

[데이터 셋 예]

csv 파일 : 콤마(,) 로 구분되는 파일 (메모장으로 열림. 엑셀로도 열림.

excel 파일

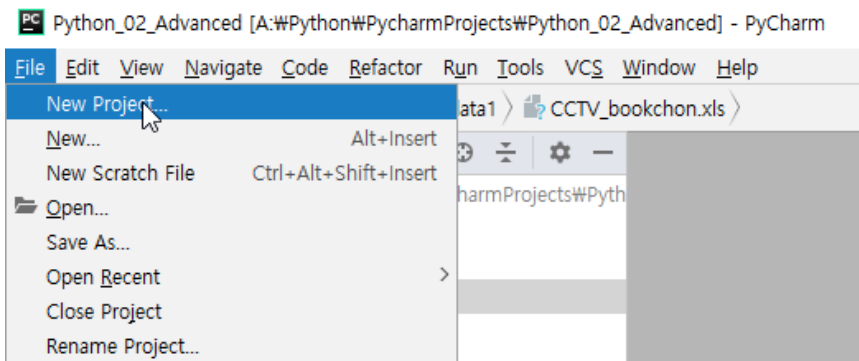
json 파일

xml 파일

csv 파일과 excel 파일을 손쉽게 읽을 수 있는 모듈 - **pandas 모듈**

[csv 파일 처리 예]

* 파이참에서 새 프로젝트 생성. (프로젝트 명: Python_02_Advanced)



* 파이참 프로젝트에 panda 폴더 생성

1) 분석 대상 데이터 셋 준비

서울시 열린 광장 <http://data.seoul.go.kr/>

서울시 자치구 연도별 CCTV 설치 현황 검색

서울특별시 제5회 서울강장문화제 개최

서울특별시 서울소식 응답소 정보공개

로그인 회원가입 사이트맵

서울열린데이터광장 데이터 이용하기 데이터 즐기기 데이터 참여소통

데이터 활용 데이터 분류

열린데이터광장에서 무엇을 하시겠습니까?

CCTV 설치현황

자동완성 검색어

해당 단어로 시작하는 검색어가 없습니다.

보건 일반행정 문화관광 산업/경제 복지 환경 교통 도시관리 교육 안전 인구/가구 주택/건설

데이터셋 (10건)

원천 SHEET CHART FILE

서울시 자치구 년도별 CCTV 설치 현황

<안전> 시설물 안전

제공기관 : 서울특별시 수정일 : 2018-06-08

Sheet Chart File

필터선택

XLS CSV JSON

기관명	소계	2013년도 이전	2014년	2015년	2016년
강남구	3,238	1,292	430	584	932
강동구	1,010	379	99	155	377
강북구	831	369	120	138	204
강서구	911	388	258	184	81
관악구	2,109	846	260	390	613
광진구	878	573	78	53	174
구로구	1,884	1,142	173	246	323
금천구	1,348	674	51	269	354
노원구	1,566	542	57	451	516
도봉구	825	238	159	42	386
동대문구	1,870	1,070	23	198	579
동작구	1,302	544	341	103	314
민포구	800	314	110	160	270

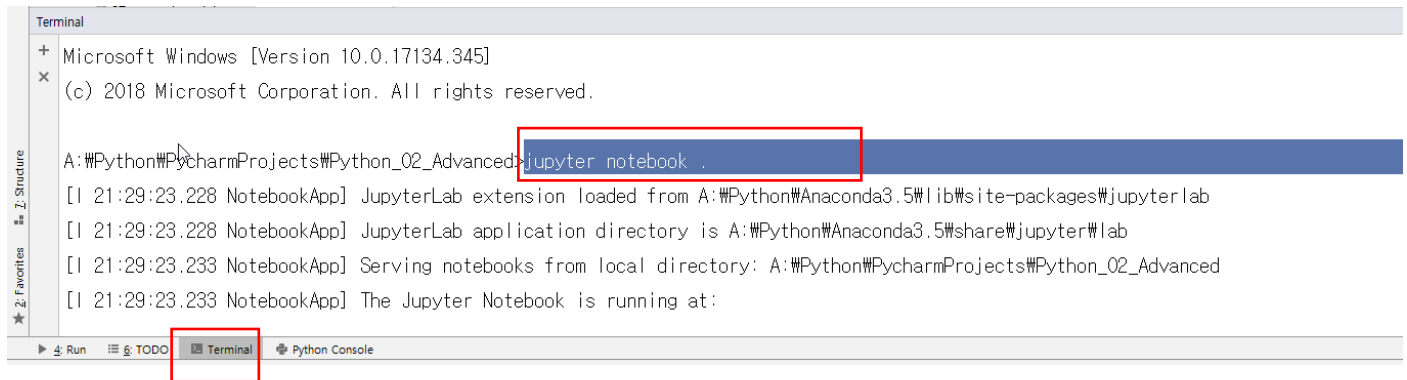
1 / 1 page(s) 25 count(s)

다운로드 받은 파일을 파이썬 프로젝트 01_panda 폴더에 ex01_data. CCTV_in_Seoul.csv로 저장

2. pandas로 csv 파일 읽기

1) 01_panda/ex01_pandas_csv.ipynb 생성

2) jupyter notebook 실행

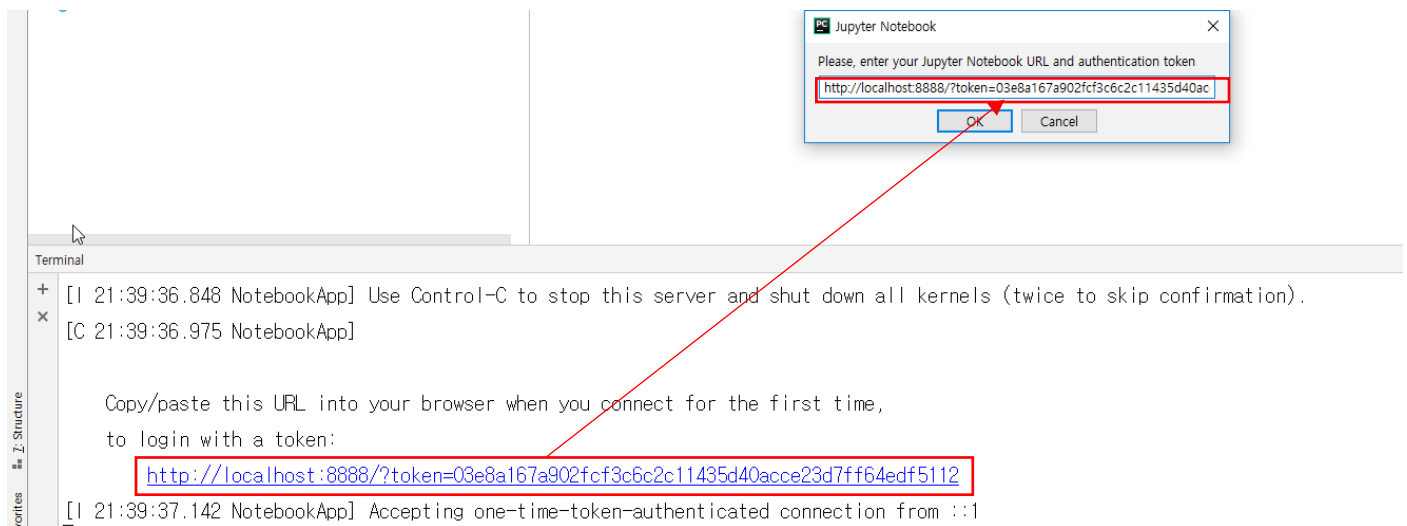


```
Terminal
+ Microsoft Windows [Version 10.0.17134.345]
x (c) 2018 Microsoft Corporation. All rights reserved.

A:\Python\PycharmProjects\Python_02_Advanced>jupyter notebook .
[I 21:29:23.228 NotebookApp] JupyterLab extension loaded from A:\Python\Anaconda3.5\lib\site-packages\jupyterlab
[I 21:29:23.228 NotebookApp] JupyterLab application directory is A:\Python\Anaconda3.5\share\jupyter\lab
[I 21:29:23.233 NotebookApp] Serving notebooks from local directory: A:\Python\PycharmProjects\Python_02_Advanced
[I 21:29:23.233 NotebookApp] The Jupyter Notebook is running at:

Run TODO Terminal Python Console
```

3) Shift +enter 또는 control+enter 로 실행



```
Jupyter Notebook
Please, enter your Jupyter Notebook URL and authentication token
http://localhost:8888/?token=03e8a167a902fcf3c6c2c11435d40ac
OK Cancel

Terminal
+ [I 21:39:36.848 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
x [C 21:39:36.975 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=03e8a167a902fcf3c6c2c11435d40acce23d7ff64edf5112
[I 21:39:37.142 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

4) ex01_pandas_csv.ipynb 구현

<CSV 파일 읽기>

```

In [1]: import pandas as pd
import os
print(os.getcwd()) #현재 경로 확인
#현재 파일이 있는 경로로 현재 경로 변경
os.chdir("A:/Python/PycharmProjects/Python_02_Advanced/01_pandas")
print(os.getcwd())

```

```

A:\Python\PycharmProjects\Python_02_Advanced
A:\Python\PycharmProjects\Python_02_Advanced\01_pandas

```

```

In [2]: CCTV_Seoul = pd.read_csv('../data/01. CCTV_in_Seoul.csv', encoding='utf-8')
print(CCTV_Seoul.head()) #5개 행만 읽음

```

	기관명	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	2780	1292	430	584	932
1	강동구	773	379	99	155	377
2	강북구	748	369	120	138	204
3	강서구	884	388	258	184	81
4	관악구	1496	846	260	390	613

```

In [4]: print(CCTV_Seoul.columns) #컬럼 명만
Index(['기관명', '소계', '2013년도 이전', '2014년', '2015년', '2016년'], dtype='object')

```

```

In [5]: print(CCTV_Seoul.columns[0]) # 첫번째 컬럼
기관명

```

```

In [6]: CCTV_Seoul.rename(columns={CCTV_Seoul.columns[0]: '구별'}, inplace=True) #첫번째 컬럼명 변경
print(CCTV_Seoul.head())

```

	구별	소계	2013년도 이전	2014년	2015년	2016년
0	강남구	2780	1292	430	584	932
1	강동구	773	379	99	155	377
2	강북구	748	369	120	138	204
3	강서구	884	388	258	184	81
4	관악구	1496	846	260	390	613

[엑셀 파일 처리 예]

1) 분석 대상 데이터 셋 준비

공유폴더 파일 복사 (ex02_data_population_in_Seoul.xls)

2) ex02_pandas_xls.ipynb 구현

```
In [1]: import pandas as pd
import os
print(os.getcwd()) #현재 경로 확인
#현재 파일이 있는 경로로 현재 경로 변경
os.chdir("A:/Python/PycharmProjects/Python_02_Advanced/01_pandas")
print(os.getcwd())
```

```
A:\Python\PycharmProjects\Python_02_Advanced
A:\Python\PycharmProjects\Python_02_Advanced\01_pandas
```

```
In [5]: pop_Seoul = pd.read_excel('ex02_data_population_in_Seoul.xls', encoding='utf-8')
print_(pop_Seoul.head())
```

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	₩
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	
1	기간	자치구	세대	계	남자	여자	계	남자	
2	2017.1/4	합계	4202888	10197604	5000005	5197599	9926968	4871560	
3	2017.1/4	종로구	72654	162820	79675	83145	153589	75611	
4	2017.1/4	중구	59481	133240	65790	67450	124312	61656	

	인구.5	인구.6	인구.7	인구.8	세대당인구	65세이상고령자
0	한국인	등록외국인	등록외국인	등록외국인	세대당인구	65세이상고령자
1	여자	계	남자	여자	세대당인구	65세이상고령자
2	5055408	270636	128445	142191	2.36	1321458
3	77978	9231	4064	5167	2.11	25425
4	62656	8928	4134	4794	2.09	20764

날짜 데이터 형 date_range 가 있음. 기본 날짜를 지정하고 periods 옵션으로 6일간 지정

<DataFrame> 파이썬의 딕셔너리로 생성 가능

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

df4

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

index의 값[2,3,6,7]은 행의 키. [B,D,F]는 열의 키

```
In [277]: df = pd.DataFrame(np.random.randn(6,4), index=dates,
                           columns=['A','B','C','D'])
df
```

```
Out [277]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828
2013-01-05	-1.262868	1.046419	0.958146	1.846223
2013-01-06	0.614887	-0.081613	-0.916182	0.773205

np.random.randn(6,4) → 6행 4열의 난수 발생함. index 는 행의 키 지정, columns는 열의 키 지정 .

```
In [278]: df.head()
```

```
Out [278]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828
2013-01-05	-1.262868	1.046419	0.958146	1.846223

```
In [279]: df.head(3)
```

```
Out [279]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317

head() 함수는 첫 5행만 보여줌.

괄호안에 숫자를 넣어두면 그 숫자만큼의 행을 볼 수 있음.

```
In [280]: df.index
```

```
Out [280]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                          '2013-01-05', '2013-01-06'],
                          dtype='datetime64[ns]', freq='D')
```

```
In [281]: df.columns
```

```
Out [281]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

Index 는 키(행) 이름, columns는 열 이름

```
In [282]: df.values
Out [282]: array([[ -0.20283216, -1.74580141,  0.82832879,  1.19187261],
 [ 0.46770329,  0.74444238,  0.01285395, -0.31656556],
 [ 0.41661431, -0.56408944, -0.44002039, -1.3753174 ],
 [-0.15419456, -0.93337606,  0.61420063,  1.03782821],
 [-1.26286813,  1.04641935,  0.95814574,  1.84622283],
 [ 0.61488744, -0.0816134 , -0.91618171,  0.77320477]])
```

Values 데이터

```
In [284]: df.describe()
Out [284]:
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.020115	-0.255670	0.176221	0.526208
std	0.696811	1.048891	0.752099	1.169134
min	-1.262868	-1.745801	-0.916182	-1.375317
25%	-0.190673	-0.841054	-0.326802	-0.044123
50%	0.131210	-0.322851	0.313527	0.905516
75%	0.454931	0.537928	0.774797	1.153362
max	0.614887	1.046419	0.958146	1.846223

describe() 함수는 통계적 개요. 평균, 표준편차, 최소, 최대, 1/4 지점

정렬

```
In [285]: df.sort_values(by='B', ascending=False)
Out [285]:
```

	A	B	C	D
2013-01-05	-1.262868	1.046419	0.958146	1.846223
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-06	0.614887	-0.081613	-0.916182	0.773205
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828
2013-01-01	-0.202832	-1.745801	0.828329	1.191873

B컬럼을 기준으로 정렬. ascending=False는 내림차순, ascending=True 올림차순

```
In [286]: df
Out [286]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828
2013-01-05	-1.262868	1.046419	0.958146	1.846223
2013-01-06	0.614887	-0.081613	-0.916182	0.773205

데이터 프레임 슬라이싱.


```
In [288]: df[0:3]
Out[288]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317

df[0:3] 0행~3바로 앞 행

```
In [289]: df['20130102':'20130104']
Out[289]:
```

	A	B	C	D
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828

행 데이터 중 20130102 부터 20130104

```
In [290]: df.loc[dates[0]]
Out[290]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873

Name: 2013-01-01 00:00:00, dtype: float64

df.loc[dates[0]] → df.loc['20130101'] → 행 데이터 중 '20130101' 만

```
In [291]: df.loc[:,['A','B']]
Out[291]:
```

	A	B
2013-01-01	-0.202832	-1.745801
2013-01-02	0.467703	0.744442
2013-01-03	0.416614	-0.564089
2013-01-04	-0.154195	-0.933376
2013-01-05	-1.262868	1.046419
2013-01-06	0.614887	-0.081613

: → 모든 행
 ['A','B'] → A,B 열

```
In [292]: df.loc['20130102':'20130104',['A','B']]
Out[292]:
```

	A	B
2013-01-02	0.467703	0.744442
2013-01-03	0.416614	-0.564089
2013-01-04	-0.154195	-0.933376

20130102, 20130104 행
 A, B 열

```
In [293]: df.loc['20130102',['A','B']]
Out [293]: A    0.467703
           B    0.744442
           Name: 2013-01-02 00:00:00, dtype: float64
```

20130102행

A, B 열

```
In [294]: df.loc[dates[0], 'A']
Out [294]: -0.20283216493809628
```

20130101행

A 열

[iloc 함수] 행번호로 조회 (참고: loc 함수는 인덱스로 조회)

```
In [295]: df.iloc[3]
Out [295]: A    -0.154195
           B    -0.933376
           C     0.614201
           D     1.037828
           Name: 2013-01-04 00:00:00, dtype: float64
```

3행

```
In [296]: df.iloc[3:5,0:2]
Out [296]:
```

	A	B
2013-01-04	-0.154195	-0.933376
2013-01-05	-1.262868	1.046419

3~4행. 0~1열

```
In [297]: df.iloc[[1,2,4],[0,2]]
Out [297]:
```

	A	C
2013-01-02	0.467703	0.012854
2013-01-03	0.416614	-0.440020
2013-01-05	-1.262868	0.958146

1,2,4, 행 0,2 열

```
In [298]: df.iloc[1:3,: ]
Out [298]:
```

	A	B	C	D
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317

1~2행, 모든 열

```
In [299]: df.iloc[:,1:3]
```

```
Out [299]:
```

	B	C
2013-01-01	-1.745801	0.828329
2013-01-02	0.744442	0.012854
2013-01-03	-0.564089	-0.440020
2013-01-04	-0.933376	0.614201
2013-01-05	1.046419	0.958146
2013-01-06	-0.081613	-0.916182

모든 행 1~2행

```
In [300]: df
```

```
Out [300]:
```

	A	B	C	D
2013-01-01	-0.202832	-1.745801	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-04	-0.154195	-0.933376	0.614201	1.037828
2013-01-05	-1.262868	1.046419	0.958146	1.846223
2013-01-06	0.614887	-0.081613	-0.916182	0.773205

```
In [301]: df[df.A > 0]
```

```
Out [301]:
```

	A	B	C	D
2013-01-02	0.467703	0.744442	0.012854	-0.316566
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317
2013-01-06	0.614887	-0.081613	-0.916182	0.773205

A 열 값이 0이상인 것만

```
In [302]: df[df > 0]
```

```
Out [302]:
```

	A	B	C	D
2013-01-01	NaN	NaN	0.828329	1.191873
2013-01-02	0.467703	0.744442	0.012854	NaN
2013-01-03	0.416614	NaN	NaN	NaN
2013-01-04	NaN	NaN	0.614201	1.037828
2013-01-05	NaN	1.046419	0.958146	1.846223
2013-01-06	0.614887	NaN	NaN	0.773205

모든 값이 0보다 큰 것만 출력. 그렇지 않은 것은 NaN으로 표시

```
In [303]: df2 = df.copy()
```

```
In [304]: df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
df2
```

```
Out [304]:
```

	A	B	C	D	E
2013-01-01	-0.202832	-1.745801	0.828329	1.191873	one
2013-01-02	0.467703	0.744442	0.012854	-0.316566	one
2013-01-03	0.416614	-0.564089	-0.440020	-1.375317	two
2013-01-04	-0.154195	-0.933376	0.614201	1.037828	three
2013-01-05	-1.262868	1.046419	0.958146	1.846223	four
2013-01-06	0.614887	-0.081613	-0.916182	0.773205	three

copy() 함수는 복제

df2['E'] = ['one', .. '] 열 추가.

```
In [305]: df2['E'].isin(['two', 'four'])
Out[305]: 2013-01-01    False
          2013-01-02    False
          2013-01-03     True
          2013-01-04    False
          2013-01-05     True
          2013-01-06    False
          Freq: D, Name: E, dtype: bool
```

E 열 값이 'two', 'four' 인 것은 True, 그렇지 않으면 False

```
In [1]: import pandas as pd
import os
print(os.getcwd()) #현재 경로 확인
#현재 파일이 있는 경로로 현재 경로 변경
os.chdir("A:/Python/PycharmProjects/Python_02_Advanced/01_pandas")
print(os.getcwd())

pop_Seoul = pd.read_excel('ex02_data_population_in_Seoul.xls', encoding='utf-8')
print(_pop_Seoul.head())
```

```
Out[314]:
```

	구별	인구수	한국인	외국인	고령자
0	합계	10197604.0	9926968.0	270636.0	1321458.0
1	종로구	162820.0	153589.0	9231.0	25425.0
2	중구	133240.0	124312.0	8928.0	20764.0
3	용산구	244203.0	229456.0	14747.0	36231.0
4	성동구	311244.0	303380.0	7864.0	39997.0

```
In [315]: pop_Seoul.drop([0], inplace=True)
pop_Seoul.head()
```

```
Out[315]:
```

	구별	인구수	한국인	외국인	고령자
1	종로구	162820.0	153589.0	9231.0	25425.0
2	중구	133240.0	124312.0	8928.0	20764.0
3	용산구	244203.0	229456.0	14747.0	36231.0
4	성동구	311244.0	303380.0	7864.0	39997.0
5	광진구	372164.0	357211.0	14953.0	42214.0

drop[0] 첫 행 삭제

```
In [316]: pop_Seoul['구별'].unique()
Out[316]: 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', nan],
               dtype=object)
```

'구별' 열 값의 유일한 값만

< dataframe 생성 >

```
In [325]: df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                             'B': ['B0', 'B1', 'B2', 'B3'],
                             'C': ['C0', 'C1', 'C2', 'C3'],
                             'D': ['D0', 'D1', 'D2', 'D3']],
                             index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']],
                    index=[4, 5, 6, 7])

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']],
                    index=[8, 9, 10, 11])
```

```
In [326]: df1
```

```
Out[326]:
```

	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 [327]: df2
```

```
Out[327]:
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
In [328]: df3
```

```
Out[328]:
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

<dataframe 합치기 예1>

```
In [329]: result = pd.concat([df1, df2, df3])
result
```

```
Out[329]:
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

아무 옵션이 없으면 열방향으로 합침

<dataframe 합치기 예2> axis=1 옵션 : 행방향으로 합침

```
In [335]: df4 = pd.DataFrame({'B': ['B2', 'B3', 'B6', 'B7'],
                             'D': ['D2', 'D3', 'D6', 'D7'],
                             'F': ['F2', 'F3', 'F6', 'F7']},
                             index=[2, 3, 6, 7])
result = pd.concat([df1, df4], axis=1)
```

```
In [336]: df1
```

```
Out[336]:
```

	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 [337]: df4
```

```
Out[337]:
```

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

```
In [338]: result
```

```
Out[338]:
```

	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
7	NaN	NaN	NaN	NaN	B7	D7	F7

concat은 *index를 기준으로 행방향으로 합쳐짐

```
In [76]: result = pd.concat([df1, df4], axis=1, join='inner')
result
```

```
Out[76]:
```

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

join="inner" 옵션은 index가 일치하지 않는 행은 없어짐

```
In [77]: result = pd.concat([df1, df4], axis=1, join_axes=[df1.index])
result
```

```
Out[77]:
```

	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

join_axes=[df1.index] 옵션은 df1.index를 기준으로 합쳐짐. df1은 모두 출력.

<dataframe merge> 인덱스 외에 다른 열값을 기준으로 합쳐질 수 있음

```
In [79]: 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 [80]: left
```

```
Out[80]:
```

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

```
In [81]: right
```

```
Out[81]:
```

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

```
In [82]: pd.merge(left, right, on='key')
```

```
Out[82]:
```

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

key 컬럼 값이 일치하는 행끼리 합쳐짐


```
In [83]: pd.merge(left, right, how='left', on='key')
```

```
Out[83]:
```

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

```
In [84]: pd.merge(left, right, how='right', on='key')
```

```
Out[84]:
```

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

```
In [85]: pd.merge(left, right, how='outer', on='key')
```

```
Out[85]:
```

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

```
pd.merge(left, right, how='left', on='key')
```

왼쪽을 기준으로 key 컬럼 값이 일치하는 행끼리 합쳐짐

```
pd.merge(left, right, how='right', on='key')
```

오른쪽을 기준으로 key 컬럼 값이 일치하는 행끼리 합쳐짐

```
pd.merge(left, right, how='outer', on='key')
```

key 컬럼 값이 일치하는 행끼리 합쳐지고, 일치하지 않는 컬럼도 모두 나타남.

```
In [86]: pd.merge(left, right, how='inner', on='key')
```

```
Out[86]:
```

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

```
pd.merge(left, right, how='inner', on='key')
```

key 컬럼 값이 일치하는 행끼리 합쳐지고, 일치하지 않는 컬럼은 없어짐. default