

2강. 데이터 전처리

2023.01.03.

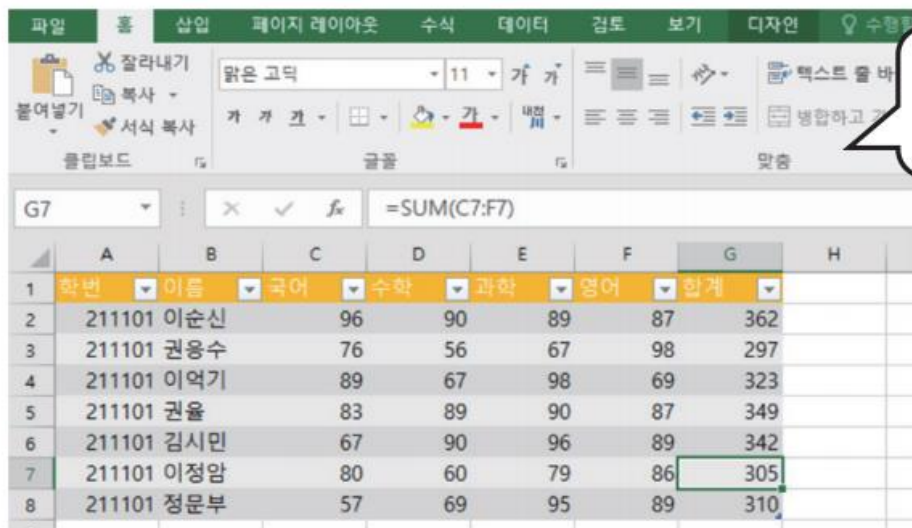
양희철

hcyang@cnu.ac.kr

Pandas를 이용한 데이터 전처리

판다스의 특징

- 넘파이는 2차원 행렬 형태의 데이터를 지원하지만, 데이터의 속성을 표시하는 행이나 열의 레이블을 가지고 있지 않다는 한계가 있다. 하지만 판다스 패키지를 사용하면 이러한 문제를 해결할 수 있다.



	A	B	C	D	E	F	G	H
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 엑셀
이라고 해요!



판다스는 엑셀같은 스프레드시트
프로그램보다 연산을 빠르고
효율적으로 할 수 있어요.

Pandas를 이용한 데이터 전처리

판다스의 특징



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

Pandas를 이용한 데이터 전처리

데이터 불러오기, 보기, 및 정제

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

Pandas를 이용한 데이터 전처리

데이터 불러오기, 보기, 및 정제

- 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.

Pandas를 이용한 데이터 전처리

CSV (comma separated variables)

- CSV는 테이블 형식의 데이터를 저장하고 이동하는 데 사용되는 구조화된 텍스트 파일 형식이다.
- 데이터 과학에서 사용되는 데이터 가운데 상당한 비율의 데이터들이 CSV 형식으로 공유되는 경우가 많다.



Pandas를 이용한 데이터 전처리

CSV (comma separated variables)

- CSV 파일은 필드를 나타내는 열과 레코드를 나타내는 행으로 구성
- 데이터의 중간에 구분자가 포함되어야 한다면 따옴표를 사용하여 필드를 묶어야 함
 - ex) 'Gildong, Hong'이라는 데이터는 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 파일이 잠재적으로 크기가 큰 경우 한 번에 모든 레코드를 읽지 않는 것이 좋다.
 - 현재 행을 읽고, 현재 행을 처리한 후에 삭제하고 다음 행을 가져오는 방식이 필요할 수 있다. 아니면 특정한 크기만큼의 데이터를 읽어서 처리한 뒤에, 다음으로 또 그만큼의 크기를 가져오는 방식을 사용할 수 있다.

Pandas를 이용한 데이터 전처리

CSV (comma separated variables)

- 판다스는 데이터를 처리하고 분석하기 위한 모듈이므로 다양한 종류의 데이터 파일 형식을 지원하지만 CSV로 저장된 데이터를 사용하는 것을 기본으로 삼는다.



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": "춘섬"},
8   "회사": "경기 수원시 팔달구 우만동"
9 }
```

JavaScript Object Notation(JSON) 예시

Pandas를 이용한 데이터 전처리

CSV 데이터 읽기

- 파이썬 모듈 csv는 CSV reader와 CSV writer를 제공한다. 두 객체 모두 파일 핸들을 첫 번째 매개 변수로 사용한다.

```
import csv

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']
...
```

Pandas를 이용한 데이터 전처리

CSV 데이터 읽기

- 헤더를 제거하기 위해 `next()` 함수를 사용한다.

```
import csv
```

```
f = open('d:/data/weather.csv')
```

```
data = csv.reader(f)
```

```
header = next(data)
```

```
for row in data:
```

```
    print(row)
```

```
f.close()
```

```
# CSV 파일을 열어서 f에 저장한다.
```

```
# csv의 reader() 함수를 이용하여 읽는다.
```

```
# 헤더를 제거한다.
```

```
# 반복 루프를 사용하여 데이터를 읽는다.
```

```
# 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']
```

```
...
```

Pandas를 이용한 데이터 전처리

CSV 데이터 추출

- 예시: 평균 풍속 데이터만 추출하여 사용하고 싶을 때, CSV 파일에서 평균 풍속 데이터는 4번째 열에 저장되어 있으므로 리스트에서 row[3]을 찾으면 된다.

	col[0]	col[1]	col[2]	col[3]
	A	B	C	D
1	일시	평균기온	최대풍속	평균풍속
2	2010-08-01	28.7	8.1	3.4
3	2010-08-02	25.2	8.1	3.8
4	2010-08-03	22.1	6.1	2.9
5	2010-08-04	25.3	6.1	4.2
6	2010-08-05	27.2	9.1	5.6
7	2010-08-06	26.8	9.1	8.1

```
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()
```

CSV 파일을 열어서 f에 저장한다.

reader() 함수를 이용하여 읽는다.

헤더를 제거한다.

반복 루프를 사용하여 데이터를 읽는다.

평균풍속만 출력하고, 쉼표로 연결한다.

파일을 닫는다.

```
3.4,3.8,2.9,4.2,5.6,8.1,5.4,3.1,5.5,4.8,2.6,4.6,4.4,10.3,3.2,1.6,2.1,1.9,3.2,4.2,2.5,6.2,3,1.9,2.5,1.6,2.3,4.9,6.2,4.2,2.6,5.3,1.7,3.2,3.3,4.3,7.6,6.6,2.5,7.2,3.8,1.8,3.9,1.6,2.2,1.2,2.2,3,3.5,2.8,3.6,7.9,5.8,4.1,6.1,1.8,2.8,5.6,2.1,2.2,3.3,3.2,5.9,5.1,3.1,3.4,3.7,2.7,2.6,3.1,2.5,5,3.2,2.9,4.5,2.9,2.9,2.1,3.9,6.3,3.9,2,3,6.1,7.1,4,3.5,5.8,6.6,7.2,5.6,3.5,3.2,2.9,3.2,3.3,2.5,7.5 ...
```

Pandas를 이용한 데이터 전처리

CSV 데이터 추출

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

	A	B	C	D
1	일시	평균기온	최대풍속	평균풍속
2	2010-08-01	28.7	8.3	3.4
3	2010-08-02	25.2	8.7	3.8
4	2010-08-03	22.1	6.3	2.9
5	2010-08-04	25.3	6.6	4.2
6	2010-08-05	27.2	9.1	5.6
7	2010-08-06	26.8	9.8	8

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

for row in data:
    if row[3] == '' :
        wind = 0
    else :
        wind = float(row[3])
    if max_wind < wind :
        max_wind = wind

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

지난 10년간 울릉도의 최대 풍속은 14.9 m/s

Pandas를 이용한 데이터 전처리

판다스의 데이터 구조: 시리즈와 데이터프레임

- 판다스는 데이터 저장을 위하여 다음과 같은 2가지의 기본 데이터 구조를 제공하고 있다.
- 데이터 구조는 모두 넘파이 배열을 이용하여 구현된다. 모든 데이터 구조는 값을 변경할 수 있으며, 시리즈를 제외하고는 크기도 변경할 수 있다. 각 행과 열은 이름이 부여되며, 행의 이름을 인덱스, 열의 이름을 컬럼스라 부른다.

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

Pandas를 이용한 데이터 전처리

판다스의 데이터 구조: 시리즈와 데이터프레임

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
```

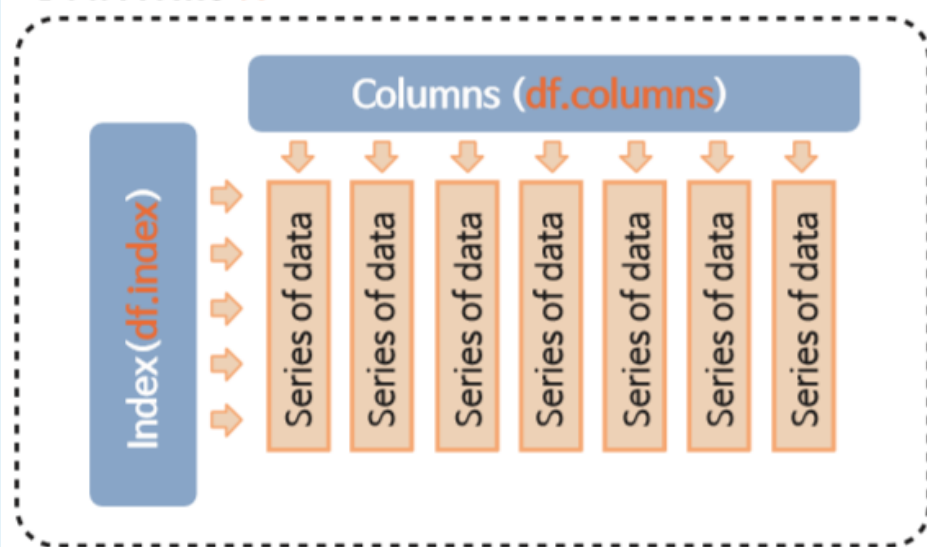
```
4    6.0
```

```
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

DataFrame df



Pandas를 이용한 데이터 전처리

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

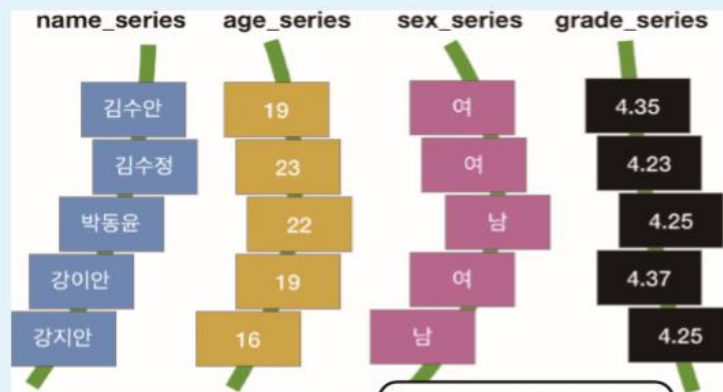
```
0 여
1 여
2 남
3 여
4 남
```

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



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

Pandas를 이용한 데이터 전처리

판다스의 데이터 구조: 시리즈와 데이터프레임

딕셔너리 형식의 데이터로
데이터 프레임을 생성함

```
>>> 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



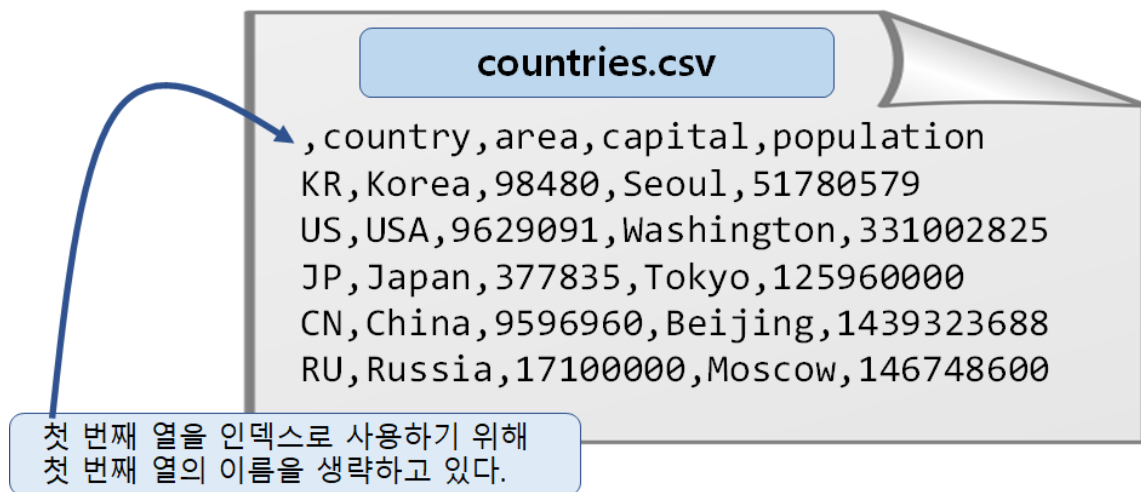
판다스의 DataFrame 클래스를
사용해서 하나의 데이터 프레임을
만들 수 있다

Pandas를 이용한 데이터 전처리

판다스로 데이터 읽기

- 판다스 모듈은 csv 파일을 읽어들이어서 데이터프레임으로 바꾸는 작업을 간단히 할 수 있다.
- CSV 파일이 데이터프레임이 될 수 있도록 각 행이 같은 구조로 되어 있고, 각 열은 동일한 자료형을 가진 시리즈로 되어 있어야 한다.

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



Pandas를 이용한 데이터 전처리

판다스로 데이터 읽기

- 판다스 모듈은 csv 파일을 읽어들이어서 데이터프레임으로 바꾸는 작업을 간단히 할 수 있다.
- CSV 파일이 데이터프레임이 될 수 있도록 각 행이 같은 구조로 되어 있고, 각 열은 동일한 자료형을 가진 시리즈로 되어 있어야 한다.

```
>>> 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

Pandas를 이용한 데이터 전처리

인덱스와 컬럼스 객체

- 데이터 프레임에서는 인덱스와 컬럼스 객체를 정의하여 사용한다. 인덱스는 행들의 레이블이고 컬럼스는 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**

Pandas를 이용한 데이터 전처리

인덱스와 컬럼스 객체

- 데이터 프레임에서는 인덱스와 컬럼스 객체를 정의하여 사용한다. 인덱스는 행들의 레이블이고 컬럼스는 열들의 레이블이 저장된 객체이다.

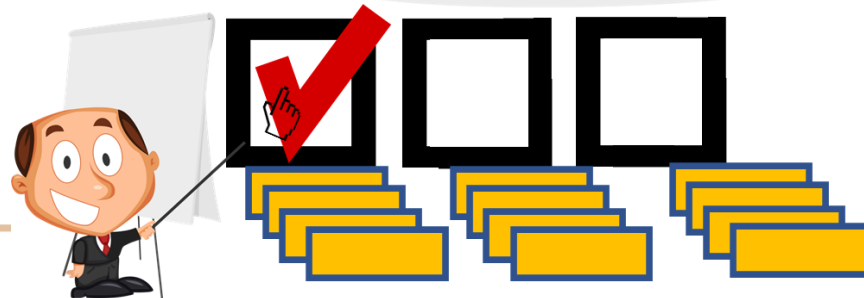
```
import pandas as pd
```

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

첫 번째 열을 인덱스로 사용하겠다는 뜻.

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

	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



Pandas를 이용한 데이터 전처리

인덱스와 컬럼스 객체

- 특정한 열만 선택하려면 아래와 같이 대괄호 안에 열의 이름을 넣으면 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

Pandas를 이용한 데이터 전처리

인덱스와 컬럼스 객체

- 특정한 열만 선택하려면 아래와 같이 대괄호 안에 열의 이름을 넣으면 된다.

```
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

Pandas를 이용한 데이터 전처리

슬라이싱을 통한 행 선택

- 데이터 프레임 중에서 몇 개의 행을 가져오고자 할 때 몇 가지의 방법이 있다. 처음 5행만 얻으려면 `head()`를 사용할 수 있고, 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

Pandas를 이용한 데이터 전처리

슬라이싱을 통한 행 선택

- 다른 예시

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

행의 레이블이 'KR'인 행만을 선택
하기

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

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

데이터프레임에서 특정한 요소 하나만을
선택하려면 loc함수에 행과 열의 레이블을 써주면 된다

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


Pandas를 이용한 데이터 전처리

새로운 열 생성

- 예시: 데이터프레임에 인구 밀도를 나타내는 열 생성

```
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

Pandas를 이용한 데이터 전처리

데이터 분석 함수

- 데이터 프레임이 저장한 데이터를 간단히 분석하려면 describe() 함수를 호출한다.

```
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

Pandas를 이용한 데이터 전처리

데이터 분석 함수

- 데이터 프레임이 저장한 데이터를 간단히 분석하려면 describe() 함수를 호출한다.

```
...  
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
```

Pandas를 이용한 데이터 전처리

데이터 분석 함수

- 데이터 집계 분석

```
>>> 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
```

Pandas를 이용한 데이터 전처리

데이터 분석 함수

- 데이터 집계 분석

```
>>> 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
```

Pandas를 이용한 데이터 전처리

데이터 그룹핑

- groupby() 함수에 그룹을 묶을 때에 사용될 열의 레이블을 인자로 전달하면, 해당 열에 있는 데이터가 동일하면 하나의 그룹으로 묶인다.

```
...
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

```
>>> weather.tail()
```

	일시	평균기온	최대풍속	평균풍속	month
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() 함수를 통해서
weather['month'] 열이 새롭게
추가되었음

Pandas를 이용한 데이터 전처리

데이터 그룹핑

- `groupby()` 함수에 그룹을 묶을 때에 사용될 열의 레이블을 인자로 전달하면, 해당 열에 있는 데이터가 동일하면 하나의 그룹으로 묶인다.

```
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

Pandas를 이용한 데이터 전처리

데이터 필터링

- 예시: 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

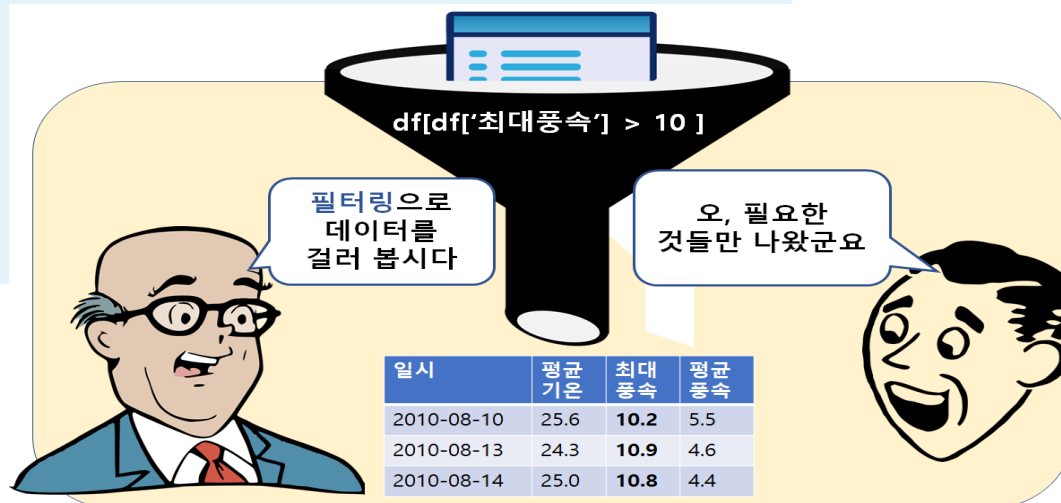
...

2020-07-29 False

2020-07-30 False

2020-07-31 False

넘파이의 논리 인덱싱과
동일한 문법



Pandas를 이용한 데이터 전처리

데이터 필터링

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

```
>>> 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

Pandas를 이용한 데이터 전처리

데이터 결손값 처리

- 실제 데이터는 완벽하지 않고 상당한 수의 결손값을 가지고 있거나 의심스러운 값을 가지고 있다. 이는 데이터가 아예 수집되지 않았거나, 측정 장치의 고장, 사건 사고 등으로 데이터를 확보할 수 없는 경우 때문이다.
- 데이터를 처리하기 전에 반드시 데이터 정제 과정을 거쳐야 한다. 판다스에서는 결손값을 NaN으로 나타나며, 이를 탐지하고 수정하는 함수를 제공한다.

Pandas를 이용한 데이터 전처리

데이터 결손값 처리

- isna() 함수를 통해 데이터에 결손값이 있는지 확인한다.

```
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

Pandas를 이용한 데이터 전처리

데이터 결손값 처리

- 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'
```

Pandas를 이용한 데이터 전처리

데이터 결손값 처리

- 빠진 값을 새로운 값으로 채우기
 - fillna() 함수를 이용하여 결손값을 채울 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
```

Pandas를 이용한 데이터 전처리

데이터 결손값 처리

- 빠진 값을 새로운 값으로 채우기
 - fillna() 함수를 이용하여 결손값을 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

Pandas를 이용한 데이터 전처리

데이터 구조 변경

- 딕셔너리 데이터를 이용하여 데이터프레임을 생성할 수 있다. 이때 키는 열의 레이블이 되고, 딕셔너리의 키에 딸린 값은 열을 채우는 데이터를 가진 리스트가 된다.

```
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

Pandas를 이용한 데이터 전처리

데이터 구조 변경

- 예시: pivot() 함수

```
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
```

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

Pandas를 이용한 데이터 전처리

데이터 구조 변경


- concat() 함수를 통해 데이터 합치기

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_1			
	A	B	C
가	a10	b10	c10
나	a11	b11	c11
다	a12	b12	c12

인덱스

df_2			
	B	C	D
다	b23	c23	d23
라	b24	c24	d24
마	b25	c25	d25

인덱스

Pandas를 이용한 데이터 전처리

데이터 구조 변경

- concat() 함수를 통해 데이터 합치기

```
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_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

Pandas를 이용한 데이터 전처리

데이터 정렬

- `sort_values()`를 사용한 데이터 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)
```

Pandas를 이용한 데이터 전처리

데이터 정렬

- `sort_values()`를 사용한 데이터 정렬

```
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