

판다스(pandas)로 데이터를 분석하기

12.1 엑셀보다 빠른 일처리는 판다스로

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

학번	이름	국어	수학	영어	합계
211101	이준신	96	90	89	362
211101	권용수	76	56	67	297
211101	이억기	89	67	98	323
211101	권율	83	89	90	349
211101	김시민	67	90	96	342
211101	이정암	80	60	79	305
211101	정문부	57	69	95	310

저는 엑셀
이라고 해요!



판다스는 엑셀같은 스프레드시트
프로그램보다 연산을 빠르고
효율적으로 할 수 있어요.



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
이종heterogeneous 자료형의 열을 가진 테이블 데이터
시계열 데이터
레이블을 가진 다양한 행렬 데이터
다양한 관측 통계 데이터
- 3 핵심 구조
시리즈Series : 1차원 구조를 가진 하나의 열
데이터프레임DataFrame : 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

12.2 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

- 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
검사와 필터링, 계산을
할 수 있어요.

• 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.

• 필터, 정렬 및 그룹화

- sort_values()로 데이터를 정렬할 수 있다.
- 조건을 사용하여 열을 필터링할 수 있다.
- groupby()를 이용하여 기준에 따라 몇 개의 그룹으로 데이터를 분할할 수 있다.

• 데이터 정제

- 데이터의 누락 값을 확인할 수 있다.
- 특정한 값을 다른 값으로 대체할 수 있다.



잠깐 - 판다스 or 판다

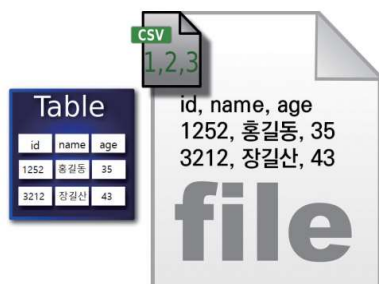
판다스라는 특이한 이름은 "panel data"라는 용어에서 유래되었다. 이 panel data라는 용어 역시 생소한 용어인데 이는 [계량경제학econometrics](#) 용어로 동일한 관찰자에 의하여 여러 회에 걸쳐 관측된 데이터 집합을 지칭하는 용어이다. 중국 쓰촨성일대에 서식하는 동물인 판다와는 관계가 없으나 용어가 비슷하므로 많은 사람들이 판다스의 로고로 판다 그림을 사용하기도 한다.



판다스는 판다하고 상관없는 데이터 분석 라이브러리예요!

12.3 CSV라고 들어봤니

- CSV는 테이블 형식의 데이터를 저장하고 이동하는 데 사용되는 구조화된 텍스트 파일 형식이다. CSV는 [쉼표로 구분한 변수comma separated variables](#)의 약자이다.
- CSV의 역사는 1972년으로 거슬러 올라가며 Microsoft Excel와 같은 [스프레드 시트spread sheet](#) 소프트웨어에 적합한 형식이다. 데이터 과학에서 사용되는 데이터 가운데 상당한 비율의 데이터들이 CSV 형식으로 공유되는 경우가 많다.



탭으로 항목을 구분하는 파일 : tsv

아하, 테이블 형태의 데이터를 한 행씩 옮겨 적어둔 파일이구나.

각 행의 항목들은 쉼표로 구분되어 있어. 그래서 CSV라고 해

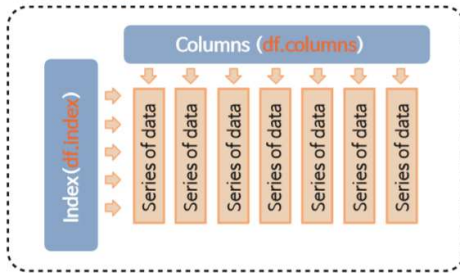
- CSV 파일은 필드를 나타내는 열과 레코드를 나타내는 행으로 구성
- 만약 데이터의 중간에 구분자가 포함되어야 한다면 따옴표를 사용하여 필드를 묶어야 함
 - 예를 들어서 'Gildong, Hong'이라는 데이터가 있다고 하자. 데이터의 중간에 쉼표(,)가 포함되어 있다. 이러한 경우에는 구분자로 사용되는 쉼표와 구분하기 위하여 반드시 데이터를 따옴표로 감싸야 한다.
- CSV 파일의 첫 번째 레코드에는 열 제목이 포함되어 있을 수 있다.
 - CSV 형식 자체의 요구사항이 아니라 단순히 일반적인 관행
- CSV 파일의 크기를 알 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.

12.4 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.
- 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 **인덱스** index, 열의 이름을 **컬럼** columns라 부른다.

데이터 구조	차원	설명
시리즈	1	레이블이 붙어있는 1차원 벡터
데이터프레임	2	행과 열로 되어있는 2차원 테이블. 각 열은 시리즈로 되어 있다.

DataFrame df



Series

1	3	4	NaN	6	8
---	---	---	-----	---	---

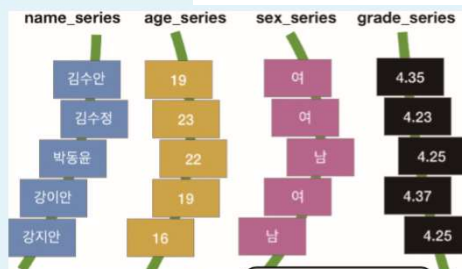
```
>>> import numpy as np
>>> import pandas as pd
>>> series = pd.Series([1, 3, 4, np.nan, 6, 8])
>>> series
0    1.0
1    3.0
2    4.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

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

```
>>> 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	4.25
강이안	19	여	4.37
강지안	16	남	4.25



네 개의 시리즈를 만들었어요. 이것들을 모아 데이터프레임을 만들 수 있어요.

```
>>> 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



판다스의 DataFrame 클래스를 사용해서 하나의 데이터 프레임을 만들 수 있다

12.5 판다스로 데이터 파일을 읽기

- 판다스 모듈을 이러한 csv 파일을 읽어들이어서 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 첫열은 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 df를 출력해보자.



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

12.5 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

12.6 데이터를 설명하는 인덱스와 컬럼스 객체

- 데이터 프레임에서는 다음과 같이 **인덱스 index**와 **컬럼스 columns** 객체를 정의하여 사용한다.
- 인덱스는 행들의 레이블이고 columns는 열들의 레이블이 저장된 객체이다.

비워 두었던 열 이름

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**

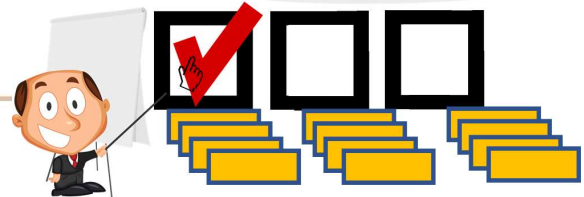

```
import pandas as pd
```

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

첫 번째 열을 인덱스로 사용하겠다는 뜻.

	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

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



12.7 열을 기준으로 데이터 선택하기

- 특정한 열만 선택하려면 아래와 같이 대괄호 안에 열의 이름을 넣으면 된다.
- 다음 코드는 countries.csv를 다시 읽고 있다. 그리고 처음에는 인덱스를 첫 열로 지정해서 df_my_index로 할당했고, 인덱스 지정 없이 만든 데이터프레임은 df_no_index로 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


```
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
선택하는 경우 : 선택을 원하는 열
의 레이블을 리스트에 넣어서 전달



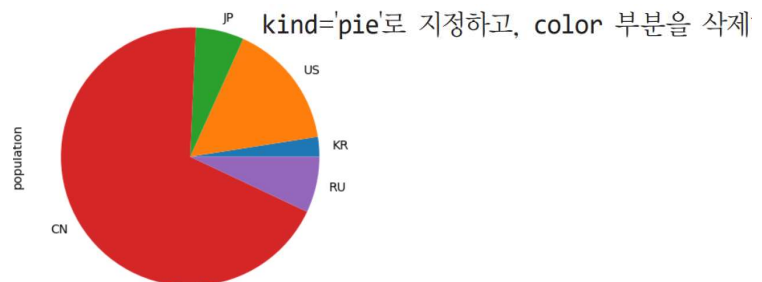
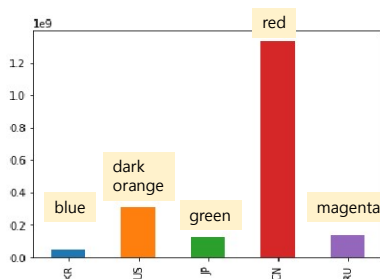
12.8 데이터 가시화하기

- 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)
```

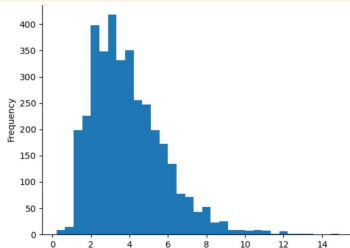
```
countries_df['population'].plot(kind='bar', color=('b', 'darkorange', 'g', 'r', 'm'))
plt.show()
```



```
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 읽어오기 위해서 사용함



데이터를 차트로 표시하는 일을 아주 간편하게 할 수 있도록 지원하고 있어요.



잠깐 - 판다스를 이용하여 csv를 여는 코드에서 눈여겨 볼 것

마이크로소프트의 윈도우 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 읽히는 것을 알 수 있다. float()를 이용하여 값을 실수로 바꿀 필요가 없는 것이다.

12.9 판다스에서도 슬라이싱으로 행 선택이 된다.

- 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

12.9 판다스에서도 슬라이싱으로 행 선택이 된다.

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

행의 레이블이 'KR'인 행만을 선택하기

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

데이터프레임에서 특정한 요소 하나만을 선택하려면 loc함수에 행과 열의 레이블을 써주면 된다

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

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

12.10 새로운 열을 쉽게 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



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.

12.11 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

```
...
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
```



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)
```

```
8.538507014753446 8.537338236838895
```



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

12.12 데이터 집계 분석도 손쉽게

- 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. 예를 들어 다음 코드를 보자.

```
>>> weather = pd.read_csv('d:/data/weather.csv', index_col = 0, encoding='CP949')
>>> weather.count()
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()
3649
```

12.12 데이터 집계 분석도 손쉽게

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

여러 개의 열을 분석하고 싶을 때는
원하는 열의 레이블들을 리스트로
제공

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

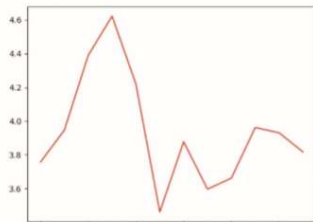
min(), max(), mean(), sum() 등도
적용 가능

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

LAB¹²⁻² 판다스로 울릉도의 바람 세기 분석하기

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



LAB¹²⁻² 판다스로 울릉도의 바람 세기 분석하기

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

weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')
monthly = [ None for x in range(12) ] # 달별로 구분된 12개의 데이터프레임
monthly_wind = [ 0 for x in range(12) ] # 각 달의 평균 풍속을 담을 리스트
# 마지막에 해당 행의 데이터가 측정된 달을 기록한 열을 추가
weather['month'] = pd.DatetimeIndex(weather['일시']).month

for i in range(12) :
    monthly[i] = weather[ weather['month'] == i + 1 ] # 달별로 분리
    monthly_wind[i] = monthly[i].mean()['평균풍속'] # 개별 데이터 분석

plt.plot(monthly_wind, 'red')
plt.show()
```

비교 : csv 모듈을 사용하는 것 보다 간결하고 강력함

```
import csv
import matplotlib.pyplot as plt

f = open('d:/data/weather.csv') # CSV 파일 열어서 f에 저장
data = csv.reader(f) # reader() 함수로 읽기
header = next(data) # 헤더를 제거

monthly_wind = [ 0 for x in range(12) ] # 매달 풍속을 담을 리스트, 초기화 0
days_counted = [ 0 for x in range(12) ] # 각 달마다 측정된 일수, 초기화 0

for row in data:
    month = int(row[0][5:7]) # 0번 열에서 달 정보 추출
    if row[3] != '' : # 풍속 데이터 존재하는지 확인
        wind = float(row[3]) # 풍속을 얻어 온다.
        monthly_wind[month-1] += wind # 해당 달에 풍속 데이터 추가
        days_counted[month-1] += 1 # 해당 달의 일수를 증가

for i in range(12) :
    monthly_wind[i] /= days_counted[i] # 일수로 나누어 월평균 구하기

plt.plot(monthly_wind, 'blue')
plt.show()

f.close() # 파일을 닫는다.
```

복잡하고 오류의 가능성도 높다!

12.13 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑

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

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

DatetimeIndex() 함수를 통해서 weather['month'] 열이 새롭게 추가되었음..

```
print(means)                                     >>> weather.tail()
```

month	평균기온	최대풍속	평균풍속	일시	평균기온	최대풍속	평균풍속	month
1	1.598387	8.158065	3.757419	3648	2020-07-27	22.1	4.2	1.7
2	2.136396	8.225357	3.946786	3649	2020-07-28	21.9	4.5	1.6
3	6.250323	8.871935	4.390291	3650	2020-07-29	21.6	3.2	1.0
4	11.064667	9.305017	4.622483	3651	2020-07-30	22.9	9.7	2.4
5	16.564194	8.548710	4.219355	3652	2020-07-31	25.7	4.8	2.5
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					

12.13 데이터를 특정한 값에 기반하여 묶는 기능 : 그룹핑

```
sum_data = weather.groupby('month').sum()
print(sum_data)
```

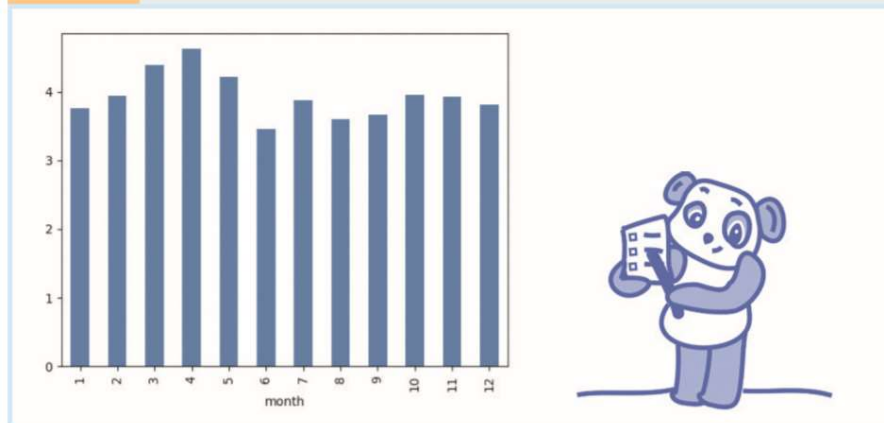
해당 데이터를 그룹별로 모두 더하여 값을 확인

month	평균기온	최대풍속	평균풍속
1	495.5	2529.0	1164.8
2	604.6	2303.1	1105.1
3	1937.6	2750.3	1356.6
4	3319.4	2782.2	1377.5
5	5134.9	2650.1	1308.0
6	5885.0	2083.7	1038.3
7	7231.8	2270.0	1202.0
8	7672.1	2124.5	1114.8
9	6097.1	2068.9	1098.5
10	4769.0	2407.7	1228.1
11	2966.9	2404.0	1179.2
12	1163.6	2494.1	1183.3

LAB¹²⁻³ 울릉도는 몇 월에 바람이 가장 강할까? groupby() 활용

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



LAB¹²⁻³ 울릉도는 몇 월에 바람이 가장 강할까? groupby() 활용

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

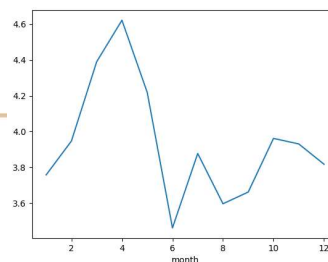
weather = pd.read_csv('d:/data/weather.csv', encoding='CP949')

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

plt.show()
```

LAB 12-1, 12-2보다 간단한
코딩 : 강력한 성능을 보임

해당 부분을 생략하면 LAB
12-1, 12-2의 결과와 동일함

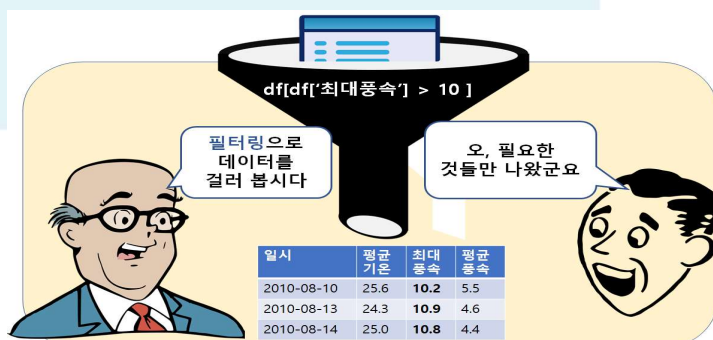


12.14 조건에 맞게 골라내자 : 필터링

- 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
동일한 문법



12.14 조건에 맞게 골라내자 : 필터링

```
>>> weather[ weather['최대풍속'] >= 10.0 ]
```

일시	평균기온	최대풍속	평균풍속
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

12.15 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(missing data)을 탐지하고 수정하는 함수를 제공한다.
- weather.csv 데이터 역시 이러한 결손값이 존재한다. 데이터에 결손값이 있는지를 확인하는 함수는 isna()이다. 평균풍속이 측정되지 않았는지를 다음과 같이 확인해 보자.

```
weather['평균풍속'].isna()
```



12.15 빠진 값을 찾고 삭제하기

- 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)
```

일시	평균기온	최대풍속	평균풍속
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

12.15 빠진 값을 찾고 삭제하기

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

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

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.



```
>>> weather.dropna(axis=0, how="any", inplace=True)
>>> weather.loc['2012-02-11']
... raise KeyError(key) from err
KeyError: '2012-02-11'
```

12.16 빠진 데이터를 깨끗하게 메워 보자

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

```
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 데이터를 수정

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

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



잠깐 - inplace 매개변수

판다스 모듈은 데이터프레임에 대한 조작을 하는 다양한 함수를 제공한다. 그리고 많은 함수들이 inplace 매개 변수를 가지고 있다. 이것은 조작이 원본 데이터에 이루어지는 것인지, 아니면 사본을 만들어 사본을 변경하는지를 결정하게 된다.

판다스를 다룰 때 흔히 범하는 실수는 inplace의 디폴트 인자가 False임을 잘 모르고, 원본 데이터프레임이 변경되었다고 생각하는 것이다.

아래 코드를 실행하면 여전히 NaN이 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'])
```

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

측정이 누락된 2012년 2월 11일의 풍속을
전체 데이터 평균으로 채울 수 있다

```
>>> weather['최대풍속'].fillna( weather['최대풍속'].mean(), inplace = True)
>>> print(weather.loc['2012-02-11'])
```

```
평균기온    -0.700000
```

```
최대풍속     7.911099
```

```
평균풍속      NaN
```

```
Name: 2012-02-11, dtype: float64
```

측정이 누락된 2012년 2월 11일의 풍속을
최대 풍속 데이터 평균으로 채울 수 있다

```
>>> weather['평균풍속'].fillna( weather['평균풍속'].mean(), inplace = True)
```

```
>>> print(weather.loc['2012-02-11'])
```

```
평균기온    -0.700000
```

```
최대풍속     7.911099
```

```
평균풍속     3.936441
```

```
Name: 2012-02-11, dtype: float64
```

측정이 누락된 2012년 2월 11일의 풍속을
평균 풍속 데이터 평균으로 채울 수 있다

12.17 데이터 구조를 변경해 보자

- 데이터프레임을 csv를 읽어서 생성할 수도 있지만, 딕셔너리 데이터를 이용하여 생성할 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.

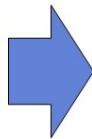
```
import pandas as pd
```

```
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

12.17 데이터 구조를 변경해 보자

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



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

```
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

12.8 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. 딕셔너리 데이터를 이용하여 만들 수 있고, `index`를 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'] )
```

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

인덱스

인덱스

합칠 데이터프레임의 리스트

축이 0이면 테이블의 행을 늘려서 붙여 나감
축이 1이면 테이블의 열을 늘려서 붙여 나감

`pandas.concat(df_list, axis=0, join='outer')`

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



```
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

LAB¹²⁻⁴ 다양한 방법으로 concat 적용해 보기

앞서 생성한 df_1과 df_2 데이터프레임을 합치는 데에 concat의 axis와 join 매개변수에 인자를 다양하게 적용하여 결과를 확인해보라.

원하는 결과

`pandas.concat([df_1, df2], axis = 0, join = 'outer')`

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

`pandas.concat([df_1, df2], axis = 0, join = 'inner')`

→ 두 데이터프레임의 공통 열

	B	C
가	b10	c10
나	b11	c11
다	b12	c12
라	b23	c23
마	b24	c24
	b25	c25

`pandas.concat([df_1, df2], axis = 1, join = 'outer')`

	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

`pandas.concat([df_1, df2], axis = 1, join = 'inner')`

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

df_1 df_2

LAB¹²⁻⁴ 다양한 방법으로 concat 적용해 보기

```
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' ) )
```


12.20 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 merge를 수행하려면 다음과 같이 코딩을 하면 된다.

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

이 방식으로 실제로 merge를 적용해 보자. 앞서 사용한 데이터프레임을 그대로 사용해 보자. 그림과 같은 모양의 테이블 데이터가 존재한다.

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

인덱스

인덱스

12.20 인덱스를 키로 활용하여 merge 적용해 보기

```
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 기준으로 조인이 이루어짐

두 데이터프레임이 동시에 가져
충돌하는 레이블은 점마사를 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

df_2

인덱스 기준으로 조인을 하면
인덱스가 사라지지 않아요

LAB¹²⁻⁵ 다양한 방법으로 merge 적용해 보기

앞서 생성한 df_1과 df_2 데이터프레임을 합치는 데에 merge()의 how 매개변수에는 네 종류의 인자를 넘길 수 있다. on='B'를 유지한채로 how를 변경하여 다양한 결과를 확인해 보라.

원하는 결과

<pre> 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 </pre>	<pre> 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: [] </pre>
--	---

LAB¹²⁻⁵ 다양한 방법으로 merge 적용해 보기

```

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' ) )
    
```

12.21 데이터를 크기에 따라 나열하자 : 정렬

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

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

	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

```
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

