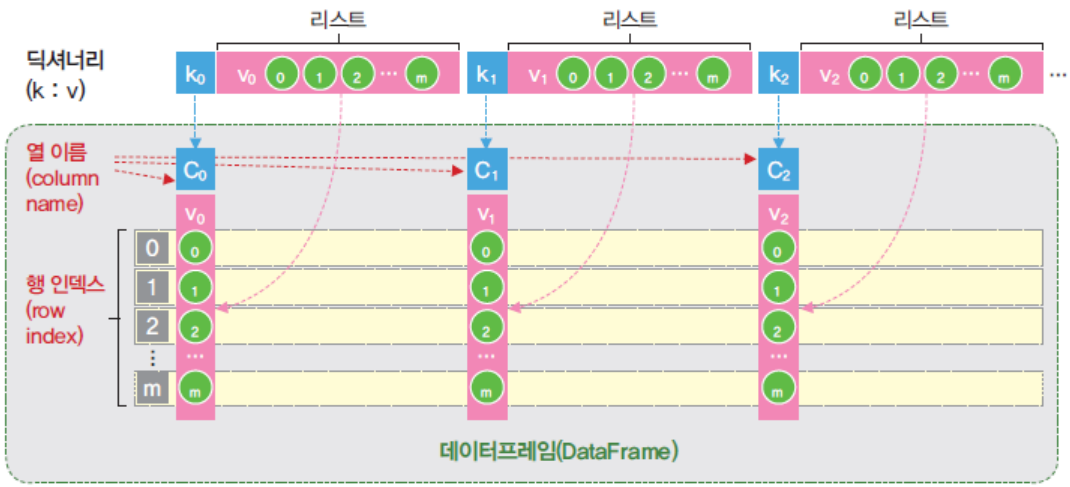
1) 같은 길이(원소의 개수가 동일한)의 배열 여러 개가 필요. 데이터프레임은 여러 개의 시리즈(열, column)를 모아 놓은 집합.

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.



[그림 1-7] 딕셔너리 → 데이터프레임 변환

• 행 인덱스/열 이름 설정

: 데이터프레임의 행 인덱스와 열 이름을 사용자가 지정 가능.

```
행 인덱스/열 이름 설정: pandas.DataFrame( 2차원 배열,
                                          index=행 인덱스 배열,
                                          columns=열 이름 배열 )
```

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

• 행/열 삭제

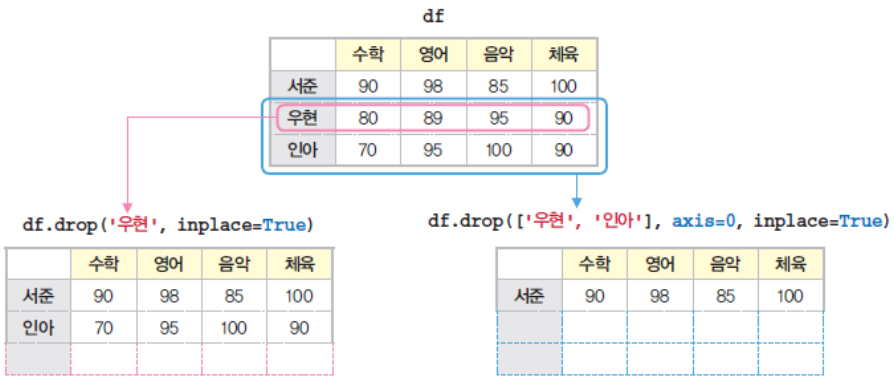
drop() 메소드에 행을 삭제할 때는 축(axis) 옵션으로 axis=0을 입력하거나, 별도로 입력하지 않는다.

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.

① 행 삭제



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

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

	수학	영어	음악	체육
서준	90	98	85	100

[그림 1-9] 행 삭제

- **행/열 삭제 *(계속)***

② 열 삭제

df

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

```
df.drop('수학', axis=1, inplace=True)
```

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

```
df.drop(['영어', '음악'], axis=1, inplace=True)
```

	수학	체육
서준	90	100
우현	80	90
인아	70	90

[그림 1-10] 열 삭제

• 행 선택

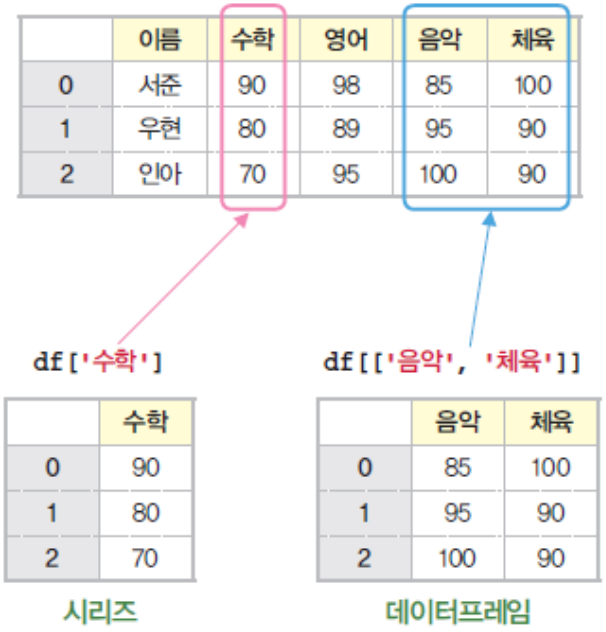
1) **loc**과 **iloc** 인덱서를 사용.

2) 인덱스 이름을 기준으로 행을 선택할 때는 loc을 이용하고, 정수형 위치 인덱스를 사용할 때는 iloc을 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

- **열 선택**



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

	수학
0	90
1	80
2	70

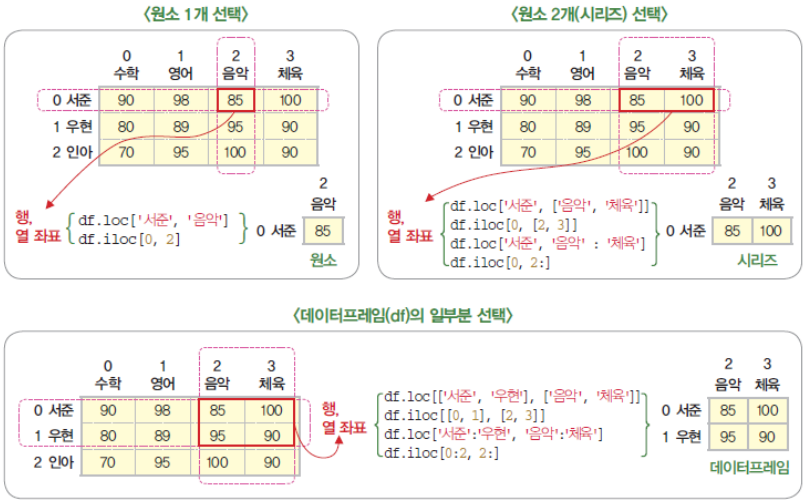
	음악	체육
0	85	100
1	95	90
2	100	90

[그림 1-11] 데이터프레임의 열 선택

- **열 선택 *(계속)***

② n개의 열 선택 (리스트 입력)

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



[그림 1-12] 데이터프레임의 [행, 열] 데이터 선택

이 때, 반환되는 객체의 자료형을 확인하면 데이터프레임이다.

• 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.

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

	국어
0	80
1	80
2	80

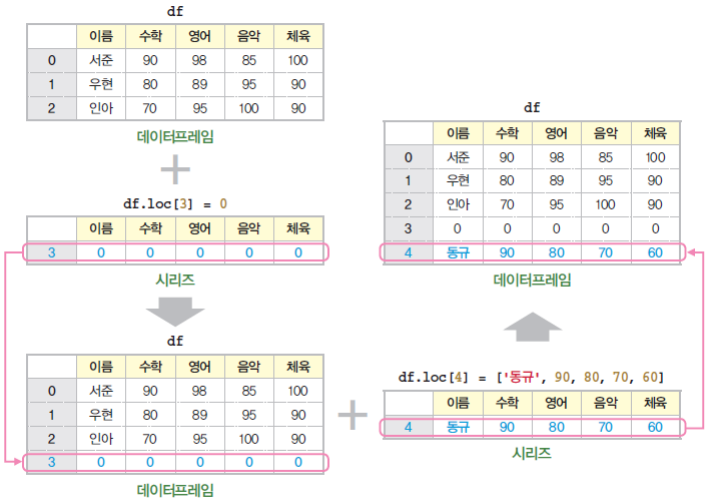
	이름	수학	영어	음악	체육	국어
0	서준	90	98	85	100	80
1	우현	80	89	95	90	80
2	인아	70	95	100	90	80

[그림 1-13] 데이터프레임의 열 추가

• 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

• 원소 값 변경

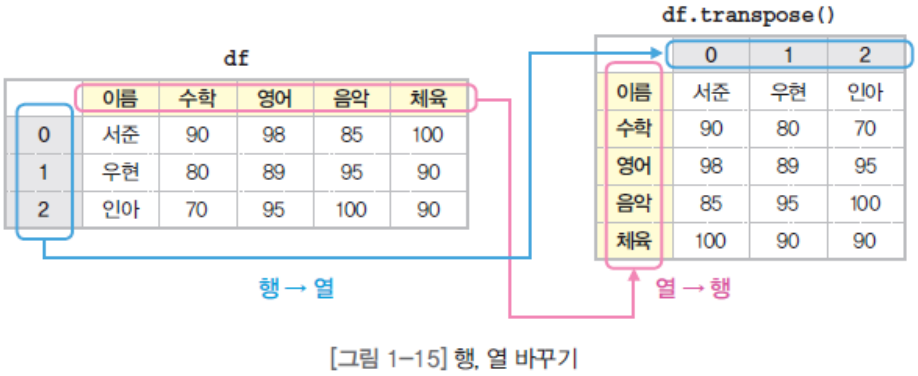
: 원소를 선택하고 새로운 데이터 값을 지정.

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

• 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.

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



[그림 1-15] 행, 열 바꾸기

• 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')
```

[그림 1-16] 데이터프레임의 특정 열을 행 인덱스로 설정

• 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.

- **행 인덱스 초기화**

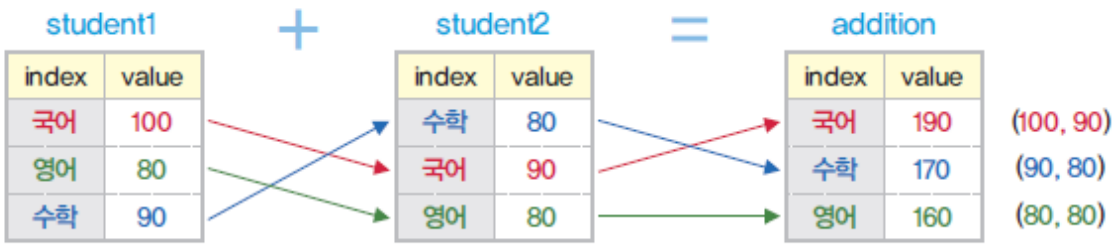
  - reset_index() 메소드로 정수형 위치 인덱스로 초기화. 기존 행 인덱스는 열로 이동. 새로운 데이터프레임 객체를 반환.

- **행 인덱스를 기준으로 데이터프레임 정렬**

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

- **판다스 객체의 산술연산은 내부적으로 3단계 프로세스를 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.

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

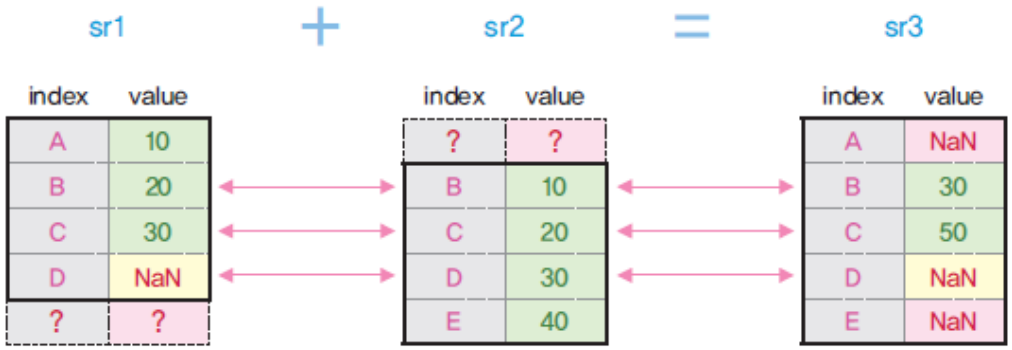


[그림 1-17] 시리즈의 산술연산(덧셈)

• 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.



[그림 1-18] NaN값이 있는 시리즈의 산술연산(덧셈)

- **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.

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.

데이터프레임과 숫자 연산: DataFrame 객체 + 연산자(+, -, *, /) + 숫자

※ 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'

- **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.

데이터프레임의 연산자 활용: `DataFrame1 + 연산자(+, -, *, /) + DataFrame2`

df1

	열 A	열 B	열 C
행 0	15	100	1
행 1	20	500	2
행 2	50	200	3
행 3	NaN	500	2
행 4	NaN	200	3

+

df2

	열 A	열 B
행 0	10	200
행 1	10	100
행 2	10	200
행 3	20	100
행 4	30	100

=

df1+df2

	열 A	열 B	열 C
행 0	25	300	NaN
행 1	30	600	NaN
행 2	60	400	NaN
행 3	NaN	600	NaN
행 4	NaN	300	NaN

[그림 1-19] 데이터프레임의 산술연산(덧셈)