

판다스 데이터 분석 (Data Analysis using Pandas)

CONTENTS

- 1 엑셀보다 빠른 일처리는 판다스로
- 2 CSV라고 들어봤니?
- 3 CSV에서 원하는 데이터를 뽑아 보자
- LAB 1** 울릉도는 몇 월에 바람이 가장 강할까?
- 4 판다스의 데이터 구조 : 시리즈와 데이터프레임
- 5 판다스로 데이터 파일을 읽기
- 6 열을 기준으로 데이터 선택하기
- 7 슬라이싱으로 행 선택하기
- 8 데이터를 간편하게 분석할 수 있는 기능이 있다
- 9 연도와 월, 일을 다루는 DatetimeIndex와 그룹핑
- 10 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑
- LAB 2** 울릉도는 몇 월에 바람이 가장 강할까? - groupby() 활용
- 11 조건에 맞게 골라내자 : 필터링
- 12 빠진 데이터를 깨끗하게 메워 보자
- 13 데이터 구조를 변경해 보자
- LAB 3** 다양한 방법으로 concat 적용해 보기
- 14 데이터베이스 join 방식의 데이터 병합 - merge
- LAB 4** 다양한 방법으로 merge 적용해 보기

이 장에서 배울 것들

- 데이터 분석 라이브러리인 판다스에 대해서 살펴봅시다.
- 쉼표로 분리된 대표적인 텍스트 자료인 csv 자료를 판다스를 이용하여 읽고 분석해 봅시다.
- csv 파일을 테이블 형식의 자료로 만든 후 분석하고 정렬하는 일을 해 봅시다.
- 데이터에서 손실된 부분인 결측 데이터를 보정해 봅시다.
- 데이터프레임을 합하여 새로운 테이블을 만들어 봅시다.
- 판다스 데이터를 matplotlib으로 시각화하는 연습도 해 봅시다.
- 조건을 이용하여 다양한 방식으로 데이터에 접근하고 연산하는 방법을 살펴보자.

1 엑셀보다 빠른 일처리는 판다스로 Panel Data

- 엑셀Excel은 행과 열로 이루어진 표에 입력된 데이터를 처리하는데 탁월한 성능을 보이고 있다.
- 앞장에서 살펴본 넘파이가 2차원 행렬matrix 형태의 데이터를 지원하지만, 넘파이는 데이터의 속성을 표시하는 행이나 열의 레이블을 가지고 있지 않다는 한계가 있다. 하지만 파이썬의 판다스pandas 패키지를 사용하면 이러한 문제를 해결할 수 있다.

Pandas : 빠른 기반 데이터 처리 지원
(인덱스와 열 이름 사용)

	A	B	C	D	E	F	G
1	학번	이름	국어	수학	과학	영어	합계
2	211101	이순신	96	90	89	87	362
3	211101	권용수	76	56	67	98	297
4	211101	이억기	89	67	98	69	323
5	211101	권율	83	89	90	87	349
6	211101	김시민	67	90	96	89	342
7	211101	이정암	80	60	79	86	305
8	211101	정문부	57	69	95	89	310

저는 행과 열로
이루어진 표의 데
이터를 잘 처리하
는 스프레드시트
프로그램인 엑셀
이라고 해요!



1 엑셀보다 빠른 일처리는 판다스로



잠깐 - 판다스의 특징

판다스는 다음과 같은 특징들을 갖는다.

1. 빠르고 효율적이며 다양한 표현력을 갖춘 자료구조.

실세계 데이터 분석을 위해 만들어진 파이썬 패키지

2. 다양한 형태의 데이터에 적합

이종*heterogeneous* 자료형의 열을 가진 테이블 데이터

시계열 데이터

레이블을 가진 다양한 행렬 데이터

다양한 관측 통계 데이터

- 3 핵심 구조

시리즈*Series* : 1차원 구조를 가진 하나의 열

데이터프레임*DataFrame* : 복수의 열을 가진 2차원 데이터

4. 판다스가 잘 하는 일

결측 데이터 처리

데이터 추가 삭제 (새로운 열의 추가, 특정 열의 삭제 등)

데이터 정렬과 다양한 데이터 조작

1 엑셀보다 빠른 일처리는 판다스로

판다스로 어떤 일을 할 수 있나

- 파이썬 리스트, 딕셔너리, 넘파이 배열을 데이터 프레임으로 변환할 수 있다.
- 판다스로 CSV 파일이나, 엑셀 파일 등을 열 수 있다.
- URL을 통해 웹 사이트의 CSV 또는 JSON과 같은 원격 파일 또는 데이터베이스를 열 수 있다.

데이터 불러오기 및 저장하기 `pd.read_csv('인수명.csv')`

- 파이썬 리스트, 파이썬 딕셔너리, 넘파이 배열을 데이터 프레임으로 변환할 수 있다.
- 판다스로 CSV 파일이나 TSV 파일, 엑셀 파일 등을 열 수 있다.
- URL을 통해 웹 사이트의 CSV 또는 JSON과 같은 원격 파일 또는 데이터베이스를 열 수 있다.



판다스는 데이터를 읽어서 테이블 형식으로 배치하고 결측 검사, 필터링 등 다양한 계산을 할 수 있어요.

1 엑셀보다 빠른 일처리는 판다스로

판다스로 어떤 일을 할 수 있나 (계속)

• 데이터 보기 및 검사

- mean()로 모든 열의 평균을 계산할 수 있다.
- corr()로 데이터 프레임의 열 사이의 상관 관계를 계산할 수 있다.
- count()로 각 데이터 프레임 열에서 null이 아닌 값의 개수를 계산할 수 있다.

`df.info()`, `df.describe()` : 개별 열의 데이터 정리를 한다

• 필터, 정렬 및 그룹화

- sort_values()로 데이터를 정렬할 수 있다.
- 조건을 사용하여 열을 필터링할 수 있다. `df['열 이름']`, `df[df['열 이름']]`
- groupby()를 이용하여 기준에 따라 몇 개의 그룹으로 데이터를 분할할 수 있다.

• 데이터 정제

- 데이터의 누락 값을 확인할 수 있다. `isna()`, `fillna()`, 중복 데이터 제거
- 특정한 값을 다른 값으로 대체할 수 있다.

① 데이터 병합 / 결합

여러 데이터프레임을
합쳐거나 결합

(concat, merge)

② 데이터 시각화

`df.plot(kind='bar')`

1 엑셀보다 빠른 일처리는 판다스로

판다스로 어떤 일을 할 수 있나 (계속)



잠깐 - 판다스 or 판다

판다스라는 특이한 이름은 "panel data"라는 용어에서 유래되었다. 이 panel data라는 용어 역시 생소한 용어인데 이는 **계량경제학**econometrics 용어로 동일한 관찰자에 의하여 여러 회에 걸쳐 관측된 데이터 집합을 지칭하는 용어이다. 중국 쓰촨성일대에 서식하는 동물인 판다와는 관계가 없으나 용어가 비슷하므로 많은 사람들이 판다스의 로고로 판다 그림을 사용하기도 한다.



판다스는 판다
하고 상관없는
데이터 분석
라이브러리예요!

2 CSV라고 들어봤니

- CSV는 테이블 형식의 데이터를 저장하고 이동하는 데 사용되는 구조화된 텍스트 파일 형식이다. CSV는 **쉼표로 구분한 변수***comma separated variables*의 약자이다.
- CSV의 역사는 1972년으로 거슬러 올라가며 Microsoft Excel와 같은 **스프레드 시트***spread sheet* 소프트웨어에 적합한 형식이다. 데이터 과학에서 사용되는 데이터 가운데 상당한 비율의 데이터들이 CSV 형식으로 공유되는 경우가 많다.



2 CSV라고 들어봤니

feature

sample

- CSV 파일은 필드를 나타내는 열과 레코드를 나타내는 행으로 구성
- 만약 데이터의 중간에 구분자가 포함되어야 한다면 따옴표를 사용하여 필드를 묶어야 함
 - 예를 들어서 'Gildong, Hong'이라는 데이터가 있다고 하자. 데이터의 중간에 쉼표(,)가 포함되어 있다. 이러한 경우에는 구분자로 사용되는 쉼표와 구분하기 위하여 반드시 데이터를 따옴표로 감싸야 한다.
- CSV 파일의 첫 번째 레코드에는 열 제목이 포함되어 있을 수 있다.
 - CSV 형식 자체의 요구사항이 아니라 단순히 일반적인 관행
- CSV 파일의 크기를 알 수 없고 잠재적으로 크기가 큰 경우 한 번에 모든 레코드를 읽지 않는 것이 좋다.
 - 이때는 현재 행을 읽고, 현재 행을 처리한 후에 삭제하고 다음 행을 가져오는 방식이 필요할 수도 있다. 아니면 특정한 크기만큼의 데이터를 읽어서 처리한 뒤에, 다음으로 또 그만큼의 크기를 가져오는 방식을 사용할 수도 있을 것이다.

2 CSV라고 들어봤니

- 이 책에서는 CSV로 저장된 데이터를 사용하는 것을 기본으로 삼을 것이다. 본격적으로 판다스를 살펴보기 전에 CSV 데이터를 처리하는 것에 대해 먼저 살펴 보자.
- 판다스는 데이터를 처리하고 분석하기 위한 모듈이므로 다양한 종류의 데이터 파일 형식을 지원한다



예제 [편집]

다음은 한 사람에 관한 정보를 갖는 JSON 객체이다.

키-값 쌍(이름:값)의 패턴으로 표현된다.

```
1  {
2      "이름": "홍길동",
3      "나이": 25,
4      "성별": "여",
5      "주소": "서울특별시 양천구 목동",
6      "특기": ["농구", "도술"],
7      "가족관계": {"#": 2, "아버지": "홍판서", "어머니": "춘섭"},
8      "회사": "경기 수원시 팔달구 우만동"
9 }
```

JavaScript Object Notation(JSON) 예시

2 CSV라고 들어봤니

CSV 데이터의 내용을 읽어 보자

- 파이썬 모듈 csv는 CSV reader와 CSV writer를 제공한다. 두 객체 모두 파일 핸들을 첫 번째 매개 변수로 사용한다. 필요한 경우 delimiter 매개 변수를 사용하여 구분자를 제공할 수 있다.
- 이 파일을 d: 드라이브의 data 폴더에 'weather.csv'로 저장했다고 가정하고, 그러면 이 파일의 경로^{path}는 'd:/data/weather.csv'가 된다.

```
import csv      # 판다스가 아닌 파이썬 csv 모듈을 사용함

f = open('d:/data/weather.csv')    # CSV 파일을 열어서 f에 저장한다.
data = csv.reader(f)              # reader() 함수를 이용하여 읽는다.
for row in data:
    print(row)
f.close()
```

```
✓ ['일시', '평균기온', '최대풍속', '평균풍속']
['2010-08-01', '28.7', '8.3', '3.4']
['2010-08-02', '25.2', '8.7', '3.8']
['2010-08-03', '22.1', '6.3', '2.9']
...
```

import pandas as pd
df = pd.read_csv('weather.csv')
print(df.head())

2 CSV라고 들어봤니

CSV 데이터의 내용을 읽어 보자 (계속)

The screenshot shows a GitHub repository page for 'dongupak / DataSciPy'. The 'Code' tab is selected. A blue callout box points from the 'weather.csv' file in the list to a detailed view of its contents. The callout box contains the following text:

2010년 8월~2020년 8월
까지 울릉도의 기온과 풍
속 데이터가 저장된 기상
데이터
출처 : 기상자료개발포털사이트

The detailed view of 'weather.csv' shows the following data:

일시	평균기온	최대풍속	평균풍속
2010-08-01	28.7	8.3	3.4
2010-08-02	25.2	8.7	3.8
2010-08-03	22.1	6.3	2.9
2010-08-04	25.3	6.6	4.2
2010-08-05	27.2	9.1	5.6
2010-08-06	26.8	9.8	8
2010-08-07	27.5	9.1	5
2010-08-08	26.6	5.9	4
2010-08-09	26.9	5.1	3.1
2010-08-10	25.6	10.2	5.5
2010-08-11	24.6	9.4	4.8
2010-08-12	23.7	8.7	2.6

2 CSV라고 들어봤니

CSV 데이터의 내용을 읽어 보자 (계속)

- 헤더를 제거하는 방법
- next() 함수를 사용함

```
import csv          # 판다스가 아닌 파이썬 csv 모듈을 사용함

f = open('d:/data/weather.csv')    # CSV 파일을 열어서 f에 저장한다.
data = csv.reader(f)              # csv의 reader() 함수를 이용하여 읽는다.
header = next(data)             # 헤더를 제거한다.
for row in data:                # 반복 루프를 사용하여 데이터를 읽는다.
    print(row)
f.close()                      # 파일을 닫는다.
```

```
['2010-08-01', '28.7', '8.3', '3.4']
['2010-08-02', '25.2', '8.7', '3.8']
['2010-08-03', '22.1', '6.3', '2.9']
['2010-08-04', '25.3', '6.6', '4.2']
...
```

Pandas 주요 메서드 구조

1) Series

- 1D 데이터 (리스트)
[1, 2, 3]

2) DataFrame

- 2D 데이터 (二维, 행렬)
- 다양한 데이터 연산 도출 가능

3 CSV에서 원하는 데이터를 뽑아 보자

- 기상자료개방 포털 사이트에서 다운 받은 데이터 weather.csv를 계속해서 사용해 보자.
- 이제 이 데이터에서 평균 풍속 데이터만 추출하여 사용하고 싶다. CSV 파일에서 평균 풍속 데이터는 4번째 열에 저장되어 있다. 인덱스로는 3이 된다. 따라서 리스트에서 row[3]을 찾으면 된다

	A	B	C	D
1	일시	평균기온	최대풍속	평균풍속
2	2010-08-01	28.7	8.	3.4
3	2010-08-02	25.2	8.	3.8
4	2010-08-03	22.1	6.	2.9
5	2010-08-04	25.3	6.	4.2
6	2010-08-05	27.2	9.	5.6
7	2010-08-06	26.8	9.	8

```
col[0] col[1] col[2] col[3]
```

```
import csv

f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data)
for row in data:
    print(row[3], end=',')
f.close()

3.4,3.8,2.9,4.2,5.6,8,5,4,3.1,5.5,4.8,2.6,4.6,4.4,10.3,3.2,1.6,2.1,1.9,3.2,4.2,2.5,6.2,3,1.9,2.5,1,6,2.3,4.9,6.2,4.2,2.6,5.3,1.7,3.2,3.3,4.3,7.6,6.6,2.5,7.2,3.8,1.8,3.9,1.6,2.2,1.2,2.2,3,3.5,2.8,3.6,7.9,5.8,4.1,6.1,1.8,2.8,5.6,2.1,2.2,3.3,3.2,5.9,5,5.1,3.1,3.4,3.7,2.7,2.6,3.1,2.5,5,3.2,2.9,4.5,2.9,2.9,2.1,3.9,6.3,3.9,2,3,6.1,7.1,4,3.5,5.8,6.6,7.2,5.6,3.5,3.2,2.9,3.2,3.3,2.5,7.5 ...
```

3 CSV에서 원하는 데이터를 뽑아 보자

- 반복문을 사용하여 import한 데이터의 네번째 열의 원소값 중에서 최대 값을 구하자.

```
# 위의 코드 import .. 부터 header =.. 까지가 생략되었음
max_wind = 0.0

for row in data:                      # 반복 루프를 사용하여 데이터를 읽는다.
    if row[2] == '' : {설명이 있어}   # 최대 풍속 데이터가 없는 경우 0을 처리
        wind = 0
    else :
        wind = float(row[2])         # 최대 풍속 데이터를 실수로 변환해 저장
    if max_wind < wind :            # 최대 풍속을 갱신하는지 검사
        max_wind = wind             # 현재까지의 최대 풍속보다 크면 새로 기록

print('지난 10년간 울릉도의 최대 풍속은 ', max_wind, 'm/s')
```

지난 10년간 울릉도의 최대 풍속은 26.0 m/s

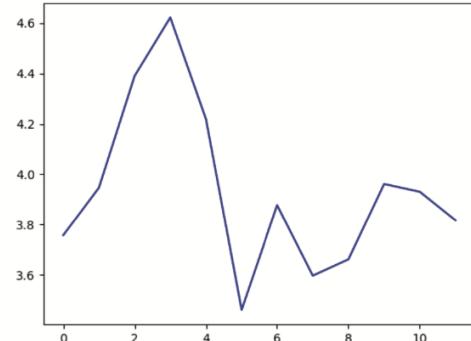
LAB¹ 울릉도는 몇 월에 바람이 가장 강할까?

실습 시간



앞서 사용했던 울릉도의 기상 데이터에는 일일 평균 풍속 데이터가 있다. 이 데이터를 바탕으로 몇 월의 울릉도가 가장 바람이 강한지 알아보고 싶다. 결과는 아래와 같다. 달이 0부터 시작하여 11(실제는 1월부터 12월)까지 있고, 3일 때에 가장 큰 풍속을 보이므로, 울릉도는 4월에 가장 강한 바람이 분다는 것을 알 수 있다.

원하는 결과



힌트



각 달의 평균 풍속을 구하기 위한 12개 항목을 가진 리스트 `monthly_wind`를 만든다. 각 달마다 측정 데이터가 존재하는 일수를 담을 12개 항목의 리스트 `days_counted`도 함께 만든다.

각 행 데이터를 읽어 해당 데이터가 몇 월의 데이터인지 확인하고, 풍속 정보가 있는지 확인한다. 여기서 첫 열에 있는 '2012-01-24'와 같은 문자열의 [5:7]을 읽으면 달의 정보가 된다. 풍속이 존재하면 해당 월의 풍속 정보에 이 데이터를 더하고, 고려된 일수 `days_counted`도 해당 월 정보를 증가시킨다. 마지막으로 누적된 풍속 데이터를 계산된 일수로 나누어 월 평균 풍속을 구한다.

LAB¹ 울릉도는 몇 월에 바람이 가장 강할까?

해답 코드



```
import csv
import matplotlib.pyplot as plt

f = open('d:/data/weather.csv')
data = csv.reader(f)
header = next(data)

monthly_wind = [ 0 for x in range(12) ]          # CSV 파일 열어 f에 저장
days_counted = [ 0 for x in range(12) ]           # reader() 함수로 읽기
                                                    # 헤더를 제거

                                                    # 매달 풍속을 담을 리스트, 초기화 0
                                                    # 각 달마다 측정된 일수, 초기화 0

for row in data:
    month = int(row[0][5:7])
    if row[3] != '' :
        wind = float(row[3])
        monthly_wind[month-1] += wind
        days_counted[month-1] += 1

for i in range(12) :
    monthly_wind[i] /= days_counted[i]               # 0번 열에서 달 정보 추출
                                                    # 풍속 데이터 존재하는지 확인
                                                    # 풍속을 얻어 온다.
                                                    # 해당 달에 풍속 데이터 추가
                                                    # 해당 달의 일수를 증가

plt.plot(monthly_wind, 'blue')                      # 일수로 나누어 월평균 구하기

plt.show()

f.close()                                            # 파일을 닫는다.
```

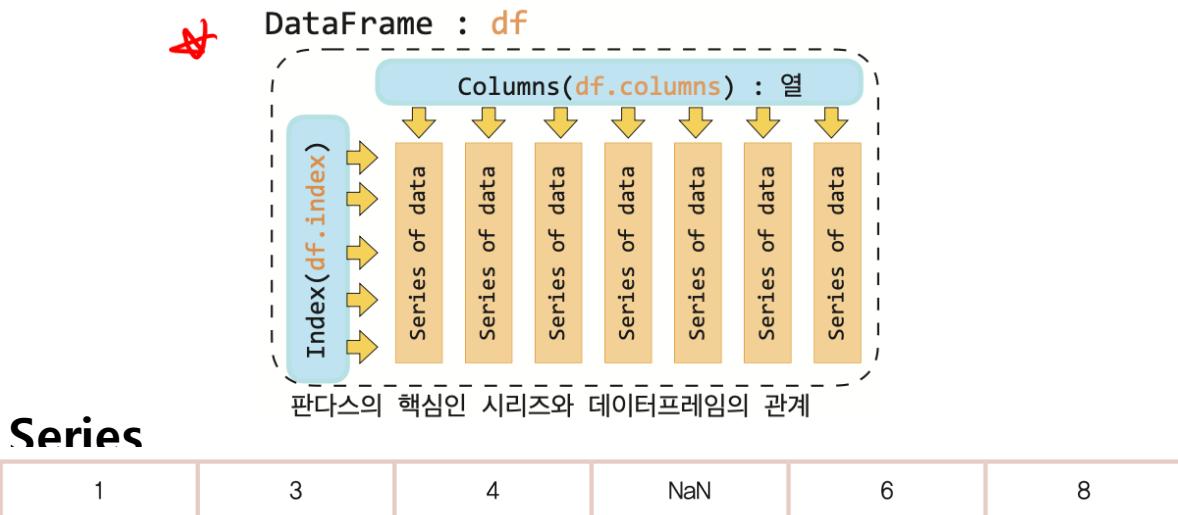
4 판다스의 데이터 구조 : 시리즈와 데이터프레임

- 앞서 다루어본 csv 모듈 이외에도 CSV 데이터를 처리할 수 있는 모듈이 있다. 이들 중 가장 강력한 외부 라이브러리인 판다스를 알아보자.
- 판다스는 데이터 저장을 위하여 다음과 같은 2가지의 기본 데이터 구조를 제공하고 있다.
+ 인덱스 / 레이블
- 이들 데이터 구조는 모두 넘파이 배열을 이용하여 구현된다. 따라서 속도가 빠르다. 모든 데이터 구조는 값을 변경할 수 있으며, 시리즈를 제외하고는 크기도 변경할 수 있다. 각 행과 열은 이름이 부여되며, 행의 이름을 **인덱스index**, 열의 이름을 **컬럼스columns**라 부른다.

데이터 구조	차원	설명
시리즈	1	레이블이 붙어있는 1차원 벡터 <i>인덱스</i> 레이터의 값과 그 값의 위치(index)
데이터프레임	2	행과 열로 되어있는 2차원 테이블, 각 열은 시리즈로 되어 있다.



4 판다스의 데이터 구조 : 시리즈와 데이터프레임



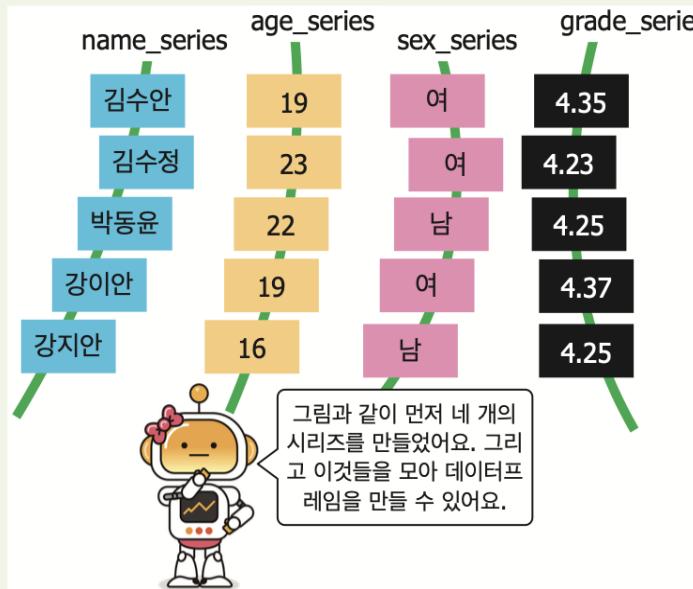
```
>>> import numpy as np
>>> import pandas as pd
>>> series = pd.Series([1, 3, 4, np.nan, 6, 8])
>>> series
0    1.0
1    3.0
2    4.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

print(series[2])
=>

np.nan은 값 없음(Not a Number)을 의미하는 것으로
수치 데이터가 없을 경우 이를 표기하는 방법이다.
NaN과 동일한 표기임.
nan이 있을 경우 실수형

4 판다스의 데이터 구조 : 시리즈와 데이터프레임

```
>>> name_series = pd.Series(['김수안', '김수정', '박동윤', '강이안', '강지안'])
>>> age_series = pd.Series([19, 23, 22, 19, 16])
>>> sex_series = pd.Series(['여', '여', '남', '여', '남'])
>>> grade_series = pd.Series([4.35, 4.23, 4.25, 4.37, 4.25])
>>> print(name_series, age_series, sex_series, grade_series)
0    김수안
1    김수정
2    박동윤
3    강이안
4    강지안
dtype: object
0    19
1    23
2    22
3    19
4    16
dtype: int64
0    여
1    여
2    남
3    여
4    남
dtype: object
0    4.35
1    4.23
2    4.25
3    4.37
4    4.25
dtype: float64
```



이름	나이	성별	평점
김수안	19	여	4.35
김수정	23	여	4.23
박동윤	22	남	4.45
강이안	19	여	4.37
강지안	16	남	4.25

4 판다스의 데이터 구조 : 시리즈와 데이터프레임

```
>>> df = pd.DataFrame({'이름': name_series, '나이': age_series,  
                      '성별': sex_series, '평점': grade_series})  
>>> print(df)  
    이름 나이 성별 평점  
0 김수안 19 여 4.35  
1 김수정 23 여 4.23  
2 박동윤 22 남 4.25  
3 강이안 19 여 4.37  
4 강지안 16 남 4.25
```

딕셔너리 형식의 데이터로
데이터 프레임을 생성함

Keys	이름	나이	성별	columns
Values	김수안	19	여	평점
김수정	23	여	여	4.23
박동윤	22	남	남	4.25
강이안	19	여	여	4.37
강지안	16	남	남	4.25

판다스의 DataFrame 클래스를
사용해서 하나의 데이터 프레임을
만들 수 있다

5 판다스로 데이터 파일을 읽기

- 판다스 모듈을 이러한 csv 파일을 읽어들여서 데이터프레임으로 바꾸는 작업을 간단히 할 수 있게 한다. 다음과 같이 `read_csv` 함수를 이용하면 된다. `countries.csv` 파일의 제 1행 제 1열은 비어 있음을 확인할 수 있다. 이것은 첫 열은 데이터가 아니라 각 행의 인덱스로 사용되도록 하기 위해서이다.
- 이때 CSV 파일이 데이터프레임이 될 수 있도록 각 행이 같은 구조로 되어 있고, 각 열은 동일한 자료형을 가진 시리즈로 되어 있어야 한다. 예러가 없이 csv 파일을 읽어왔다면 `df`를 출력해보자.

countries.csv				
	,country,area,capital,population			
KR	Korea,98480,Seoul,51780579			
US	USA,9629091,Washington,331002825			
JP	Japan,377835,Tokyo,125960000			
CN	China,9596960,Beijing,1439323688			
RU	Russia,17100000,Moscow,146748600			

첫 번째 열을 인덱스로 사용하기 위해
첫 번째 열의 이름을 생략하고 있다.

```
>>> import pandas as pd  
>>> df = pd.read_csv('d:/data/countries.csv')
```

5 판다스로 데이터 파일을 읽기

```
>>> df
```

	Unnamed: 0	country	area	capital	population
0	KR	Korea	98480	Seoul	51780579
1	US	USA	9629091	Washington	331002825
2	JP	Japan	377835	Tokyo	125960000
3	CN	China	9596960	Beijing	1439323688
4	RU	Russia	17100000	Moscow	146748600

각 열은 서로 다른 속성 레이블을
나타낸다.

인덱스 번호는 판다스가
추가한 열이다

5 판다스로 데이터 파일을 읽기

데이터를 설명하는 인덱스와 컬럼스 객체

비워 두었던 열이름

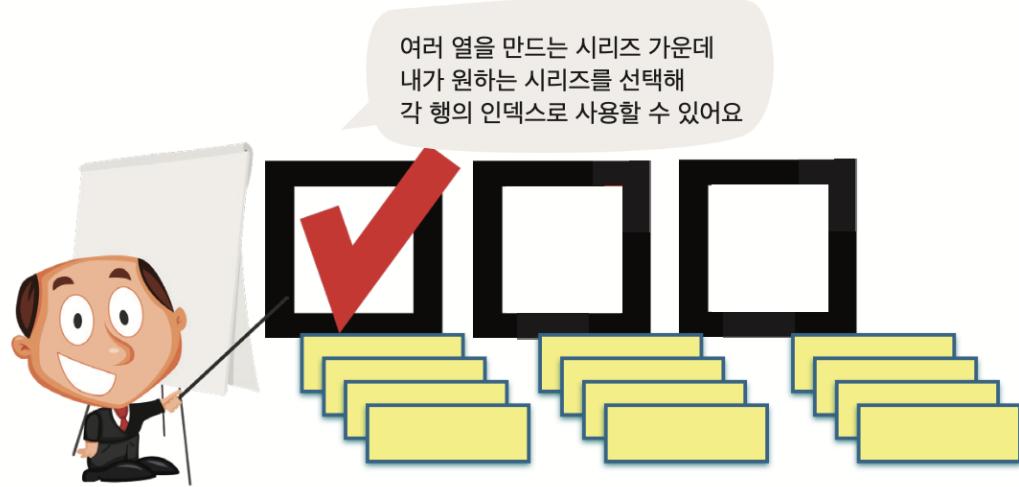
CSV 파일의 첫 행으로 만들어진 **columns**

	Unnamed: 0	country	area	capital	population
0	KR	Korea	98480	Seoul	51780579
1	US	USA	9629091	Washington	331002825
2	JP	Japan	377835	Tokyo	125960000
3	CN	China	9596960	Beijing	1439323688
4	RU	Russia	17100000	Moscow	146748600

자동으로 생성된 **index**

5 판다스로 데이터 파일을 읽기

데이터를 설명하는 인덱스와 컬럼스 객체 (계속)



여러 열을 만드는 시리즈 가운데
내가 원하는 시리즈를 선택해
각 행의 인덱스로 사용할 수 있어요

```
import pandas as pd

df = pd.read_csv('d:/data/countries.csv', index_col = 0) # 첫 열을 인덱스 컬럼으로 사용
print(df)
```

	country	area	capital	population
KR	Korea	98480	Seoul	51780579
US	USA	9629091	Washington	331002825
JP	Japan	377835	Tokyo	125960000
CN	China	9596960	Beijing	1439323688
RU	Russia	17100000	Moscow	146748600

6 열을 기준으로 데이터 선택하기

- 특정한 열만 선택하려면 아래와 같이 대괄호 안에 열의 이름을 넣으면 된다.
- 다음 코드는 `countries.csv`를 다시 읽고 있다. 그리고 처음에는 인덱스를 첫 열로 지정해서 `df_my_index`로 할당했고, 인덱스 지정 없이 만든 데이터프레임은 `df_no_index`로 할당했다. 두 데이터프레임에서 `population` 레이블을 가진 열을 추출하기 위해서는 `df['population']`이라고 하면 된다.

```
import pandas as pd

df_my_index = pd.read_csv('d:/data/countries.csv', index_col = 0)
df_no_index = pd.read_csv('d:/data/countries.csv')
print(df_my_index['population'])
print(df_no_index['population'])

KR      51780579
US      331002825
JP      125960000
CN      1439323688
RU      146748600
Name: population, dtype: int64
0      51780579
1      331002825
2      125960000
3      1439323688
4      146748600
Name: population, dtype: int64
```

인덱스 컬럼이 0이므로 KR,
US, JP..가 인덱스가 된다

인덱스 컬럼이 없을 경우 0, 1,
2, ..가 인덱스가 됨

6 열을 기준으로 데이터 선택하기

```
import pandas as pd  
  
df_my_index = pd.read_csv('d:/data/countries.csv', index_col = 0)  
print(df_my_index[ ['area', 'population'] ])
```

	area	population
KR	98480	51780579
US	9629091	331002825
JP	377835	125960000
CN	9596960	1439323688
RU	17100000	146748600

전체 데이터 중에서 두 개의 열을 선택
하는 경우 : 선택을 원하는 열의 레이블
을 리스트에 넣어서 전달



6 열을 기준으로 데이터 선택하기

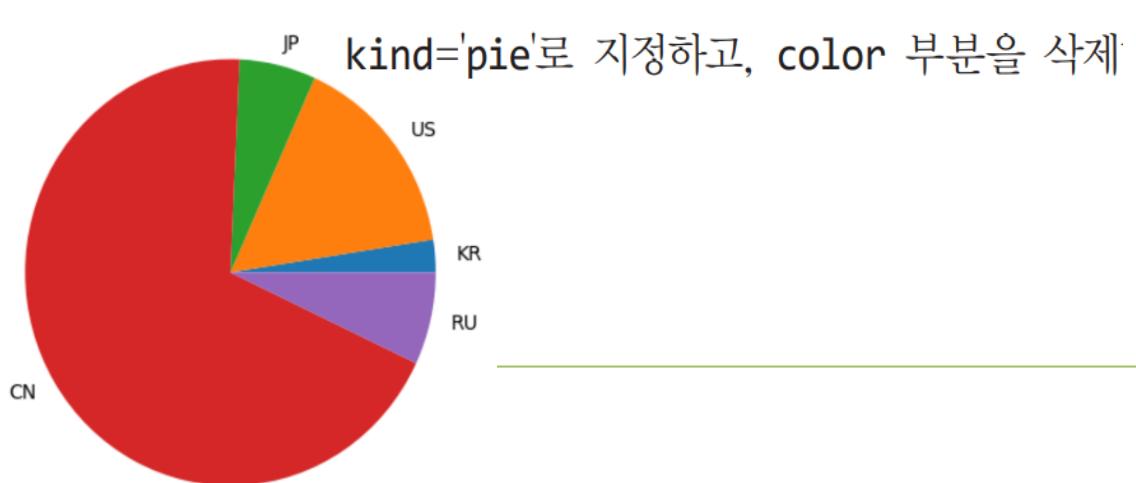
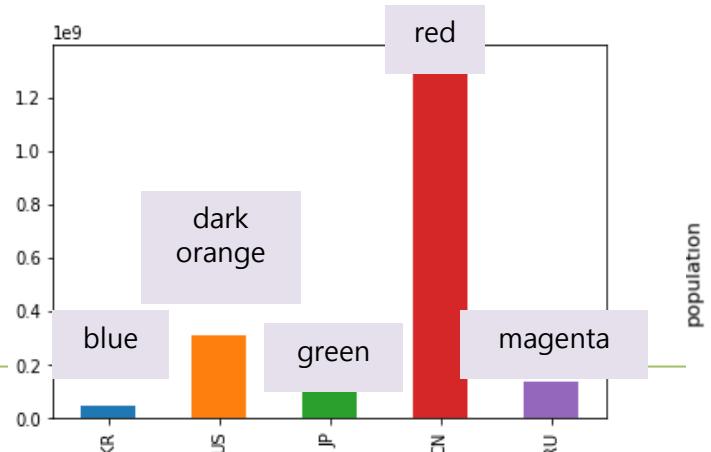
데이터 시각화하기

- 우리는 선택된 열을 그래프로 그릴 수 있다. 이를 위하여 데이터 프레임의 이름 다음에 `plot()` 메소드만 추가하면 된다. 각 국가의 인구만을 추출하여서 막대 그래프로 그려보면 다음과 같다.

```
import pandas as pd
import matplotlib.pyplot as plt

countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)

countries_df['population'].plot(kind='bar', color=('b', 'darkorange', 'g', 'r', 'm') )
plt.show()
```

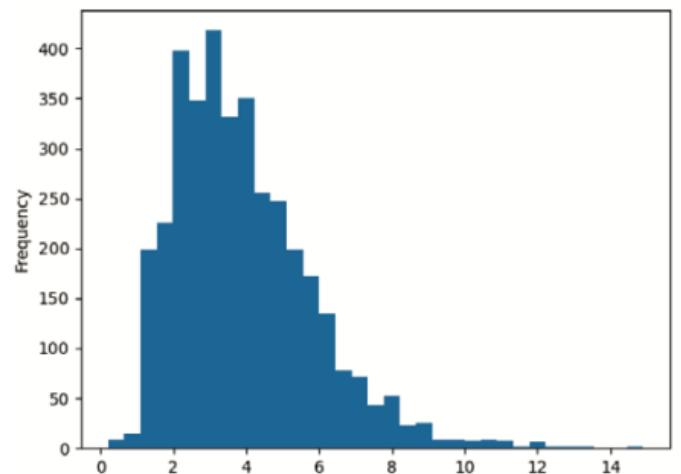


6 열을 기준으로 데이터 선택하기

데이터 시각화하기 (계속)

```
import pandas as pd  
import matplotlib.pyplot as plt  
  
weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')  
weather['평균풍속'].plot(kind='hist', bins=33)  
plt.show()
```

한글 인코딩 문자를 읽어
오기 위해서 사용함



판다스를 이용하면 데이터
를 차트로 표시하는 일을
아주 간편하게 할 수 있답
니다.



6 열을 기준으로 데이터 선택하기

데이터 시각화하기 (계속)



잠깐 – 판다스를 이용하여 csv를 여는 코드에서 눈여겨 볼 것

マイ크로소프트의 윈도 시스템에서 생성되고 한글을 포함하고 있는 weather.csv는 읽을 때 한글을 어떤 **인코딩 encoding** 방식으로 처리할지 지정해야 한다. 위의 코드에 encoding='CP949'라고 표시된 것을 확인할 수 있을 것이다. 또 주의해서 볼 것은 csv 모듈로 읽을 때와 달리 판다스로 읽으면 풍속이 문자열이 아니라 실수 데이터로 바로 읽힌다는 것을 알 수 있다. float()를 이용하여 값을 실수로 바꿀 필요가 없는 것이다.

7 슬라이싱으로 행 선택하기

- 데이터 프레임 중에서 몇 개의 행만을 가져오고자 할 때는 몇 가지의 방법이 있다. 우선 처음 5행만 얻으려면 head()를 사용할 수 있다. 마지막 5행만을 얻으려면 tail()을 사용한다.

```
>>> countries_df.head() # countries_df[0:5]와 같다
```

	country	area	capital	population
KR	Korea	98480	Seoul	48422644
US	USA	9629091	Washington	310232863
JP	Japan	377835	Tokyo	127288000
CN	China	9596960	Beijing	1330044000
RU	Russia	17100000	Moscow	140702000

```
>>> countries_df[:3]
```

	country	area	capital	population
KR	Korea	98480	Seoul	48422644
US	USA	9629091	Washington	310232863
JP	Japan	377835	Tokyo	127288000

7 슬라이싱으로 행 선택하기

```
>>> countries_df.loc['KR']
   country      Korea
   area        98480
   capital     Seoul
   population  48422644
```

행의 레이블이 'KR'인 행만을 선택하기

데이터 프레임

location

loc

- 레이블 기반 데이터 선택
~~~

```
>>> countries_df['population'][::3]
   KR      48422644
   US      310232863
   JP      127288000
```

```
>>> countries_df.loc['US', 'capital']
   'Washington'
```

데이터프레임에서 특정한 요소 하나만을  
선택하려면 loc 속성에 행과 열의 레이블을  
써주면 된다

```
>>> countries_df['capital'].loc['US']
   'Washington'
```

# 7 슬라이싱으로 행 선택하기

## 새로운 열을 생성해 보자

- 판다스를 이용하면 다른 열의 정보를 토대로 새로운 열을 생성할 수도 있다.
- 우리의 데이터프레임에 인구 밀도를 나타내는 열을 생성해보자. 앞장에서 넘파이 배열에는 어떤 수를 곱하고 더하는 것이 가능하다고 하였다. 판다스는 넘파이를 기반으로 하기 때문에 판다스 데이터 프레임에도 동일하게 적용할 수 있다. 인구를 면적으로 나눠주면 된다.

```
import pandas as pd
import matplotlib.pyplot as plt

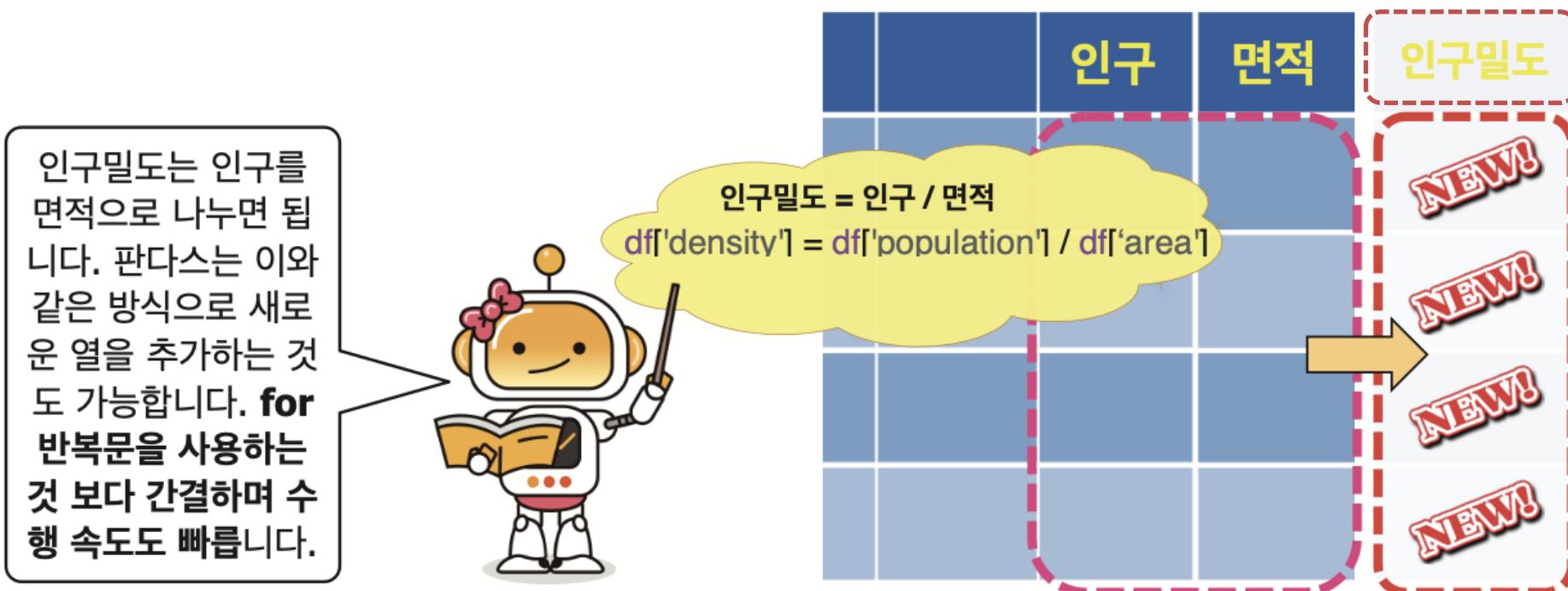
countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
countries_df['density'] = countries_df['population'] / countries_df['area']
print(countries_df)
```

|    | country | area     | capital    | population | density    |
|----|---------|----------|------------|------------|------------|
| KR | Korea   | 98480    | Seoul      | 51780579   | 525.797918 |
| US | USA     | 9629091  | Washington | 331002825  | 34.375293  |
| JP | Japan   | 377835   | Tokyo      | 125960000  | 333.373033 |
| CN | China   | 9596960  | Beijing    | 1439323688 | 149.977044 |
| RU | Russia  | 17100000 | Moscow     | 146748600  | 8.581789   |

countries\_df['density'] 열이 새롭게 추가되었음

# 7 슬라이싱으로 행 선택하기

## 새로운 열을 생성해 보자 (계속)



# 7 슬라이싱으로 행 선택하기

## 새로운 열을 생성해 보자 (계속)



### 잠깐 – 데이터프레임의 열을 이용한 연산

인구밀도를 구하기 위해 새로운 열을 'density'라는 레이블로 생성해 보았다. 그리고 이 열의 데이터는 기존에 존재하던 데이터 중에서 인구수를 면적으로 나누어 얻을 수 있다. 그런데, 이를 위해서 각 행을 차례로 접근하여 해당 데이터 항목마다 연산을 수행하지 않는다. 데이터프레임의 어떤 열이 다른 열들의 값에 의해 결정될 때는 이 계산을 행별로 반복하여 일을 하는 것이 아니라 필요한 열을 통째로 접근하여 한번에 계산이 이루어지게 한다. 이것은 코드가 간결할 뿐만 아니라, 계산도 훨씬 빠르다. 데이터프레임이나 행렬 데이터를 다루면서 `for` 문을 사용한다면 '내가 잘못하고 있지 않는가? 이 `for` 문을 꼭 써야 하는가?'라는 생각을 항상 해야 한다.

# 8 데이터를 간편하게 분석할 수 있는 기능이 있다.

- 이제 우리는 외부 파일을 읽어서 데이터 프레임을 생성해서 필요한 행과 열을 선택할 수 있다. 데이터 프레임이 저장한 데이터를 간단히 분석하려면 `describe()` 함수를 호출해주면 된다.

```
import pandas as pd  
weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
```

```
print(weather.describe())
```

|         | 평균기온        | 최대풍속        | 평균풍속        |
|---------|-------------|-------------|-------------|
| ✓ count | 3653.000000 | 3649.000000 | 3647.000000 |
| ✓ mean  | 12.942102   | 7.911099    | 3.936441    |
| ✓ std   | 8.538507    | 3.029862    | 1.888473    |
| ✓ min   | -9.000000   | 2.000000    | 0.200000    |
| ✓ 25%   | 5.400000    | 5.700000    | 2.500000    |
| ✓ 50%   | 13.800000   | 7.600000    | 3.600000    |
| ✓ 75%   | 20.100000   | 9.700000    | 5.000000    |
| ✓ max   | 31.300000   | 26.000000   | 14.900000   |

# 8 데이터를 간편하게 분석할 수 있는 기능이 있다.

```
...
print('평균 분석 -----')
print(weather.mean())
print('표준편차 분석 -----')
print(weather.std())
```

```
평균 분석 -----
평균기온      12.942102
최대풍속      7.911099
평균풍속      3.936441
dtype: float64
표준편차 분석 -----
평균기온      8.538507
최대풍속      3.029862
평균풍속      1.888473
dtype: float64
```

# 8 데이터를 간편하게 분석할 수 있는 기능이 있다.



## 잠깐 - 판다스의 표준편차와 넘파이의 표준편차의 차이

평균기온 표준편차를 넘파이로 계산하면 다른 값이 나온다. 판다스 표준편차와 넘파이 표준편차를 각각 구하는 방법은 다음과 같다.

```
pandas_std = weather['평균기온'].std()  
numpy_std = np.std(weather['평균기온'])  
print(pandas_std, numpy_std)
```

'ddof=1'

8.538507014753446 8.537338236838895

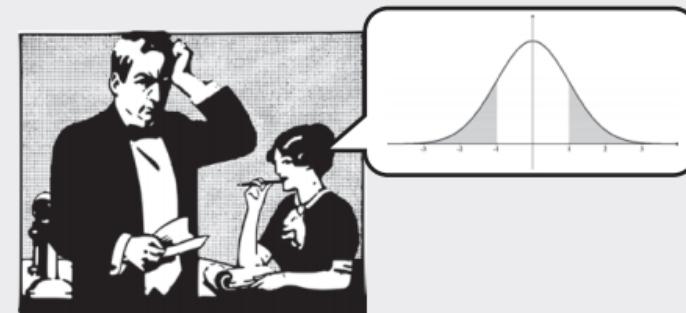
$$\sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n}}$$

↓ numpy

→ n-1

비교

pandas



판다스의 표준편차는 디폴트로 베셀 보정 Bessel's correction을 적용하는데, 이것은 표준편차를 구할 때, 표본 크기 n 대신에 n-1을 적용하는 것으로 모분산 추정에서 편향을 보정하는 역할을 한다.

# 8 데이터를 간편하게 분석할 수 있는 기능이 있다.

## 데이터 집계 분석도 쉽게 해 보자

- 데이터의 전체적인 특징이 어떠한지를 분석하는 것은 매우 중요한 일이다. 앞서 살펴본 `describe()` 함수는 데이터프레임의 데이터 특성을 전체적으로 요약해 주는데, 이 분석 내용 각각은 하나씩 떼어서 적용할 수도 있다.
  - 다음 코드는 데이터의 수를 알려준다

```
>>> weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
>>> weather.count() # 각 열의 데이터 수를 알려준다
평균기온    3653
최대풍속    3649
평균풍속    3647
dtype: int64
```

weather.csv 파일이 담고 있는 데이터가  
3 개의 열을 가지고 있고, 각각의 열에 담긴  
데이터가 3653, 3649, 3647개라는 것을 알 수 있다

```
>>> weather['최대풍속'].count() # '최대풍속' 열의 데이터 수를 알려준다
3649
```

# 8 데이터를 간편하게 분석할 수 있는 기능이 있다.

## 데이터 집계 분석도 쉽게 해 보자 (계속)

```
>>> weather[['최대풍속','평균풍속']].count() # '최대풍속', '평균풍속' 열의 데이터 수  
최대풍속      3649  
평균풍속      3647  
dtype: int64
```

여러 개의 열을 분석하고 싶을 때  
는 원하는 열의 레이블들을 리스트  
로 제공

```
>>> weather[['최대풍속','평균풍속']].mean() # '최대풍속', '평균풍속' 열의 평균값  
최대풍속      7.911099  
평균풍속      3.936441  
dtype: float64
```

min(), max(), mean(), sum() 등  
도 적용 가능

```
>>> weather.mean()[['최대풍속', '평균풍속']]  
최대풍속      7.911099  
평균풍속      3.936441  
dtype: float64
```



# 9 연도와 월, 일을 다루는 DatetimeIndex와 그룹핑

- 조금 더 효율적인 방법이 있는데 그것은 groupby()라는 함수이다. groupby() 함수에 넘길 인자로는 우리가 그룹을 묶을 때에 사용될 열의 레이블이다. 해당 열에 있는 데 이터가 동일하면 하나의 그룹으로 묶이는 것이다. 그리고 여기에 mean()을 적용하면 해당 그룹의 데이터들이 가진 값의 평균을 구할 수 있다.

```
import pandas as pd

# DatetimeIndex를 이용하여 연도를 추출
print(pd.DatetimeIndex(['2010-08-01', '2011-09-21']).year)
Int64Index([2010, 2011], dtype='int64')

# DatetimeIndex를 이용하여 월을 추출
print(pd.DatetimeIndex(['2010-08-01', '2011-09-21']).month)
Int64Index([8, 9], dtype='int64')

# DatetimeIndex를 이용하여 일을 추출
print(pd.DatetimeIndex(['2010-08-01', '2011-09-21']).day)
Int64Index([1, 21], dtype='int64')
```

[ '2010-08-01', '2011-09-21' ]



pd.DatetimeIndex(['2010-08-01', '2011-09-21']).year



[2010, 2011]



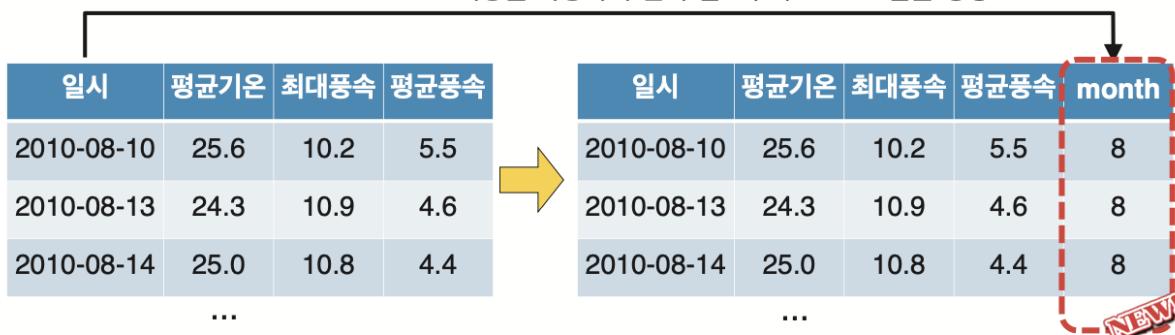
DatetimeIndex  
는 다양한 형식의  
연-월-일 데이터에  
서 특정한 연도,  
월, 또는 일을 추출  
할 수 있습니다.

# 9 연도와 월, 일을 다루는 DatetimeIndex와 그룹핑

```
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
weather['month'] = pd.DatetimeIndex(weather['일시']).month # 새로운 'month' 열을 만든다
print(weather)
```

|      | 일시         | 평균기온 | 최대풍속 | 평균풍속 | month |
|------|------------|------|------|------|-------|
| 0    | 2010-08-01 | 28.7 | 8.3  | 3.4  | 8     |
| 1    | 2010-08-02 | 25.2 | 8.7  | 3.8  | 8     |
| 2    | 2010-08-03 | 22.1 | 6.3  | 2.9  | 8     |
| 3    | 2010-08-04 | 25.3 | 6.6  | 4.2  | 8     |
| 4    | 2010-08-05 | 27.2 | 9.1  | 5.6  | 8     |
| ...  | ...        | ...  | ...  | ...  | ...   |
| 3648 | 2020-07-27 | 22.1 | 4.2  | 1.7  | 7     |
| 3649 | 2020-07-28 | 21.9 | 4.5  | 1.6  | 7     |
| 3650 | 2020-07-29 | 21.6 | 3.2  | 1.0  | 7     |
| 3651 | 2020-07-30 | 22.9 | 9.7  | 2.4  | 7     |
| 3652 | 2020-07-31 | 25.7 | 4.8  | 2.5  | 7     |

DatetimeIndex 기능을 사용하여 일시 열로부터 month 열을 생성



# 9 연도와 월, 일을 다루는 DatetimeIndex와 그룹핑

```
...  
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')  
weather['month'] = pd.DatetimeIndex(weather['일시']).month  
means = weather.groupby('month').mean()  
  
print(means)
```

| month | 평균기온      | 최대풍속     | 평균풍속     |
|-------|-----------|----------|----------|
| 1     | 1.598387  | 8.158065 | 3.757419 |
| 2     | 2.136396  | 8.225357 | 3.946786 |
| 3     | 6.250323  | 8.871935 | 4.390291 |
| 4     | 11.064667 | 9.305017 | 4.622483 |
| 5     | 16.564194 | 8.548710 | 4.219355 |
| 6     | 19.616667 | 6.945667 | 3.461000 |
| 7     | 23.328387 | 7.322581 | 3.877419 |
| 8     | 24.748710 | 6.853226 | 3.596129 |
| 9     | 20.323667 | 6.896333 | 3.661667 |
| 10    | 15.383871 | 7.766774 | 3.961613 |
| 11    | 9.889667  | 8.013333 | 3.930667 |
| 12    | 3.753548  | 8.045484 | 3.817097 |

# 10 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑

```
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
weather['year'] = pd.DatetimeIndex(weather['일시']).year # 새로운 'year' 열을 만든다
print(weather)
```

|     | 일시         | 평균기온 | 최대풍속 | 평균풍속 | year |
|-----|------------|------|------|------|------|
| 0   | 2010-08-01 | 28.7 | 8.3  | 3.4  | 2010 |
| 1   | 2010-08-02 | 25.2 | 8.7  | 3.8  | 2010 |
| 2   | 2010-08-03 | 22.1 | 6.3  | 2.9  | 2010 |
| 3   | 2010-08-04 | 25.3 | 6.6  | 4.2  | 2010 |
| 4   | 2010-08-05 | 27.2 | 9.1  | 5.6  | 2010 |
| ... | ...        | ...  | ...  | ...  | ...  |

```
y_means = weather.groupby('year').mean(numeric_only=True)
print(y_means)
```

종인 연도로 grouping

| year | 평균기온      | 최대풍속     | 평균풍속     |
|------|-----------|----------|----------|
| 2010 | 15.238562 | 8.205229 | 4.069281 |
| 2011 | 12.073425 | 8.355616 | 4.251233 |
| 2012 | 11.892896 | 7.794490 | 3.863912 |
| 2013 | 12.795068 | 7.859726 | 3.874795 |
| 2014 | 12.844110 | 7.458904 | 3.816438 |
| 2015 | 13.162466 | 7.694247 | 3.799449 |
| 2016 | 13.243443 | 7.963934 | 3.977869 |
| 2017 | 13.111233 | 8.001370 | 3.934795 |
| 2018 | 13.041644 | 8.158630 | 4.085479 |
| 2019 | 13.767671 | 7.796703 | 3.854396 |
| 2020 | 12.233333 | 7.897183 | 3.786385 |

# 10 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑

```
y_max = weather.groupby('year').max(numeric_only=True)  
print(y_max)
```

숫자형 데이터만 대상  
~~문자열 제거~~

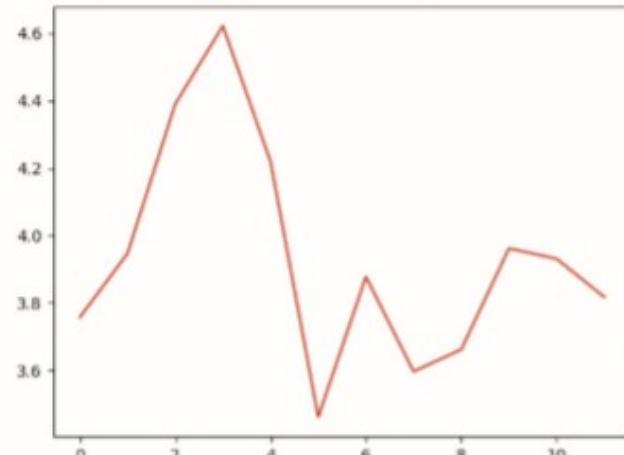
평균기온 최대풍속 평균풍속

| year | 평균기온 | 최대풍속 | 평균풍속 |
|------|------|------|------|
| 2010 | 28.9 | 16.9 | 10.3 |
| 2011 | 27.6 | 21.6 | 12.5 |
| 2012 | 28.3 | 17.5 | 9.2  |
| 2013 | 31.3 | 17.8 | 10.1 |
| 2014 | 27.5 | 16.1 | 11.2 |
| 2015 | 29.6 | 25.3 | 14.9 |
| ...  |      |      |      |

# LAB<sup>1</sup> 울릉도는 몇 월에 바람이 가장 강할까?

앞서 사용했던 울릉도의 기상 데이터에 기록된 매일의 평균 풍속 데이터를 바탕으로 몇 월의 바람이 가장 강한지 분석해 보았다. 이번에는 이 작업을 판다스를 이용하여 해 보려고 한다. 결과는 앞의 것도 동일하게 나올 것이다.

## 원하는 결과



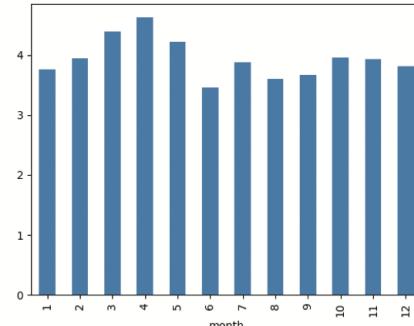
# LAB<sup>2</sup> 울릉도는 몇 월에 바람이 가장 강할까? groupby( ) 활용

## 실습 시간



울릉도의 기상 데이터에 기록된 매일의 평균 풍속 데이터를 바탕으로 몇 월의 바람이 가장 강한지 분석해 보았다. 이번에는 판다스가 제공하는 `groupby()` 함수로 더욱 효율적으로 만들어 보라.

## 원하는 결과



## 힌트



`groupby()`를 이용하면 각 달별로 별도의 데이터를 만들 필요가 없다. 원래의 데이터에 'month'라는 열을 추가하는 것은 이미 다루어 본 것과 동일한 방식으로 할 수 있다.

```
weather['month'] = pd.DatetimeIndex(weather['일시']).month
```

그리고 그 다음은 `month` 열의 데이터를 기준으로 그룹을 만들면 1월에서 12월까지 달별로 데이터가 묶이게 될 것이고 여기에 `mean()` 함수를 적용하면 12개월 각각의 데이터 평균을 구할 수 있다.

여기서 관심이 있는 '평균풍속'만을 추출하여 판다스의 차트 기능으로 그려보았다.

```
means['평균풍속'].plot(kind = 'bar')
```

# LAB<sup>2</sup> 울릉도는 몇 월에 바람이 가장 강할까? groupby( ) 활용

해답 코드



```
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt

weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')

weather['month'] = pd.DatetimeIndex(weather['일시']).month
means = weather.groupby('month').mean(numeric_only = True)
means['평균풍속'].plot(kind = 'bar')

plt.show()
```

# 11 조건에 맞게 골라내자 : 필터링

- weather 데이터프레임에서 '최대풍속' 레이블로 되어 있는 열의 값이 10.0을 넘는지 확인하여 참과 거짓을 얻는 방법은 다음과 같다.

```
>>> weather['최대풍속'] >= 10.0
```

일시

2010-08-01 False

2010-08-02 False

2010-08-03 False

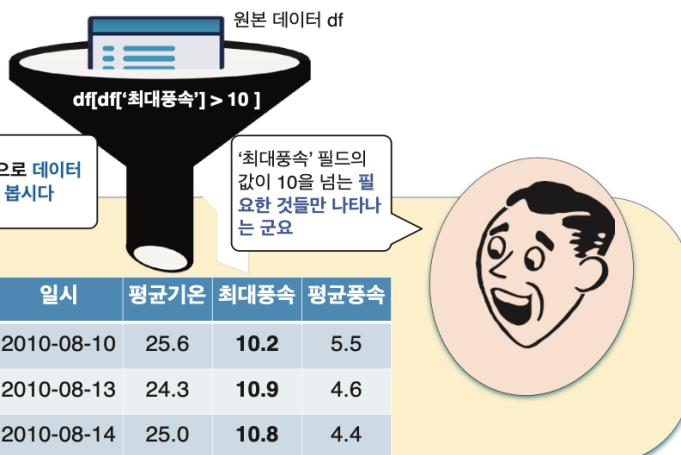
...

2020-07-29 False

2020-07-30 False

2020-07-31 False

넘파이의 논리 인덱싱과 동  
일한 문법



# 11 조건에 맞게 골라내자 : 필터링

☞

```
>>> weather[ weather['최대풍속'] >= 10.0 ]  
          평균기온  최대풍속  평균풍속  
일시  
2010-08-10    25.6      10.2      5.5  
2010-08-13    24.3      10.9      4.6  
2010-08-14    25.0      10.8      4.4  
...           ...       ...       ...  
2020-07-13    17.8      10.3      4.6  
2020-07-14    17.8      12.7      9.4  
2020-07-20    23.0      11.2      7.3
```

'최대풍속' 레이블로 되어 있는 열의 값이  
10.0을 넘는지 확인하여 참값을 얻는 방법

# 11 조건에 맞게 골라내자 : 필터링

## 빠진 값을 찾고 삭제하기

- 데이터 과학자가 사용하는 실제 데이터는 완벽하지 않고 상당한 수의 결손값을 가지고 있거나 의심스러운 값을 가지고 있다. 결손값은 왜 생길까? 데이터가 아예 수집되지 않았거나, 측정 장치의 고장, 사건 사고 등으로 데이터를 확보할 수 없을 수도 있다. 따라서 데이터를 처리하기 전에 반드시 거쳐야 하는 절차가 데이터 정제이다. 판다스에서는 결손값을 NaN으로 나타낸다(혹은 NA로 표기함). 판다스는 결손값<sup>missing data</sup>을 탐지하고 수정하는 함수를 제공한다.

Data Cleansing

# 11 조건에 맞게 골라내자 : 필터링

## 빠진 값을 찾고 삭제하기 (계속)

- weather.csv 데이터 역시 이러한 결손값이 존재한다. 데이터에 결손값이 있는지를 확인하는 함수는 `isna()`이다. 평균풍속이 측정되지 않았는지를 다음과 같이 확인해 보자.

```
weather['평균풍속'].isna()      null()
```



# 11 조건에 맞게 골라내자 : 필터링

## 빠진 값을 찾고 삭제하기 (계속)

- weather[ weather['평균풍속'].isna() ]라고 하면, 이 조건을 이용하여 데이터프레임의 일부를 가져올 것이다. 즉, 평균 풍속이 측정되지 않은 날들만 추출해 보고 싶다면 아래 코드로 가능하다. 지난 10년의 데이터 가운데 평균풍속이 기록되지 않은 날은 6일임을 알 수 있다.

```
import pandas as pd

weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
missing_data = weather [ weather['평균풍속'].isna() ]
print(missing_data)
```

|            | 평균기온 | 최대풍속 | 평균풍속 |
|------------|------|------|------|
| 일시         |      |      |      |
| 2012-02-11 | -0.7 | NaN  | NaN  |
| 2012-02-12 | 0.4  | NaN  | NaN  |
| 2012-02-13 | 4.0  | NaN  | NaN  |
| 2015-03-22 | 10.1 | 11.6 | NaN  |
| 2015-04-01 | 7.3  | 12.1 | NaN  |
| 2019-04-18 | 15.7 | 11.7 | NaN  |

# 11 조건에 맞게 골라내자 : 필터링

## 빠진 값을 찾고 삭제하기 (계속)

dropna

결손값 (missing values)이 있는 행 / 열 제거  
삭제

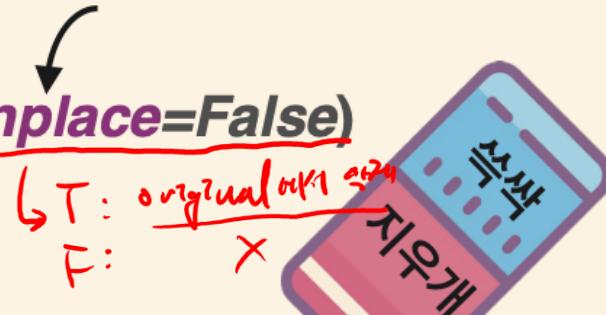
축이 0이면 결손데이터를 포함한 행을 삭제하고  
축이 1이면 결손데이터를 포함한 열을 삭제한다.

0: 행  
1: 열

pandas.DataFrame.dropna(axis=0, how='any', inplace=False)

how의 값이 'any'이면 결손 데이터가 하나라도 포함되면 제거 대상이 되고,  
'all'이면 axis 인자에 따라서 행 혹은 열 전체가 결손 데이터이어야 제거한다.

inplace가 True이면 원본 데이터에서 결손데이터를 삭제하고  
False인 경우는 원본은 그대로 두고 고쳐진 데이터프레임 반환



```
>>> weather.dropna(axis=0, how="any", inplace=True)
>>> weather.loc['2012-02-11'] # 2012년 2월 11일 데이터가 삭제되어 오류발생
... raise KeyError(key) from err
KeyError: '2012-02-11'
```

# 12 빠진 데이터를 깨끗하게 메워 보자

- 우리가 사용하고 있는 `weather.csv`의 결손값을 새로운 값으로 채워보자. 아래의 코드는 `fillna()` 함수를 이용하여 결손값을 0으로 채우고 있다. 그리고 이러한 작업이 원본에 반영되도록 `inplace=True`로 설정했음을 유의해서 보자. 평균풍속의 결손값이 존재했던 2012년 2월 11일의 데이터를 출력해 보자. 결손값 NaN이 아니라 0이 채워진 것을 확인할 수 있다.

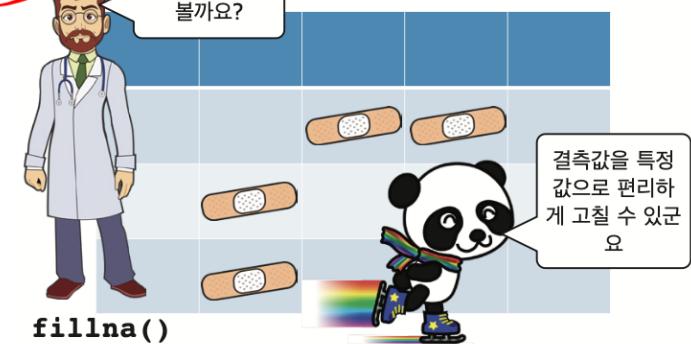
`fillna` 결손값 대체  
→ 정수  
통계학

```
import pandas as pd

weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
weather.fillna(0, inplace = True) # 결손값을 0으로 채움, inplace를 True로 설정해 원본 데이터
                                # 를 수정
print(weather.loc['2012-02-11'])

# 결손값을 고정하는 대신 평균을 사용하는 경우
# weather['평균풍속'].fillna(weather['평균풍속'].mean(), inplace=True)
```

```
평균기온      -0.7
최대풍속       0.0
✓ 평균풍속      0.0
Name: 2012-02-11, dtype: float64
```



# 12 빠진 데이터를 깨끗하게 메워 보자



## 잠깐 – inplace 매개변수

판다스 모듈은 데이터프레임에 대한 조작을 하는 다양한 함수를 제공한다. 그리고 많은 함수들이 `inplace` 매개변수를 가지고 있다. 이것은 조작이 원본 데이터에 이루어지는 것인지, 아니면 사본을 만들어 사본을 변경하는지를 결정하게 된다. 판다스를 다룰 때 흔히 범하는 실수는 `inplace`의 디폴트 인자가 `False`임을 잘 모르고, 원본 데이터프레임이 변경되었다고 생각하는 것이다.

아래 코드를 실행하면 여전히 `NaN`이 출력될 것이다.

```
weather.fillna(0)  
print(weather.loc['2012-02-11'][2])
```

```
weather.fillna( weather['평균풍속'].mean(), inplace = True)  
print(weather.loc['2012-02-11']) # 2012년 2월 11일 데이터가 삭제되지 않음
```

```
평균기온    -0.700000  
최대풍속     3.936441  
평균풍속     3.936441  
Name: 2012-02-11, dtype: float64
```

# 12 빠진 데이터를 깨끗하게 메워 보자

## 데이터를 크기에 따라 나열하자 : 정렬

- 데이터 분석에 있어서 우리가 가지고 있는 데이터를 크기에 따라 순서대로 나열하는 정렬 sort은 필수적이다. 정렬은 판다스에서 그리 어렵지 않다. sort\_values() 함수를 호출하면 된다. 이 함수의 인자로 정렬할 열의 이름과 오름차순, 내림차순 결과를 지정할 수 있다.

```
import pandas as pd
import matplotlib.pyplot as plt

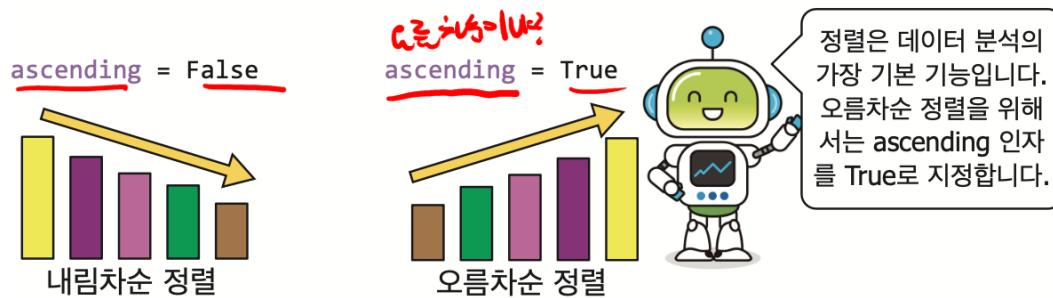
countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
sorted_df = countries_df.sort_values('population') # 인구 수(population)의 오름차순 정렬
print(sorted_df)
```

|    | country | area     | capital    | population |
|----|---------|----------|------------|------------|
| KR | Korea   | 98480    | Seoul      | 51780579   |
| JP | Japan   | 377835   | Tokyo      | 125960000  |
| RU | Russia  | 17100000 | Moscow     | 146748600  |
| US | USA     | 9629091  | Washington | 331002825  |
| CN | China   | 9596960  | Beijing    | 1439323688 |

# 12 빠진 데이터를 깨끗하게 메워 보자

## 데이터를 크기에 따라 나열하자 : 정렬 (계속)

```
countries_df = pd.read_csv('d:/data/countries.csv', index_col = 0)
countries_df.sort_values('population', inplace = True)
print(countries_df)
```



```
countries = pd.read_csv('d:/data/countries.csv', index_col = 0, encoding='CP949')
countries.sort_values(['population', 'area'], ascending = False, inplace = True)
print(countries)
```

|    | country | area     | capital    | population |
|----|---------|----------|------------|------------|
| CN | China   | 9596960  | Beijing    | 1439323688 |
| US | USA     | 9629091  | Washington | 331002825  |
| RU | Russia  | 17100000 | Moscow     | 146748600  |
| JP | Japan   | 377835   | Tokyo      | 125960000  |
| KR | Korea   | 98480    | Seoul      | 51780579   |

# 13 데이터 구조를 변경해 보자

- 데이터프레임을 csv를 읽어서 생성할 수도 있지만, 딕셔너리 데이터를 이용하여 생성 할 수도 있다. 이때 키는 열의 레이블이 되고, 딕셔너리의 키에 딸린 값은 열을 채우는 데이터를 가진 리스트가 된다.
- 딕셔너리의 한 항목이 시리즈 데이터가 되는 것이다. 다음과 같은 방식으로 데이터프레임을 만들어 보자.

```
import pandas as pd

df_1 = pd.DataFrame({'item' : ['ring0', 'ring0', 'ring1', 'ring1'],
                     'type' : ['Gold', 'Silver', 'Gold', 'Bronze'],
                     'price': [20000, 10000, 50000, 30000]})
```

|   | item  | type   | price |
|---|-------|--------|-------|
| 0 | ring0 | Gold   | 20000 |
| 1 | ring0 | Silver | 10000 |
| 2 | ring1 | Gold   | 50000 |
| 3 | ring1 | Bronze | 30000 |

# 13 데이터 구조를 변경해 보자

| index | item  | type   | price |
|-------|-------|--------|-------|
| 0     | ring0 | Gold   | 20000 |
| 1     | ring0 | Silver | 10000 |
| 2     | ring1 | Gold   | 50000 |
| 3     | ring1 | Bronze | 30000 |

pivot 이전의 데이터프레임



| item  | Bronze | Gold  | Silver |
|-------|--------|-------|--------|
| ring0 | NaN    | 20000 | 10000  |
| ring1 | 30000  | 50000 | NaN    |

pivot 이후의 데이터프레임



데이터프레임의  
pivot() 기능을 사용하  
여 테이블의 구조를 변  
경할 수 있습니다.

```
df_2 = df_1.pivot(index='item', columns='type', values='price')
print(df_2)
```

```
type    Bronze      Gold     Silver
item
ring0      NaN  20000.0  10000.0
ring1  30000.0  50000.0      NaN
```

# 13 데이터 구조를 변경해 보자

## concat() 함수로 데이터프레임을 합쳐보자

- 일반적으로 데이터들은 하나의 큰 테이블로 저장되지 않고, 작은 테이블로 나누어져 있는 경우가 많다. 이것은 저장과 관리의 편의성 때문이기도 하고, 데이터 수집의 시기, 주체 등이 달라 별도로 생성된 경우가 많기 때문이다. 이 절에서는 이러한 데이터를 하나로 합치는 방법 가운데 하나인 concat() 함수를 살펴보자.
- 우선 다음과 같이 데이터프레임을 두 개 준비해 보자.
  - 다음과 같이 키:값의 딕셔너리 데이터를 이용하여 만들 수 있고, index를 원하는 값으로 설정할 수 있다.

```
df_1 = pd.DataFrame( {'A' : ['a10', 'a11', 'a12'],
                      'B' : ['b10', 'b11', 'b12'],
                      'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame( {'B' : ['b23', 'b24', 'b25'],
                      'C' : ['c23', 'c24', 'c25'],
                      'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )
```

# 13 데이터 구조를 변경해 보자

## concat() 함수로 데이터프레임을 합쳐보자 (계속)

|     | A   | B   | C   |
|-----|-----|-----|-----|
| 인덱스 | a10 | b10 | c10 |
| 가   | a11 | b11 | c11 |
| 나   | a12 | b12 | c12 |
| 다   |     |     |     |

|     | B   | C   | D   |
|-----|-----|-----|-----|
| 인덱스 | b23 | c23 | d23 |
| 다   | b24 | c24 | d24 |
| 라   | b25 | c25 | d25 |
| 마   |     |     |     |

합칠 데이터프레임의 리스트

pandas.concat(df\_list, axis=0, join='outer')

join 매개변수는 테이블들을 붙일 때 레이블들을 어떻게 사용할지 결정한다.  
이 키워드 인자가 'outer'이면 레이블들의 합집합으로 생성하고 'inner'이면  
레이블들의 교집합으로 생성된다.

축이 0이면 테이블의 행을 늘려서 붙여 나감

축이 1이면 테이블의 열을 늘려며 붙여 나감



❶ axis = 0 헉 가능 결합  
| 열 //

❷ join = outer 레이블의 합집합  
Tuner // 결합

```
df_3 = pd.concat([df_1, df_2]) # df_1, df_2 두 데이터프레임을 합쳐서 df_3을 생성  
print(df_3)
```

|   | A   | B   | C   | D   |
|---|-----|-----|-----|-----|
| 가 | a10 | b10 | c10 | NaN |
| 나 | a11 | b11 | c11 | NaN |
| 다 | a12 | b12 | c12 | NaN |
| 다 | NaN | b23 | c23 | d23 |
| 라 | NaN | b24 | c24 | d24 |
| 마 | NaN | b25 | c25 | d25 |

# LAB<sup>3</sup> 다양한 방법으로 concat 적용해 보기

실습 시간



앞서 생성한 df\_1과 df\_2 데이터프레임을 여러 가지 방법으로 합쳐보자. 이를 위하여 concat() 함수의 axis와 join 키워드 인자에 다음과 같이 다양하게 적용하여 결과를 확인해보라.

원하는 결과

pandas.concat([df\_1, df\_2], axis = 0, join = 'outer')

|      | A   | B   | C   | D   |
|------|-----|-----|-----|-----|
| 가    | a10 | b10 | c10 | NaN |
| df_1 | a11 | b11 | c11 | NaN |
| 다    | a12 | b12 | c12 | NaN |
| df_2 | NaN | b23 | c23 | d23 |
| 라    | b24 | c24 | d24 |     |
| 마    | NaN | b25 | c25 | d25 |

axis=0과 outer로 결합한 결과

pandas.concat([df\_1, df\_2], axis = 1, join = 'outer')

|      | A   | B   | C   | B   | C   | D   |
|------|-----|-----|-----|-----|-----|-----|
| 가    | a10 | b10 | c10 | NaN | NaN | NaN |
| df_1 | a11 | b11 | c11 | NaN | NaN | NaN |
| 다    | a12 | b12 | c12 | b23 | c23 | d23 |
| df_2 | NaN | NaN | NaN | b24 | c24 | d24 |
| 라    | NaN | NaN | NaN | b25 | c25 | d25 |
| 마    | NaN | NaN | NaN |     |     |     |

axis=1과 outer 조인의 결과

pandas.concat([df\_1, df\_2], axis = 0, join = 'inner')

|      | B   | C   |
|------|-----|-----|
| df_1 | b10 | c10 |
| 다    | b11 | c11 |
| df_2 | b12 | c12 |
| 라    | b23 | c23 |
| 마    | b24 | c24 |
| 라    | b25 | c25 |

두 데이터프레임의  
공통 열

axis = 0과 inner 결합한 결과

pandas.concat([df\_1, df\_2], axis = 1, join = 'inner')

|   | A   | B   | C   | B   | C   | D   |
|---|-----|-----|-----|-----|-----|-----|
| 다 | a12 | b12 | c12 | b23 | c23 | d23 |

df\_1                    df\_2

axis =1과 inner 조인의 결과

힌트



axis 매개변수에는 0과 1, join 매개변수에는 'outer'와 'inner' 두 종류의 인자를 넘길 수 있으므로 모두 네 가지의 조합이 가능하다. 이때, axis 키워드 인자가 1인 경우 열 방향으로 테이블을 결합한다.

# LAB<sup>3</sup> 다양한 방법으로 concat 적용해 보기

해답 코드



```
import pandas as pd

df_1 = pd.DataFrame(
    {'A' : ['a10', 'a11', 'a12'],
     'B' : ['b10', 'b11', 'b12'],
     'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame(
    {'B' : ['b23', 'b24', 'b25'],
     'C' : ['c23', 'c24', 'c25'],
     'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )

print( pd.concat( [df_1, df_2] , axis = 0, join = 'outer' ) )
print( pd.concat( [df_1, df_2] , axis = 0, join = 'inner' ) )
print( pd.concat( [df_1, df_2] , axis = 1, join = 'outer' ) )
print( pd.concat( [df_1, df_2] , axis = 1, join = 'inner' ) )
```

# LAB<sup>3</sup> 다양한 방법으로 concat 적용해 보기

수행 결과



|   | A   | B   | C   | D   |     |     |
|---|-----|-----|-----|-----|-----|-----|
| 가 | a10 | b10 | c10 | NaN |     |     |
| 나 | a11 | b11 | c11 | NaN |     |     |
| 다 | a12 | b12 | c12 | NaN |     |     |
| 다 | NaN | b23 | c23 | d23 |     |     |
| 라 | NaN | b24 | c24 | d24 |     |     |
| 마 | NaN | b25 | c25 | d25 |     |     |
|   | B   | C   |     |     |     |     |
| 가 | b10 | c10 |     |     |     |     |
| 나 | b11 | c11 |     |     |     |     |
| 다 | b12 | c12 |     |     |     |     |
| 다 | b23 | c23 |     |     |     |     |
| 라 | b24 | c24 |     |     |     |     |
| 마 | b25 | c25 |     |     |     |     |
|   | A   | B   | C   | B   | C   | D   |
| 가 | a10 | b10 | c10 | NaN | NaN | NaN |
| 나 | a11 | b11 | c11 | NaN | NaN | NaN |
| 다 | a12 | b12 | c12 | b23 | c23 | d23 |
| 라 | NaN | NaN | NaN | b24 | c24 | d24 |
| 마 | NaN | NaN | NaN | b25 | c25 | d25 |
|   | A   | B   | C   | B   | C   | D   |
| 다 | a12 | b12 | c12 | b23 | c23 | d23 |

# LAB<sup>3</sup> 다양한 방법으로 concat 적용해 보기

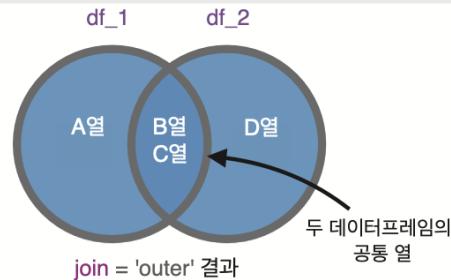


## 잠깐 - inner와 outer의 결합방식 차이를 알아보자

`pd.concat( [df_1, df_2], join='outer' )`과 `pd.concat( [df_1, df_2], join='inner' )`의 결과는 두 데이터프레임 df\_1과 df\_2를 결합하는 방법 중에서 합집합과 교집합을 구하는 방식과도 유사하다.

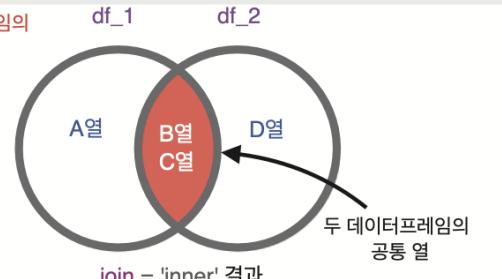
outer를 사용하는 경우 그림과 같이 데이터프레임에서 공통적인 열인 B열과 C열과 함께 A열과 D열도 포함한 새로운 데이터프레임을 반환한다.

|      | A | B   | C   | D   |
|------|---|-----|-----|-----|
| df_1 | 가 | a10 | b10 | c10 |
|      | 다 | a11 | b11 | c11 |
|      | 다 | NaN | b12 | c12 |
| df_2 | 마 | NaN | b23 | c23 |
|      | 마 | NaN | b24 | c24 |
|      | 마 | NaN | b25 | c25 |



반면 inner를 사용하는 경우 그림과 같이 데이터프레임에서 공통적인 열인 B열과 C열 만을 포함한 새로운 데이터프레임을 반환한다.

|      | B   | C   |
|------|-----|-----|
| df_1 | b10 | c10 |
|      | b11 | c11 |
|      | b12 | c12 |
|      | b23 | c23 |
| df_2 | b24 | c24 |
|      | b25 | c25 |



# 14 데이터베이스 join 방식의 데이터 병합 – merge

- 데이터베이스는 **조인join**이라는 연산을 지원한다. 이 조인 연산과 같은 방식의 데이터 병합을 지원하는 판다스 함수가 `merge()` 함수이다. `merge()` 함수는 데이터프레임 `df_1`을 `df_2`와 병합하려고 할 때, 다음과 같은 방식을 사용한다.

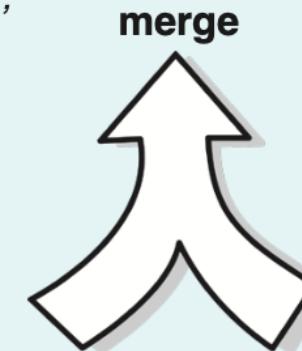
`df_1.merge(df_2, 다양한 선택 사항을 결정하는 키워드 인자들... )`

현재 데이터프레임과 결합할  
오른쪽의 데이터프레임

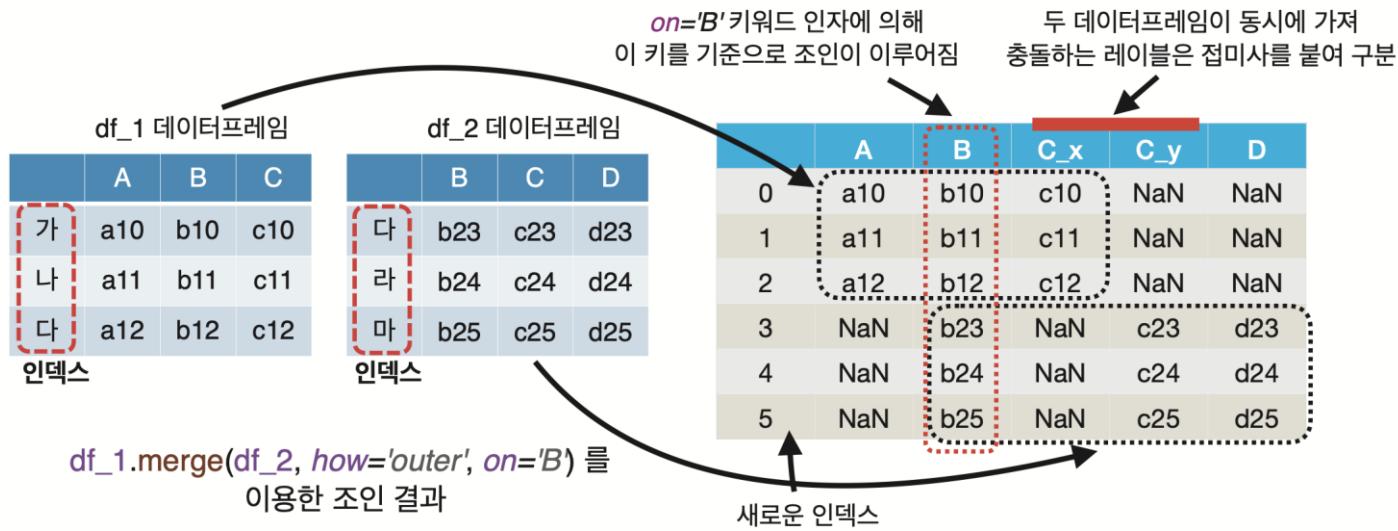
결합의 방식으로 'inner', 'outer', 'left', 'right',  
방식이 있음

**DataFrame.merge( *right\_df*, *how='inner'*, *on=None* )**

결합 연산을 수행하기 위해 사용할 레이블  
(왼쪽과 오른쪽 데이터프레임 모두에 존재해야 함)

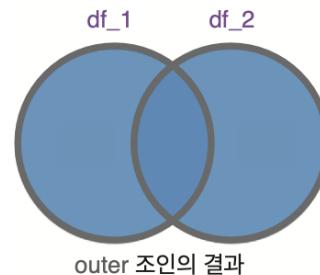


# 14 데이터베이스 join 방식의 데이터 병합 – merge



```
df_3 = df_1.merge(df_2, how='outer', on='B')
print(df_3)
```

|   | A   | B   | C_x | C_y | D   |
|---|-----|-----|-----|-----|-----|
| 0 | a10 | b10 | c10 | NaN | NaN |
| 1 | a11 | b11 | c11 | NaN | NaN |
| 2 | a12 | b12 | c12 | NaN | NaN |
| 3 | NaN | b23 | NaN | c23 | d23 |
| 4 | NaN | b24 | NaN | c24 | d24 |
| 5 | NaN | b25 | NaN | c25 | d25 |



# 인덱스를 키로 활용하여 merge 적용해 보기

## 인덱스를 키로 활용하여 merge 적용해 보기

- 앞에서 살펴본 `merge()` 함수의 동작 결과를 살펴보면, 두 데이터를 결합할 때 사용할 키가 될 레이블을 지정하면, 해당 레이블에 있는 값들을 이용하여 테이블을 생성하고 있다. 그런데, 원래 테이블에 있던 인덱스는 사라진 것을 확인할 수 있다.
- 많은 경우 인덱스가 키의 역할을 수행하는 경우도 있다. 이런 경우에 인덱스를 키로 사용하라고 할 수도 있다. 이러한 방식으로 `merge`를 수행하려면 다음과 같이 코딩하면 된다.

```
df_1.merge(df_2, how = 'outer', left_index = True, right_index = True )
```

```
df_3 = df_1.merge(df_2, how='outer', left_index = True, right_index = True )
print(df_3)
```

|   | A   | B_x | C_x | B_y | C_y | D   |
|---|-----|-----|-----|-----|-----|-----|
| 가 | a10 | b10 | c10 | NaN | NaN | NaN |
| 나 | a11 | b11 | c11 | NaN | NaN | NaN |
| 다 | a12 | b12 | c12 | b23 | c23 | d23 |
| 라 | NaN | NaN | NaN | b24 | c24 | d24 |
| 마 | NaN | NaN | NaN | b25 | c25 | d25 |

# 인덱스를 키로 활용하여 merge 적용해 보기

## 인덱스를 키로 활용하여 merge 적용해 보기 (계속)

인덱스를 기준으로 조인이 이루어짐

두 데이터프레임이 동시에 가져  
충돌하는 레이블은 접미사를 붙여 구분

|   | A   | B_x | C_x | B_y | C_y | D   |
|---|-----|-----|-----|-----|-----|-----|
| 가 | a10 | b10 | c10 | NaN | NaN | NaN |
| 나 | a11 | b11 | c11 | NaN | NaN | NaN |
| 다 | a12 | b12 | c12 | b23 | c23 | d23 |
| 마 | NaN | b23 | NaN | b24 | c24 | d24 |
| 바 | NaN | b24 | NaN | b25 | c25 | d25 |

df\_1 데이터프레임      df\_2 데이터프레임



인덱스를 기준으로 조인을  
하면 인덱스가 사라지지 않  
아요. 하지만 두 데이터프레  
임이 동시에 가지는 열의 레  
이블은 접미사를 붙여서  
B\_x, C\_x, B\_y, C\_y와 같  
이 새롭게 만들지요.

# 인덱스를 키로 활용하여 merge 적용해 보기

## 인덱스를 키로 활용하여 merge 적용해 보기 (계속)



### 도전문제 12.1

- (1) 위의 df\_1, df\_2 프레임워크에 대하여 다음과 같은 조인 연산을 수행하고 그 결과를 출력하여라.

```
df_1.merge(df_2, how='outer')
```

- (2) 위의 df\_1, df\_2 프레임워크에 대하여 다음과 같은 조인 연산을 수행하고 그 결과를 출력하여라.

```
df_1.merge(df_2, how='outer', on='C')
```

# LAB<sup>4</sup> 다양한 방법으로 merge 적용해 보기

실습 시간



앞서 생성한 df\_1과 df\_2 데이터프레임을 합치는 데에 merge()의 how 매개변수에는 네 종류의 인자를 넘길 수 있다. on='B'를 유지한 채로 how를 변경하여 다양한 결과를 확인해 보라.

원하는 결과

왼쪽 데이터프레임  
인덱스 기준 결합

오른쪽 데이터프레임  
인덱스 기준 결합

| left outer |     |     |     |     |     |
|------------|-----|-----|-----|-----|-----|
|            | A   | B   | C_x | C_y | D   |
| 0          | a10 | b10 | c10 | NaN | NaN |
| 1          | a11 | b11 | c11 | NaN | NaN |
| 2          | a12 | b12 | c12 | NaN | NaN |

| right outer |     |     |     |     |     |
|-------------|-----|-----|-----|-----|-----|
|             | A   | B   | C_x | C_y | D   |
| 0           | NaN | b23 | NaN | c23 | d23 |
| 1           | NaN | b24 | NaN | c24 | d24 |
| 2           | NaN | b25 | NaN | c25 | d25 |

| full outer |     |     |     |     |     |
|------------|-----|-----|-----|-----|-----|
|            | A   | B   | C_x | C_y | D   |
| 0          | a10 | b10 | c10 | NaN | NaN |
| 1          | a11 | b11 | c11 | NaN | NaN |
| 2          | a12 | b12 | c12 | NaN | NaN |
| 3          | NaN | b23 | NaN | c23 | d23 |
| 4          | NaN | b24 | NaN | c24 | d24 |
| 5          | NaN | b25 | NaN | c25 | d25 |

| inner                        |  |  |  |  |
|------------------------------|--|--|--|--|
| Empty DataFrame              |  |  |  |  |
| Columns: [A, B, C_x, C_y, D] |  |  |  |  |
| Index: []                    |  |  |  |  |

모든 열은 있음

공통 인덱스만 남음

힌트



'outer' 방식을 기본으로 생각하면 된다. 'left'는 'outer' 조인의 결과에서 왼쪽 데이터프레임에 존재하는 키를 가진 것만 뽑아내면 되고, 'right'는 오른쪽 테이블에서 발견되는 키를 가진 것만 뽑으면 된다. inner는 왼쪽과 오른쪽 테이블 모두에서 발견되는 키를 가져야 추출되는데, 이 예에서는 공백이다.

| on = 'B' |     |     |     |     |     |
|----------|-----|-----|-----|-----|-----|
|          | A   | B   | C_x | C_y | D   |
| 0        | a10 | b10 | c10 | NaN | NaN |
| 1        | a11 | b11 | c11 | NaN | NaN |
| 2        | a12 | b12 | c12 | NaN | NaN |
| 3        | NaN | b23 | NaN | c23 | d23 |
| 4        | NaN | b24 | NaN | c24 | d24 |
| 5        | NaN | b25 | NaN | c25 | d25 |

inner: 교집합 없음  
{b10, b11, b12}  
{b23, b24, b25}

right outer: b10, b11, b12, b23, b24, b25

# LAB<sup>4</sup> 다양한 방법으로 merge 적용해 보기

해답 코드



```
import pandas as pd

df_1 = pd.DataFrame( {'A' : ['a10', 'a11', 'a12'],
                      'B' : ['b10', 'b11', 'b12'],
                      'C' : ['c10', 'c11', 'c12']} , index = ['가', '나', '다'] )

df_2 = pd.DataFrame( {'B' : ['b23', 'b24', 'b25'],
                      'C' : ['c23', 'c24', 'c25'],
                      'D' : ['d23', 'd24', 'd25']} , index = ['다', '라', '마'] )

print('left outer \n' , df_1.merge(df_2, how='left', on='B' ) )
print('right outer \n' ,df_1.merge(df_2, how='right', on='B' ) )
print('full outer \n' ,df_1.merge(df_2, how='outer', on='B' ) )
print('inner \n' ,df_1.merge(df_2, how='inner', on='B' ) )
```

- CSV는 쉼표 단위로 데이터를 구분하여 저장하는 텍스트 파일이다.
- 판다스는 테이블 형태의 데이터를 다루는 데에 최적화된 모듈이다.
- 판다스의 데이터는 1차원 시리즈와 2차원 데이터프레임을 기본으로 한다.
- 판다스는 맷플롯립과 완벽히 연동되어 데이터 시각화 기능도 강력하다.
- 데이터프레임에 담긴 데이터의 일부를 다루기 위해서 인덱스와 컬럼스 정보를 이용한다.
- 데이터프레임을 결합하거나 구조를 변경하는 다양한 기능이 제공된다.

판다스로 데이터를 다룰 때 사용하는 기본적인 자료 구조가 무엇이죠?



1차원 데이터는 시리즈라고 부르고  
이 시리즈가 모여서 2차원 구조를 이루는 것을 데이터프레임이라고 하죠.

맞아요. 데이터프레임은 판다스를 이용한 데이터 처리의 핵심  
자료구조라고 할 수 있어요.

데이터프레임 내에 존재하는 데이터 항목을 다룰 때에 인덱스와 컬럼스 정보를  
이용하는 건 이제 잘 이해하겠지요?





네, 인덱스는 특정 데이터 행을 가리키고, 컬럼스는 각 열의 레이블을 담고 있죠.  
각 열은 동일한 특성을 나타내는 데이터를 담고 있고요.

잘 설명했어요. 판다스는 아주 빠르게 데이터의 열을 추가하고, 연산할 수 있는데,  
이를 위해서는 데이터 항목 하나 하나를 접근하는 코드를 작성하는 것이 아니라  
인덱스와 컬럼스를 이용하여 시리즈 단위로 연산이 이루어지게 하면 효율적이랍니다.





**Questions?**