pandas - Series

김지성 강사

Pandas(Panel Datas)란

- ✓ Pandas는 주로 데이터 분석에 사용된다.
- ✓ 대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있다. Pandas 패키지는 이러한 데이터를 다루기 위한 Series 클래스와 DataFrame 클래스를 제공.
- ✓ 숫자 테이블과 시계열 을 조작하기 위한 데이터 구조 와 연산을 제공.

Pandas package import

- ✓ NumPy와 마찬가지로 Pandas 패키지를 사용하기 위해선 해당 패키지를 임포트를 해야한다.
- ✓ Pandas 패키지는 pd라는 별칭으로 임포트하는 것이 관례.

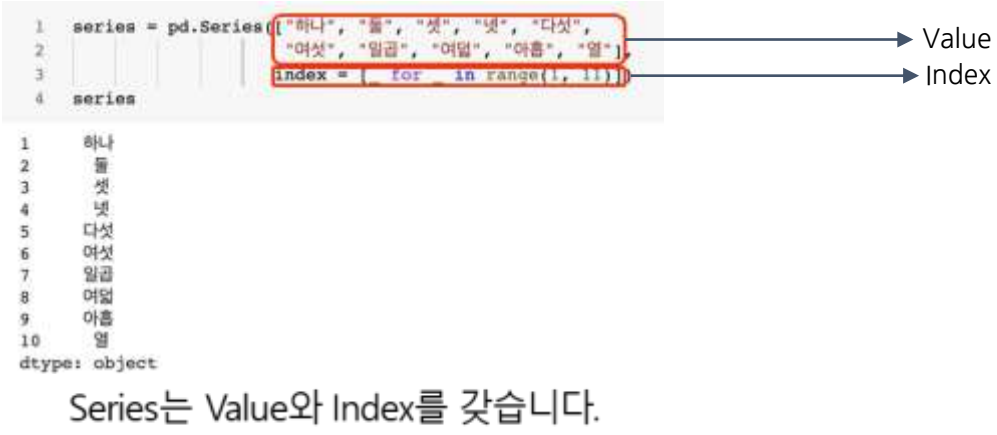
```
import pandas as pd
```

Series class

✓ Series 클래스는 Numpy에서 제공하는 1차원 배열과 그 모양이 비슷하다.

✓ 하지만 Series class는 배열과 다르게 각 데이터의 의미를 표시하는 index(index)를 붙일 수 있다.

데이터 자체는 값(value)이라고 표현.



Series 생성하기

- ✓ Series 객체를 만들 때 **첫 인수로 data**, **두 번째 인수로는 index**를 넣는다.
- ✓ data 값으로 iterable, 배열, scalar value, dict(key와 index를 동일하게 사용하거나 생략)를 사용할 수 있다.
- ✓ index는 label이라고도 한다. index는 data와 length가 동일해야 한다. label은 꼭 unique할 필요는 없다. 만약 index를 생략할 경우 RangeIndex(0, 1, … , n)를 제공한다.

pandas.Series

```
class pandas.Series(data=None, index=None, dtype=None, name=None,
copy=False, fastpath=False)                                    [source]
```

One-dimensional ndarray with axis labels (including time series).

Series 생성하기

✓ 다음 예제를 통해 각 도시의 2015년 인구 데이터를 Series로 만들어보자.

✓ 자리수가 긴 숫자의 경우에 쉽게 읽기 위해 콤마로 3자리씩 끊어 표기한다. 파이썬에서도 이처럼 사용할 수 있는 방법이 있는데, 아래 예제처럼 언더바를 숫자 사이사이 넣으면 된다.

```
1   s = pd.Series([9_904_312, 3_448_737, 2_890_451, 2_466_052],
2                 index=["서울", "부산", "인천", "대구"])
3   s
```

```
서울    9904312
부산    3448737
인천    2890451
대구    2466052
dtype: int64
```

Series 생성하기 - 연습 문제

✓ 다음과 같이 10, 20, 30, 40, 50, 60, 70, 80, 90의 값을 갖는 Series를 생성해보세요.

```
0    10
1    20
2    30
3    40
4    50
5    60
6    70
7    80
8    90
dtype: int64
```

Series 생성하기 - 연습 문제 해답

✓ 다음과 같이 10, 20, 30, 40, 50, 60, 70, 80, 90의 값을 갖는 Series를 생성해보세요.

```
1   test = pd.Series(list(range(10, 100, 10)))
2   test
```

```
0    10
1    20
2    30
3    40
4    50
5    60
6    70
7    80
8    90
dtype: int64
```

Series 생성하기

✓ 만약 index를 지정하지 않고 Series를 만들면 Series의 index는 0부터 시작하는 정수 값이 된다.

```
1   pd.Series(range(10, 14))
```

```
0    10
1    11
2    12
3    13
dtype: int64
```

Series 생성하기

✓ Series의 index는 index 속성으로 접근할 수 있다.

✓ Series의 value는 1차원 배열(ndarray) 이며 values 속성으로 접근할 수 있다.

```
1  s.index
```

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
```

```
1  s.values
```

```
array([9904312, 3448737, 2890451, 2466052])
```

Series 생성하기 - 연습 문제

✓ 앞선 연습 문제에서 만든 Series 객체의 값 중 50보다 큰 값의 개수를 구해보세요.

✓ 결과 : 4

```
1   test = pd.Series(list(range(10, 100, 10)))
2   test
```

```
0    10
1    20
2    30
3    40
4    50
5    60
6    70
7    80
8    90
dtype: int64
```

Series 생성하기 - 연습 문제

✓ 앞선 연습 문제에서 만든 Series 객체의 값 중 50보다 큰 값의 개수를 구해보세요.

```
1   sum(test.values > 50)
```

```
4
```

Series 생성하기

✓ name 속성을 이용하여 Series 데이터에 이름을 붙일 수 있다.

✓ index.name 속성으로 Series의 index에도 이름을 붙일 수 있다.

```
1   s.name = "인구"
2   s.index.name = "도시"
3   s
```

```
도시
서울      9904312
부산      3448737
인천      2890451
대구      2466052
Name: 인구, dtype: int64
```

Series 생성하기

✓ Series 객체를 만들 때 data에 dict를 사용할 수 있다.

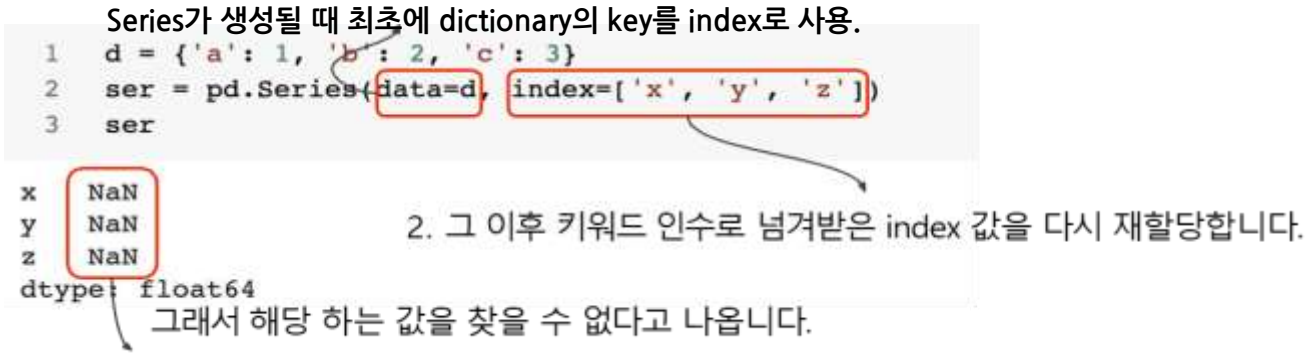
✓ 만들어진 Series 객체를 조회해보면 그 값이 정상적으로 조회되는 것을 확인할 수 있다.

```
1   d = {'a': 1, 'b': 2, 'c': 3}
2   ser = pd.Series(data=d, index=['a', 'b', 'c'])
3   ser
```

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

Series 생성하기

- ✓ 이번에는 dict의 key와 Series 객체의 index를 다르게 설정해보자.
- ✓ data가 dict일 때 index가 최초에 dict의 key로 만들어진다. 그 후 Series는 index 키워드로 전달 받은 인수로 index를재할당한다.
- ✓ 밑에 예제와 같이 Series 객체의 값이 NaN의 결과를 출력하는 것을 확인할 수 있다.



Series 생성하기

✓ index 지정 없이 dict 객체만 가지고 Series를 만들 수도 있습니다. dic의 key가 index로 사용되는 것을 확인할 수 있다.

```
1   s2 = pd.Series({"서울":9_904_312,
2                   "부산":3_448_737,
3                   "인천":2_890_451,
4                   "대구":2_466_052})
5   s2
```

```
서울    9904312
부산    3448737
인천    2890451
대구    2466052
dtype: int64
```

Series 생성하기 - 연습 문제

✓ 사회 점수가 다음과 같을 때 이름을 index로 하고 점수를 values로 하는 Series를 만들어보세요.

이름	사회 점수
철수	88
영희	95
길동	100
몽룡	67

Series 생성하기 - 연습 문제 해답

✓ 사회 점수가 다음과 같을 때 이름을 index로 하고 점수를 values로 하는 Series를 만들어보세요.

```
1  scores = {"철수": 88, "영희": 95, "길동": 100, "몽룡": 67}
2  s_scores = pd.Series(scores)
3  s_scores
```

```
철수     88
영희     95
길동    100
몽룡     67
dtype: int64
```

Series index를 속성처럼 활용하기

✓ 만약 label 값이 공백 없는 문자열인 경우에는 index label이 속성인것처럼 마침표(.)를 활용하여 해당 index 값에 접근할 수도 있다.

```
1  d = {'a': 1, 'b': 2, 'c': 3}
2  ser = pd.Series(data=d, index=['a', 'b', 'c'])
3  ser
```

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

```
1  ser.a, ser.b, ser.c
```

```
(1, 2, 3)
```

Series의 특징

- ✓ Series 객체는 index label을 키(key)로 사용하기에 딕셔너리 자료형과 비슷한 특징을 갖는다.
- ✓ 그래서 Series를 딕셔너리와 같은 방식으로 사용할 수 있게 구현해놨음.
  - ○ 예를 들어 in 연산도 가능하고, items() 메서드를 사용해서 for문 루프를 돌려 각 요소의 키(key)와 값(value)에 접근할 수도 있다.

```
1  "서울" in s  # 인덱스 레이블 중에 서울이 있는가
```

```
True
```

```
1  "대전" in s  # 인덱스 레이블 중에 대전이 있는가
```

```
False
```

```
1  for k, v in s.items():
2      print(f"{k}, {v}")
```

```
서울, 9904312
부산, 3448737
인천, 2890451
대구, 2466052
```

Series 연습 문제

✓ 도시의 인구가 300만이 넘는 곳을 찾아 다음과 같이 출력해보세요.

○ for문과 items() 메서드를 활용하세요.

```
서울의 인구는 300만이 넘습니다.
부산의 인구는 300만이 넘습니다.
```

Series 연습 문제 해답

✓ 도시의 인구가 300만이 넘는 곳을 찾아 다음과 같이 출력해보세요.

○ for문과 items() 메서드를 활용하세요.

```
1   for i, v in s.items():
2       if v > 3_000_000:
3           print(f'{i}의 인구는 300만이 넘습니다.')
```

```
서울의 인구는 300만이 넘습니다.
부산의 인구는 300만이 넘습니다.
```

Series 연산하기

✓ 넘파이 배열처럼 Series도 벡터화 연산을 할 수 있다. 다만 연산은 Series의 value에만 적용되며 index 값은 변하지 않는다.

○ 예를 들어 인구 숫자를 백만 단위로 만들기 위해 Series 객체를 1,000,000 으로 나누어도 index label에는 영향을 미치지 않는 것을 볼 수 있다.

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

```
1  s / 1000000
```

```
도시
서울    9.904312
부산    3.448737
인천    2.890451
대구    2.466052
Name: 인구, dtype: float64
```

Series 인덱싱

✓ Series는 **넘파이 배열에서 가능한 index 방법** 이외에도 **index label을 이용한 인덱싱**도 할 수 있습니다. 배열 인덱싱이나 index label을 이용한 슬라이싱(slicing)도 가능합니다.

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

```
1  s[1], s["부산"]
```

```
(3448737, 3448737)
```

```
1  s[3], s["대구"]
```

```
(2466052, 2466052)
```

Series 인덱싱

✓ 배열 인덱싱을 하면 부분적인 값을 가지는 Series 자료형을 반환한다. 자료의 순서를 바꾸거나 특정한 자료만 취사 선택할 수 있다.

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

```
1  s[[0, 3, 1]]
```

```
도시
서울    9904312
대구    2466052
부산    3448737
Name: 인구, dtype: int64
```

```
1  s[["서울", "대구", "부산"]]
```

```
도시
서울    9904312
대구    2466052
부산    3448737
Name: 인구, dtype: int64
```

Series 인덱싱

✓ 단 하나의 값을 시리즈 형태로 가져오고 싶으면 다음과 같이 값이 하나인 리스트로 인덱싱하여 작성할 수 있다.

```
1  s[[0]]
```

```
서울     9904312
dtype: int64
```

Series 슬라이싱

✓ 슬라이싱을 해도 부분적인 Series를 반환한다.

✓ 이 때 문자열 label을 이용한 슬라이싱을 하는 경우에는 숫자 인덱싱과 달리 콜론(:) 기호 뒤에 오는 값도 결과에 포함되므로 주의해야 한다.

```
도시
서울     9904312
부산     3448737
인천     2890451
대구     2466052
Name: 인구, dtype: int64
```

```
1  s[1:3]  # 두번째(1)부터 세번째(2)까지 (네번째(3) 미포함)
```

```
도시
부산     3448737
인천     2890451
Name: 인구, dtype: int64
```

```
1  s["부산":"대구"]  # 부산에서 대구까지 (대구도 포함)
```

```
도시
부산     3448737
인천     2890451
대구     2466052
Name: 인구, dtype: int64
```

Series index 기반 연산

- ✓ 이번에는 2015년도와 2010년의 인구 증가를 계산해 보자. Series에 대해 연산을 하는 경우 index가 같은 데이터에 대해서만 차이를 구합니다.
- ✓ 대구와 대전의 경우에는 2010년 자료와 2015년 자료가 모두 존재하지 않기 때문에 계산이 불가능하므로 NaN(Not a Number)이라는 값을 가지게 됩니다.

Series index 기반 연산

```
1  s = pd.Series([9904312, 3448737, 2890451, 2466052],
2                index=["서울", "부산", "인천", "대구"])
3  s.name = "인구"
4  s.index.name = "도시"
5  s
```

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

```
1  s2 = pd.Series({"서울": 9631482, "부산": 3393191, "인천": 2632035, "대전": 1490158})
2  s2
```

```
서울    9631482
부산    3393191
인천    2632035
대전    1490158
dtype: int64
```

```
1  ds = s - s2
2  ds
```

```
대구         NaN
대전         NaN
부산     55546.0
서울    272830.0
인천    258416.0
dtype: float64
```

NaN 값이 float 자료형에서만 표현 가능하므로 다른 계산 결과도 모두 float 자료형이 되었다는 점에 주의해야 합니다.

Series에서 값이 NaN인지 확인

✓ Series의 값이 **NaN이면 True NaN이 아니면 False**인 bool type의 Series를 구하려면 isnull() 메서드를 사용하면 된다.

```
대구          NaN
대전          NaN
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

```
]  1  ds.isnull()
```

```
대구     True
대전     True
부산    False
서울    False
인천    False
dtype: bool
```

Series에서 값이 NaN인지 확인

✓ Series의 값이 **NaN이 아니면 True NaN이면 False** 값을 갖는 bool type의 Series를 구하려면 notnull() 메서드를 사용하면 된다.

```
대구          NaN
대전          NaN
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

```
1  ds.notnull()
```

```
대구    False
대전    False
부산     True
서울     True
인천     True
dtype: bool
```

Series에서 NaN이 아닌 값만 인덱싱으로 구하기

✓ notnull() 메서드로 구한 True / False 값을 갖는 시리즈를 활용하여 NaN인 값을 배제한 Series 객체를 인덱싱하여 만들 수 있다.

```
대구        NaN
대전        NaN
부산    55546.0
서울   272830.0
인천   258416.0
dtype: float64
```

```
1  ds.notnull()
```

```
대구    False
대전    False
부산     True
서울     True
인천     True
dtype: bool
```

```
1  ds[ds.notnull()]
```

```
부산    55546.0
서울   272830.0
인천   258416.0
dtype: float64
```

Series에서 NaN이 아닌 값 구하기

✓ 마찬가지로 NaN 값인 것을 배제하고 2010년 대비 2015년 인구 증가율(%)은 다음과 같이 구할 수 있다.

```
5   s # 2015년 도시별 인구
```

```
도시
서울    9904312
부산    3448737
인천    2890451
대구    2466052
Name: 인구, dtype: int64
```

```
2   s2 # 2010년 도시별 인구
```

```
서울    9631482
부산    3393191
인천    2632035
대전    1490158
dtype: int64
```

```
1   rs = (s - s2) / s2 * 100
2   rs = rs[rs.notnull()]
3   rs
```

```
부산    1.636984
서울    2.832690
인천    9.818107
dtype: float64
```

Series 연습 문제

✓ 2010년 대비 2015년 인구 증가를 구하세요.(NaN 값인 것을 배제) 인구수 증가가 가장 많은 도시의 이름과 증가한 인구수를 Series 객체로 출력해보세요.

```
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

```
서울     272830.0
dtype: float64
```

Series 연습 문제 해답

✓ 2010년 대비 2015년 인구 증가를 구하세요.(NaN 값인 것을 배제) 인구수 증가가 가장 많은 도시의 이름과 증가한 인구수를 Series 객체로 출력해보세요.

```
1  ds = s - s2
2  ds
```

```
대구          NaN
대전          NaN
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

```
1  ds = ds[ds.notnull()]
2  ds
```

```
부산      55546.0
서울     272830.0
인천     258416.0
dtype: float64
```

```
1  ds[[ds.values.argmax()]]
```

```
서울     272830.0
dtype: float64
```

Series 데이터 추가, 갱신, 삭제

✓ 없는 index에 값을 할당하면 Series에 데이터가 추가(add)된다. 아래 예제에서는 “대구"라는 index는 현재 없는데 그 index에 값을 1.41 할당하여 데이터를 추가하고 있다.

```
2   rs
```

```
부산    1.630000
서울    2.832690
인천    9.818107
dtype: float64
```

```
1   rs["대구"] = 1.41
2   rs
```

```
부산    1.630000
서울    2.832690
인천    9.818107
대구    1.410000
dtype: float64
```

Series 데이터 추가, 갱신, 삭제

✓ 데이터를 삭제할 때도 딕셔너리처럼 del 명령을 사용한다. 아래 예제에서는 “서울" 이라는 index 에 접근하여 del 명령을 사용하여 데이터를 삭제하고 있다.

```
2   rs
```

```
부산    1.630000
서울    2.832690
인천    9.818107
대구    1.410000
dtype: float64
```

```
1   del rs["서울"]
2   rs
```

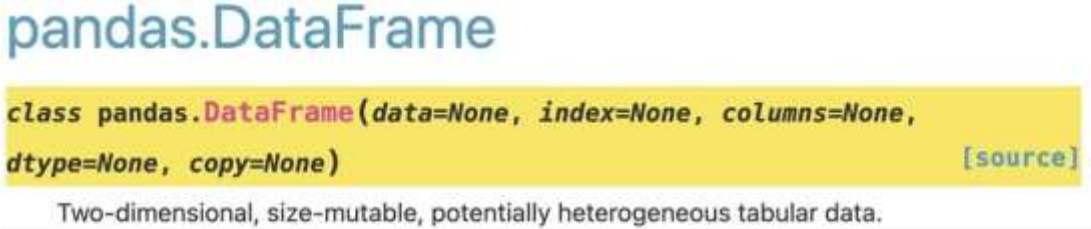
```
부산    1.630000
인천    9.818107
대구    1.410000
dtype: float64
```

pandas - DataFrame

김지성 강사

DataFrame class

- ✓ DataFrame은 Pandas의 주요 데이터 구조이다.
- ✓ label된 **row와 column, 두 개의 축을 갖는 데이터 구조**입니다. 산술 연산은 row와 column 모두 적용됩니다. Series 객체를 갖는 dictionary라고 생각하면 비슷합니다.
- ✓ 첫 인자로 **data**, 두 번째 인자로 **index**를 전달한다.



DataFrame은 각 column마다 자료형이 다를 수 있다.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame

DataFrame 생성

✓ DataFrame을 만드는 방법은 다양하다. 가장 간단한 방법으로는 다음과 같다.

1. 우선 하나의 열이 되는 데이터를 리스트나 일차원 배열을 준비합니다.
2. 이 각각의 열에 대한 이름(label)을 키로 가지는 딕셔너리를 만듭니다.
3. 이 데이터를 DataFrame 클래스 생성자에 넣는다. 동시에 열방향 index는 columns 인수로, 행방향 index는 index 인수로 지정합니다.

```
1  d = {'col1': [1, 2], 'col2': [3, 4]}
2  df = pd.DataFrame(data=d)
3  df
```

	col1	col2
0	1	3
1	2	4

DataFrame 생성

✓ 좀 더 스케일을 확장해서 데이터를 늘리면 아래와 같이 DataFrame을 만들 수 있다.

```
1  data = {
2      "2015": [9904312, 3448737, 2890451, 2466052],
3      "2010": [9631482, 3393191, 2632035, 2431774],
4      "2005": [9762546, 3512547, 2517680, 2456016],
5      "2000": [9853972, 3655437, 2466338, 2473990],
6      "지역": ["수도권", "경상권", "수도권", "경상권"],
7      "2010-2015 증가율": [0.0283, 0.0163, 0.0982, 0.0141]
8  }
9  columns = ["지역", "2015", "2010", "2005", "2000", "2010-2015 증가율"]
10 index = ["서울", "부산", "인천", "대구"]
11 df = pd.DataFrame(data, index=index, columns=columns)
12 df
```

	지역	2015	2010	2005	2000	2010-2015 증가율
서울	수도권	9904312	9631482	9762546	9853972	0.0283
부산	경상권	3448737	3393191	3512547	3655437	0.0163
인천	수도권	2890451	2632035	2517680	2466338	0.0982
대구	경상권	2466052	2431774	2456016	2473990	0.0141

```
1  df.dtypes
```

```
지역                object
2015                int64
2010                int64
2005                int64
2000                int64
2010-2015 증가율    float64
dtype: object
```

지역과 인구와 증가율은 각각 object, int, float입니다.

앞서 이야기했듯 DataFrame의 각 column은 자료형이 다를 수 있습니다.

DataFrame의 속성 values, columns, index

✓ Series와 마찬가지로 데이터만 접근하려면 values 속성을 사용.

✓ 열방향 index와 행방향 index는 각각 columns, index 속성으로 접근.

```
1  df.values
```

```
array([['수도권', 9904312, 9631482, 9762546, 9853972, 0.0283],
       ['경상권', 3448737, 3393191, 3512547, 3655437, 0.0163],
       ['수도권', 2890451, 2632035, 2517680, 2466338, 0.0982],
       ['경상권', 2466052, 2431774, 2456016, 2473990, 0.0141]], dtype=object)
```

```
1  df.columns
```

```
Index(['지역', '2015', '2010', '2005', '2000', '2010-2015 증가율'], dtype='object')
```

```
1  df.index
```

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
```

DataFrame 이름 붙이기

✓ Series에서 처럼 열방향 index와 행방향 index에 이름을 붙이는 것도 가능하다.

```
1  df.index.name = "도시"
2  df.columns.name = "특성"
3  df
```

특성 / 도시	지역	2015	2010	2005	2000	2010-2015 증가율
서울	수도권	9904312	9631482	9762546	9853972	0.0283
부산	경상권	3448737	3393191	3512547	3655437	0.0163
인천	수도권	2890451	2632035	2517680	2466338	0.0982
대구	경상권	2466052	2431774	2456016	2473990	0.0141

DataFrame 연습 문제

✓ 아래 조건을 만족하는 DataFrame을 직접 만들어보세요!

(1) column의 개수와 row의 개수가 각각 4개 이상이어야 합니다.

(2) column에는 정수, 문자열, 실수 자료형 데이터가 각각 1개 이상씩 포함되어 있어야 합니다.

DataFrame 전치(Transpose)

✓ DataFrame은 전치(transpose)를 포함하여 넘파이 2차원 배열이 가지는 대부분의 속성이나 메서드를 지원한다.

```
1  df.T
```

도시 특성	서울	부산	인천	대구
지역	수도권	경상권	수도권	경상권
2015	9904312	3448737	2890451	2466052
2010	9631482	3393191	2632035	2431774
2005	9762546	3512547	2517680	2456016
2000	9853972	3655437	2466338	2473990
2010-2015 증가율	0.0283	0.0163	0.0982	0.0141

DataFrame column 추가, 갱신, 삭제

✓ DataFrame은 column을 Series의 딕셔너리으로 볼 수 있는데, Column 단위로 데이터를 갱신하거나 추가, 삭제할 수 있다. 아래 예제는 값을 갱신하고 있음.

```
1  # "2010-2015 증가율"이라는 이름의 열의 값을 갱신
2  df["2010-2015 증가율"] = df["2010-2015 증가율"] * 100
3  df
```

특성	지역	2015	2010	2005	2000	2010-2015 증가율
도시						
서울	수도권	9904312	9631482	9762546	9853972	2.83
부산	경상권	3448737	3393191	3512547	3655437	1.63
인천	수도권	2890451	2632035	2517680	2466338	9.82
대구	경상권	2466052	2431774	2456016	2473990	1.41

갱신

2010-2015 증가율
0.0283
0.0163
0.0982
0.0141

DataFrame column 추가, 갱신, 삭제

✓ 아래 예제에서는 “2005-2010 증가율”이라는 이름의 Column을 추가하고 있다.

○ 기존에 있는 column의 “2005-2010 증가율"에 값을 할당해서 추가합니다

```
# "2005-2010 증가율"이라는 이름의 열 추가
df["2005-2010 증가율"] = ((df["2010"] - df["2005"]) / df["2005"] * 100).round(2)
df
```

특성	지역	2015	2010	2005	2000	2010-2015 증가율	2005-2010 증가율
도시							
서울	수도권	9904312	9631482	9762546	9853972	2.83	-1.34
부산	경상권	3448737	3393191	3512547	3655437	1.63	-3.40
인천	수도권	2890451	2632035	2517680	2466338	9.82	4.54
대구	경상권	2466052	2431774	2456016	2473990	1.41	-0.99

DataFrame column 추가, 갱신, 삭제

✓ 아래 예제에서는 “2010-2015 증가율”이라는 이름의 column을 삭제하고 있다.

✓ del 명령을 통해 해당 column에 접근하여 삭제합니다.

```
1  # "2010-2015 증가율"이라는 이름의 열 삭제
2  del df["2010-2015 증가율"]
3  df
```

특성	지역	2015	2010	2005	2000	2005-2010 증가율
도시						
서울	수도권	9904312	9631482	9762546	9853972	-1.34
부산	경상권	3448737	3393191	3512547	3655437	-3.40
인천	수도권	2890451	2632035	2517680	2466338	4.54
대구	경상권	2466052	2431774	2456016	2473990	-0.99

DataFrame column 인덱싱

✓ DataFrame을 인덱싱을 할 때도 column label을 키(key)로 생각하여 인덱싱을 할 수 있다.

✓ index로 label 값을 하나만 넣으면 Series 객체가 반환된다.

```
1  # 하나의 column만 인덱싱하면 Series가 반환된다.
2  df["지역"]
```

```
도시
서울      수도권
부산      경상권
인천      수도권
대구      경상권
Name: 지역, dtype: object
```

DataFrame column 인덱싱

✓ index로 label 값을 하나의 column만 넣으면 Series 객체가 반환된다.

```
1  # 2010이라는 column을 반환하면서 Series 자료형으로 변환
2  df["2010"]
```

```
도시
서울    9631482
부산    3393191
인천    2632035
대구    2431774
Name: 2010, dtype: int64
```

```
1  type(df["2010"])
```

```
pandas.core.series.Series
```

DataFrame column 인덱싱

✓ label의 배열 또는 리스트로 인덱싱하면 DataFrame 타입이 반환.

```
1  # 여러 개의 columns을 인덱싱하면 부분적인 DataFrame이 반환된다.
2  df[["2010", "2015"]]
```

특성	2010	2015
도시		
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

DataFrame column 인덱싱

✓ 만약 하나의 column만 빼내어더라도 DataFrame 자료형을 유지하고 싶다면 요소가 하나인 리스트 자료형을 사용해서 인덱싱하면 된다.

```
1  # 2010이라는 column을 반환하면서 DataFrame 자료형을 유지
2  df[["2010"]]
```

특성	2010
도시	
서울	9631482
부산	3393191
인천	2632035
대구	2431774

```
1  type(df[["2010"]])
```

```
pandas.core.frame.DataFrame
```

DataFrame column 인덱싱

✓ 문자열이 아닌 정수형 column index를 가지는 경우에는 index 값으로 정수를 사용할 수 있다.

```
1  df2 = pd.DataFrame(np.arange(12).reshape(3, 4))
2  df2
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

별도의 columns 키워드 인수를 전달하지 않으면 RangeIndex를 기본 값으로 부여합니다.

```
1  df2[2]
```

```
0     2
1     6
2    10
Name: 2, dtype: int64
```

```
1  df2[[1, 2]]
```

	1	2
0	1	2
1	5	6
2	9	10

DataFrame row 슬라이싱

✓ 만약 row 단위로 인덱싱을 하고자 하면 항상 슬라이싱(slicing)을 해야 한다.

✓ index의 값이 문자 label이면 label 슬라이싱도 가능.

```
1  df
```

특성	지역	2015	2010	2005	2000	2005-2010 증가율
도시						
서울	수도권	9904312	9631482	9762546	9853972	-1.34
부산	경상권	3448737	3393191	3512547	3655437	-3.40
인천	수도권	2890451	2632035	2517680	2466338	4.54
대구	경상권	2466052	2431774	2456016	2473990	-0.99

```
1  df[:1] # df[:"서울"]  --> 문자는 포함, 숫자는 미포함
```

특성	지역	2015	2010	2005	2000	2005-2010 증가율
도시						
서울	수도권	9904312	9631482	9762546	9853972	-1.34

DataFrame row 슬라이싱

✓ row가 부산인 결과만 보고 싶을 경우에는 아래의 예제 코드처럼 작성해야 한다.

✓ 단 한 줄이기 때문에 [“부산":”부산"]으로 슬라이싱하고 있다.

숫자 슬라이싱과
문자 슬라이싱의 결과가
다름을 꼭 인지하세요.

↓

그도 그럴 것이 문자는
초과하는 범위를 표현할
수 없습니다.

```
1  df[1:2]
```

특성 / 도시	지역	2015	2010	2005	2000	2005-2010 증가율
부산	경상권	3448737	3393191	3512547	3655437	-3.4

```
1  df["부산":"부산"]
```

특성 / 도시	지역	2015	2010	2005	2000	2005-2010 증가율
부산	경상권	3448737	3393191	3512547	3655437	-3.4

DataFrame 개별 데이터 인덱싱

- ✓ DataFrame에서 column label로 인덱싱하면 Series가 된다. 이 Series를 다시 row label로 인덱싱하면 개별 데이터가 나온다.
- ✓ 즉 column label -> row label 순으로 인덱싱

```
1   df["2015"]["서울"]
```

```
9904312
```

```
1   type(df["2015"]["서울"])
```

```
numpy.int64
```

DataFrame 연습 문제

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

- ○ 모든 학생의 수학 점수를 Series로 나타낸다.
- ○ 모든 학생의 국어와 영어 점수를 데이터 프레임으로 나타낸다.
- ○ 모든 학생의 각 과목 평균 점수를 새로운 열로 추가한다.
- ○ 춘향의 점수를 DataFrame으로 나타낸다.
- ○ 향단의 점수를 Series로 나타낸다.

```
data = {
    "국어": [80, 90, 70, 30],
    "영어": [90, 70, 60, 40],
    "수학": [90, 60, 80, 70],
}
columns = ["국어", "영어", "수학"]
index = ["춘향", "몽룡", "향단", "방자"]
df = pd.DataFrame(data,  index=index, columns=columns)
```

DataFrame 연습 문제 해답

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

○ 모든 학생의 수학 점수를 Series로 나타낸다.

```
1  # (1) 모든 학생의 수학 점수를 Series로 나타낸다.
2  df["수학"]
```

```
춘향    90
몽룡    60
향단    80
방자    70
Name: 수학, dtype: int64
```

DataFrame 연습 문제 해답

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

○ 모든 학생의 국어와 영어 점수를 데이터 프레임으로 나타낸다.

```
1  # (2) 모든 학생의 국어와 영어 점수를 DataFrame으로 나타낸다.
2  df[["국어", "영어"]]
```

	국어	영어
춘향	80	90
몽룡	90	70
향단	70	60
방자	30	40

DataFrame 연습 문제 해답

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

○ 모든 학생의 각 과목 평균 점수를 새로운 열로 추가한다.

```
1  # (3) 모든 학생의 각 과목 평균 점수를 새로운 column로 추가한다.
2  df["평균"] = round((df["국어"] + df["영어"] + df["수학"]) / 3, 2)
3  # round(df.mean(axis=1), 2)을 활용해도 된다.
4  df
```

	국어	영어	수학	평균
춘향	80	90	90	86.67
몽룡	90	70	60	73.33
향단	70	60	80	70.00
방자	30	40	70	46.67

DataFrame 연습 문제 해답

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

○ 춘향의 점수를 DataFrame으로 나타낸다.

```
2   df[:1]
3   # df["춘향":"춘향"]
```

	국어	영어	수학	평균
춘향	80	90	90	86.67

DataFrame 연습 문제 해답

✓ 다음 DataFrame을 활용하여 아래 문제를 해결해보세요.

○ 향단의 점수를 Series로 나타낸다.

```
2   df.T["향단"]
```

```
국어    70.0
영어    60.0
수학    80.0
평균    70.0
Name: 향단, dtype: float64
```

Pandas 데이터 입출력

✓ 데이터 출력하기에 앞서 다음과 같은 DataFrame을 만들어 보자.

	c1	c2	c3
0	1	1.11	one
1	2		two
2	누락	3.33	three

Pandas 데이터 입출력

✓ 데이터 출력하기에 앞서 다음과 같은 DataFrame을 만들어 보자.

```
1  data = {
2      "c1": [1, 2, "누락"],
3      "c2": [1.11, "", 3.33],
4      "c3": ["one", "two", "three"]
5  }
6  df_csv = pd.DataFrame(data)
7  df_csv
```

	c1	c2	c3
0	1	1.11	one
1	2		two
2	누락	3.33	three

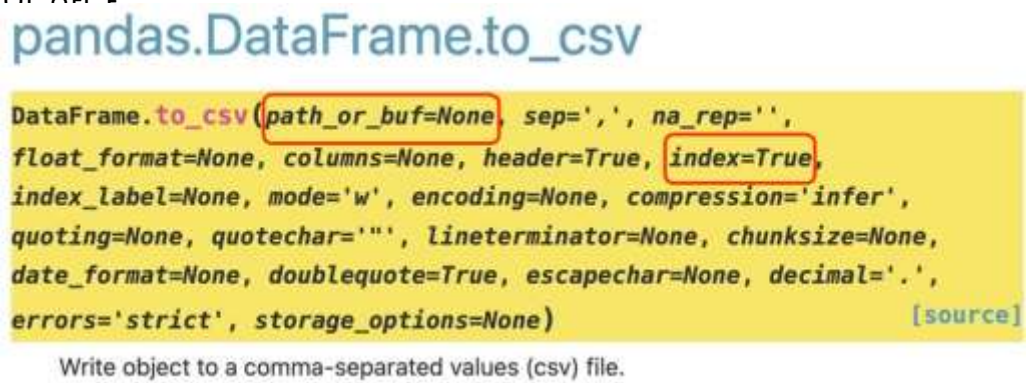
Pandas 데이터 입출력

✓ 데이터를 csv 파일로 출력할 땐 to_csv() 메서드를 활용한다.

✓ 첫 인자로는 파일 경로를 입력한다.

○ 현재 만든 DataFrame의 index는 의미 없는 값이므로 출력할 때 고려하지 않는다.

○ to_csv()의 기본값 인자인 index의 default가 True이니 index=False 키워드를 활용하여 설정해줘야 한다.



Pandas 데이터 입출력

- ✓ df_csv.to_csv(“파일이름 및 확장자", index=False)와 같이 사용한다.
- ✓ 따로 경로를 지정하지 않으면 해당 노트북 폴더에 파일이 생성된다.

```
df_csv.to_csv("sample1.csv", index=False)
```

Pandas 데이터 입출력

✓ 파일을 확인했을 때 아래와 같은 내용으로 생성된 것을 확인할 수 있다.

✓ 엑셀과 같은 형태로 보이기도, 텍스트처럼 보일수도 있는데, 둘 다 같은 파일이다.

Delimiter: ,

	c1	c2	c3
1	1	1.11	one
2	2		two
3	누락	3.33	three

Pandas 데이터 입출력

✓ 이번에는 만든 csv 파일로부터 데이터를 불러오는 작업을 진행해보자.

✓ 이때는 read_csv() 메서드를 사용.

pandas.read_csv

```
pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default,
delimiter=None, header='infer', names=_NoDefault.no_default,
index_col=None, usecols=None, squeeze=None,
prefix=_NoDefault.no_default, mangle_dupe_cols=True, dtype=None,
engine=None, converters=None, true_values=None, false_values=None,
skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None,
na_values=None, keep_default_na=True, na_filter=True, verbose=False,
skip_blank_lines=True, parse_dates=None, infer_datetime_format=False,
keep_date_col=False, date_parser=None, dayfirst=False,
cache_dates=True, iterator=False, chunksize=None, compression='infer',
thousands=None, decimal='.', lineterminator=None, quotechar='"',
quoting=0, doublequote=True, escapechar=None, comment=None,
encoding=None, encoding_errors='strict', dialect=None,
error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None,
delim_whitespace=False, low_memory=True, memory_map=False,
float_precision=None, storage_options=None)                [source]
```

Read a comma-separated values (csv) file into DataFrame.

Pandas 데이터 입출력

✓ CSV 파일로부터 데이터를 읽어 DataFrame을 만들 때는 pandas.read_csv 함수를 사용한다.

✓ 함수의 첫 번째 인수로 “파일 이름.확장자” 문자열로 넣는다. 그럼 아래와 같이 DataFrame을 잘 불러오는 것을 확인할 수 있다.

```
df_read = pd.read_csv("sample.csv")
df_read
```

	c1	c2	c3
0	1	1.11	one
1	2	NaN	two
2	누락	3.33	three

Pandas 데이터 입출력

✓ 이번에는 column 인덱스를 배제하고 저장해보자.

✓ 아래의 예제 코드와 같이 header=False 키워드 인수를 추가해주면 된다.

```
df_csv.to_csv("sample2.csv", index=False, header=False)
```

Pandas 데이터 입출력

✓ column을 지정하지 않았기 때문에 1행의 데이터가 column으로 지정되었다.

✓ 이러한 문제를 해결하기 위해서 column을 직접 넣을 수 있다.

```
pd.read_csv("sample2.csv")
```

	1	1.11	one
0	2	NaN	two
1	누락	3.33	three

```
pd.read_csv("sample2.csv", names=["c1", "c2", "c3"])
```

	c1	c2	c3
0	1	1.11	one
1	2	NaN	two
2	누락	3.33	three

Pandas 데이터 입출력

✓ 데이터를 다운받아서 폴더에 넣고 파일을 읽어보자.

```
pd.read_csv("pokemon.csv")
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

800 rows × 13 columns

Pandas 데이터 입출력

✓ 파일 중에 건너 뛰어야 할 상단 행이 있으면 skiprows 인수를 사용하면 된다.

✓ 건너 뛸 줄을 리스트 안에 작성해도 되고 리스트가 아닌 range(2)를 활용할 수도 있다.

```
pd.read_csv("pokemon.csv", skiprows=[0,1])
```

	2	Ivysaur	Grass	Poison	405	60	62	63	80	80.1	60.1	1	False
0	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
1	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
2	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
3	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
4	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
793	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
794	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
795	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
796	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
797	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

798 rows × 13 columns

```
pd.read_csv("pokemon.csv", skiprows=range(2))
```

	2	Ivysaur	Grass	Poison	405	60	62	63	80	80.1	60.1	1	False
0	3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
1	3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
2	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
3	5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
4	6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
793	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
794	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
795	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
796	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
797	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

798 rows × 13 columns

Pandas 데이터 입출력

✓ 데이터로 불러올 자료 안 특정한 값을 NaN으로 취급하고 싶으면 na_values 인수에 NaN 값으로 취급할 값을 넣는다

```
pd.read_csv("pokemon.csv", na_values=["Grass"])
```

	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	1	Bulbasaur	NaN	Poison	318	45	49	49	65	65	45	1	False
1	2	Ivysaur	NaN	Poison	405	60	62	63	80	80	60	1	False
2	3	Venusaur	NaN	Poison	525	80	82	83	100	100	80	1	False
3	3	VenusaurMega Venusaur	NaN	Poison	625	80	100	123	122	120	80	1	False
4	4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

800 rows × 13 columns

Pandas 데이터 입출력

✓ 불러올 때와 마찬가지로 저장할 때도 na_rep 키워드 인수를 사용해서 NaN 표시값을 바꿀 수도 있습니다. 아래의 코드를 보면 NaN 값을 '누락'으로 변경해서 저장한다.

```
df_na = pd.read_csv("pokemon.csv", na_values=["Grass"])
df_na.to_csv("df_na_sample.csv", na_rep="누락")
```

```
pd.read_csv("df_na_sample.csv")
```

	Unnamed: 0	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0	0	1	Bulbasaur	누락	Poison	318	45	49	49	65	65	45	1	False
1	1	2	Ivysaur	누락	Poison	405	60	62	63	80	80	60	1	False
2	2	3	Venusaur	누락	Poison	525	80	82	83	100	100	80	1	False
3	3	3	VenusaurMega Venusaur	누락	Poison	625	80	100	123	122	120	80	1	False
4	4	4	Charmander	Fire	누락	309	39	52	43	60	50	65	1	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
795	795	719	Diancie	Rock	Fairy	600	50	100	150	100	150	50	6	True
796	796	719	DiancieMega Diancie	Rock	Fairy	700	50	160	110	160	110	110	6	True
797	797	720	HoopaHoopa Confined	Psychic	Ghost	600	80	110	60	150	130	70	6	True
798	798	720	HoopaHoopa Unbound	Psychic	Dark	680	80	160	60	170	130	80	6	True
799	799	721	Volcanion	Fire	Water	600	80	110	120	130	90	70	6	True

800 rows × 14 columns

Pandas 데이터 입출력

✓ 웹상에는 다양한 데이터 파일이 CSV 파일 형태로 제공되는데, read_csv 명령 사용시 path 대신 URL을 지정하면 Pandas가 지정 해당 파일을 다운로드하여 읽어들인다

```
titanic = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/train.csv")
titanic
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y
...	...	...	...	...	...	...	...	...	...	...
622	0	male	28.0	0	0	10.5000	Second	unknown	Southampton	y
623	0	male	25.0	0	0	7.0500	Third	unknown	Southampton	y
624	1	female	19.0	0	0	30.0000	First	B	Southampton	y
625	0	female	28.0	1	2	23.4500	Third	unknown	Southampton	n
626	0	male	32.0	0	0	7.7500	Third	unknown	Queenstown	y

627 rows × 10 columns

데이터 출력 - head()/tail()

✓ 만약 앞이나 뒤의 특정 개수만 보고 싶다면 head() 메서드나 tail() 메서드를 사용하면 된다.

✓ 메서드 인수로 출력할 행의 수를 넣으면 된다.

```
titanic.head()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

```
titanic.tail()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
622	0	male	28.0	0	0	10.50	Second	unknown	Southampton	y
623	0	male	25.0	0	0	7.05	Third	unknown	Southampton	y
624	1	female	19.0	0	0	30.00	First	B	Southampton	y
625	0	female	28.0	1	2	23.45	Third	unknown	Southampton	n
626	0	male	32.0	0	0	7.75	Third	unknown	Queenstown	y

데이터 출력 - head()/tail()

✓ 만약 앞이나 뒤의 특정 개수만 보고 싶다면 head() 메서드나 tail() 메서드를 사용하면 된다.

✓ 메서드 인수로 출력할 행의 수를 넣으면 된다.

```
titanic.head()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y

```
titanic.tail()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone
622	0	male	28.0	0	0	10.50	Second	unknown	Southampton	y
623	0	male	25.0	0	0	7.05	Third	unknown	Southampton	y
624	1	female	19.0	0	0	30.00	First	B	Southampton	y
625	0	female	28.0	1	2	23.45	Third	unknown	Southampton	n
626	0	male	32.0	0	0	7.75	Third	unknown	Queenstown	y

데이터 출력 - nunique()

✓ nunique() 메서드는 고**유한 값의 개수를 계산**할 때 사용한다.

✓ DataFrame 객체는 nunique() 메서드에 대해서 각 컬럼마다 갖는 고유 값을 Series 객체로 반환한다.

```
titanic.nunique()
```

```
survived               2
sex                    2
age                   76
n_siblings_spouses     7
parch                  6
fare                 216
class                  3
deck                   8
embark_town            4
alone                  2
dtype: int64
```

데이터 출력 - count()

- ✓ DataFrame 객체의 count() 메서드는 컬럼마다의 데이터의 개수를 계산한다.
- ✓ 이때 nan인 값에 대해서는 개수에 포함시키지 않는다.
  - ○ 컬럼마다의 개수를 보여줘야해서 Series 객체로 값을 반환한다.
- ✓ DataFrame 객체를 인수로 해서 len() 함수를 사용하면 row index의 전체 크기를 알려주는데, nan 값과 관계없이 전체를 세기 때문에 단 하나의 값을 정수로 반환.

데이터 출력 - value_counts()

- ✓ count()와 마찬가지로 고유값의 개수를 카운팅해주는 메소드이다.
- ✓ 다양한 옵션값을 설정할 수 있다.
  - ○ 오름차순 정렬 : ascending=True
  - ○ Na값을 집계에 포함시키려면 dropna=True
  - ○ 노말라이즈 : normalize=True
  - ○ 그 이외에도 bins, sort 등의 다양한 옵션이 존재한다.

데이터 출력 - value_counts()

```
titanic['survived'].value_counts()
```

```
survived
0    384
1    243
Name: count, dtype: int64
```

```
titanic['class'].value_counts(normalize=True)
```

```
class
Third     0.543860
First     0.253589
Second    0.202552
Name: proportion, dtype: float64
```

```
titanic[['class','sex']].value_counts()
```

```
class   sex
Third   male      248
        female     93
First   male       90
Second  male       72
First   female     69
Second  female     55
Name: count, dtype: int64
```

데이터 출력 - count()

✓ titanic.count()의 개수와 len(titanic)의 개수가 다른 것을 볼 수 있다.

```
titanic.count()
```

```
survived             627
sex                  627
age                  627
n_siblings_spouses   627
parch                627
fare                 627
class                627
deck                 627
embark_town          627
alone                627
dtype: int64
```

```
len(titanic)
```

```
627
```

데이터 출력 - dtypes

- ✓ titanic의 각 컬럼에 대해 dtype을 조회해볼 수 있는데 이때는 dtypes 속성을 사용한다.
- ✓ object type은 주로 문자열 혹은 문자열+숫자의 혼합일 때 주로 나타난다.

```
titanic.dtypes
```

```
survived                int64
sex                    object
age                   float64
n_siblings_spouses      int64
parch                   int64
fare                  float64
class                  object
deck                   object
embark_town            object
alone                  object
dtype: object
```

데이터 출력 - describe()

✓ describe() 메서드는 수치 값을 갖는 DataFrame의 각 컬럼에 대해 count, mean, std, min, 25%, median(50%), 75%, max에 대한 모든 통계를 구해준다.

```
titanic.describe()
```

	survived	age	n_siblings_spouses	parch	fare
count	627.000000	627.000000	627.000000	627.000000	627.000000
mean	0.387560	29.631308	0.545455	0.379585	34.385399
std	0.487582	12.511818	1.151090	0.792999	54.597730
min	0.000000	0.750000	0.000000	0.000000	0.000000
25%	0.000000	23.000000	0.000000	0.000000	7.895800
50%	0.000000	28.000000	0.000000	0.000000	15.045800
75%	1.000000	35.000000	1.000000	0.000000	31.387500
max	1.000000	80.000000	8.000000	5.000000	512.329200

결측치 다루기

✓ series때 다뤘던 것과 마찬가지로 동일한 메소드를 사용할 수 있다.

✓ 특히 .isnull().sum()과 같은 결측치 확인 명령어는 알아두면 유용하다.

결측값 확인	내용
• isnull(데이터명) / 데이터명. Isnull	관측치가 결측이면 True
• notnull(데이터명) / 데이터명.notnull	관측치가 결측이면 False
• 데이터명.isnull().sum()	칼럼별 결측값 개수
• 데이터명.isnull().sum(1)	행(row) 단위로 결측값 개수
• 데이터명.notnull().sum(1)	행(row) 단위로 실측값 개수
• dropna()	결측값이 있는 축 제외
• fillnma()	누락된 값을 대체하거나 ffill 이나 bfill 메소드를 이용해 대체

결측치 다루기

./ 다음과 같은 데이터를 만들자

```
import numpy as np
# 결측치가 포함된 샘플 데이터셋 생성
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eva"],
    "Age": [25, np.nan, 35, 40, np.nan],
    "Salary": [50000, 60000, np.nan, 80000, 90000],
    "Department": ["HR", "Finance", np.nan, "IT", "Marketing"]
}

df = pd.DataFrame(data)
```

```
df
```

	Name	Age	Salary	Department
0	Alice	25.0	50000.0	HR
1	Bob	NaN	60000.0	Finance
2	Charlie	35.0	NaN	NaN
3	David	40.0	80000.0	IT
4	Eva	NaN	90000.0	Marketing

결측치 다루기

✓ isnull()/notnull()은 결과값이 반대이다.

```
df.isnull()
```

	Name	Age	Salary	Department
0	False	False	False	False
1	False	True	False	False
2	False	False	True	True
3	False	False	False	False
4	False	True	False	False

```
df.notnull()
```

	Name	Age	Salary	Department
0	True	True	True	True
1	True	False	True	True
2	True	True	False	False
3	True	True	True	True
4	True	False	True	True

```
df.isnull().sum()
```

```
Name          0
Age           2
Salary        1
Department    1
dtype: int64
```

```
df.notnull().sum()
```

```
Name          5
Age           3
Salary        4
Department    4
dtype: int64
```

결측치 다루기

✓ dropna()는 결측치가 존재하는 행을 제외, fillna()는 결측치를 원하는 값으로 채울 수 있다.

```
df.dropna()
```

	Name	Age	Salary	Department
0	Alice	25.0	50000.0	HR
3	David	40.0	80000.0	IT

```
df.fillna("누락")
```

	Name	Age	Salary	Department
0	Alice	25.0	50000.0	HR
1	Bob	누락	60000.0	Finance
2	Charlie	35.0	누락	누락
3	David	40.0	80000.0	IT
4	Eva	누락	90000.0	Marketing

DataFrame 차원 확인

✓ 데이터의 차원과 형태를 확인하기 위해서는 ndim과 shape 속성을 사용할 수 있다.

- ○ DataFrame은 2차원 형태를 가지는 경우가 많기 numpy에서 더 자주 사용하게 된다.

✓ ndim은 차원의 수를 반환하고 shape은 데이터의 형태를 반환한다.

```
titanic.ndim
```

```
2
```

```
titanic.shape
```

```
(627, 10)
```

pandas 연습 문제

✓ 데이터를 다운받아서 파일을 읽어오고 다음과 같은 문제에 답하세요.

○ nba 데이터에 대한 파악을 위해 각 컬럼이 어떤 타입을 갖는지 확인해봅시다.

○ 또 각각의 데이터 유형이 몇개씩 존재하나 확인해 봅시다.

```
Name          object
Team          object
Number       float64
Position      object
Age          float64
Height        object
Weight       float64
College       object
Salary       float64
dtype: object
```

```
object     5
float64    4
Name: count, dtype: int64
```

pandas 연습 문제 해답

```
nba_df.dtypes
```

```
Name          object
Team          object
Number       float64
Position      object
Age          float64
Height        object
Weight       float64
College       object
Salary       float64
dtype: object
```

```
nba_df.dtypes.value_counts()
```

```
object     5
float64    4
Name: count, dtype: int64
```

pandas 연습 문제

✓ 데이터를 다운받아서 파일을 읽어오고 다음과 같은 문제에 답하세요.

○ nba 데이터의 차원의 수, 모양, 컬럼 인덱스, 로우 인덱스를 각각 구해보세요.

pandas 연습 문제 해답

```
nba_df.ndim
```

```
2
```

```
nba_df.shape
```

```
(458, 9)
```

```
nba_df.columns
```

```
Index(['Name', 'Team', 'Number', 'Position', 'Age', 'Height', 'Weight',
       'College', 'Salary'],
      dtype='object')
```

```
nba_df.index
```

```
RangeIndex(start=0, stop=458, step=1)
```

pandas 연습 문제

✓ 데이터를 다운받아서 파일을 읽어오고 다음과 같은 문제에 답하세요.

- ○ nba 데이터에 결측치를 갖는 컬럼이 존재하는지 확인해 봅시다. 그리고 존재한다면 몇개의 데이터나 결측치 값을 갖는지 확인해보세요.
- ○ 또한 각 컬럼별 결측치를 제외한 데이터의 수를 확인해보세요.

pandas 연습 문제 해답

```
nba_df.isnull().sum()
```

```
Name          1
Team          1
Number        1
Position      1
Age           1
Height        1
Weight        1
College      85
Salary       12
dtype: int64
```

```
nba_df.count()
```

```
Name         457
Team         457
Number       457
Position     457
Age          457
Height       457
Weight       457
College      373
Salary       446
dtype: int64
```

pandas 연습 문제

✓ 데이터를 다운받아서 파일을 읽어오고 다음과 같은 문제에 답하세요.

- ○ nba 데이터에 각 컬럼마다 고유한 값을 몇개씩 갖는지 조회해보세요.
- ○ Salary 컬럼의 평균, 최대 최소 등 다양한 통계량을 확인해보세요.

pandas 연습 문제 해답

```
nba_df.nunique()
```

```
Name        457
Team         30
Number       53
Position      5
Age          22
Height       18
Weight       87
College     118
Salary      309
dtype: int64
```

```
# 과학적 표기법 비활성화
pd.set_option('display.float_format', '{:.0f}'.format)

nba_df['Salary'].describe()
```

```
count        446
mean     4842684
std      5229238
min        30888
25%      1044792
50%      2839073
75%      6500000
max     25000000
Name: Salary, dtype: float64
```

고급 인덱싱

- ✓ DataFrame에서 특정한 데이터만 골라내는 것을 인덱싱(indexing)이라고 한다.
- ✓ 그런데 Pandas는 NumPy 배열과 같이 콤마(,)를 사용한 (row 인덱스, column 인덱스) 형식의 2차원 인덱싱을 지원하기 위해 다음과 같은 특별한 인덱서(indexer) 속성도 제공한다.
  - ○ loc : label 값 기반의 2차원 인덱싱
  - ○ iloc : 순서를 나타내는 정수 기반의 2차원 인덱싱

고급 인덱싱 - loc

✓ 다음과 같은 데이터를 만들어보자

```
# 10x10 데이터프레임 생성 (인덱스를 처음부터 row_로 설정)
data = np.arange(1, 101).reshape(10, 10)
index = [f'row_{i+1}' for i in range(10)]
columns = [f'col_{i+1}' for i in range(10)]
df = pd.DataFrame(data, columns=columns, index=index)
```

df

	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
row_1	1	2	3	4	5	6	7	8	9	10
row_2	11	12	13	14	15	16	17	18	19	20
row_3	21	22	23	24	25	26	27	28	29	30
row_4	31	32	33	34	35	36	37	38	39	40
row_5	41	42	43	44	45	46	47	48	49	50
row_6	51	52	53	54	55	56	57	58	59	60
row_7	61	62	63	64	65	66	67	68	69	70
row_8	71	72	73	74	75	76	77	78	79	80
row_9	81	82	83	84	85	86	87	88	89	90
row_10	91	92	93	94	95	96	97	98	99	100

고급 인덱싱 - loc

✓ loc 인덱서는 다음처럼 사용할 수 있다.

df.loc[row 인덱스]

df.loc[row 인덱스, column 인덱스]

```
df.loc["row_1"]
```

```
col_1      1
col_2      2
col_3      3
col_4      4
col_5      5
col_6      6
col_7      7
col_8      8
col_9      9
col_10    10
Name: row_1, dtype: int32
```

```
df.loc["row_1","col_1"]
```

```
1
```

고급 인덱싱 - loc

✓ 인덱스 데이터의 슬라이스도 가능하다. 이 때는 사실 loc를 쓰지 않을 때와 결과가 동일하다.

```
df.loc["row_1":"row_3"]
```

	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
row_1	1	2	3	4	5	6	7	8	9	10
row_2	11	12	13	14	15	16	17	18	19	20
row_3	21	22	23	24	25	26	27	28	29	30

```
df["row_1":"row_3"]
```

	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
row_1	1	2	3	4	5	6	7	8	9	10
row_2	11	12	13	14	15	16	17	18	19	20
row_3	21	22	23	24	25	26	27	28	29	30

고급 인덱싱 - loc

✓ 인덱스 데이터의 리스트 자료형도 사용 가능하다.

○ 이 때는 loc를 쓰지 않으면 KeyError 오류가 발생

```
df.loc[["row_1","row_3"]]
```

	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9	col_10
row_1	1	2	3	4	5	6	7	8	9	10
row_3	21	22	23	24	25	26	27	28	29	30

고급 인덱싱 - loc

✓ Boolean Series로 row를 기준으로 인덱싱할 수 있다.

○ 아래 예제에서는 df.col_1(영어 문자열은 속성처럼 접근 가능)의 값 중 50 초과인 결과를 Boolean Series 값을 얻을 수 있습니다. 이 Boolean Series를 활용해 인덱싱하고 있다.

```
select_value = df.col_1>50
print(select_value)
```

```
row_1     False
row_2     False
row_3     False
row_4     False
row_5     False
row_6      True
row_7      True
row_8      True
row_9      True
row_10     True
Name: col_1, dtype: bool
```

```
df.loc[select_value].col_1
```

```
row_6     51
row_7     61
row_8     71
row_9     81
row_10    91
Name: col_1, dtype: int32
```

고급 인덱싱 - loc

✓ Boolean Series로 row를 기준으로 인덱싱할 수 있다.

○ 아래 예제에서는 df.col_1(영어 문자열은 속성처럼 접근 가능)의 값 중 50 초과인 결과를 Boolean Series 값을 얻을 수 있습니다. 이 Boolean Series를 활용해 인덱싱하고 있다.

```
select_value = df.col_1>50
print(select_value)
```

```
row_1     False
row_2     False
row_3     False
row_4     False
row_5     False
row_6      True
row_7      True
row_8      True
row_9      True
row_10     True
Name: col_1, dtype: bool
```

```
df.loc[select_value].col_1
```

```
row_6     51
row_7     61
row_8     71
row_9     81
row_10    91
Name: col_1, dtype: int32
```

고급 인덱싱 - loc vs iloc

✓ iloc은 loc과 다르게 label 인덱스가 아닌 숫자로된 인덱스에 접근하기에 우리가 아는 슬라이싱 방식과 동일하게 포함하지 않습니다.

○ 데이터의 인덱스와 컬럼을 0부터 시작되게 변경하세요.

```
df.iloc[1:2]
```

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9
row_1	11	12	13	14	15	16	17	18	19	20

```
df.iloc[1,3]
```

```
14
```

고급 인덱싱 - loc vs iloc

✓ 일반적인 슬라이싱과 마찬가지로 행과 열에 각각 슬라이싱을 적용할 수 있다.

```
df.iloc[1:5,3:6]
```

	col_3	col_4	col_5
row_1	14	15	16
row_2	24	25	26
row_3	34	35	36
row_4	44	45	46

```
df.iloc[1:4,2:7:2]
```

	col_2	col_4	col_6
row_1	13	15	17
row_2	23	25	27
row_3	33	35	37

고급 인덱싱 - 조건 필터링

✓ 조건을 걸어서 특정 원하는 데이터만 필터링이 가능하다.

`col_5`가 `50`보다 큰 행만 필터링

```
filtered_df = df[df['col_5'] > 50]
print(filtered_df)
```

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9
row_5	51	52	53	54	55	56	57	58	59	60
row_6	61	62	63	64	65	66	67	68	69	70
row_7	71	72	73	74	75	76	77	78	79	80
row_8	81	82	83	84	85	86	87	88	89	90
row_9	91	92	93	94	95	96	97	98	99	100

`col_5`가 30보다 크고, `col_7`이 70보다 작은 행 필터링

```
filtered_df = df[(df['col_5'] > 30) & (df['col_7'] < 70)]
print(filtered_df)
```

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8	col_9
row_3	31	32	33	34	35	36	37	38	39	40
row_4	41	42	43	44	45	46	47	48	49	50
row_5	51	52	53	54	55	56	57	58	59	60
row_6	61	62	63	64	65	66	67	68	69	70

연습 문제

1. 타이타닉호 승객의 평균 나이를 구하세요. 29.6
2. 타이타닉호 승객중 여성 승객의 평균 나이를 구하세요. 28.7
3. 타이타닉호 승객중 1등실(class=="First") 선실의 여성 승객의 평균 나이를 구하세요. 34.3

평균은 **mean()** 메소드를 통해서 구할 수 있다.

```
df['col_1'].mean()
```

```
47.0
```

연습 문제 해답

```
# 승객의 평균 나이
round(titanic['age'].mean(),1)
```

```
29.6
```

```
# 여성 승객의 평균 나이
round(titanic[titanic['sex']=='female']['age'].mean(),1)
```

```
28.7
```

```
# 1등급 선실의 여성 승객의 평균 나이
round(titanic[(titanic['class']=="First") & (titanic['sex']=='female')]['age'].mean(),1)
```

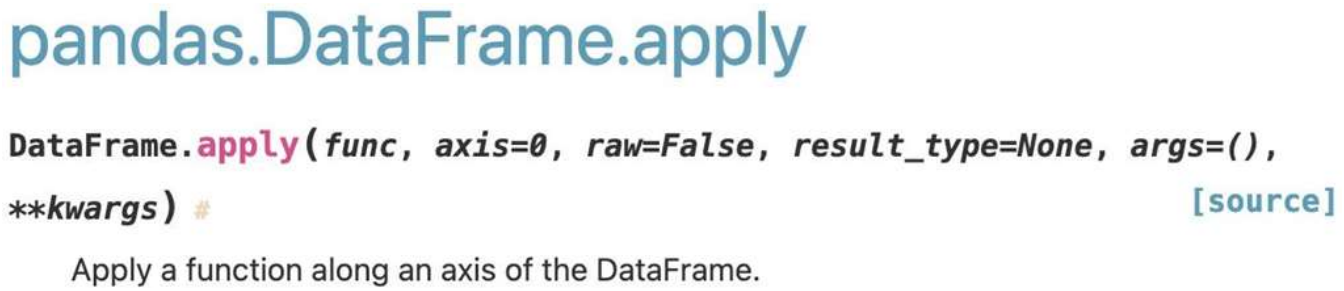
```
34.3
```

각 값에 대해 함수 적용하기 - apply()

✓ DataFrame에 대해 Function을 적용할 때에는 apply()를 활용하면 좋다.

✓ 이 메서드는 첫 인자로 함수를 필수 값으로 받는다. 경우에 따라 두 번째 인자로 axis를 사용할 수 있는데, axis 인자는 0이 default 이다.

- ○ axis가 0인 경우 행 방향으로 내려가면서 열마다 함수를 적용합니다.
- ○ axis가 1인 경우 열 방향으로 이동하면서 행마다 함수를 적용합니다.



pandas.DataFrame.apply

```
DataFrame.apply(func, axis=0, raw=False, result_type=None, args=(),
**kwargs) #                                                    [source]
```

Apply a function along an axis of the DataFrame.

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

```
# 데이터프레임 생성
df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
})
df
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9

```
df.apply(np.sqrt)
```

	A	B	C
0	1.000000	2.000000	2.645751
1	1.414214	2.236068	2.828427
2	1.732051	2.449490	3.000000

```
df.apply(lambda x: x.sum(), axis=0)
```

```
A     6
B    15
C    24
dtype: int64
```

```
df.apply(lambda x: x.sum(), axis=1)
```

```
0     9
1    12
2    15
dtype: int64
```

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

```python
# 각 열의 평균 계산
result = df.apply(lambda x: x.mean(), axis=0)

print("원본 데이터프레임:")
print(df)
print("\n각 열의 평균:")
print(result)
```

```
원본 데이터프레임:
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

각 열의 평균:
A    2.0
B    5.0
C    8.0
dtype: float64
```

```python
# 각 행의 평균 계산
result = df.apply(lambda x: x.mean(), axis=1)

print("원본 데이터프레임:")
print(df)
print("\n각 행의 평균:")
print(result)
```

```
원본 데이터프레임:
   A  B  C
0  1  4  7
1  2  5  8
2  3  6  9

각 행의 평균:
0    4.0
1    5.0
2    6.0
dtype: float64
```

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

```
# A열과 B열의 차이를 계산하여 새로운 열 생성
df['A-B'] = df.apply(lambda x: x['A'] - x['B'], axis=1)

print("원본 데이터프레임:")
print(df)
```

```
원본 데이터프레임:
   A  B  C  A-B
0  1  4  7   -3
1  2  5  8   -3
2  3  6  9   -3
```

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

○ 데이터 프레임에 새로운 열을 추가할 때 행마다 매핑을 하기 때문에 axis=1로 계산한다.

```
# 각 열의 최대값과 최소값의 차이를 구함
df.apply(lambda x: x.max() - x.min(), axis=0)
```

```
A    2
B    2
C    2
dtype: int64
```

```
# 각 행의 최대값과 최소값의 차이를 구함
df.apply(lambda x: x.max() - x.min(), axis=1)
```

```
0    6
1    6
2    6
dtype: int64
```

```
# 데이터 프레임에 새로운 열을 추가
df['max-min']=df.apply(lambda x: x.max() - x.min(), axis=1)
df
```

	A	B	C	max-min
0	1	4	7	6.0
1	2	5	8	6.0
2	3	6	9	6.0

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

○ 열마다의 계산값을 추가하고 싶다면 행에 추가하면 된다.

```
# 데이터 프레임에 새로운 열을 추가
col_max_min = df.apply(lambda x: x.max() - x.min(), axis=0)
```

```
col_max_min
```

```
A    2
B    2
C    2
dtype: int64
```

```
df.loc['max-min'] = col_max_min
```

```
df
```

	A	B	C
0	1	4	7
1	2	5	8
2	3	6	9
max-min	2	2	2

각 값에 대해 함수 적용하기 - apply()

✓ 다음과 같은 데이터 프레임을 만들고 함수를 적용해보자.

○ 람다식을 활용하여 조건을 줄 수 있다.

```
titanic["adult/child"] = titanic.apply(lambda r: "adult" if r.age >= 20 else "child", axis=1)
titanic.tail()
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone	adult/child
622	0	male	28.0	0	0	10.50	Second	unknown	Southampton	y	adult
623	0	male	25.0	0	0	7.05	Third	unknown	Southampton	y	adult
624	1	female	19.0	0	0	30.00	First	B	Southampton	y	child
625	0	female	28.0	1	2	23.45	Third	unknown	Southampton	n	adult
626	0	male	32.0	0	0	7.75	Third	unknown	Queenstown	y	adult

apply() 연습 문제

✓ 타이타닉호의 승객에 대해 나이와 성별에 의한 카테고리 column인 category1 열을 만들어보세요. category1 카테고리는 다음과 같이 정의됩니다.

1. 20살이 넘으면 성별을 그대로 사용합니다.
2. 20살 미만이면 성별에 관계없이 “child”라고 합니다.

apply() 연습 문제 해답

```
titanic['category1'] = titanic.apply(lambda x: x['sex'] if x['age'] >= 20 else 'child', axis=1)
titanic
```

	survived	sex	age	n_siblings_spouses	parch	fare	class	deck	embark_town	alone	adult/child	category1
0	0	male	22.0	1	0	7.2500	Third	unknown	Southampton	n	adult	male
1	1	female	38.0	1	0	71.2833	First	C	Cherbourg	n	adult	female
2	1	female	26.0	0	0	7.9250	Third	unknown	Southampton	y	adult	female
3	1	female	35.0	1	0	53.1000	First	C	Southampton	n	adult	female
4	0	male	28.0	0	0	8.4583	Third	unknown	Queenstown	y	adult	male
...	...	...	...	...	...	...	...	...	...	...	...	...
622	0	male	28.0	0	0	10.5000	Second	unknown	Southampton	y	adult	male
623	0	male	25.0	0	0	7.0500	Third	unknown	Southampton	y	adult	male
624	1	female	19.0	0	0	30.0000	First	B	Southampton	y	child	child
625	0	female	28.0	1	2	23.4500	Third	unknown	Southampton	n	adult	female
626	0	male	32.0	0	0	7.7500	Third	unknown	Queenstown	y	adult	male

DataFrame 합성

✓ pandas는 두 개 이상의 DataFrame을 하나로 합치는 데이터 병합(merge)이나 연결(concatenate)을 지원한다.

pandas.DataFrame.join

```
DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='',
sort=False, validate=None) #                                    [source]
```

Join columns of another DataFrame.

pandas.DataFrame.merge

```
DataFrame.merge(right, how='inner', on=None, left_on=None,
right_on=None, left_index=False, right_index=False, sort=False,
suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)
```

Merge DataFrame or named Series objects with a database-style join. [source]

DataFrame 합성

✓ 다음과 같이 2개의 데이터 프레임을 만들자.

✓ 두 데이터 프레임의 공통 column 혹은 index를 기준으로 2개의 테이블을 합칠 수 있다.

○ 이 때 기준이 되는 column, row의 데이터를 key라고 표현한다.

```
df1 = pd.DataFrame({
    '고객번호': [1001, 1002, 1003, 1004, 1005, 1006, 1007],
    '이름': ['둘리', '도우너', '또치', '길동', '희동', '마이콜', '영희']
}, columns=['고객번호', '이름'])

df1
```

	고객번호	이름
0	1001	둘리
1	1002	도우너
2	1003	또치
3	1004	길동
4	1005	희동
5	1006	마이콜
6	1007	영희

```
df2 = pd.DataFrame({
    '고객번호': [1001, 1001, 1005, 1006, 1008, 1001],
    '금액': [10000, 20000, 15000, 5000, 100000, 30000]
}, columns=['고객번호', '금액'])

df2
```

	고객번호	금액
0	1001	10000
1	1001	20000
2	1005	15000
3	1006	5000
4	1008	100000
5	1001	30000

DataFrame 합성 - merge()

✓ merge 함수로 위의 두 DataFrame df1, df2 를 합치면 공통 column인 고객번호 column을 기준으로 데이터를 찾아서 합친다.

○ 양쪽 DataFrame에 모두 키가 존재하는 데이터만 보여주는 inner join 방식을 사용한다.

```
1  pd.merge(df1, df2)
```

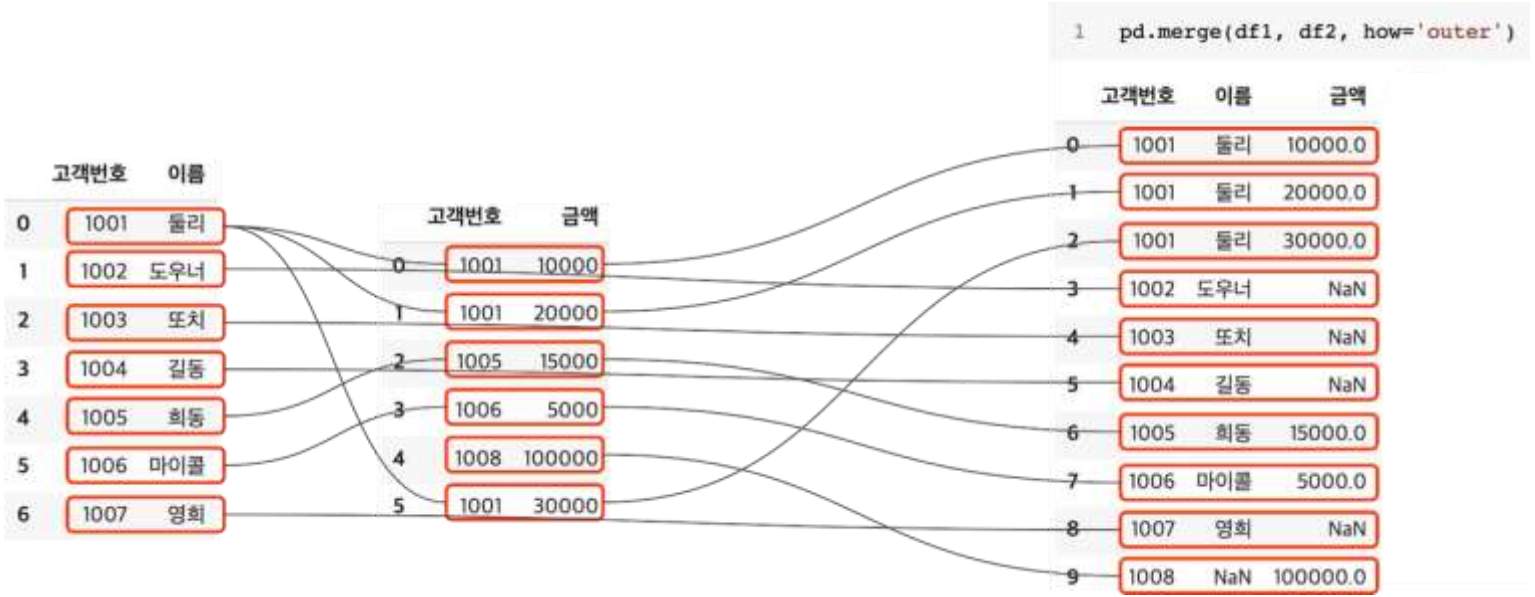
	고객번호	이름
0	1001	둘리
1	1002	도우너
2	1003	또치
3	1004	길동
4	1005	희동
5	1006	마이콜
6	1007	영희

	고객번호	금액
0	1001	10000
1	1001	20000
2	1005	15000
3	1006	5000
4	1008	100000
5	1001	30000

	고객번호	이름	금액
0	1001	둘리	10000
1	1001	둘리	20000
2	1001	둘리	30000
3	1005	희동	15000
4	1006	마이콜	5000

DataFrame 합성 - merge()

✓ outer join을 진행하면 키 값이 한쪽에만 있어도 데이터를 합친다.



DataFrame 합성 - merge()

✓ left와 right를 인수를 줄 수 있는데 각각 첫 번째 인수, 두 번째 인수를 기준으로 합친다.

```
pd.merge(df1,df2, how="left")
```

	고객번호	이름	금액
0	1001	둘리	10000.0
1	1001	둘리	20000.0
2	1001	둘리	30000.0
3	1002	도우너	NaN
4	1003	또치	NaN
5	1004	길동	NaN
6	1005	희동	15000.0
7	1006	마이콜	5000.0
8	1007	영희	NaN

```
pd.merge(df1,df2, how="right")
```

	고객번호	이름	금액
0	1001	둘리	10000
1	1001	둘리	20000
2	1005	희동	15000
3	1006	마이콜	5000
4	1008	NaN	100000
5	1001	둘리	30000

DataFrame 그룹 분석

- ✓ 만약 키가 지정하는 조건에 맞는 데이터가 하나 이상이라서 데이터 그룹을 이루는 경우에는 그룹의 특성을 보여주는 그룹분석(group analysis)을 해야 한다.
- ✓ 그룹분석은 키에 의해서 결정되는 데이터가 여러개가 있을 경우 미리 지정한 연산을 통해 그 그룹 데이터의 대표값을 계산한다.
- ✓ pandas에서는 groupby 메서드를 사용하여 다음처럼 그룹분석을 진행한다.
  - ○ 1.분석하고자 하는 Series나 DataFrame에 groupby 메서드를 호출하여 그룹화를 한다.
  - ○ 2.그룹 객체에 대해 그룹연산을 수행한다.

DataFrame 그룹 분석

- ✓ groupby 메서드는 데이터를 그룹 별로 분류하는 역할을 한다.
- ✓ groupby 메서드의 인수로는 다음과 같은 값을 사용합니다.
  - ○ column 또는 column의 리스트
  - ○ row 인덱스
- ✓ 연산 결과로 그룹 데이터를 나타내는 GroupBy 클래스 객체를 반환하는데, 이 객체에는 그룹별로 연산을 할 수 있는 그룹연산 메서드가 있다.

DataFrame 그룹 분석

- ✓ groupby 결과, 즉 GroupBy 클래스 객체의 뒤에 붙일 수 있는 그룹연산 메서드는 다양하다.
- ✓ 다음은 자주 사용되는 그룹연산 메서드들이다.
  - ○ size(), count(): 그룹 데이터의 개수
  - ○ mean(), median(), min(), max(): 그룹 데이터의 평균, 중앙값, 최소, 최대
  - ○ sum(), prod(), std(), var(), quantile() : 합계, 곱, 표준편차, 분산, 사분위수
  - ○ first(), last(): 그룹 데이터 중 가장 첫번째 데이터와 가장 나중 데이터

DataFrame 그룹 분석

✓ 예를 들어 다음과 같은 데이터가 있을 때 key1의 값(A 또는 B)에 따른 data1의 합계는 어떻게 구할수 있을까?

```
df2 = pd.DataFrame({
    'key1': ['A', 'A', 'B', 'B', 'A'],
    'key2': ['one', 'two', 'one', 'two', 'one'],
    'data1': [1, 2, 3, 4, 5],
    'data2': [10, 20, 30, 40, 50]
})
df2
```

	key1	key2	data1	data2
0	A	one	1	10
1	A	two	2	20
2	B	one	3	30
3	B	two	4	40
4	A	one	5	50

DataFrame 그룹 분석

✓ groupby 명령을 사용하여 그룹 A와 그룹 B로 구분한 그룹 데이터를 만든다.

```
groups = df2.groupby(df2.key1)
groups
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001EBC0FEFFE0>
```

✓ 이 GroupBy 클래스 객체에는 각 그룹 데이터의 인덱스를 저장한 groups 속성이 있다.

```
groups.groups
```

```
{'A': [0, 1, 4], 'B': [2, 3]}
```

DataFrame 그룹 분석

✓ 이제 sum()을 이용하여 합계를 구할 수 있다.

```
groups.data1.sum()
```

```
key1
A    8
B    7
Name: data1, dtype: int64
```

```
df2.data1.groupby(df2.key1).sum()
```

```
key1
A    8
B    7
Name: data1, dtype: int64
```

DataFrame 그룹 분석

✓ 다양한 표현방식이 있을 수 있는데 groupby 후 컬럼을 지정하여 연산하거나 연산 후 컬럼을 지정해도 상관없다.

```
df2.groupby(df2.key1)['data1'].sum()
```

```
key1
A    8
B    7
Name: data1, dtype: int64
```

```
df2.groupby(df2.key1).sum()['data1']
```

```
key1
A    8
B    7
Name: data1, dtype: int64
```

DataFrame 그룹 분석

✓ 복합 키 (key1, key2) 값에 따른 data1의 합계를 구할수 있다.

✓ 분석하고자 하는 키가 복수이면 리스트를 사용.

```
df2.data1.groupby([df2.key1, df2.key2]).sum()
```

```
key1  key2
A     one     6
      two     2
B     one     3
      two     4
Name: data1, dtype: int64
```

DataFrame 피벗 테이블

✓ 피벗 테이블은 데이터프레임에서 특정 열을 기준으로 데이터를 그룹화하고, 집계 함수(합계, 평균 등)를 적용하여 요약된 표를 만드는 데 사용된다.

- ○ 엑셀의 피벗 테이블과 유사한 역할을 한다.

✓ 주요 매개변수로는 다음과 같다.

`index`: 행 인덱스로 사용할 열.

`columns`: 열로 사용할 데이터.

`values`: 계산할 데이터(값).

`aggfunc`: 데이터를 집계하는 함수(기본값은 **`mean`**) (**`sum`**, **`mean`**, **`count`**, **`min`**, **`max`** 등)

DataFrame 피벗 테이블

✓ 결과를 보면 연도별 각 지역의 매출 합을 쉽게 계산되어진 것을 볼 수 있다.

```python
# 데이터 생성
data = {
    '지역': ['서울', '서울', '부산', '부산', '서울', '대구', '대구'],
    '연도': [2020, 2021, 2020, 2021, 2021, 2020, 2021],
    '매출': [100, 150, 200, 250, 300, 400, 450]
}

df = pd.DataFrame(data)

# 피벗 테이블 생성
pivot_table = pd.pivot_table(
    df,
    index='지역',        # 행 인덱스
    columns='연도',      # 열
    values='매출',       # 집계 대상 값
    aggfunc='sum'        # 집계 함수 (합계)
)

print("원본 데이터프레임:")
print(df)
print("\n피벗 테이블:")
print(pivot_table)
```

```
원본 데이터프레임:
   지역    연도   매출
0  서울  2020  100
1  서울  2021  150
2  부산  2020  200
3  부산  2021  250
4  서울  2021  300
5  대구  2020  400
6  대구  2021  450

피벗 테이블:
연도  2020  2021
지역
대구   400   450
부산   200   250
서울   100   450
```

DataFrame 피벗 테이블

✓ 다음의 예시를 확인해보고 어떤 의미를 나타내는지 생각해보자

```
pivot_avg = pd.pivot_table(df, index='지역', columns='연도', values='매출', aggfunc='mean')
print(pivot_avg)
```

```
연도    2020    2021
지역
대구  400.0  450.0
부산  200.0  250.0
서울  100.0  225.0
```

```
pivot_count = pd.pivot_table(df, index='지역', columns='연도', values='매출', aggfunc='count')
print(pivot_count)
```

```
연도  2020  2021
지역
대구     1     1
부산     1     1
서울     1     2
```

```
pivot_max = pd.pivot_table(df, index='지역', columns='연도', values='매출', aggfunc='max')
print(pivot_max)
```

```
연도  2020  2021
지역
대구   400   450
부산   200   250
서울   100   300
```