48.(심화)Pandas

2019.07

상병 김재형

시작하기 전에

본 강의는 "파이썬으로 데이터 주무르기", 2017, 민형기, 비제이퍼블릭을 참고

- ※ 원래는 새로 만들려 하였으나...

pandas

Pandas란?

- 데이터 분석 라이브러리
- 행과 열로 이루어진 데이터 객체를 만들 수 있다.
  ※ DB와 유사하나, SQL로 값을 변경하지 않음
- 엑셀이라고 생각하면 편함

pandas-기초

import

- pandas를 임포트
- 일반적으로 pandas는 pd로 별칭을 지음
- numpy와 같이 사용되는 경우가 많음
- numpy는 np로 별칭

```
In [15]: import pandas as pd
         import numpy as np
```

https://ko.wikipedia.org/wiki/%EC%BB%AC%EB%9F%BC_(%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4)

pandas-기초

column과 row(데이터베이스)

- 컬럼(column)
  - 각 열이 어떠한 값을 갖는지 알려줌

```
In [4]: CCTV_Seoul = pd.read_csv('Seoul_CCTV.csv', encoding='utf-8')
        CCTV_Seoul.head()
```

Out [4]:

	기관명	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

column

row

https://ko.wikipedia.org/wiki/%EB%A1%9C%EC%9A%B0_(%EB%8D%B0%EC%9D%B4%ED%84%B0%EB%B2%A0%EC%9D%B4%EC%8A%A4)

pandas-기초

column과 row(데이터베이스)

- 행(row)
  - 단일 구조 데이터 항목
  - column의 값이 입력됨. ex) 회사

```
In [4]: CCTV_Seoul = pd.read_csv('Seoul_CCTV.csv', encoding='utf-8')
        CCTV_Seoul.head()
```

Out [4]:

column

row

	기관명	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

https://datascienceschool.net/view-notebook/8959673a97214e8fafdb159f254185e9/

pandas-기초

날짜형 사용

- pd.date_range(시작일, 종료일 or 기간, [freq])
  - 시작일에서 종료일 또는 기간으로 인덱스를 생성
  - Freq로 특정한 날짜만 생성하도록 할 수 있음
    (s: 초, T: 분, D:일, W: 주(일요일), M: 각 달의 마지막 날 등)

```
In [17]: pd.date_range("2019-07-01", "2019-07-07")

Out[17]: DatetimeIndex(['2019-07-01', '2019-07-02', '2019-07-03', '2019-07-04',
                        '2019-07-05', '2019-07-06', '2019-07-07'],
                       dtype='datetime64[ns]', freq='D')
```

```
In [19]: dates = pd.date_range("2019-07-01", periods=6)
         dates

Out[19]: DatetimeIndex(['2019-07-01', '2019-07-02', '2019-07-03', '2019-07-04',
                        '2019-07-05', '2019-07-06'],
                       dtype='datetime64[ns]', freq='D')
```

pandas-기초

pandas객체(Series 객체)

- 1차원 배열과 같은 자료구조
- 인덱스가 있음
- 한 개의 column에 해당하는 데이터
- Pandas객체의 기초
- NaN(Not A Number)

```
In [16]: pd.Series([1, 2, 5, np.nan, 6, 8])

Out[16]: 0    1.0
         1    2.0
         2    5.0
         3    NaN
         4    6.0
         5    8.0
         dtype: float64
```

https://eunguru.tistory.com/220?category=817455

pandas-기초

pandas객체(DataFrame)

- pandas에서 사용하는 기본 자료구조
- column이 모여 만들어짐
- column은 서로 다른 데이터 타입을 가질 수 있음
- 2차원 배열로 볼 수 있다.

```
In [5]: CCTV_Seoul?
```

```
Type:        DataFrame
String form:
기관명    소계  2013년도 이전  2014년  2015년  2016년
          0    강남구  3238       1292    430    584    932
          1   <...> 630
          23     중구  1023       413    190     72    348
          24    중랑구   916       509    121    177    109
Length:      25
```

pandas-기초

https://nittaku.tistory.com/443

pandas객체(DataFrame)

- np.random.randn(m, n)
  - 평균이 0, 표준편차가 1인 가우시안 정규분포 난수
  - m, n matrix array(행렬)로 생성
  - Index와 columns도 지정 가능

In [20]:
```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=['A', 'B', 'C', 'D'])
df
```

Out [20]:

	A	B	C	D
2019-07-01	-0.271590	0.893861	0.158598	0.010475
2019-07-02	-1.858550	0.771038	-0.185653	-1.654077
2019-07-03	-0.139784	-1.054661	0.079073	1.057681
2019-07-04	2.298534	-0.273554	-0.654312	0.247948
2019-07-05	-0.540385	-1.273276	1.237219	-0.347435
2019-07-06	0.230773	1.264430	0.302012	0.796698

pandas-기초

pandas객체(Series, DataFrame)

- 변수명만 입력하면, 전체가 나타남

In [20]:
```
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=['A', 'B', 'C', 'D'])
df
```

Out [20]:

	A	B	C	D
2019-07-01	-0.271590	0.893861	0.158598	0.010475
2019-07-02	-1.858550	0.771038	-0.185653	-1.654077
2019-07-03	-0.139784	-1.054661	0.079073	1.057681
2019-07-04	2.298534	-0.273554	-0.654312	0.247948
2019-07-05	-0.540385	-1.273276	1.237219	-0.347435
2019-07-06	-0.230773	-1.364439	-0.302013	-0.796698

pandas-기초

pandas객체(Series, DataFrame)

- 리스트와 유사하게 슬라이싱이 가능
  - Column을 지정하여 해당 column만 Series로 나타낼 수 있음
  - 행의 범위를 지정하여 원하는 행만 볼 수 있음

```
In [40]: df['A']
Out[40]: 2019-07-01   -0.271590
         2019-07-02   -1.858550
         2019-07-03   -0.139784
         2019-07-04    2.298534
         2019-07-05   -0.540385
         2019-07-06   -0.230773
         Freq: D, Name: A, dtype: float64
```

```
In [41]: df[0:3]
Out[41]:
```

	A	B	C	D
2019-07-01	-0.271590	0.893861	0.158598	0.010475
2019-07-02	-1.858550	0.771038	-0.185653	-1.654077
2019-07-03	-0.139784	-1.054661	0.079073	1.057681

pandas-기초

pandas객체(Series, DataFrame)

- 리스트와 유사하게 슬라이싱이 가능
  - Index가 날짜일 경우, 날짜값을 입력해 슬라이싱이 가능

```
In [42]: df['20190703':'20190705']
```

Out [42]:

	A	B	C	D
2019-07-03	-0.139784	-1.054661	0.079073	1.057681
2019-07-04	2.298534	-0.273554	-0.654312	0.247948
2019-07-05	-0.540385	-1.273276	1.237219	-0.347435

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
  - 인덱싱 값
    - 인덱스데이터
    - 인덱스데이터 슬라이스
    - 인덱스데이터 리스트
    - 같은 행 인덱스를 가지는 블리언 시리즈(행 인덱싱)
    - 위의 값을 반환하는 함수

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
    - 인덱스데이터

```
In [43]: df.loc[dates[0]]

Out[43]: A   -0.271590
         B    0.893861
         C    0.158598
         D    0.010475
         Name: 2019-07-01 00:00:00, dtype: float64
```

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
    - 인덱스데이터와 인덱스데이터 리스트

```
In [50]: df.loc['20190704',['A','B']]

Out[50]: A     2.298534
         B    -0.273554
         Name: 2019-07-04 00:00:00, dtype: float64
```

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
    - 인덱스데이터 슬라이스와 데이터 리스트

```
In [48]: df.loc[:, ['A','B']]
```

Out[48]:

	A	B
2019-07-01	-0.271590	0.893861
2019-07-02	-1.858550	0.771038
2019-07-03	-0.139784	-1.054661
2019-07-04	2.298534	-0.273554
2019-07-05	-0.540385	-1.273276
2019-07-06	-0.230773	-1.364439

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
    - 같은 행 인덱스를 가지는 블리언 시리즈(행 인덱싱)

```
In [37]: df
```

Out [37]:

	A			
2019-07-01	-1.018053	0.832		
2019-07-02	0.515277	-0.603		
2019-07-03	0.024126	0.179		
2019-07-04	-1.335043	-0.240		
2019-07-05	0.470217	0.566		
2019-07-06	1.781308	1.002232	1.069235	-0.083919

```
In [36]: df.loc[df.A > 0]
```

Out [36]:

	A	B	C	D
2019-07-02	0.515277	-0.603213	-1.343303	-0.716284
2019-07-03	0.024126	0.179468	0.135117	-0.715872
2019-07-05	0.470217	0.566410	-0.236959	1.591827
2019-07-06	1.781308	1.002232	1.069235	-0.083919

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe loc인덱서

- loc인덱서를 통해 원하는 대로 슬라이싱 가능
  - df.loc[행 인덱싱 값] 또는 df.loc[행 인덱싱 값, 열 인덱싱 값]
    - 같은 행 인덱스를 가지는 블리언 시리즈(행 인덱싱)
    - 값이 없을 경우, NaN으로 표시
    - loc인덱서가 없어도 됨

```
In [43]: df[df > 0]
```

Out [43]:

	A	B	C	D
2019-07-01	NaN	0.832888	1.082031	0.772655
2019-07-02	0.515277	NaN	NaN	NaN
2019-07-03	0.024126	0.179468	0.135117	NaN
2019-07-04	NaN	NaN	NaN	0.047769
2019-07-05	0.470217	0.566410	NaN	1.591827
2019-07-06	1.781308	1.002232	1.069235	NaN

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe iloc인덱서

- 행과 열의 번호를 이용해서 데이터에 바로 접근

```
In [38]: df.iloc[3]

Out[38]: A   -1.335043
         B   -0.240084
         C   -0.977388
         D    0.047769
         Name: 2019-07-04 00:00:00, dtype: float64
```

https://datascienceschool.net/view-notebook/704731b41f794b8ea00768f5b0904512/

pandas-기초

Dataframe iloc인덱서

- 리스트와 같은 방식으로 행이나 열의 범위 지정가능

```
In [42]: df.iloc[1:3, :]
```

Out [42]:

	A	B	C	D
2019-07-02	0.515277	-0.603213	-1.343303	-0.716284
2019-07-03	0.024126	0.179468	0.135117	-0.715872

```
In [40]: df.iloc[[1, 3], [0, 2]]
```

Out [40]:

	A	C
2019-07-02	0.515277	-1.343303
2019-07-04	-1.335043	-0.977388

pandas-기초

DataFrame.head(n)메서드

- 상위 n개의 행을 출력
- 기본값은 5

```
In [21]: df.head(3)
```

Out[21]:

	A	B	C	D
2019-07-01	-0.271590	0.893861	0.158598	0.010475
2019-07-02	-1.858550	0.771038	-0.185653	-1.654077
2019-07-03	-0.139784	-1.054661	0.079073	1.057681

pandas-기초

DataFrame.index

- 각 행의 제목을 가진 객체
- 지정하지 않으면 0부터 시작하도록 자동으로 정해줌
- 리스트처럼 인덱스로 값을 불러올 수 있음

```
In [27]: df.index

Out[27]: DatetimeIndex(['2019-07-01', '2019-07-02', '2019-07-03', '2019-07-04',
                        '2019-07-05', '2019-07-06'],
                       dtype='datetime64[ns]', freq='D')

In [28]: df.index[0]

Out[28]: Timestamp('2019-07-01 00:00:00', freq='D')
```

pandas-기초

DataFrame.columns

- column들의 제목을 가진 객체
- 리스트처럼 인덱스로 값을 가져올 수 있음

```
In [23]: df.columns
Out[23]: Index(['A', 'B', 'C', 'D'], dtype='object')

In [29]: df.columns[1]
Out[29]: 'B'
```

pandas-기초

DataFrame.values

- DataFrame이 가진 값을 가진 객체
- 인덱스로 값을 슬라이싱 하거나 가져올 수 있음

```
In [30]: df.values

Out[30]: array([[-0.27158975,  0.89386054,  0.15859787,  0.01047534],
                [-1.85854974,  0.77103782, -0.18565305, -1.65407709],
                [-0.13978376, -1.05466071,  0.0790734 ,  1.05768149],
                [ 2.29853446, -0.2735535 , -0.65431186,  0.24794802],
                [-0.54038503, -1.27327576,  1.23721884, -0.34743548],
                [-0.23077274, -1.36443932, -0.30201261, -0.79669782]])

In [32]: df.values[1]

Out[32]: array([-1.85854974,  0.77103782, -0.18565305, -1.65407709])

In [33]: df.values[1][1]

Out[33]: 0.7710378206371521
```

pandas-기초

DataFrame.info()메서드

- DataFrame의 개요를 볼 수 있음

```
In [35]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6 entries, 2019-07-01 to 2019-07-06
Freq: D
Data columns (total 4 columns):
A    6 non-null float64
B    6 non-null float64
C    6 non-null float64
D    6 non-null float64
dtypes: float64(4)
memory usage: 240.0 bytes
```

pandas-기초

DataFrame.describe()메서드

- 통계적 개요 확인
  - 개수, 평균, 최소, 최대, 표준편차, 1/4지점
  - 문자이면, 문자에 맞는 개요가 나타남

```
In [36]: df.describe()
```

Out [36]:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.123758	-0.383505	0.055485	-0.247018
std	1.347898	1.017948	0.647680	0.927964
min	-1.858550	-1.364439	-0.654312	-1.654077
25%	-0.473186	-1.218622	-0.272923	-0.684382
50%	-0.251181	-0.664107	-0.053290	-0.168480
75%	-0.162531	0.509890	0.138717	0.188580
max	2.298534	0.893861	1.237219	1.057681

pandas-기초

DataFrame.sort_values(by, [ascending])메서드

- by로 지정된 컬럼을 기준으로 정렬
- ascending 기본값은 True로 오름차순으로 정렬

```
In [39]: df.sort_values(by='B', ascending=False)
```

Out[39]:

	A	B	C	D
2019-07-01	-0.271590	0.893861	0.158598	0.010475
2019-07-02	-1.858550	0.771038	-0.185653	-1.654077
2019-07-04	2.298534	-0.273554	-0.654312	0.247948
2019-07-03	-0.139784	-1.054661	0.079073	1.057681
2019-07-05	-0.540385	-1.273276	1.237219	-0.347435
2019-07-06	-0.230773	-1.364439	-0.302013	-0.796698

https://3months.tistory.com/292

pandas-기초

DataFrame.drop(라벨, [axis=0], [inplace=False])

- 행이나 열을 삭제
- axis가 0이면 행, 1이면 열

```
In [223]: df.drop(df.index[1],inplace=True)
```

```
In [214]: df
```

Out [214]:

	A	B	C	D
2019-07-01	0.955438	-0.402158	1.429367	0.555802
2019-07-02	-0.248572	-0.872710	-1.738065	0.639109
2019-07-03	-2.087119	-0.394603	0.503518	2.049374
2019-07-04	-0.736823	-0.885039	-0.899339	-0.418095
2019-07-05	0.812940	0.935347	0.307478	-1.692276
2019-07-06	-0.288200	0.533767	1.128501	-0.998093

```
df
```

	A	B	C	D
2019-07-01	0.955438	-0.402158	1.429367	0.555802
2019-07-03	-2.087119	-0.394603	0.503518	2.049374
2019-07-04	-0.736823	-0.885039	-0.899339	-0.418095
2019-07-05	0.812940	0.935347	0.307478	-1.692276
2019-07-06	-0.288200	0.533767	1.128501	-0.998093

pandas-기초

DataFrame.copy()

- '='으로 새 변수명 지정 시, 메모리 주소만 할당
- 내장 메서드 copy를 통해 복사 가능

```
In [44]: df2 = df.copy()
```

```
In [45]: df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
```

```
In [46]: df
```

Out [46]:

	A	B	C	D
2019-07-01	-1.018053	0.832888	1.082031	0.772655
2019-07-02	0.515277	-0.603213	-1.343303	-0.716284
2019-07-03	0.024126	0.179468	0.135117	-0.715872
2019-07-04	-1.335043	-0.240084	-0.977388	0.047769
2019-07-05	0.470217	0.566410	-0.236959	1.591827
2019-07-06	1.781308	1.002232	1.069235	-0.083919

```
47]: df2
```

47]:

	A	B	C	D	E
2019-07-01	-1.018053	0.832888	1.082031	0.772655	one
2019-07-02	0.515277	-0.603213	-1.343303	-0.716284	one
2019-07-03	0.024126	0.179468	0.135117	-0.715872	two
2019-07-04	-1.335043	-0.240084	-0.977388	0.047769	three
2019-07-05	0.470217	0.566410	-0.236959	1.591827	four
2019-07-06	1.781308	1.002232	1.069235	-0.083919	three

pandas-기초

Series.isin(데이터)

- Series의 값이 데이터 값과 같은지 확인하여 반환

```
In [51]: df2['E'].isin(['two','four'])

Out[51]: 2019-07-01    False
         2019-07-02    False
         2019-07-03     True
         2019-07-04    False
         2019-07-05     True
         2019-07-06    False
         Freq: D, Name: E, dtype: bool
```

pandas-기초

Series.isin(데이터)

- 이를 통해 원하는 값이 있는 행만 추출 가능
  - 같은 행 인덱스를 가지는 블리언 시리즈(행 인덱싱)

```
In [52]: df2[df2['E'].isin(['two','four'])]
```

Out[52]:

	A	B	C	D	E
2019-07-03	0.024126	0.179468	0.135117	-0.715872	two
2019-07-05	0.470217	0.566410	-0.236959	1.591827	four

pandas-기초

DataFrame.apply(함수)

- 함수를 적용하여 값을 바꿈
- np.cumsum: 누적합

```
In [53]: df
```

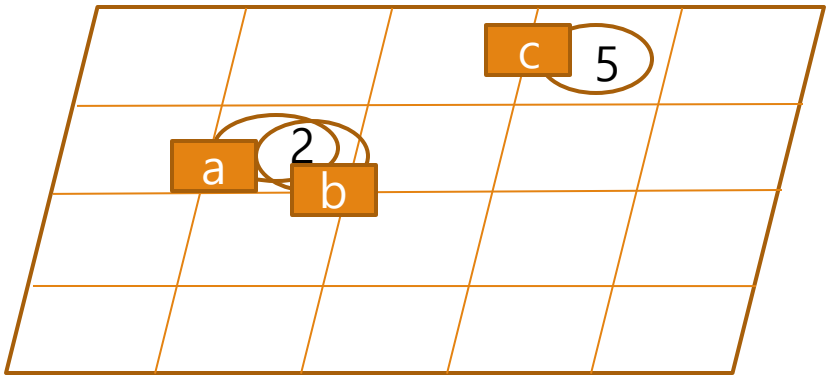
Out [53]:

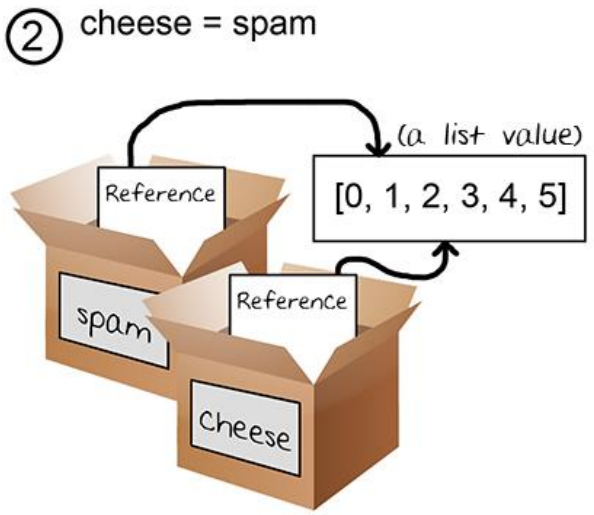
	A	B	C	D
2019-07-01	-1.018053	0.832888	1.082031	0.772655
2019-07-02	0.515277	-0.603213	-1.343303	-0.716284
2019-07-03	0.024126	0.179468	0.135117	-0.715872
2019-07-04	-1.335043	-0.240084	-0.977388	0.047769
2019-07-05	0.470217	0.566410	-0.236959	1.591827
2019-07-06	1.781308	1.002232	1.069235	-0.083919

```
df.apply(np.cumsum)
```

	A	B	C	D
2019-07-01	-1.018053	0.832888	1.082031	0.772655
2019-07-02	-0.502775	0.229675	-0.261271	0.056371
2019-07-03	-0.478650	0.409143	-0.126155	-0.659501
2019-07-04	-1.813692	0.169059	-1.103543	-0.611732
2019-07-05	-1.343475	0.735469	-1.340502	0.980095
2019-07-06	0.437833	1.737701	-0.271267	0.896175

Review-변수의 생성-Python





이름 객체

a → 2

서울시 구별 CCTV 현황 분석

목표

- 서울시 내 구별 CCTV의 대수 확인
- 인구 대비 비율 확인

데이터 구하기

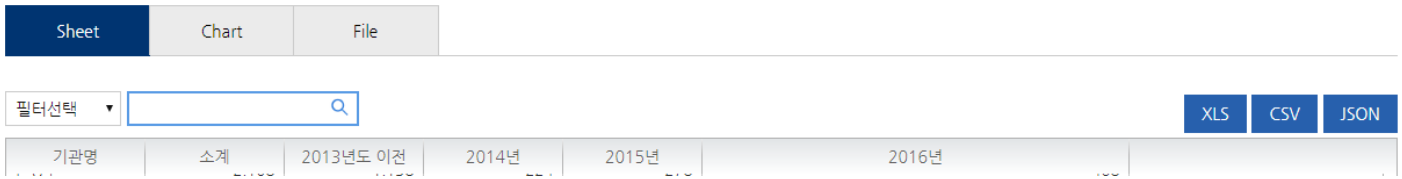
CCTV 데이터

- 최근 공공데이터가 많이 공개되고 있다.
- 최근 수정일자는 2018.06

데이터 구하기

CCTV 데이터

- 접속하여 데이터를 다운로드 받는다.
- XLS: 엑셀
- CSV(comma-seperated value)
  각 필드(나눠진)내용을 ","로 분리된 데이터
- JSON
  데이터를 저장할 수 있는 포맷



데이터 구하기

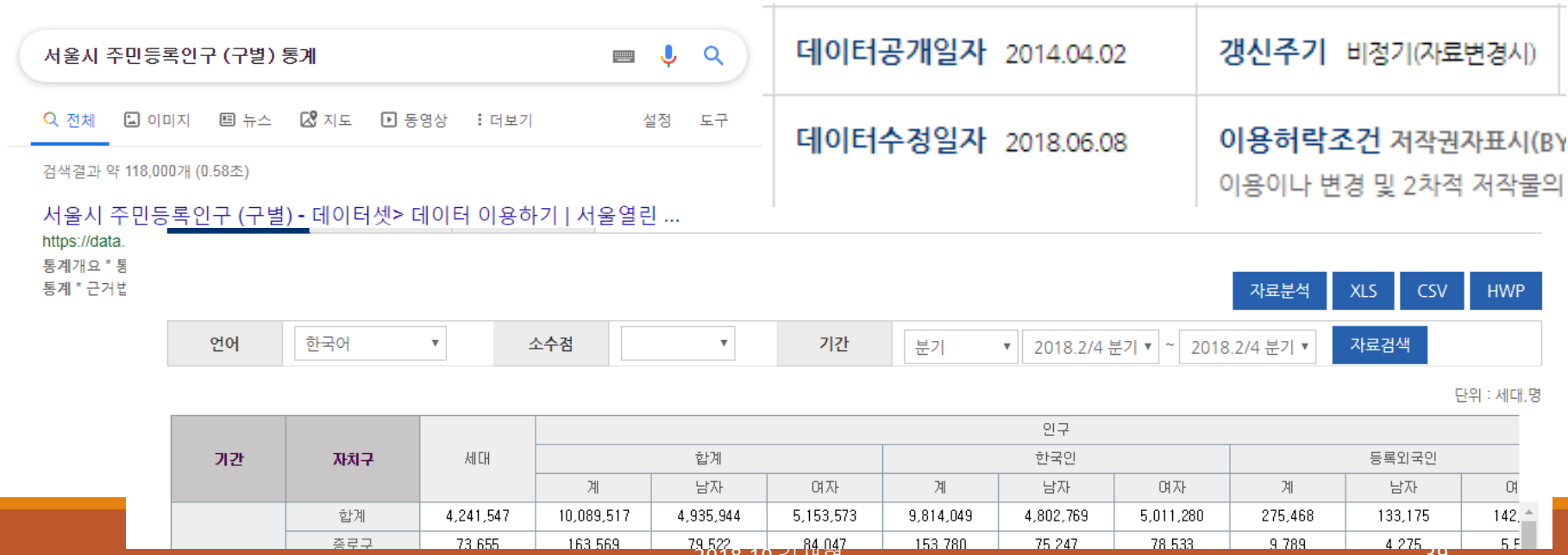
CCTV 데이터

- CSV로 다운로드
- TeachingMaterials/insert_file/48 lecture file에 두었음
- 현재, CSV로 다운로드 받을 경우
  값이 2016년까지만 다운로드됨
- 수정된 파일을 올려 둠

데이터 구하기

서울시 구별 주민등록 인구

- CCTV가 2018년 6월 자료 => 같은 기간 다운로드
- 기간을 변경하고 다운로드
- xls로 다운로드 한다.



데이터 구하기

서울시 구별 주민등록 인구

- txt파일일 경우
- 내부 필드값이 tap으로 구분
- tsv(tap-seperated value) 파일이라고 한다.
- ','로 자리수를 나타내어서 tap으로 구분

pandas

csv파일 불러오기

- 한글을 불러옴으로 encoding에 주의한다.
- ※ 24.파일 입출력을 참고

In [4]:
```
CCTV_Seoul = pd.read_csv('Seoul_CCTV.csv', encoding='utf-8')
CCTV_Seoul.head()
```

Out[4]:

	기관명	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	3238	1292	430	584	932
1	강동구	1010	379	99	155	377
2	강북구	831	369	120	138	204
3	강서구	911	388	258	184	81
4	관악구	2109	846	260	390	613

pandas

csv파일 불러오기

- DataFrame.columns
  - 리스트처럼 인덱스로 접근 가능

```
In [7]: CCTV_Seoul.columns
Out[7]: Index(['기관명', '소계', '2013년도 이전', '2014년', '2015년', '2016년'], dtype='objec
        t')
```

```
In [8]: CCTV_Seoul.columns[0]
Out[8]: '기관명'
```

pandas

xls파일 불러오기

- Anaconda를 설치한 것이 아니여서 모듈 error발생
- xlrd를 설치
- pip3 install xlrd 사용
- ※ 이미 설치하여, 구름IDE에서 진행할 필요 없음

```
In [12]: pop_Seoul = pd.read_excel('Seoul.Population.xls', encoding='utf-8')
         pop_Seoul.head()
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/usr/local/lib/python3.6/site-packages/pandas/io/excel.py in __init__(self, io
s)
    351         try:
--> 352             import xlrd
    353         except ImportError:

ModuleNotFoundError: No module named 'xlrd'
```

pandas

xls파일 불러오기

- 확인해보니 첫 행이 이상
- 이는 3개의 행을 제목으로 사용하였기 때문
- 파일을 열 때, 옵션을 걸어서 원하는 것만 읽도록 함

```
In [9]: pop_Seoul = pd.read_excel('Seoul_Population.xls', encoding='utf-8')
        pop_Seoul.head()
```

Out[9]:

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계
2	2018.2/4	합계	4241547	10089517	4935944	5153573	9814049	4802769	5011280	275468

pandas

xls파일 불러오기

- 필요한 항목
  - 3번째 행을 제목으로 함
  - 전체 인구, 한국인 인구, 외국인 인구, 65세 고령자
  - B열, D열, G열, J열, N열

A	B	C	D	E	F	G	H	I	J	K	L	M	N
기간	자치구	세대	인구									세대당인구	5세이상고령자
			합계			한국인			등록외국인				
			계	남자	여자	계	남자	여자	계	남자	여자		
	합계	4,241,547	10,089,517	4,935,944	5,153,573	9,814,049	4,802,769	5,011,280	275,468	133,175	142,293	2.31	1,393,671
	종로구	73,655	163,569	79,522	84,047	153,780	75,247	78,533	9,789	4,275	5,514	2.09	26,512
	중구	61,091	135,427	66,673	68,754	126,032	62,260	63,772	9,395	4,413	4,982	2.06	21,798
	용산구	108,516	245,245	119,963	125,282	229,677	111,078	118,599	15,568	8,885	6,683	2.12	37,331

pandas

xls파일 불러오기

- Header옵션, usecols옵션
  - header(첫 행은 0)
  - usecols(최근 변경, 예전에는 parse_cols)

```
In [13]: pop_Seoul = pd.read_excel('Seoul_Population.xls',
                                   header=2,
                                   usecols='B, D, G, J, N',
                                   encoding='utf-8')
         pop_Seoul.head()
```

Out[13]:

	자치구	계	계.1	계.2	65세이상고령자
0	합계	10089517	9814049	275468	1393671
1	종로구	163569	153780	9789	26512
2	중구	135427	126032	9395	21798
3	용산구	245245	229677	15568	37331
4	성동구	316068	308066	8002	42171

pandas

xls파일 불러오기

- column이름 변경
  - rename을 통해 구별, 인구수, 한국인, 외국인, 고령자로 변경

In [14]:
```
pop_Seoul.rename(columns={pop_Seoul.columns[0] : '구별',
                          pop_Seoul.columns[1] : '인구수',
                          pop_Seoul.columns[2] : '한국인',
                          pop_Seoul.columns[3] : '외국인',
                          pop_Seoul.columns[4] : '고령자',}, inplace=True)
pop_Seoul.head()
```

Out [14]:

	구별	인구수	한국인	외국인	고령자
0	합계	10089517	9814049	275468	1393671
1	종로구	163569	153780	9789	26512
2	중구	135427	126032	9395	21798
3	용산구	245245	229677	15568	37331
4	성동구	316068	308066	8002	42171

pandas

CCTV 데이터 확인

- CCTV 전체 개수인 소계로 정렬
- CCTV가 가장 적은 구 별로 볼 수 있음

```
In [155]: CCTV_Seoul.sort_values(by='소계', ascending=True).head(5)
```

Out [155]:

	구별	소계	2015년 이전	2016년	2017년	2018년
24	중랑구	1,068	872	121	66	9
2	강북구	1,265	691	254	1	319
22	종로구	1,471	941	148	281	101
23	중구	1,544	629	270	317	328
5	광진구	1,581	746	21	468	346

pandas

CCTV 데이터 확인

- CCTV 전체 개수인 소계로 정렬
- CCTV가 가장 많은 구 별로 볼 수 있음

```
In [156]: CCTV_Seoul.sort_values(by='소계', ascending=False).head(5)
```

Out[156]:

	구별	소계	2015년 이전	2016년	2017년	2018년
9	도봉구	858	515	155	117	71
0	강남구	5,221	3,431	765	577	448
4	관악구	3,985	2,001	619	694	671
6	구로구	3,227	1,875	326	540	486
16	성북구	3,003	1,687	388	285	643

pandas

CCTV 데이터 확인

- 2016-2018년 간 CCTV증가율
  - 2016-2018년을 더한 뒤, 2015년 이전으로 나눈 뒤 * 100

```
In [210]: CCTV_Seoul['최근증가율'] = (CCTV_Seoul['2016년']+CCTV_Seoul['2017년']+₩
                                  CCTV_Seoul['2018년']/CCTV_Seoul['2015년 이전'] * 100)

In [212]: CCTV_Seoul.sort_values(by='최근증가율', ascending=False).head()
```

Out [212]:

	구별	소계	2015년 이전	2016년	2017년	2018년	최근증가율
0	강남구	5221	3431	765	577	448	1355.057418
4	관악구	3985	2001	619	694	671	1346.533233
15	성동구	2679	1251	201	933	294	1157.501199
6	구로구	3227	1875	326	540	486	891.920000
12	마포구	1935	1009	359	372	195	750.326065

pandas

인구 현황 데이터 확인

- 합계가 있는 것을 볼 수 있음

```
In [213]: pop_Seoul.head()
```

Out [213]:

	구별	인구수	한국인	외국인	고령자
0	합계	10089517	9814049	275468	1393671
1	종로구	163569	153780	9789	26512
2	중구	135427	126032	9395	21798
3	용산구	245245	229677	15568	37331
4	성동구	316068	308066	8002	42171

pandas

인구 현황 데이터 확인

- 합계를 삭제

In [225]:
```
pop_Seoul.drop([0], inplace=True)
pop_Seoul.head()
```

Out [225]:

	구별	인구수	한국인	외국인	고령자
1	종로구	163569	153780	9789	26512
2	중구	135427	126032	9395	21798
3	용산구	245245	229677	15568	37331
4	성동구	316068	308066	8002	42171
5	광진구	370519	355748	14771	44806

https://rfriend.tistory.com/267

pandas

인구 현황 데이터 확인

- 값이 잘 들어가 있는지 확인
- Series.unique(): 유일한 값을 찾아줌

```
In [226]: pop_Seoul['구별'].unique()
Out[226]: array(['종로구', '중구', '용산구', '성동구', '광진구', '동대문구', '중랑구', '성북구', '강북구',
                 '도봉구', '노원구', '은평구', '서대문구', '마포구', '양천구', '강서구', '구로구', '금천구',
                 '영등포구', '동작구', '관악구', '서초구', '강남구', '송파구', '강동구'], dtype=object)
```

https://rfriend.tistory.com/267

pandas

인구 현황 데이터 확인

- Series.unique(): 유일한 값을 찾아줌
- 만일, NaN이 있을 경우 isnull()메서드 사용
  pop_Seoul[pop_Seoul['구별'].isnull()]
- 이후, 인덱스값을 확인하여 drop한다.

https://rfriend.tistory.com/267

pandas

인구 현황 데이터 확인

- Series.value_counts()
- 각 값이 몇 개가 있는지 확인하는 메서드

```
In [227]: pop_Seoul['구별'].value_counts()
Out[227]: 마포구      1
          송파구      1
          양천구      1
          용산구      1
          금천구      1
          강동구      1
          동대문구     1
          종로구      1
          관악구      1
          동작구      1
          강남구      1
          강서구      1
          성동구      1
          강북구      1
          구로구      1
          은평구      1
          광진구      1
          중구       1
          노원구      1
          성북구      1
          중랑구      1
          영등포구     1
          서대문구     1
          도봉구      1
          서초구      1
          Name: 구별, dtype: int64
```

pandas

인구 현황 데이터 확인

- 외국인 비율과 고령자 비율을 작성

In [229]:
```
pop_Seoul['외국인비율'] = pop_Seoul['외국인'] / pop_Seoul['인구수'] * 100
pop_Seoul['고령자비율'] = pop_Seoul['고령자'] / pop_Seoul['인구수'] * 100
pop_Seoul.head()
```

Out [229]:

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
1	종로구	163569	153780	9789	26512	5.984630	16.208450
2	중구	135427	126032	9395	21798	6.937317	16.095756
3	용산구	245245	229677	15568	37331	6.347938	15.221921
4	성동구	316068	308066	8002	42171	2.531734	13.342382
5	광진구	370519	355748	14771	44806	3.986570	12.092767

pandas

인구 현황 데이터 확인

- 총 인구수

```
In [230]: pop_Seoul.sort_values(by='인구수', ascending=False).head()
```

Out [230]:

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
24	송파구	673161	666439	6722	79093	0.998572	11.749492
16	강서구	606981	600257	6724	78042	1.107778	12.857404
11	노원구	553177	549365	3812	75741	0.689110	13.692001
23	강남구	551888	546952	4936	66011	0.894384	11.960941
21	관악구	521960	504048	17912	71317	3.431681	13.663308

pandas

인구 현황 데이터 확인

- 외국인 및 외국인 비율

```
In [233]: pop_Seoul.sort_values(by='외국인', ascending=False).head()
```

Out [233]:

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
19	영등포구	404501	369003	35498	54994	8.775751	13.595516
17	구로구	440305	407235	33070	60564	7.510703	13.755011
18	금천구	252752	233263	19489			
21	관악구	521960	504048	17912			
6	동대문구	364527	348903	15624			

```
In [234]: pop_Seoul.sort_values(by='외국인비율', ascending=False)
```

Out [234]:

	구별	인구수	한국인	외국인	고령자	외국인비율	고
19	영등포구	404501	369003	35498	54994	8.775751	13.
18	금천구	252752	233263	19489	34945	7.710720	13.
17	구로구	440305	407235	33070	60564	7.510703	13
2	중구	135427	126032	9395	21798	6.937317	16.
3	용산구	245245	229677	15568	37331	6.347938	15

pandas

인구 현황 데이터 확인

- 고령자 및 고령자 비율

```
pop_Seoul.sort_values(by='고령자', ascending=False).head()
```

	구별	인구수	한국인	외국인	고령자
24	송파구	673161	666439	6722	79093
16	강서구	606981	600257	6724	78042
12	은평구	489045	484642	4403	76097
11	노원구	553177	549365	3812	75741
21	관악구	521960	504048	17912	71317

```
pop_Seoul.sort_values(by='고령자비율', ascending=False).head()
```

	구별	인구수	한국인	외국인	고령자	외국인비율	고령자비율
9	강북구	326063	322385	3678	57401	1.128003	17.604267
1	종로구	163569	153780	9789	26512	5.984630	16.208450
2	중구	135427	126032	9395	21798	6.937317	16.095756
10	도봉구	344096	341928	2168	54969	0.630057	15.974902
12	은평구	489045	484642	4403	76097	0.900326	15.560327

pandas-데이터 병합

데이터 병합방법(concat)

In [238]:
```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']})
df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']})
df3 = pd.DataFrame({'A': ['A8', 'A5', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']})
```

In [239]: df1

Out [239]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [240]: df2

Out [240]:

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

In [241]: df3

Out [241]:

	A	B	C	D
0	A8	B8	C8	D8
1	A5	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

pandas-데이터 병합

데이터 병합방법(concat)

- pandas.concat([DataFrame1, DataFrame2, …] [axis=0])
- 옵션 없이 사용하면 열 방향으로 단순히 합쳐준다.
- Index가 중복 될 수 있다.
- axis=1을 주면 행 방향으로 합쳐진다.

```
In [245]: result = pd.concat([df1, df2, df3])
          result
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7
0	A8	B8	C8	D8
1	A5	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

pandas-데이터 병합

데이터 병합방법(concat)

- pandas.concat([DataFrame1, DataFrame2, …] keys=[1, 2, …])
- 각 dataframe을 key로 묶어서 합쳐준다.

```
In [246]: result = pd.concat([df1, df2, df3], keys=['x', 'y', 'z'])
          result
```

		A	B	C	D
x	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
y	0	A4	B4	C4	D4
	1	A5	B5	C5	D5
	2	A6	B6	C6	D6
	3	A7	B7	C7	D7
z	0	A8	B8	C8	D8
	1	A5	B9	C9	D9
	2	A10	B10	C10	D10
	3	A11	B11	C11	D11

pandas-데이터 병합

데이터 병합방법(concat)

- 중간이 빈 df4를 작성

In [252]:
```
df4 = pd.DataFrame({'B': ['B2', 'B3', 'B6', 'B7'],
                    'D': ['D2', 'D3', 'D6', 'D7'],
                    'F': ['F2', 'F3', 'F6', 'F7'],},
                    index=[2, 3, 6, 7])
df4
```

Out [252]:

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

pandas-데이터 병합

데이터 병합방법(concat)

- df1과 df4 병합
  - axis=0으로 했을 때, 경고가 뜬다.
  - 이는 pandas의 sort의 기본값이 변경되기 때문

```
In [264]: result = pd.concat([df1, df4], axis=0)
          result
```

Out [264]:

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
2	NaN	B2	NaN	D2	F2
3	NaN	B3	NaN	D3	F3
6	NaN	B6	NaN	D6	F6

```
In [267]: result = pd.concat([df1, df4], axis=1)
          result
```

Out [267]:

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6

pandas-데이터 병합

데이터 병합방법(concat)

- df1과 df4 병합
  - 둘 다 있는 부분만 합치고 싶다.
  - Join옵션을 사용한다.

```
In [268]: result = pd.concat([df1, df4], axis=1, join='inner')
          result
```

Out [268]:

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

pandas-데이터 병합

데이터 병합방법(merge)

```
In [274]: left = pd.DataFrame({'key': ['K0', 'K4', 'K2', 'K3'],
                               'A': ['A0', 'A1', 'A2', 'A3'],
                               'B': ['B0', 'B1', 'B2', 'B3'],})
          right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
                                'C': ['C0', 'C1', 'C2', 'C3'],
                                'D': ['D0', 'D1', 'D2', 'D3'],})
```

```
In [275]: left
```

Out [275]:

	key	A	B
0	K0	A0	B0
1	K4	A1	B1
2	K2	A2	B2
3	K3	A3	B3

```
In [276]: right
```

Out [276]:

	key	C	D
0	K0	C0	D0
1	K1	C1	D1
2	K2	C2	D2
3	K3	C3	D3

pandas-데이터 병합

데이터 병합방법(merge)

- pandas.merge(dataframe1, dataframe2, …, on='값')
- on을 기준으로 공통된 key를 합친다.

```
In [277]: pd.merge(left, right, on='key')
```

Out[277]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A2	B2	C2	D2
2	K3	A3	B3	C3	D3

pandas-데이터 병합

데이터 병합방법(merge)

- pandas.merge(dataframe1, dataframe2, …, on='값')
- how='dataframe이름'을 넣으면
  기본이 되는 데이터프레임을 잡는다.

```
In [278]: pd.merge(left, right, how='left', on='key')
```

Out [278]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K4	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3

```
In [279]: pd.merge(left, right, how='right', on='key')
```

Out [279]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A2	B2	C2	D2
2	K3	A3	B3	C3	D3
3	K1	NaN	NaN	C1	D1

pandas-데이터 병합

데이터 병합방법(merge)

- pandas.merge(dataframe1, dataframe2, …, on='값')
- how='outer'/'inner'
  'outer': 합집합, inner: 교집합

```
In [280]: pd.merge(left, right, how='outer', on='key')
```

Out [280]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K4	A1	B1	NaN	NaN
2	K2	A2	B2	C2	D2
3	K3	A3	B3	C3	D3
4	K1	NaN	NaN	C1	D1

```
In [281]: pd.merge(left, right, how='inner', on='key')
```

Out [281]:

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A2	B2	C2	D2
2	K3	A3	B3	C3	D3

pandas

CCTV데이터와 인구 데이터 병합

- 필요 없는 이전데이터를 삭제(drop)

```
In [284]: data_result.drop(['2015년 이전'], axis=1, inplace=True)
          data_result.drop(['2016년'], axis=1, inplace=True)

In [286]: data_result.head()
```

Out [286]:

	구별	소계	2017년	2018년	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율
0	강남구	5221	577	448	1355.057418	551888	546952	4936	66011	0.894384	11.960941
1	강동구	1879	273	385	504.487829	437050	432749	4301	57680	0.984098	13.197575
2	강북구	1265	1	319	301.164978	326063	322385	3678	57401	1.128003	17.604267
3	강서구	1617	264	254	481.942794	606981	600257	6724	78042	1.107778	12.857404
4	관악구	3985	694	671	1346.533233	521960	504048	17912	71317	3.431681	13.663308

pandas

CCTV데이터와 인구 데이터 병합

- 필요 없는 이전데이터를 삭제
- 열은 del을 통해서 지울 수도 있다

In [287]:
```
del data_result['2017년']
del data_result['2018년']
data_result.head()
```

Out[287]:

	구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율
0	강남구	5221	1355.057418	551888	546952	4936	66011	0.894384	11.960941
1	강동구	1879	504.487829	437050	432749	4301	57680	0.984098	13.197575
2	강북구	1265	301.164978	326063	322385	3678	57401	1.128003	17.604267
3	강서구	1617	481.942794	606981	600257	6724	78042	1.107778	12.857404
4	관악구	3985	1346.533233	521960	504048	17912	71317	3.431681	13.663308

pandas

CCTV데이터와 인구 데이터 병합

- 그래프를 그릴 때, index가 '구'가 되면 편하다.

In [288]:
```
data_result.set_index('구별', inplace=True)
data_result.head()
```

Out [288]:

구별	소계	최근증가율	인구수	한국인	외국인	고령자	외국인비율	고령자비율
강남구	5221	1355.057418	551888	546952	4936	66011	0.894384	11.960941
강동구	1879	504.487829	437050	432749	4301	57680	0.984098	13.197575
강북구	1265	301.164978	326063	322385	3678	57401	1.128003	17.604267
강서구	1617	481.942794	606981	600257	6724	78042	1.107778	12.857404
관악구	3985	1346.533233	521960	504048	17912	71317	3.431681	13.663308

pandas

CCTV데이터와 인구 데이터 데이터 분석

- 상관관계