Pandas 한번에 제대로 배우기

- 관계 또는 레이블링 데이터로 쉽고 직관적으로 작업할 수 있도록 고안된 빠르고 유연하며 표현력이 뛰어난 데이터 구조를 제공하는 Python 패키지

Pandas 특징

- 부동 소수점이 아닌 데이터 뿐만 아니라 부동 소수점 데이터에서도 결측 데이터(NaN으로 표시됨)를 쉽게 처리
- 크기 변이성(Size mutability): DataFrame 및 고차원 객체에서 열을 삽입 및 삭제 가능
- 자동 및 명시적(explicit) 데이터 정렬: 객체를 라벨 집합에 명시적으로 정렬하거나, 사용자가 라벨을 무시하고 Series, DataFrame 등의 계산에서 자동으로 데이터 조정 가능
- 데이터 세트에서 집계 및 변환을 위한 분할(split), 적용(apply), 결합(combine) 작업을 수행할 수 있는 강력하고 유연한 group-by 함수 제공
- 누락된 데이터 또는 다른 Python 및 NumPy 데이터 구조에서 서로 다른 인덱싱 데이터를 DataFrame 개체로 쉽게 변환
- 대용량 데이터 세트의 지능형 라벨 기반 슬라이싱, 고급 인덱싱 및 부분 집합 구하기 가능
- 직관적인 데이터 세트 병합 및 결합
- 데이터 세트의 유연한 재구성 및 피벗
- 축의 계층적 라벨링(눈금당 여러 개의 라벨을 가질 수 있음)
- 플랫 파일(CSV 및 구분), Excel 파일, 데이터베이스 로딩 및 초고속 HDF5 형식의 데이터 저장/로드에 사용되는 강력한 IO 도구
- 시계열 특정 기능: 날짜 범위 생성 및 주파수 변환, 무빙 윈도우(moving window) 통계, 날짜 이동 및 지연

```
import numpy as np
import pandas as pd
```

```
import pandas as pd
pd.__version__
```

```
'1.0.5'
```

Pandas 객체

Series 객체

```
s = pd.Series([0, 0.25, 0.5, 0.75, 1.0])
s
```

```
0    0.00
1    0.25
2    0.50
3    0.75
4    1.00
dtype: float64
```

```
s.values
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

```
s.index
```

```
RangeIndex(start=0, stop=5, step=1)
```



```
s[1]
```

```
0.25
```

```
s[1:4]
```

```
1    0.25
2    0.50
3    0.75
dtype: float64
```

```
s = pd.Series([0, 0.25, 0.5, 0.75, 1.0],
              index=['a', 'b', 'c', 'd', 'e'])
s
```

```
a    0.00
b    0.25
c    0.50
d    0.75
e    1.00
dtype: float64
```

```
s['c']
```

```
0.5
```

```
s[['c', 'd', 'e']]
```

```
c    0.50
d    0.75
e    1.00
dtype: float64
```

```
'b' in s
```

```
True
```

```
s = pd.Series([0, 0.25, 0.5, 0.75, 1.0],
              index=[2, 4, 6, 8, 10])
s
```

```
2     0.00
4     0.25
6     0.50
8     0.75
10    1.00
dtype: float64
```



```
s[4]
```

```
0.25
```

```
s[2:]
```

```
6     0.50
8     0.75
10    1.00
dtype: float64
```

```
s.unique()
```

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

```
s.value_counts()
```

```
1.00    1
0.75    1
0.50    1
0.25    1
0.00    1
dtype: int64
```

```
s.isin([0.25, 0.75])
```

```
2     False
```

```
4      True
6     False
8      True
10    False
dtype: bool
```

```
pop_tuple = {'서울특별시': 9720846,
             '부산광역시': 3404423,
             '인천광역시': 2947217,
             '대구광역시': 2427954,
             '대전광역시': 1471040,
             '광주광역시': 1455048}
population = pd.Series(pop_tuple)
population
```

```
서울특별시    9720846
부산광역시    3404423
인천광역시    2947217
대구광역시    2427954
대전광역시    1471040
광주광역시    1455048
dtype: int64
```

```
population['서울특별시']
```

```
9720846
```



```
population['서울특별시':'인천광역시']
```

```
서울특별시    9720846
부산광역시    3404423
인천광역시    2947217
dtype: int64
```

DataFrame 객체

```
pd.DataFrame([{'A':2, 'B':4, 'D':3}, {'A':4, 'B':5, 'C':7}])
```

	A	B	D	C
0	2	4	3.0	NaN
1	4	5	NaN	7.0

```
pd.DataFrame(np.random.rand(5, 5),
             columns=['A', 'B', 'C', 'D', 'E'],
             index=[1, 2, 3, 4, 5])
```

	A	B	C	D	E
1	0.008643	0.294440	0.437378	0.112411	0.076327
2	0.560204	0.944527	0.800915	0.978915	0.464241
3	0.385648	0.013000	0.558273	0.453111	0.510526

```
male_tuple = {'서울특별시': 4732275,
              '부산광역시': 1668618,
              '인천광역시': 1476813,
              '대구광역시': 1198815,
              '대전광역시': 734441,
              '광주광역시': 720060}
male = pd.Series(male_tuple)
male
```

```
서울특별시    4732275
부산광역시    1668618
인천광역시    1476813
대구광역시    1198815
대전광역시     734441
광주광역시     720060
dtype: int64
```

```
female_tuple = {'서울특별시': 4988571,
                '부산광역시': 1735805,
                '인천광역시': 1470404,
                '대구광역시': 1229139,
                '대전광역시': 736599,
                '광주광역시': 734988}
female = pd.Series(female_tuple)
female
```



```
서울특별시    4988571
부산광역시    1735805
인천광역시    1470404
대구광역시    1229139
대전광역시     736599
광주광역시     734988
dtype: int64
```

```
korea_df = pd.DataFrame({'인구수': population,
                         '남자인구수': male,
                         '여자인구수': female})
korea_df
```

	인구수	남자인구수	여자인구수
서울특별시	9720846	4732275	4988571

```
korea_df.index
```

```
Index(['서울특별시', '부산광역시', '인천광역시', '대구광역시', '대전광역시', '광주광역시'],
```

```
korea_df.columns
```

```
Index(['인구수', '남자인구수', '여자인구수'], dtype='object')
```

```
korea_df['여자인구수']
```

```
서울특별시    4988571
부산광역시    1735805
인천광역시    1470404
대구광역시    1229139
대전광역시     736599
광주광역시     734988
Name: 여자인구수, dtype: int64
```

```
korea_df['서울특별시':'인천광역시']
```

	인구수	남자인구수	여자인구수
서울특별시	9720846	4732275	4988571
부산광역시	3404423	1668618	1735805
인천광역시	2947217	1476813	1470404

Index 객체

클래스	설명
Index	일반적인 Index 객체이며, NumPy 배열 형식으로 축의 이름 표현
Int64Index	정수 값을 위한 Index
MultiIndex	단일 축에 여러 단계 색인을 표현하는 계층적 Index 객체 (튜플의 배열과 유사)
DatetimeIndex	NumPy의 datetime64 타입으로 타임스탬프 저장
PeriodIndex	기간 데이터를 위한 Index

```
idx = pd.Index([2, 4, 6, 8, 10])
idx
```

```
Int64Index([2, 4, 6, 8, 10], dtype='int64')
```

```
idx[1]
```

```
4
```

```
idx[1:2:2]
```

```
Int64Index([4], dtype='int64')
```

```
idx[-1::]
```

```
Int64Index([10], dtype='int64')
```

```
idx[::2]
```

```
Int64Index([2, 6, 10], dtype='int64')
```

```
print(idx)
print(idx.size)
print(idx.shape)
print(idx.ndim)
print(idx.dtype)
```

```
Int64Index([2, 4, 6, 8, 10], dtype='int64')
5
(5,)
1
int64
```


Index 연산

연산자	메소드	설명
	append	색인 객체를 추가한 새로운 색인 반환
	difference	색인의 차집합 반환
&	intersection	색인의 교집합 반환
|	union	색인의 합집합 반환
	isin	색인이 존재하는지 여부를 불리언 배열로 반환
	delete	색인이 삭제된 새로운 색인 반환
	drop	값이 삭제된 새로운 색인 반환
	insert	색인이 추가된 새로운 색인 반환
	is_monotonic	색인이 단조성을 가지면 True
	is_unique	중복되는 색인이 없다면 True
	unique	색인에서 중복되는 요소를 제거하고 유일한 값만 반환

```
idx1 = pd.Index([1, 2, 4, 6, 8])
idx2 = pd.Index([2, 4, 5, 6, 7])
print(idx1.append(idx2))
print(idx1.difference(idx2))
print(idx1 - idx2)
print(idx1.intersection(idx2))
print(idx1 & idx2)
print(idx1.union(idx2))
print(idx1 | idx2)
print(idx1.delete(0))
```

```
print(idx1.drop(1))
print(idx1 ^ idx2)
```

```
Int64Index([1, 2, 4, 6, 8, 2, 4, 5, 6, 7], dtype='int64')
Int64Index([1, 8], dtype='int64')
Int64Index([-1, -2, -1, 0, 1], dtype='int64')
Int64Index([2, 4, 6], dtype='int64')
Int64Index([2, 4, 6], dtype='int64')
Int64Index([1, 2, 4, 5, 6, 7, 8], dtype='int64')
Int64Index([1, 2, 4, 5, 6, 7, 8], dtype='int64')
Int64Index([2, 4, 6, 8], dtype='int64')
Int64Index([2, 4, 6, 8], dtype='int64')
Int64Index([1, 5, 7, 8], dtype='int64')
```

인덱싱(Indexing)

```
s = pd.Series([0, 0.25, 0.5, 0.75, 1.0],
              index=['a', 'b', 'c', 'd', 'e'])
s
```

```
a    0.00
b    0.25
c    0.50
d    0.75
e    1.00
dtype: float64
```



```
s['b']
```

```
0.25
```

```
'b' in s
```

```
True
```

```
s.keys()
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
list(s.items())
```

```
[('a', 0.0), ('b', 0.25), ('c', 0.5), ('d', 0.75), ('e', 1.0)]
```

```
s['f'] = 1.25
s
```

```
a    0.00
b    0.25
```

```
c    0.50
d    0.75
e    1.00
f    1.25
dtype: float64
```

```
s['a':'d']
```

```
a    0.00
b    0.25
c    0.50
d    0.75
dtype: float64
```

```
s[0:4]
```

```
a    0.00
b    0.25
c    0.50
d    0.75
dtype: float64
```

```
s[(s > 0.4) & (s < 0.8)]
```

```
c    0.50
d    0.75
dtype: float64
```

```
s[['a', 'c', 'e']]
```

```
a    0.0
c    0.5
e    1.0
dtype: float64
```


Series 인덱싱

```
s = pd.Series(['a', 'b', 'c', 'd', 'e'],
              index=[1, 3, 5, 7, 9])
s
```

```
1    a
3    b
5    c
7    d
9    e
dtype: object
```

```
s[1]
```

```
'a'
```

```
s[2:4]
```

```
5    c
7    d
dtype: object
```

```
s.iloc[1]
```

```
'b'
```

```
s.iloc[2:4]
```

```
5    c
7    d
dtype: object
```

```
s.reindex(range(10))
```

```
0    NaN
1      a
2    NaN
3      b
4    NaN
5      c
6    NaN
7      d
8    NaN
9      e
dtype: object
```



```
s.reindex(range(10), method='bfill')
```

```
0    a
1    a
2    b
3    b
4    c
5    c
6    d
7    d
8    e
9    e
dtype: object
```

DataFrame 인덱싱

사용 방법	설명
`df[val]`	하나의 컬럼 또는 여러 컬럼을 선택
`df.loc[val]`	라벨값으로 로우의 부분집합 선택
`df.loc[:, val]`	라벨값으로 컬럼의 부분집합 선택
`df.loc[val1, val2]`	라벨값으로 로우와 컬럼의 부분집합 선택
`df.iloc[where]`	정수 색인으로 로우의 부분집합 선택

사용 방법	설명
`df.iloc[:, where]`	정수 색인으로 컬럼의 부분집합 선택
`df.iloc[where_i, where_j]`	정수 색인으로 로우와 컬럼의 부분집합 선택
`df.at[label_i, label_j]`	로우와 컬럼의 라벨로 단일 값 선택
`df.iat[i, j]`	로우와 컬럼의 정수 색인으로 단일 값 선택
`reindex`	하나 이상의 축을 새로운 색인으로 재색인
`get_value, set_value`	로우와 컬럼의 이름으로 값 선택

```
korea_df
```

	인구수	남자인구수	여자인구수
서울특별시	9720846	4732275	4988571
부산광역시	3404423	1668618	1735805
인천광역시	2947217	1476813	1470404
대구광역시	2427954	1198815	1229139
대전광역시	1471040	734441	736599
광주광역시	1455048	720060	734988


```
korea_df['남자인구수']
```

```
서울특별시     4732275
부산광역시     1668618
인천광역시     1476813
대구광역시     1198815
대전광역시      734441
광주광역시      720060
Name: 남자인구수, dtype: int64
```

```
korea_df.남자인구수
```

```
서울특별시     4732275
부산광역시     1668618
인천광역시     1476813
대구광역시     1198815
대전광역시      734441
광주광역시      720060
Name: 남자인구수, dtype: int64
```

```
korea_df.여자인구수
```

```
서울특별시     4988571
부산광역시     1735805
인천광역시     1470404
대구광역시     1229139
대전광역시      736599
광주광역시      734988
Name: 여자인구수, dtype: int64
```

```
korea_df['남여비율'] = (korea_df['남자인구수'] * 100 / korea_df['여자인구수'])
```

```
korea_df['남여비율'] = (korea_df['남자인구수'] * 100 / korea_df['여자인구수'])
```

```
korea_df.남여비율
```

```
서울특별시      94.862336
부산광역시      96.129346
인천광역시     100.435867
대구광역시      97.532907
대전광역시      99.707032
광주광역시      97.968946
Name: 남여비율, dtype: float64
```

```
korea_df.values
```

```
array([[9.72084600e+06, 4.73227500e+06, 4.98857100e+06, 9.48623363e+01],
       [3.40442300e+06, 1.66861800e+06, 1.73580500e+06, 9.61293463e+01],
       [2.94721700e+06, 1.47681300e+06, 1.47040400e+06, 1.00435867e+02],
       [2.42795400e+06, 1.19881500e+06, 1.22913900e+06, 9.75329072e+01],
       [1.47104000e+06, 7.34441000e+05, 7.36599000e+05, 9.97070319e+01],
       [1.45504800e+06, 7.20060000e+05, 7.34988000e+05, 9.79689464e+01]])
```

```
korea_df.T
```

	서울특별시	부산광역시	인천광역시	대구광역시	대전광역시	광주광역시
인구수	9.720846e+06	3.404423e+06	2.947217e+06	2.427954e+06	1.471040e+06	1.455048e+06
남자인구	4.732275e+06	1.668618e+06	1.476813e+06	1.198815e+06	7.344410e+05	7.200600e+05


```
korea_df.values[0]
```

```
array([9.72084600e+06, 4.73227500e+06, 4.98857100e+06, 9.48623363e+01])
```

```
korea_df['인구수']
```

```
서울특별시     9720846
부산광역시     3404423
인천광역시     2947217
대구광역시     2427954
대전광역시     1471040
광주광역시     1455048
Name: 인구수, dtype: int64
```

```
korea_df.loc[:'인천광역시', :'남자인구수']
```

	인구수	남자인구수

```
korea_df.loc[(korea_df.여자인구수 > 1000000)]
```

	인구수	남자인구수	여자인구수	남여비율
서울특별시	9720846	4732275	4988571	94.862336
부산광역시	3404423	1668618	1735805	96.129346
인천광역시	2947217	1476813	1470404	100.435867
대구광역시	2427954	1198815	1229139	97.532907

```
korea_df.loc[(korea_df.인구수 < 2000000)]
```

	인구수	남자인구수	여자인구수	남여비율
대전광역시	1471040	734441	736599	99.707032
광주광역시	1455048	720060	734988	97.968946

```
korea_df.loc[(korea_df.인구수 > 2500000)]
```

	인구수	남자인구수	여자인구수	남여비율
서울특별시	9720846	4732275	4988571	94.862336
부산광역시	3404423	1668618	1735805	96.129346
인천광역시	2947217	1476813	1470404	100.435867


```
korea_df.loc[korea_df.남여비율 > 100]
```

	인구수	남자인구수	여자인구수	남여비율
인천광역시	2947217	1476813	1470404	100.435867

```
korea_df.loc[(korea_df.인구수 > 2500000) & (korea_df.남여비율 > 100)]
```

	인구수	남자인구수	여자인구수	남여비율
인천광역시	2947217	1476813	1470404	100.435867

```
korea_df.iloc[:3, :2]
```

다중 인덱싱(Multi Indexing)

- 1차원의 Series와 2차원의 DataFrame 객체를 넘어 3차원, 4차원 이상의 고차원 데이터 처리
- 단일 인덱스 내에 여러 인덱스를 포함하는 다중 인덱싱

다중 인덱스 Series

```
korea_df
```

	인구수	남자인구수	여자인구수	남여비율
서울특별시	9720846	4732275	4988571	94.862336
부산광역시	3404423	1668618	1735805	96.129346
인천광역시	2947217	1476813	1470404	100.435867
대구광역시	2427954	1198815	1229139	97.532907
대전광역시	1471040	734441	736599	99.707032
광주광역시	1455048	720060	734988	97.968946


```
idx_tuples = [('서울특별시', 2010), ('서울특별시', 2020),
              ('부산광역시', 2010), ('부산광역시', 2020),
              ('인천광역시', 2010), ('인천광역시', 2020),
              ('대구광역시', 2010), ('대구광역시', 2020),
              ('대전광역시', 2010), ('대전광역시', 2020),
              ('광주광역시', 2010), ('광주광역시', 2020)]
idx_tuples
```

```
[('서울특별시', 2010),
 ('서울특별시', 2020),
 ('부산광역시', 2010),
 ('부산광역시', 2020),
 ('인천광역시', 2010),
 ('인천광역시', 2020),
 ('대구광역시', 2010),
 ('대구광역시', 2020),
 ('대전광역시', 2010),
 ('대전광역시', 2020),
 ('광주광역시', 2010),
 ('광주광역시', 2020)]
```

```
pop_tuples = [10312545, 9720846,
              2567910, 3404423,
              2758296, 2947217,
              2511676, 2427954,
              1503664, 1471040,
              1454636, 1455048]
population = pd.Series(pop_tuples, index=idx_tuples)
population
```

```
(서울특별시, 2010)    10312545
(서울특별시, 2020)     9720846
(부산광역시, 2010)     2567910
(부산광역시, 2020)     3404423
(인천광역시, 2010)     2758296
(인천광역시, 2020)     2947217
(대구광역시, 2010)     2511676
(대구광역시, 2020)     2427954
(대전광역시, 2010)     1503664
(대전광역시, 2020)     1471040
(광주광역시, 2010)     1454636
(광주광역시, 2020)     1455048
dtype: int64
```

```
midx = pd.MultiIndex.from_tuples(idx_tuples)
midx
```

```
MultiIndex([('서울특별시', 2010),
            ('서울특별시', 2020),
            ('부산광역시', 2010),
            ('부산광역시', 2020),
            ('인천광역시', 2010),
            ('인천광역시', 2020),
            ('대구광역시', 2010),
            ('대구광역시', 2020),
            ('대전광역시', 2010),
            ('대전광역시', 2020),
            ('광주광역시', 2010),
            ('광주광역시', 2020)],
           )
```



```
population = population.reindex(midx)
population
```

```
서울특별시  2010    10312545
       2020     9720846
부산광역시  2010     2567910
       2020     3404423
인천광역시  2010     2758296
       2020     2947217
대구광역시  2010     2511676
       2020     2427954
대전광역시  2010     1503664
       2020     1471040
광주광역시  2010     1454636
       2020     1455048
dtype: int64
```

```
population[:, 2010]
```

```
서울특별시    10312545
부산광역시     2567910
인천광역시     2758296
대구광역시     2511676
대전광역시     1503664
광주광역시     1454636
dtype: int64
```

```
population['대전광역시', :]
```

```
2010    1503664
2020    1471040
dtype: int64
```

```
korea_mdf = population.unstack()
korea_mdf
```

	2010	2020
광주광역시	1454636	1455048
대구광역시	2511676	2427954
대전광역시	1503664	1471040
부산광역시	2567910	3404423
서울특별시	10312545	9720846
인천광역시	2758296	2947217


```
korea_mdf.stack()
```

```
광주광역시  2010    1454636
        2020    1455048
대구광역시  2010    2511676
        2020    2427954
대전광역시  2010    1503664
        2020    1471040
부산광역시  2010    2567910
        2020    3404423
서울특별시  2010    10312545
        2020    9720846
인천광역시  2010    2758296
        2020    2947217
dtype: int64
```

```
male_tuples = [5111259, 4732275,
               1773170, 1668618,
               1390356, 1476813,
               1255245, 1198815,
               753648, 734441,
               721780, 720060]
male_tuples
```

```
[5111259,
 4732275,
 1773170,
 1668618,
 1390356,
 1476813,
 1255245,
 1198815,
```

```
 753648,
 734441,
 721780,
 720060]
```

```
korea_mdf = pd.DataFrame({'총인구수': population,
                          '남자인구수': male_tuples})
korea_mdf
```

		총인구수	남자인구수
서울특별시	**2010**	10312545	5111259
	2020	9720846	4732275
부산광역시	**2010**	2567910	1773170
	2020	3404423	1668618
인천광역시	**2010**	2758296	1390356
	2020	2947217	1476813
대구광역시	**2010**	2511676	1255245
	2020	2427954	1198815
대전광역시	**2010**	1503664	753648
	2020	1471040	734441
광주광역시	**2010**	1454636	721780
	2020	1455048	720060


```
female_tuples = [5201286, 4988571,
                 1794740, 1735805,
                 1367940, 1470404,
                 1256431, 1229139,
                 750016, 736599,
                 732856, 734988]
female_tuples
```

```
[5201286,
 4988571,
 1794740,
 1735805,
 1367940,
 1470404,
 1256431,
 1229139,
 750016,
 736599,
 732856,
 734988]
```

```
korea_mdf = pd.DataFrame({'총인구수': population,
                          '남자인구수': male_tuples,
                          '여자인구수': female_tuples})
```

```
korea_mdf
```

		총인구수	남자인구수	여자인구수
서울특별시	**2010**	10312545	5111259	5201286
	2020	9720846	4732275	4988571
부산광역시	**2010**	2567910	1773170	1794740
	2020	3404423	1668618	1735805
인천광역시	**2010**	2758296	1390356	1367940
	2020	2947217	1476813	1470404
대구광역시	**2010**	2511676	1255245	1256431
	2020	2427954	1198815	1229139
대전광역시	**2010**	1503664	753648	750016
	2020	1471040	734441	736599
광주광역시	**2010**	1454636	721780	732856
	2020	1455048	720060	734988


```
ratio = korea_mdf['남자인구수'] * 100 / korea_mdf['여자인구수']
ratio
```

```
서울특별시  2010      98.269140
        2020      94.862336
부산광역시  2010      98.798155
        2020      96.129346
인천광역시  2010     101.638668
        2020     100.435867
대구광역시  2010      99.905606
        2020      97.532907
대전광역시  2010     100.484256
        2020      99.707032
광주광역시  2010      98.488653
        2020      97.968946
dtype: float64
```

```
ratio.unstack()
```

```
korea_mdf = pd.DataFrame({'총인구수': population,
                          '남자인구수': male_tuples,
                          '여자인구수': female_tuples,
                          '남여비율':ratio})
korea_mdf
```

		총인구수	남자인구수	여자인구수	남여비율
서울특별시	**2010**	10312545	5111259	5201286	98.269140
	2020	9720846	4732275	4988571	94.862336
부산광역시	**2010**	2567910	1773170	1794740	98.798155
	2020	3404423	1668618	1735805	96.129346
인천광역시	**2010**	2758296	1390356	1367940	101.638668
	2020	2947217	1476813	1470404	100.435867
대구광역시	**2010**	2511676	1255245	1256431	99.905606
	2020	2427954	1198815	1229139	97.532907
대전광역시	**2010**	1503664	753648	750016	100.484256
	2020	1471040	734441	736599	99.707032
광주광역시	**2010**	1454636	721780	732856	98.488653
	2020	1455048	720060	734988	97.968946

다중 인덱스 생성

```
df = pd.DataFrame(np.random.rand(6, 3),
                  index=[['a', 'a', 'b', 'b', 'c', 'c'], [1, 2, 1, 2, 1, 2]],
                  columns=['c1', 'c2', 'c3'])
df
```

		c1	c2	c3
a	**1**	0.787723	0.488996	0.598397
	2	0.982661	0.120751	0.718474
b	**1**	0.385435	0.858297	0.051731
	2	0.550690	0.356138	0.521237
c	**1**	0.155001	0.019116	0.938892
	2	0.339056	0.175161	0.864092

```
pd.MultiIndex.from_arrays([['a', 'a', 'b', 'b', 'c', 'c'], [1, 2, 1, 2, 1, 2]])
```

```
MultiIndex([('a', 1),
            ('a', 2),
```

```
                ('b', 1),
                ('b', 2),
                ('c', 1),
                ('c', 2)],
               )
```

```
pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('b', 1), ('b', 2), ('c', 1), ('c', 2)])
```

```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2),
            ('c', 1),
            ('c', 2)],
           )
```

```
pd.MultiIndex.from_product([['a', 'b', 'c'], [1, 2]])
```

```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2),
            ('c', 1),
            ('c', 2)],
           )
```

```
pd.MultiIndex(levels=[['a', 'b', 'c'], [1, 2]],
              codes=[[0, 0, 1, 1, 2, 2], [0, 1, 0, 1, 0, 1]])
```

```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2),
            ('c', 1),
            ('c', 2)],
           )
```

```
population
```

```
서울특별시  2010    10312545
       2020     9720846
부산광역시  2010     2567910
       2020     3404423
인천광역시  2010     2758296
       2020     2947217
대구광역시  2010     2511676
       2020     2427954
대전광역시  2010     1503664
       2020     1471040
광주광역시  2010     1454636
       2020     1455048
dtype: int64
```

```
population.index.names = ['행정구역', '년도']
population
```

```
행정구역    년도
서울특별시  2010    10312545
        2020     9720846
부산광역시  2010     2567910
        2020     3404423
인천광역시  2010     2758296
        2020     2947217
대구광역시  2010     2511676
        2020     2427954
대전광역시  2010     1503664
        2020     1471040
광주광역시  2010     1454636
        2020     1455048
dtype: int64
```

```
idx = pd.MultiIndex.from_product([['a', 'b', 'c'], [1, 2]],
                                 names=['name1', 'name2'])
cols = pd.MultiIndex.from_product([['c1', 'c2', 'c3'], [1, 2]],
                                  names=['col_name1', 'col_name2'])
data = np.round(np.random.randn(6, 6), 2)
mdf = pd.DataFrame(data, index=idx, columns=cols)
mdf
```

	col_name1	c1		c2		c3	
	col_name2	1	2	1	2	1	2
name1	name2						
a	1	-0.13	-1.66	0.58	-0.28	0.58	1.08
	2	2.08	-0.30	0.10	1.12	0.60	-0.08
b	1	0.68	-2.13	0.92	0.65	0.88	-0.02
	2	0.26	-2.44	1.27	0.65	1.68	0.97
c	1	-0.26	0.08	0.28	0.78	0.95	-0.11
	2	2.69	-0.62	0.60	-0.67	0.25	0.40


```
mdf['c2']
```

	col_name2	1	2
name1	name2		
a	1	0.58	-0.28
	2	0.10	1.12
b	1	0.92	0.65
	2	1.27	0.65
c	1	0.28	0.78
	2	0.60	-0.67

인덱싱 및 슬라이싱

```
population
```

```
행정구역    년도
서울특별시   2010    10312545
        2020     9720846
부산광역시   2010     2567910
        2020     3404423
인천광역시   2010     2758296
        2020     2947217
대구광역시   2010     2511676
        2020     2427954
대전광역시   2010     1503664
        2020     1471040
광주광역시   2010     1454636
        2020     1455048
dtype: int64
```

```
population['인천광역시', 2010]
```

```
2758296
```

```
population[:, 2010]
```

```
행정구역
서울특별시    10312545
부산광역시     2567910
인천광역시     2758296
대구광역시     2511676
대전광역시     1503664
광주광역시     1454636
dtype: int64
```



```
population[population > 3000000]
```

```
행정구역    년도
서울특별시   2010    10312545
        2020     9720846
부산광역시   2020     3404423
dtype: int64
```

```
population[['대구광역시', '대전광역시']]
```

```
행정구역    년도
대구광역시   2010    2511676
        2020    2427954
대전광역시   2010    1503664
        2020    1471040
dtype: int64
```

```
mdf
```

	col_name1	c1		c2		c3	
	col_name2	1	2	1	2	1	2
name1	name2						
a	1	-0.13	-1.66	0.58	-0.28	0.58	1.08
	2	2.08	-0.30	0.10	1.12	0.60	-0.08
b	1	0.68	-2.13	0.92	0.65	0.88	-0.02
	2	0.26	-2.44	1.27	0.65	1.68	0.97
c	1	-0.26	0.08	0.28	0.78	0.95	-0.11
	2	2.69	-0.62	0.60	-0.67	0.25	0.40

```
mdf['c2', 1]
```

```
name1  name2
a      1        0.58
       2        0.10
b      1        0.92
       2        1.27
c      1        0.28
       2        0.60
Name: (c2, 1), dtype: float64
```



```
mdf.iloc[:3, :4]
```

	col_name1	c1		c2	
	col_name2	1	2	1	2
name1	name2				
a	1	-0.13	-1.66	0.58	-0.28
	2	2.08	-0.30	0.10	1.12
b	1	0.68	-2.13	0.92	0.65

```
mdf.loc[:, ('c2', 1)]
```

```
name1  name2
a      1        0.58
       2        0.10
b      1        0.92
       2        1.27
c      1        0.28
       2        0.60
Name: (c2, 1), dtype: float64
```

```
idx_slice = pd.IndexSlice
mdf.loc[idx_slice[:, 2], idx_slice[:, 2]]
```

	col_name1	c1	c2	c3
	col_name2	2	2	2
name1	name2			
a	2	-0.30	1.12	-0.08
b	2	-2.44	0.65	0.97
c	2	-0.62	-0.67	0.40

다중 인덱스 재정렬

```
idx
```

```
MultiIndex([('a', 1),
            ('a', 2),
            ('b', 1),
            ('b', 2),
            ('c', 1),
            ('c', 2)],
           names=['name1', 'name2'])
```

```
korea_mdf
```

		총인구수	남자인구수	여자인구수	남여비율
행정구역	년도				
서울특별시	2010	10312545	5111259	5201286	98.269140
	2020	9720846	4732275	4988571	94.862336
부산광역시	2010	2567910	1773170	1794740	98.798155
	2020	3404423	1668618	1735805	96.129346
인천광역시	2010	2758296	1390356	1367940	101.638668
	2020	2947217	1476813	1470404	100.435867
대구광역시	2010	2511676	1255245	1256431	99.905606
	2020	2427954	1198815	1229139	97.532907
대전광역시	2010	1503664	753648	750016	100.484256
	2020	1471040	734441	736599	99.707032
광주광역시	2010	1454636	721780	732856	98.488653
	2020	1455048	720060	734988	97.968946


```
korea_mdf = korea_mdf.sort_index()
korea_mdf
```

행정구역	년도	총인구수	남자인구수	여자인구수	남여비율
광주광역시	2010	1454636	721780	732856	98.488653
	2020	1455048	720060	734988	97.968946
대구광역시	2010	2511676	1255245	1256431	99.905606
	2020	2427954	1198815	1229139	97.532907
대전광역시	2010	1503664	753648	750016	100.484256
	2020	1471040	734441	736599	99.707032
부산광역시	2010	2567910	1773170	1794740	98.798155
	2020	3404423	1668618	1735805	96.129346
서울특별시	2010	10312545	5111259	5201286	98.269140
	2020	9720846	4732275	4988571	94.862336
인천광역시	2010	2758296	1390356	1367940	101.638668

```
korea_mdf['서울특별시':'인천광역시']
```

행정구역	년도	총인구수	남자인구수	여자인구수	남여비율
서울특별시	2010	10312545	5111259	5201286	98.269140
	2020	9720846	4732275	4988571	94.862336
인천광역시	2010	2758296	1390356	1367940	101.638668
	2020	2947217	1476813	1470404	100.435867


```
korea_mdf.unstack(level=0)
```

	총인구수						남자인구수		
행정구역	광주광역시	대구광역시	대전광역시	부산광역시	서울특별시	인천광역시	광주광역시	대구광역시	대전광역시
년도									
2010	1454636	2511676	1503664	2567910	10312545	2758296	721780	1255245	75364
2020	1455048	2427954	1471040	3404423	9720846	2947217	720060	1198815	73444

```
korea_mdf.unstack(level=1)
```

	총인구수		남자인구수		여자인구수		남여비율	
년도	2010	2020	2010	2020	2010	2020	2010	2020
행정구역								
광주광역시	1454636	1455048	721780	720060	732856	734988	98.488653	97.968946
대구광역	2511676	2427954	1255245	1198815	1256431	1229139	99.905606	97.532907

```
korea_mdf.stack()
```

```
행정구역     년도
광주광역시   2010  총인구수       1.454636e+06
                   남자인구수      7.217800e+05
                   여자인구수      7.328560e+05
                   남여비율       9.848865e+01
             2020  총인구수       1.455048e+06
                   남자인구수      7.200600e+05
                   여자인구수      7.349880e+05
                   남여비율       9.796895e+01
대구광역시   2010  총인구수       2.511676e+06
                   남자인구수      1.255245e+06
                   여자인구수      1.256431e+06
                   남여비율       9.990561e+01
             2020  총인구수       2.427954e+06
                   남자인구수      1.198815e+06
                   여자인구수      1.229139e+06
                   남여비율       9.753291e+01
대전광역시   2010  총인구수       1.503664e+06
                   남자인구수      7.536480e+05
                   여자인구수      7.500160e+05
                   남여비율       1.004843e+02
             2020  총인구수       1.471040e+06
                   남자인구수      7.344410e+05
                   여자인구수      7.365990e+05
                   남여비율       9.970703e+01
부산광역시   2010  총인구수       2.567910e+06
                   남자인구수      1.773170e+06
                   여자인구수      1.794740e+06
                   남여비율       9.879815e+01
             2020  총인구수       3.404423e+06
                   남자인구수      1.668618e+06
                   여자인구수      1.735805e+06
                   남여비율       9.612935e+01
서울특별시   2010  총인구수       1.031254e+07
                   남자인구수      5.111259e+06
                   여자인구수      5.201286e+06
                   남여비율       9.826914e+01
             2020  총인구수       9.720846e+06
                   남자인구수      4.732275e+06
                   여자인구수      4.988571e+06
                   남여비율       9.486234e+01
인천광역시   2010  총인구수       2.758296e+06
                   남자인구수      1.390356e+06
                   여자인구수      1.367940e+06
                   남여비율       1.016387e+02
```

```
        2020  총인구수       2.947217e+06
              남자인구수     1.476813e+06
              여자인구수     1.470404e+06
              남여비율       1.004359e+02
dtype: float64
```

```
korea_mdf
```

행정구역	년도	총인구수	남자인구수	여자인구수	남여비율
광주광역시	**2010**	1454636	721780	732856	98.488653
	2020	1455048	720060	734988	97.968946
대구광역시	**2010**	2511676	1255245	1256431	99.905606
	2020	2427954	1198815	1229139	97.532907
대전광역시	**2010**	1503664	753648	750016	100.484256
	2020	1471040	734441	736599	99.707032
부산광역시	**2010**	2567910	1773170	1794740	98.798155
	2020	3404423	1668618	1735805	96.129346
서울특별시	**2010**	10312545	5111259	5201286	98.269140
	2020	9720846	4732275	4988571	94.862336
인천광역시	**2010**	2758296	1390356	1367940	101.638668
	2020	2947217	1476813	1470404	100.435867


```
idx_flat = korea_mdf.reset_index(level=0)
idx_flat
```

행정구역	총인구수	남자인구수	여자인구수	남여비율
년도				

```
idx_flat = korea_mdf.reset_index(level=(0, 1))
idx_flat
```

	행정구역	년도	총인구수	남자인구수	여자인구수	남여비율
0	광주광역시	2010	1454636	721780	732856	98.488653
1	광주광역시	2020	1455048	720060	734988	97.968946
2	대구광역시	2010	2511676	1255245	1256431	99.905606
3	대구광역시	2020	2427954	1198815	1229139	97.532907
4	대전광역시	2010	1503664	753648	750016	100.484256
5	대전광역시	2020	1471040	734441	736599	99.707032
6	부산광역시	2010	2567910	1773170	1794740	98.798155
7	부산광역시	2020	3404423	1668618	1735805	96.129346
8	서울특별시	2010	10312545	5111259	5201286	98.269140
9	서울특별시	2020	9720846	4732275	4988571	94.862336
10	인천광역시	2010	2758296	1390356	1367940	101.638668
11	인천광역시	2020	2947217	1476813	1470404	100.435867


```
idx_flat.set_index(['행정구역', '년도'])
```

데이터 연산

광주광역시	2010	1454636	721780	732856	98.488653

```
s = pd.Series(np.random.randint(0, 10, 5))
s
```

```
0    6
1    4
2    4
3    8
4    3
dtype: int64
```

부산광역시	2010	2567910	1773170	1794740	98.798135

```
df = pd.DataFrame(np.random.randint(0, 10, (3, 3)),
                  columns=['A', 'B', 'C'])
df
```

	A	B	C
0	7	0	5
1	0	3	6
2	8	4	6


```
np.exp(s)
```

```
0     403.428793
1      54.598150
2      54.598150
3    2980.957987
4      20.085537
dtype: float64
```

```
np.cos(df * np.pi / 4)
```

	A	B	C
0	0.707107	1.000000	-7.071068e-01
1	1.000000	-0.707107	-1.836970e-16
2	1.000000	-1.000000	-1.836970e-16

```
s1 = pd.Series([1, 3, 5, 7, 9], index=[0, 1, 2, 3, 4])
s2 = pd.Series([2, 4, 6, 8, 10], index=[1, 2, 3, 4, 5])
s1 + s2
```

```
0     NaN
1     5.0
2     9.0
3    13.0
```

```
4    17.0
5     NaN
dtype: float64
```

```
s1.add(s2, fill_value=0)
```

```
0     1.0
1     5.0
2     9.0
3    13.0
4    17.0
5    10.0
dtype: float64
```

```
df1 = pd.DataFrame(np.random.randint(0, 20, (3, 3)),
                   columns=list('ACD'))
df1
```

	A	C	D
0	3	4	9
1	6	19	9
2	3	6	0


```
df2 = pd.DataFrame(np.random.randint(0, 20, (5, 5)),
                   columns=list('BAECD'))
df2
```

	B	A	E	C	D
0	13	14	9	0	5
1	14	6	2	14	13
2	11	14	19	17	3
3	18	18	8	7	13
4	14	15	2	6	1

```
df1 + df2
```

	A	B	C	D	E
0	17.0	NaN	4.0	14.0	NaN
1	12.0	NaN	33.0	22.0	NaN
2	17.0	NaN	23.0	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN

```
fvalue = df1.stack().mean()
df1.add(df2, fill_value=fvalue)
```

	A	B	C	D	E
0	17.000000	19.555556	4.000000	14.000000	15.555556
1	12.000000	20.555556	33.000000	22.000000	8.555556
2	17.000000	17.555556	23.000000	3.000000	25.555556
3	24.555556	24.555556	13.555556	19.555556	14.555556
4	21.555556	20.555556	12.555556	7.555556	8.555556

연산자 범용 함수

Python 연산자	Pandas 메소드
+	add, radd
-	sub, rsub, subtract
*	mul, rmul, multiply
/	truediv, div, rdiv, divide
//	floordiv, rfloordiv
%	mod
**	pow, rpow

add()

```
a = np.random.randint(1, 10, size=(3, 3))
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a + a[0]
```

```
array([[14, 16, 12],
       [14, 13, 12],
       [14, 11, 14]])
```

```
df = pd.DataFrame(a, columns=list('ABC'))
df
```

```
df + df.iloc[0]
```

	A	B	C
0	14	16	12
1	14	13	12
2	14	11	14

```
df.add(df.iloc[0])
```

	A	B	C
0	14	16	12
1	14	13	12
2	14	11	14

sub() / subtract()


```
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a - a[0]
```

```
array([[ 0,  0,  0],
       [ 0, -3,  0],
       [ 0, -5,  2]])
```

```
df
```

	A	B	C
0	7	8	6
1	7	5	6
2	7	3	8

```
df - df.iloc[0]
```

	A	B	C

```
df.sub(df.iloc[0])
```

	A	B	C
0	0	0	0
1	0	-3	0
2	0	-5	2

```
df.subtract(df['B'], axis=0)
```

	A	B	C
0	-1	0	-2
1	2	0	1
2	4	0	5

mul() / multply()


```
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a * a[1]
```

```
array([[49, 40, 36],
       [49, 25, 36],
       [49, 15, 48]])
```

```
df
```

	A	B	C
0	7	8	6
1	7	5	6
2	7	3	8

```
df * df.iloc[1]
```

	A	B	C
0	49	40	36

```
df.mul(df.iloc[1])
```

	A	B	C
0	49	40	36
1	49	25	36
2	49	15	48

```
df.multiply(df.iloc[2])
```

	A	B	C
0	49	24	48
1	49	15	48
2	49	9	64

truediv() / div() / divide() / floordiv()

```
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a / a[0]
```

```
array([[1.        , 1.        , 1.        ],
       [1.        , 0.625     , 1.        ],
       [1.        , 0.375     , 1.33333333]])
```

```
df
```

	A	B	C
0	7	8	6
1	7	5	6
2	7	3	8

```
df / df.iloc[0]
```

	A	B	C
0	1.0	1.000	1.000000
1	1.0	0.625	1.000000

```
df.truediv(df.iloc[0])
```

	A	B	C
0	1.0	1.000	1.000000
1	1.0	0.625	1.000000
2	1.0	0.375	1.333333

```
df.div(df.iloc[1])
```

	A	B	C
0	1.0	1.6	1.000000
1	1.0	1.0	1.000000
2	1.0	0.6	1.333333


```
df.divide(df.iloc[2])
```

	A	B	C
0	1.0	2.666667	0.75
1	1.0	1.666667	0.75
2	1.0	1.000000	1.00

```
a // a[0]
```

```
array([[1, 1, 1],
       [1, 0, 1],
       [1, 0, 1]])
```

```
df.floordiv(df.iloc[0])
```

	A	B	C
0	1	1	1
1	1	0	1
2	1	0	1

mod()

```
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a % a[0]
```

```
array([[0, 0, 0],
       [0, 5, 0],
       [0, 3, 2]])
```

```
df
```

	A	B	C
0	7	8	6
1	7	5	6
2	7	3	8

```
df.mod(df.iloc[0])
```

	A	B	C
0	0	0	0
1	0	5	0
2	0	3	2

pow()

```
a
```

```
array([[7, 8, 6],
       [7, 5, 6],
       [7, 3, 8]])
```

```
a ** a[0]
```

```
array([[  823543, 16777216,    46656],
       [  823543,   390625,    46656],
       [  823543,     6561,   262144]])
```

```
df
```

	A	B	C
0	7	8	6

```
df.pow(df.iloc[0])
```

	A	B	C
0	823543	16777216	46656
1	823543	390625	46656
2	823543	6561	262144

```
row = df.iloc[0, ::2]
row
```

```
A    7
C    6
Name: 0, dtype: int64
```

```
df - row
```

	A	B	C
0	0.0	NaN	0.0
1	0.0	NaN	0.0
2	0.0	NaN	2.0

정렬(Sort)

```
s = pd.Series(range(5), index=['A', 'D', 'B', 'C', 'E'])
s
```

```
A    0
D    1
B    2
C    3
E    4
dtype: int64
```

```
s.sort_index()
```

```
A    0
B    2
C    3
D    1
E    4
dtype: int64
```

```
s.sort_values()
```

```
A    0
D    1
B    2
C    3
E    4
dtype: int64
```

```
df = pd.DataFrame(np.random.randint(0, 10, (4, 4)),
                  index=[2, 4, 1, 3],
                  columns=list('BDAC'))
df
```

	B	D	A	C
2	0	9	4	4
4	8	0	4	6
1	2	1	8	0
3	3	5	1	6

```
df.sort_index()
```

	B	D	A	C
1	2	1	8	0
2	0	9	4	4
3	3	5	1	6
4	8	0	4	6


```
df.sort_index(axis=1)
```

	A	B	C	D
2	4	0	4	9
4	4	8	6	0
1	8	2	0	1
3	1	3	6	5

```
df.sort_values(by='A')
```

	B	D	A	C
3	3	5	1	6

```
df.sort_values(by=['A', 'C'])
```

	B	D	A	C
3	3	5	1	6
2	0	9	4	4
4	8	0	4	6
1	2	1	8	0

순위(Ranking)

메소드	설명
average	기본값. 순위에 같은 값을 가지는 항목들의 평균값을 사용
min	같은 값을 가지는 그룹을 낮은 순위로 지정
max	같은 값을 가지는 그룹을 높은 순위로 지정
first	데이터 내의 위치에 따라 순위 지정
dense	같은 그룹 내에서 모두 같은 순위를 적용하지 않고 1씩 증가


```
s = pd.Series([-2, 4, 7, 3, 0, 7, 5, -4, 2, 6])
s
```

```
0   -2
1    4
2    7
3    3
4    0
5    7
6    5
7   -4
8    2
9    6
dtype: int64
```

```
s.rank()
```

```
0    2.0
1    6.0
2    9.5
3    5.0
4    3.0
5    9.5
6    7.0
7    1.0
8    4.0
9    8.0
dtype: float64
```

```
s.rank(method='first')
```

```
s.rank(method='first')
```

```
0     2.0
1     6.0
2     9.0
3     5.0
4     3.0
5    10.0
6     7.0
7     1.0
8     4.0
9     8.0
dtype: float64
```

```
s.rank(method='max')
```

```
0     2.0
1     6.0
2    10.0
3     5.0
4     3.0
5    10.0
6     7.0
7     1.0
8     4.0
9     8.0
dtype: float64
```


고성능 연산

```
nrows, ncols = 100000, 100
df1, df2, df3, df4 = (pd.DataFrame(np.random.rand(nrows, ncols)) for i in range(4))
```

```
%timeit df1 + df2 + df3 + df4
```

```
10 loops, best of 3: 62.7 ms per loop
```

```
%timeit pd.eval('df1 + df2 + df3 + df4')
```

```
10 loops, best of 3: 41.5 ms per loop
```

```
%timeit df1 * -df2 / (-df3 * df4)
```

```
10 loops, best of 3: 94.4 ms per loop
```

```
%timeit pd.eval('df1 * -df2 / (-df3 * df4)')
```

```
10 loops, best of 3: 52.2 ms per loop
```

```
%timeit (df1 < df2) & (df2 <= df3) & (df3 != df4)
```

```
1 loop, best of 3: 330 ms per loop
```

```
%timeit pd.eval('(df1 < df2) & (df2 <= df3) & (df3 != df4)')
```

```
10 loops, best of 3: 93.1 ms per loop
```

```
df = pd.DataFrame(np.random.rand(1000000, 5), columns=['A', 'B', 'C', 'D', 'E'])
df.head()
```

	A	B	C	D	E
0	0.809568	0.914763	0.602827	0.486493	0.535205
1	0.271222	0.670036	0.400662	0.525909	0.161296
2	0.651753	0.925138	0.942498	0.087579	0.216714
3	0.061394	0.784605	0.952853	0.435291	0.176985
4	0.615251	0.195590	0.981553	0.741863	0.096994

```
%timeit df['A'] + df['B'] / df['C'] - df['D'] * df['E']
```

```
10 loops, best of 3: 15.9 ms per loop
```



```
%timeit pd.eval('df.A + df.B / df.C - df.D * df.E')
```

```
100 loops, best of 3: 8.23 ms per loop
```

```
%timeit df.eval('A + B / C - D * E')
```

```
100 loops, best of 3: 12.9 ms per loop
```

```
df.eval('R = A + B / C - D * E', inplace=True)
df.head()
```

	A	B	C	D	E	R
0	0.809568	0.914763	0.602827	0.486493	0.535205	2.066649
1	0.271222	0.670036	0.400662	0.525909	0.161296	1.858715
2	0.651753	0.925138	0.942498	0.087579	0.216714	1.614353
3	0.061394	0.784605	0.952853	0.435291	0.176985	0.807782
4	0.615251	0.195590	0.981553	0.741863	0.096994	0.742560

```
df.eval('R = A - B / C + D * E', inplace=True)
df.head()
```

	A	B	C	D	E	R
0	0.809568	0.914763	0.602827	0.486493	0.535205	-0.447513
1	0.271222	0.670036	0.400662	0.525909	0.161296	-1.316270
2	0.651753	0.925138	0.942498	0.087579	0.216714	-0.310848

```
col_mean = df.mean(1)
df['A'] + col_mean
```

```
0          1.293125
1          0.390031
2          1.070559
3          0.349084
4          1.135117
             ...
999995     0.796291
999996     1.345668
999997     0.273292
999998     0.872099
999999     1.150862
Length: 1000000, dtype: float64
```

```
df.eval('A + @col_mean')
```

```
0          1.293125
1          0.390031
2          1.070559
3          0.349084
4          1.135117
             ...
999995     0.796291
999996     1.345668
999997     0.273292
999998     0.872099
999999     1.150862
Length: 1000000, dtype: float64
```



```
df[(df.A < 0.5) & (df.B < 0.5) & (df.C > 0.5)]
```

	A	B	C	D	E	R
21	0.378554	0.318906	0.503215	0.956104	0.453821	0.178717
22	0.386906	0.078570	0.912289	0.227901	0.963534	0.520373
30	0.317711	0.030924	0.885734	0.520469	0.844921	0.722554

```
pd.eval('df[(df.A < 0.5) & (df.B < 0.5) & (df.C > 0.5)]')
```

	A	B	C	D	E	R
21	0.378554	0.318906	0.503215	0.956104	0.453821	0.178717
22	0.386906	0.078570	0.912289	0.227901	0.963534	0.520373
30	0.317711	0.030924	0.885734	0.520469	0.844921	0.722554
31	0.430064	0.165408	0.743194	0.724131	0.252827	0.390580
44	0.294310	0.379110	0.506650	0.357761	0.258580	-0.361448
...	...	...	...	...	...	...
999925	0.175797	0.497737	0.681946	0.186990	0.445728	-0.470734
999971	0.157704	0.092293	0.994735	0.982039	0.014898	0.079552
999974	0.425083	0.185591	0.924796	0.611062	0.777083	0.699247
999976	0.248551	0.162089	0.633810	0.471437	0.105643	0.042617
999983	0.307258	0.271711	0.863673	0.060059	0.422888	0.018057

124861 rows × 6 columns


```
df.query('(A < 0.5) and (B < 0.5) and (C > 0.5)')
```

	A	B	C	D	E	R
21	0.378554	0.318906	0.503215	0.956104	0.453821	0.178717
22	0.386906	0.078570	0.912289	0.227901	0.963534	0.520373
30	0.317711	0.030924	0.885734	0.520469	0.844921	0.722554
31	0.430064	0.165408	0.743194	0.724131	0.252827	0.390580
44	0.294310	0.379110	0.506650	0.357761	0.258580	-0.361448
...	...	...	...	...	...	...
999925	0.175797	0.497737	0.681946	0.186990	0.445728	-0.470734
999971	0.157704	0.092293	0.994735	0.982039	0.014898	0.079552
999974	0.425083	0.185591	0.924796	0.611062	0.777083	0.699247
999976	0.248551	0.162089	0.633810	0.471437	0.105643	0.042617
999983	0.307258	0.271711	0.863673	0.060059	0.422888	0.018057

124861 rows × 6 columns

```
col_mean = df['D'].mean()
df[(df.A < col_mean) & (df.B < col_mean)]
```

	A	B	C	D	E	R
9	0.206366	0.332854	0.452255	0.587352	0.250940	-0.382231
16	0.186270	0.420888	0.122973	0.805942	0.109413	-3.148163
21	0.378554	0.318906	0.503215	0.956104	0.453821	0.178717
22	0.386906	0.078570	0.912289	0.227901	0.963534	0.520373
24	0.445190	0.267078	0.088809	0.940300	0.577578	-2.019055
...	...	...	...	...	...	...
999975	0.122675	0.385851	0.231283	0.880432	0.064430	-1.488905
999976	0.248551	0.162089	0.633810	0.471437	0.105643	0.042617
999980	0.059664	0.415475	0.017191	0.681780	0.874583	-23.512904
999983	0.307258	0.271711	0.863673	0.060059	0.422888	0.018057
999988	0.457082	0.009105	0.082372	0.001740	0.118373	0.346750

249846 rows × 6 columns


```
df.query('A < @col_mean and B < @col_mean')
```

	A	B	C	D	E	R
9	0.206366	0.332854	0.452255	0.587352	0.250940	-0.382231
16	0.186270	0.420888	0.122973	0.805942	0.109413	-3.148163
21	0.378554	0.318906	0.503215	0.956104	0.453821	0.178717
22	0.386906	0.078570	0.912289	0.227901	0.963534	0.520373
24	0.445190	0.267078	0.088809	0.940300	0.577578	-2.019055
...	...	...	...	...	...	...
999975	0.122675	0.385851	0.231283	0.880432	0.064430	-1.488905
999976	0.248551	0.162089	0.633810	0.471437	0.105643	0.042617
999980	0.059664	0.415475	0.017191	0.681780	0.874583	-23.512904
999983	0.307258	0.271711	0.863673	0.060059	0.422888	0.018057
999988	0.457082	0.009105	0.082372	0.001740	0.118373	0.346750

249846 rows × 6 columns

데이터 결합

Concat() / Append()

```
s1 = pd.Series(['a', 'b'], index=[1, 2])
s2 = pd.Series(['c', 'd'], index=[3, 4])
pd.concat([s1, s2])
```

```
1    a
2    b
3    c
4    d
dtype: object
```

```
def create_df(cols, idx):
  data = {c: [str(c.lower()) + str(i) for i in idx] for c in cols}
  return pd.DataFrame(data, idx)
```

```
df1 = create_df('AB', [1, 2])
df1
```

	A	B
1	a1	b1
2	a2	b2


```
df2 = create_df('AB', [3, 4])
df2
```

	A	B
3	a3	b3
4	a4	b4

```
pd.concat([df1, df2])
```

	A	B
1	a1	b1
2	a2	b2
3	a3	b3
4	a4	b4

```
df3 = create_df('AB', [0, 1])
df3
```

	A	B
0	a0	b0

```
df4 = create_df('CD', [0, 1])
df4
```

	C	D
0	c0	d0
1	c1	d1

```
pd.concat([df3, df4])
```

	A	B	C	D
0	a0	b0	NaN	NaN
1	a1	b1	NaN	NaN
0	NaN	NaN	c0	d0
1	NaN	NaN	c1	d1


```
pd.concat([df3, df4], axis=1)
```

	A	B	C	D
0	a0	b0	c0	d0
1	a1	b1	c1	d1

```
pd.concat([df1, df3])
```

	A	B
1	a1	b1
2	a2	b2
0	a0	b0
1	a1	b1

```
#pd.concat([df1, df3], verify_integrity=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-239-09c85839a3e0> in <module>()
----> 1 pd.concat([df1, df3], verify_integrity=True)

 5 frames 

/usr/local/lib/python3.6/dist-packages/pandas/core/reshape/concat.py in
_maybe_check_integrity(self, concat_index)
    580                 raise ValueError(
    581                     "Indexes have overlapping values: "
--> 582                     "{overlap!s}".format(overlap=overlap)
```

```
pd.concat([df1, df3], ignore_index=True)
```

	A	B
0	a1	b1
1	a2	b2
2	a0	b0
3	a1	b1

```
pd.concat([df1, df3], keys=['X', 'Y'])
```

		A	B
X	1	a1	b1
	2	a2	b2
Y	0	a0	b0
	1	a1	b1


```
df5 = create_df('ABC', [1, 2])
df6 = create_df('BCD', [3, 4])
pd.concat([df5, df6])
```

	A	B	C	D
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	NaN	b3	c3	d3
4	NaN	b4	c4	d4

```
pd.concat([df5, df6], join='inner')
```

	B	C
1	b1	c1

```
df5.append(df6)
```

	A	B	C	D
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	NaN	b3	c3	d3
4	NaN	b4	c4	d4

병합과 조인

```
df1 = pd.DataFrame({'학생': ['홍길동', '이순신', '임꺽정', '김유신'],
                    '학과': ['경영학과', '교육학과', '컴퓨터학과', '통계학과']})
df1
```

	학생	학과
0	홍길동	경영학과
1	이순신	교육학과
2	임꺽정	컴퓨터학과
3	김유신	통계학과


```
df2 = pd.DataFrame({'학생': ['홍길동', '이순신', '임꺽정', '김유신'],
                    '입학년도': [2012, 2016, 2019, 2020]})
df2
```

	학생	입학년도
0	홍길동	2012
1	이순신	2016
2	임꺽정	2019
3	김유신	2020

```
df3 = pd.merge(df1, df2)
df3
```

	학생	학과	입학년도
0	홍길동	경영학과	2012
1	이순신	교육학과	2016

```
df4 = pd.DataFrame({'학과': ['경영학과', '교육학과', '컴퓨터학과', '통계학과'],
                    '학과장': ['황희', '장영실', '안창호', '정약용']})
df4
```

	학과	학과장
0	경영학과	황희
1	교육학과	장영실
2	컴퓨터학과	안창호
3	통계학과	정약용

```
pd.merge(df3, df4)
```

	학생	학과	입학년도	학과장
0	홍길동	경영학과	2012	황희
1	이순신	교육학과	2016	장영실
2	임꺽정	컴퓨터학과	2019	안창호
3	김유신	통계학과	2020	정약용


```
df5 = pd.DataFrame({'학과': ['경영학과', '교육학과', '교육학과', '컴퓨터학과', '컴퓨터학과', '통계
                    '과목': ['경영개론', '기초수학', '물리학', '프로그래밍', '운영체제', '확률론']
df5
```

	학과	과목
0	경영학과	경영개론
1	교육학과	기초수학
2	교육학과	물리학
3	컴퓨터학과	프로그래밍
4	컴퓨터학과	운영체제
5	통계학과	확률론

```
pd.merge(df1, df5)
```

	학생	학과	과목
0	홍길동	경영학과	경영개론
1	이순신	교육학과	기초수학
2	이순신	교육학과	물리학

```
pd.merge(df1, df2, on='학생')
```

	학생	학과	입학년도
0	홍길동	경영학과	2012
1	이순신	교육학과	2016
2	임꺽정	컴퓨터학과	2019
3	김유신	통계학과	2020

```
df6 = pd.DataFrame({'이름': ['홍길동', '이순신', '임꺽정', '김유신'],
                    '성적': ['A', 'A+', 'B', 'A+']})
df6
```

	이름	성적
0	홍길동	A
1	이순신	A+
2	임꺽정	B
3	김유신	A+


```
pd.merge(df1, df6, left_on="학생", right_on="이름")
```

	학생	학과	이름	성적
0	홍길동	경영학과	홍길동	A
1	이순신	교육학과	이순신	A+
2	임꺽정	컴퓨터학과	임꺽정	B
3	김유신	통계학과	김유신	A+

```
pd.merge(df1, df6, left_on="학생", right_on="이름").drop("이름", axis=1)
```

학생 학과 성적

```
mdf1 = df1.set_index('학생')
mdf2 = df2.set_index('학생')
```

1 이순신 교육학과 A+

```
mdf1
```

학생	학과
홍길동	경영학과
이순신	교육학과
임꺽정	컴퓨터학과
김유신	통계학과

```
mdf2
```

학생	입학년도
홍길동	2012
이순신	2016
임꺽정	2019
김유신	2020


```
pd.merge(mdf1, mdf2, left_index=True, right_index=True)
```

학생	학과	입학년도
홍길동	경영학과	2012
이순신	교육학과	2016
임꺽정	컴퓨터학과	2019
김유신	통계학과	2020

```
mdf1.join(mdf2)
```

	학과	입학년도
학생		

```
pd.merge(mdf1, df6, left_index=True, right_on='이름')
```

	학과	이름	성적
0	경영학과	홍길동	A
1	교육학과	이순신	A+
2	컴퓨터학과	임꺽정	B
3	통계학과	김유신	A+

```
df7 = pd.DataFrame({'이름': ['홍길동', '이순신', '임꺽정'],
                    '주문음식': ['햄버거', '피자', '짜장면']})
df7
```

	이름	주문음식
0	홍길동	햄버거
1	이순신	피자
2	임꺽정	짜장면


```
df8 = pd.DataFrame({'이름': ['홍길동', '이순신', '김유신'],
                    '주문음료': ['콜라', '사이다', '커피']})
df8
```

	이름	주문음료
0	홍길동	콜라
1	이순신	사이다
2	김유신	커피

```
pd.merge(df7, df8)
```

	이름	주문음식	주문음료
0	홍길동	햄버거	콜라
1	이순신	피자	사이다

```
pd.merge(df7, df8, how='inner')
```

	이름	주문음식	주문음료

```
pd.merge(df7, df8, how='outer')
```

	이름	주문음식	주문음료
0	홍길동	햄버거	콜라
1	이순신	피자	사이다
2	임꺽정	짜장면	NaN
3	김유신	NaN	커피

```
pd.merge(df7, df8, how='left')
```

	이름	주문음식	주문음료
0	홍길동	햄버거	콜라
1	이순신	피자	사이다
2	임꺽정	짜장면	NaN

```
pd.merge(df7, df8, how='right')
```

	이름	주문음식	주문음료
0	홍길동	햄버거	콜라
1	이순신	피자	사이다
2	김유신	NaN	커피


```
df9 = pd.DataFrame({'이름': ['홍길동', '이순신', '임꺽정', '김유신'],
                    '순위': [3, 2, 4, 1]})
df9
```

	이름	순위
0	홍길동	3
1	이순신	2
2	임꺽정	4
3	김유신	1

```
df10 = pd.DataFrame({'이름': ['홍길동', '이순신', '임꺽정', '김유신'],
                     '순위': [4, 1, 3, 2]})
df10
```

	이름	순위
0	홍길동	4
1	이순신	1

```
pd.merge(df9, df10, on='이름')
```

	이름	순위_x	순위_y
0	홍길동	3	4
1	이순신	2	1
2	임꺽정	4	3
3	김유신	1	2

```
pd.merge(df9, df10, on='이름', suffixes=["_인기", "_성적"])
```

	이름	순위_인기	순위_성적
0	홍길동	3	4
1	이순신	2	1
2	임꺽정	4	3
3	김유신	1	2

데이터 집계와 그룹 연산

집계 연산(Aggregation)

집계	설명
`count`	전체 개수
`head`, `tail`	앞의 항목 일부 반환, 뒤의 항목 일부 반환
`describe`	Series, DataFrame의 각 컬럼에 대한 요약 통계
`min`, `max`	최소값, 최대값
`cummin`, `cummax`	누적 최소값, 누적 최대값
`argmin`, `argmax`	최소값과 최대값의 색인 위치
`idxmin`, `idxmax`	최소값과 최대값의 색인값
`mean`, `median`	평균값, 중앙값
`std`, `var`	표준편차(Standard deviation), 분산(Variance)
`skew`	왜도(skewness) 값 계산
`kurt`	첨도(kurtosis) 값 계산
`mad`	절대 평균 편차(Mean Absolute Deviation)
`sum`, `cumsum`	전체 항목 합, 누적합
`prod`, `cumprod`	전체 항목 곱, 누적곱

집계	설명
quantile	0부터 1까지의 분위수 계산
diff	1차 산술차 계산
pct_change	퍼센트 변화율 계산
corr, cov	상관관계, 공분산 계산

```
df = pd.DataFrame([[1, 1.2, np.nan],
                   [2.4, 5.5, 4.2],
                   [np.nan, np.nan, np.nan],
                   [0.44, -3.1, -4.1]],
                  index=[1, 2, 3, 4],
                  columns=['A', 'B', 'C'])
df
```

	A	B	C
1	1.00	1.2	NaN
2	2.40	5.5	4.2
3	NaN	NaN	NaN
4	0.44	-3.1	-4.1


```
df.head(2)
```

	A	B	C
1	1.0	1.2	NaN
2	2.4	5.5	4.2

```
df.tail(2)
```

	A	B	C
3	NaN	NaN	NaN
4	0.44	-3.1	-4.1

```
df.describe()
```

	A	B	C
count	3.000000	3.00	2.000000
mean	1.280000	1.20	0.050000

```
print(df)
print(np.argmin(df), np.argmax(df))
```

```
     A    B    C
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
2 2
```

max	2.400000	5.50	4.200000

```
print(df)
print(df.idxmin())
print(df.idxmax())
```

```
     A    B    C
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
A    4
B    4
C    4
dtype: int64
A    2
B    2
C    2
dtype: int64
```



```
print(df)
print(df.std())
print(df.var())
```

```
     A    B    C
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
A    1.009554
B    4.300000
C    5.868986
dtype: float64
A     1.0192
B    18.4900
C    34.4450
dtype: float64
```

```
print(df)
print(df.skew())
print(df.kurt())
```

```
     A    B    C
```

```
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
A    1.15207
B    0.00000
C        NaN
dtype: float64
A   NaN
B   NaN
C   NaN
dtype: float64
```

```
print(df)
print(df.sum())
print(df.cumsum())
```

```
       A    B    C
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
A    3.84
B    3.60
C    0.10
dtype: float64
       A    B    C
1  1.00  1.2  NaN
2  3.40  6.7  4.2
3   NaN  NaN  NaN
4  3.84  3.6  0.1
```



```
print(df)
print(df.prod())
print(df.cumprod())
```

```
       A    B    C
1  1.00  1.2  NaN
2  2.40  5.5  4.2
3   NaN  NaN  NaN
4  0.44 -3.1 -4.1
A     1.056
B   -20.460
C   -17.220
dtype: float64
       A      B      C
1  1.000   1.20    NaN
2  2.400   6.60   4.20
3    NaN    NaN    NaN
4  1.056 -20.46 -17.22
```

```
df.diff()
```

	A	B	C
1	NaN	NaN	NaN
2	1.4	4.3	NaN

```
df.quantile()
```

```
A    1.00
B    1.20
C    0.05
Name: 0.5, dtype: float64
```

```
df.pct_change()
```

	A	B	C
1	NaN	NaN	NaN
2	1.400000	3.583333	NaN
3	0.000000	0.000000	0.00000
4	-0.816667	-1.563636	-1.97619


```
df.corr()
```

	A	B	C
A	1.000000	0.970725	1.0
B	0.970725	1.000000	1.0
C	1.000000	1.000000	1.0

```
df.corrwith(df.B)
```

```
A    0.970725
B    1.000000
C    1.000000
dtype: float64
```

```
df.cov()
```

	A	B	C
A	1.0192	4.214	8.134
B	4.2140	18.490	35.690
C	8.1340	35.690	34.445

```
df['B'].unique()
```

```
array([ 1.2,  5.5,  nan, -3.1])
```

```
df['A'].value_counts()
```

```
0.44    1
2.40    1
1.00    1
Name: A, dtype: int64
```

GroupBy 연산

```
df = pd.DataFrame({'c1':['a', 'a', 'b', 'b', 'c', 'd', 'b'],
                   'c2':['A', 'B', 'B', 'A', 'D', 'C', 'C'],
                   'c3':np.random.randint(7),
                   'c4':np.random.random(7)})
df
```

	c1	c2	c3	c4
0	a	A	3	0.082461
1	a	B	3	0.560924
2	b	B	3	0.005640
3	b	A	3	0.207320
4	c	D	3	0.841396
5	d	C	3	0.098452
6	b	C	3	0.965033


```
df.dtypes
```

```
c1     object
c2     object
c3      int64
c4    float64
dtype: object
```

```
df['c3'].groupby(df['c1']).mean()
```

```
c1
a    3
b    3
c    3
d    3
Name: c3, dtype: int64
```

```
df['c4'].groupby(df['c2']).std()
```

```
c2
A    0.088288
B    0.392645
```

```
C    0.612765
D         NaN
Name: c4, dtype: float64
```

```
df['c4'].groupby([df['c1'], df['c2']]).mean()
```

```
c1  c2
a   A     0.082461
    B     0.560924
b   A     0.207320
    B     0.005640
    C     0.965033
c   D     0.841396
d   C     0.098452
Name: c4, dtype: float64
```

```
df['c4'].groupby([df['c1'], df['c2']]).mean().unstack()
```

c2	A	B	C	D
c1				
a	0.082461	0.560924	NaN	NaN
b	0.207320	0.005640	0.965033	NaN
c	NaN	NaN	NaN	0.841396
d	NaN	NaN	0.098452	NaN


```
df.groupby('c1').mean()
```

	c3	c4
c1		
a	3	0.321692
b	3	0.392664
c	3	0.841396
d	3	0.098452

```
df.groupby(['c1', 'c2']).mean()
```

		c3	c4
c1	c2		
a	A	3	0.082461

```
df.groupby(['c1', 'c2']).size()
```

```
c1  c2
a   A     1
    B     1
b   A     1
    B     1
    C     1
c   D     1
d   C     1
dtype: int64
```

```
for c1, group in df.groupby('c1'):
  print(c1)
  print(group)
```

```
a
  c1 c2  c3        c4
0  a  A   3  0.082461
1  a  B   3  0.560924
b
  c1 c2  c3        c4
2  b  B   3  0.005640
3  b  A   3  0.207320
6  b  C   3  0.965033
c
  c1 c2  c3        c4
4  c  D   3  0.841396
d
  c1 c2  c3        c4
5  d  C   3  0.098452
```



```
for (c1, c2), group in df.groupby(['c1', 'c2']):
  print((c1, c2))
  print(group)
```

```
('a', 'A')
  c1 c2  c3        c4
0  a  A   3  0.082461
('a', 'B')
  c1 c2  c3        c4
1  a  B   3  0.560924
('b', 'A')
  c1 c2  c3       c4
3  b  A   3  0.20732
('b', 'B')
  c1 c2  c3       c4
2  b  B   3  0.00564
('b', 'C')
  c1 c2  c3        c4
6  b  C   3  0.965033
('c', 'D')
```

```
  c1 c2  c3        c4
4  c  D   3  0.841396
('d', 'C')
  c1 c2  c3        c4
5  d  C   3  0.098452
```

```
df.groupby(['c1', 'c2'])[['c4']].mean()
```

		c4
c1	**c2**	
a	**A**	0.082461
	B	0.560924
b	**A**	0.207320
	B	0.005640
	C	0.965033
c	**D**	0.841396
d	**C**	0.098452

```
df.groupby('c1')['c3'].quantile()
```

```
c1
a    3.0
b    3.0
c    3.0
d    3.0
Name: c3, dtype: float64
```



```
df.groupby('c1')['c3'].count()
```

```
c1
a    2
b    3
c    1
d    1
Name: c3, dtype: int64
```

```
df.groupby('c1')['c4'].median()
```

```
c1
a    0.321692
b    0.207320
c    0.841396
d    0.098452
Name: c4, dtype: float64
```

```
df.groupby('c1')['c4'].std()
```

```
c1
```

```
a    0.338324
b    0.505839
c         NaN
d         NaN
Name: c4, dtype: float64
```

```
df.groupby(['c1', 'c2'])['c4'].agg(['mean', 'min', 'max'])
```

		mean	min	max
c1	**c2**			
a	**A**	0.082461	0.082461	0.082461
	B	0.560924	0.560924	0.560924
b	**A**	0.207320	0.207320	0.207320
	B	0.005640	0.005640	0.005640
	C	0.965033	0.965033	0.965033
c	**D**	0.841396	0.841396	0.841396
d	**C**	0.098452	0.098452	0.098452

```
df.groupby(['c1', 'c2'], as_index=False)['c4'].mean()
```

	c1	c2	c4
0	a	A	0.082461
1	a	B	0.560924
2	b	A	0.207320
3	b	B	0.005640
4	b	C	0.965033
5	c	D	0.841396
6	d	C	0.098452


```
df.groupby(['c1', 'c2'], group_keys=False)['c4'].mean()
```

```
c1  c2
a   A     0.082461
    B     0.560924
b   A     0.207320
    B     0.005640
    C     0.965033
c   D     0.841396
d   C     0.098452
Name: c4, dtype: float64
```

```
def top(df, n=3, column='c1'):
  return df.sort_values(by=column)[-n:]
```

```
top(df, n=5)
```

	c1	c2	c3	c4
2	b	B	3	0.005640
3	b	A	3	0.207320
6	b	C	3	0.965033
4	c	D	3	0.841396
5	d	C	3	0.098452

```
df.groupby('c1').apply(top)
```

		c1	c2	c3	c4
c1					
a	**0**	a	A	3	0.082461
	1	a	B	3	0.560924
b	**2**	b	B	3	0.005640
	3	b	A	3	0.207320
	6	b	C	3	0.965033
c	**4**	c	D	3	0.841396
d	**5**	d	C	3	0.098452

피벗 테이블(Pivot Table)

함수	설명
`values`	집계하려는 컬럼 이름 혹은 이름의 리스트. 기본적으로 모든 숫자 컬럼 집계
`index`	피벗테이블의 로우를 그룹으로 묶을 컬럼 이름이나 그룹 키
`columns`	피벗테이블의 컬럼을 그룹으로 묶을 컬럼 이름이나 그룹 키
`aggfunc`	집계 함수나 함수 리스트. 기본값으로 `mean`이 사용
`fill_value`	결과 테이블에서 누락된 값 대체를 위한 값
`dropna`	True인 경우 모든 항목이 NA인 컬럼은 포함하지 않음
`margins`	부분합이나 총계를 담기 위한 로우/컬럼 추가 여부. 기본값은 False

```
df.pivot_table(['c3', 'c4'],
               index=['c1'],
               columns=['c2'])
```

	c3				c4			
c2	A	B	C	D	A	B	C	D
c1								
a	3.0	3.0	NaN	NaN	0.082461	0.560924	NaN	NaN

```
df.pivot_table(['c3', 'c4'],
               index=['c1'],
               columns=['c2'],
               margins=True)
```

	c3					c4				
c2	A	B	C	D	All	A	B	C	D	All
c1										
a	3.0	3.0	NaN	NaN	3	0.082461	0.560924	NaN	NaN	0.321692
b	3.0	3.0	3.0	NaN	3	0.207320	0.005640	0.965033	NaN	0.392664
c	NaN	NaN	NaN	3.0	3	NaN	NaN	NaN	0.841396	0.841396
d	NaN	NaN	3.0	NaN	3	NaN	NaN	0.098452	NaN	0.098452
All	3.0	3.0	3.0	3.0	3	0.144890	0.283282	0.531742	0.841396	0.394461


```
df.pivot_table(['c3', 'c4'],
               index=['c1'],
               columns=['c2'],
               margins=True,
               aggfunc=sum)
```

	c3					c4				
c2	A	B	C	D	All	A	B	C	D	All
c1										
a	3.0	3.0	NaN	NaN	6	0.082461	0.560924	NaN	NaN	0.643385
b	3.0	3.0	3.0	NaN	9	0.207320	0.005640	0.965033	NaN	1.177993
c	NaN	NaN	NaN	3.0	3	NaN	NaN	NaN	0.841396	0.841396
d	NaN	NaN	3.0	NaN	3	NaN	NaN	0.098452	NaN	0.098452
All	6.0	6.0	6.0	3.0	21	0.289781	0.566564	1.063485	0.841396	2.761225

```
df.pivot_table(['c3', 'c4'],
               index=['c1'],
               columns=['c2'],
               margins=True,
               aggfunc=sum,
               fill_value=0)
```

	c3					c4				
c2	A	B	C	D	All	A	B	C	D	All
c1										
a	3	3	0	0	6	0.082461	0.560924	0.000000	0.000000	0.643385
b	3	3	3	0	9	0.207320	0.005640	0.965033	0.000000	1.177993
c	0	0	0	3	3	0.000000	0.000000	0.000000	0.841396	0.841396
d	0	0	3	0	3	0.000000	0.000000	0.098452	0.000000	0.098452
All	6	6	6	3	21	0.289781	0.566564	1.063485	0.841396	2.761225

```
pd.crosstab(df.c1, df.c2)
```

c2	A	B	C	D
c1				
a	1	1	0	0
b	1	1	1	0
c	0	0	0	1
d	0	0	1	0


```
pd.crosstab(df.c1, df.c2, values=df.c3, aggfunc=sum, margins=True)
```

c2	A	B	C	D	All
c1					
a	3.0	3.0	NaN	NaN	6
b	3.0	3.0	3.0	NaN	9
c	NaN	NaN	NaN	3.0	3
d	NaN	NaN	3.0	NaN	3
All	6.0	6.0	6.0	3.0	21

▾ 범주형(Categorical) 데이터

메소드	설명
add_categories	기존 카테고리에 새로운 카테고리 추가
as_ordered	카테고리에 순서 지정
as_unordered	카테고리에 순서 미지정
remove_categories	카테고리 제거
remove_unused_categories	사용안하는 카테고리 제거
rename_categories	카테고리 이름 변경

메소드	설명
reorder_categories	새로운 카테고리에 순서 지정
set_categories	새로운 카테고리로 변경

```
s = pd.Series(['c1', 'c2', 'c1', 'c2', 'c1'] * 2)
s
```

```
0    c1
1    c2
2    c1
3    c2
4    c1
5    c1
6    c2
7    c1
8    c2
9    c1
dtype: object
```

```
pd.unique(s)
```

```
array(['c1', 'c2'], dtype=object)
```

```
pd.value_counts(s)
```

```
c1    6
c2    4
dtype: int64
```



```
code = pd.Series([0, 1, 0, 1, 0] * 2)
code
```

```
0    0
1    1
2    0
3    1
4    0
5    0
6    1
7    0
8    1
9    0
dtype: int64
```

```
d = pd.Series(['c1', 'c2'])
d
```

```
0    c1
1    c2
dtype: object
```

```
d.take(code)
```

```
0    c1
```

```
1    c2
0    c1
1    c2
0    c1
0    c1
1    c2
0    c1
1    c2
0    c1
dtype: object
```

```
df = pd.DataFrame({'id': np.arange(len(s)),
                   'c': s,
                   'v': np.random.randint(1000, 5000, size=len(s))})
df
```

	id	c	v
0	0	c1	3055
1	1	c2	3809
2	2	c1	4486
3	3	c2	4612
4	4	c1	2139
5	5	c1	4757
6	6	c2	4934
7	7	c1	4948
8	8	c2	3949
9	9	c1	4398


```
c = df['c'].astype('category')
c
```

```
0    c1
1    c2
2    c1
3    c2
4    c1
5    c1
6    c2
7    c1
8    c2
9    c1
Name: c, dtype: category
Categories (2, object): [c1, c2]
```

```
c.values
```

```
[c1, c2, c1, c2, c1, c1, c2, c1, c2, c1]
Categories (2, object): [c1, c2]
```

```
c.values.categories
```

```
Index(['c1', 'c2'], dtype='object')
```

```
c.values.codes
```

```
array([0, 1, 0, 1, 0, 0, 1, 0, 1, 0], dtype=int8)
```

```
df['c'] = c
df.c
```

```
0    c1
1    c2
2    c1
3    c2
4    c1
5    c1
6    c2
7    c1
8    c2
9    c1
Name: c, dtype: category
Categories (2, object): [c1, c2]
```



```
c = pd.Categorical(['c1', 'c2', 'c3', 'c1', 'c2'])
c
```

```
[c1, c2, c3, c1, c2]
Categories (3, object): [c1, c2, c3]
```

```
categories = ['c1', 'c2', 'c3']
codes = [0, 1, 2, 0, 1]
c = pd.Categorical.from_codes(codes, categories)
c
```

```
[c1, c2, c3, c1, c2]
Categories (3, object): [c1, c2, c3]
```

```
pd.Categorical.from_codes(codes, categories, ordered=True)
```

```
[c1, c2, c3, c1, c2]
Categories (3, object): [c1 < c2 < c3]
```

```
c.as_ordered()
```

```
[c1, c2, c3, c1, c2]
Categories (3, object): [c1 < c2 < c3]
```

```
c.codes
```

```
array([0, 1, 2, 0, 1], dtype=int8)
```

```
c.categories
```

```
Index(['c1', 'c2', 'c3'], dtype='object')
```

```
c = c.set_categories(['c1', 'c2', 'c3', 'c4', 'c5'])
c.categories
```

```
Index(['c1', 'c2', 'c3', 'c4', 'c5'], dtype='object')
```

```
c.value_counts()
```

```
c1    2
c2    2
c3    1
c4    0
c5    0
dtype: int64
```

```
c[c.isin(['c1', 'c3'])]
```

```
[c1, c3, c1]
Categories (5, object): [c1, c2, c3, c4, c5]
```

```
c = c.remove_unused_categories()
```

```
c.categories
```

```
Index(['c1', 'c2', 'c3'], dtype='object')
```


문자열 연산

문자열 연산자

- 파이썬의 문자열 연산자를 거의 모두 반영

함수	설명
capitalize()	첫 문자를 대문자로하고, 나머지 문자를 소문자로 하는 문자열 반환
casefold()	모든 대소문자 구분을 제거
count(sub, [, start[, end]])	[start, end] 범위에서 부분 문자열 sub의 중복되지 않은 수를 반환
find(sub [, start [, end]])	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 인덱스를 반환. sub가 발견되지 않는
rfind(sub [, start [, end]])	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 큰 인덱스를 반환. sub가 발견되지 않
index(sub [, start [, end]])	find()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
rindex(sub [, start [, end]])	rfind()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
isalnum()	문자열의 모든 문자가 영숫자로 1개 이상 있으면 True, 아니면 False 반환
isalpha()	문자열의 모든 문자가 영문자로 1개 이상 있으면 True, 아니면 False 반환
isdecimal()	문자열의 모든 문자가 10진수 문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환

함수	설명
isdigit()	문자열의 모든 문자가 숫자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isnumeric()	문자열의 모든 문자가 수치형이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isidentifier()	문자열이 유효한 식별자인 경우 True 반환
isspace()	문자열 내에 공백 문자가 있고, 문자가 1개 이상 있으면 True, 그렇지 않으면 False
istitle()	문자열이 제목이 있는 문자열에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
islower()	문자열의 모든 문자가 소문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isupper()	문자열의 문자가 모두 대문자에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
join(iterable)	iterable에 있는 문자열에 연결된 문자열을 반환
center(width [, fillchar])	길이 너비만큼 중앙정렬된 문자열 반환
ljust(width [, fillchar])	너미만큼의 문자열에서 왼쪽 정렬된 문자열을 반환
rjust(width [, fillchar])	너미만큼의 문자열에서 오른쪽 정렬된 문자열을 반환
lower()	모든 대소문자가 소문자로 변환된 문자열을 반환
upper()	문자열에서 모든 문자를 대문자로 변환한 문자열을 반환
title()	문자열에서 첫 글자만 대문자이고 나머지는 소문자인 문자열 반환
swapcase()	문자열에서 소문자를 대문자로 대문자를 소문자로 변환한 문자열 반환
strip([chars])	문자열 양쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
lstrip([chars])	문자열 왼쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
rstrip([chars])	문자열 오른쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
partition(sep)	문자열에서 첫번째 sep를 기준으로 분할하여 3개의 튜플을 반환
rpartition(sep)	문자열에서 마지막 sep를 기준으로 분할하여 3개의 튜플을 반환
replace(old, new[,count])	문자열의 모든 old를 new로 교체한 문자열을 반환
split(sep=None, maxsplit=-1)	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
rsplit(sep=None, maxsplit=-1)	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
splitlines([keepends])	문자열에서 라인 단위로 구분하여 리스트를 반환
startswith(prefix [, start[, end]])	[start, end] 범위에서 지정한 prefix로 시작하면 True, 아니면 False 반환
endswith(suffix [, start[, end]])	[start, end] 범위에서 지정한 suffix로 끝나면 True, 아니면 False 반환
zfill(width)	너비 만큼의 문자열에서 비어있는 부분에 '0'이 채워진 문자열 반환


```
name_tuple = ['Suan Lee', 'Steven Jobs', 'Larry Page', 'Elon Musk', None, 'Bill Gates', 'Mark Zucke
names = pd.Series(name_tuple)
names
```

```
0          Suan Lee
1        Steven Jobs
2         Larry Page
3          Elon Musk
4               None
5         Bill Gates
6    Mark Zuckerberg
7         Jeff Bezos
dtype: object
```

```
names.str.lower()
```

```
0          suan lee
1        steven jobs
```

```
2         larry page
3          elon musk
4               None
5         bill gates
6    mark zuckerberg
7         jeff bezos
dtype: object
```

```
names.str.len()
```

```
0     8.0
1    11.0
2    10.0
3     9.0
4     NaN
5    10.0
6    15.0
7    10.0
dtype: float64
```

```
names.str.split()
```

```
0           [Suan, Lee]
1       [Steven, Jobs]
2        [Larry, Page]
3         [Elon, Musk]
4                 None
5        [Bill, Gates]
6    [Mark, Zuckerberg]
7        [Jeff, Bezos]
dtype: object
```


기타 연산자

메소드	설명
get()	각 요소에 인덱스 지정
slice()	각 요소에 슬라이스 적용
slice_replace()	각 요소의 슬라이스를 특정 값으로 대체
cat()	문자열 연결
repeat()	값 반복
normalize()	문자열의 유니코드 형태로 반환
pad()	문자열 왼쪽, 오른쪽, 또는 양쪽 공백 추가
wrap()	긴 문자열을 주어진 너비보다 짧은 길이의 여러 줄로 나눔
join()	Series의 각 요소에 있는 문자열을 전달된 구분자와 결합
get_dummies()	DataFrame으로 가변수(dummy variable) 추출

```
names.str[0:4]
```

```
0    Suan
1    Stev
2    Larr
3    Elon
4    None
```

```
5    Bill
6    Mark
7    Jeff
dtype: object
```

```
names.str.split().str.get(-1)
```

```
0          Lee
1         Jobs
2         Page
3         Musk
4         None
5        Gates
6    Zuckerberg
7        Bezos
dtype: object
```

```
names.str.repeat(2)
```

```
0                 Suan LeeSuan Lee
1           Steven JobsSteven Jobs
2             Larry PageLarry Page
3               Elon MuskElon Musk
4                             None
5             Bill GatesBill Gates
6    Mark ZuckerbergMark Zuckerberg
7             Jeff BezosJeff Bezos
dtype: object
```



```
names.str.join('*')
```

```
0              S*u*a*n* *L*e*e
1          S*t*e*v*e*n* *J*o*b*s
2            L*a*r*r*y* *P*a*g*e
3              E*l*o*n* *M*u*s*k
4                            None
5            B*i*l*l* *G*a*t*e*s
6    M*a*r*k* *Z*u*c*k*e*r*b*e*r*g
7            J*e*f*f* *B*e*z*o*s
dtype: object
```

정규표현식

메소드	설명
match()	각 요소에 re.match() 호출. 불리언 값 반환
extract()	각 요소에 re.match() 호출. 문자열로 매칭된 그룹 반환
findall()	각 요소에 re.findall() 호출.
replace()	패턴이 발생한 곳을 다른 문자열로 대체
contains()	각 요소에 re.search() 호출. 불리언 값 반환
count()	패턴 발생 건수 집계
split()	str.split()과 동일하지만 정규 표현식 사용
rsplit()	str.rsplit()과 동일하지만 정규 표현식 사용

```
names.str.match('([A-Za-z]+)')
```

```
0    True
1    True
2    True
3    True
4    None
5    True
6    True
7    True
dtype: object
```

```
names.str.findall('([A-Za-z]+)')
```

```
0           [Suan, Lee]
1        [Steven, Jobs]
2         [Larry, Page]
3          [Elon, Musk]
4                  None
5         [Bill, Gates]
6    [Mark, Zuckerberg]
7         [Jeff, Bezos]
dtype: object
```

시계열 처리


```
idx = pd.DatetimeIndex(['2019-01-01', '2020-01-01', '2020-02-01', '2020-02-02', '2020-03-01'])
s = pd.Series([0, 1, 2, 3, 4], index=idx)
s
```

```
2019-01-01    0
2020-01-01    1
2020-02-01    2
2020-02-02    3
2020-03-01    4
dtype: int64
```

```
s['2020-01-01':]
```

```
2020-01-01    1
2020-02-01    2
2020-02-02    3
2020-03-01    4
dtype: int64
```

```
s[:'2020-01-01']
```

```
2019-01-01    0
2020-01-01    1
dtype: int64
```

```
s['2019']
```

```
2019-01-01    0
dtype: int64
```

시계열 데이터 구조

타임스탬프(timestamp)	기간(time period)	시간 델타 또는 지속 기간
Pandas `Timestamp` 타입 제공	Pandas `Period` 타입 제공	Pandas의 `Timedelta` 타입 제공
파이썬 `datetime` 대체 타입		파이썬 `datetime.timedelta` 대체 타입
`numpy.datetime64` 타입 기반	`numpy.datetime64` 타입 기반	`numpy.timedelta64` 타입 기반
`DatetimeIndex` 인덱스 구조	`PeriodIndex` 인덱스 구조	`TimedeltaIndex` 인덱스 구조

```
from datetime import datetime
dates = pd.to_datetime(['12-12-2019', datetime(2020, 1, 1), '2nd of Feb, 2020', '2020-Mar-4', '2020
dates
```

```
DatetimeIndex(['2019-12-12', '2020-01-01', '2020-02-02', '2020-03-04',
               '2020-07-01'],
              dtype='datetime64[ns]', freq=None)
```

```
dates.to_period('D')
```

```
PeriodIndex(['2019-12-12', '2020-01-01', '2020-02-02', '2020-03-04',
             '2020-07-01'],
            dtype='period[D]', freq='D')
```

```
dates - dates[0]
```

```
TimedeltaIndex(['0 days', '20 days', '52 days', '83 days', '202 days'], dtype='timedelta64[n
```



```
pd.date_range('2020-01-01', '2020-07-01')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10',
               ...
               '2020-06-22', '2020-06-23', '2020-06-24', '2020-06-25',
               '2020-06-26', '2020-06-27', '2020-06-28', '2020-06-29',
               '2020-06-30', '2020-07-01'],
              dtype='datetime64[ns]', length=183, freq='D')
```

```
pd.date_range('2020-01-01', periods=7)
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07'],
              dtype='datetime64[ns]', freq='D')
```

```
pd.date_range('2020-01-01', periods=7, freq='M')
```

```
DatetimeIndex(['2020-01-31', '2020-02-29', '2020-03-31', '2020-04-30',
               '2020-05-31', '2020-06-30', '2020-07-31'],
              dtype='datetime64[ns]', freq='M')
```

```
pd.date_range('2020-01-01', periods=7, freq='H')
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 01:00:00',
               '2020-01-01 02:00:00', '2020-01-01 03:00:00',
               '2020-01-01 04:00:00', '2020-01-01 05:00:00',
               '2020-01-01 06:00:00'],
              dtype='datetime64[ns]', freq='H')
```

```
idx = pd.to_datetime(['2020-01-01 12:00:00', '2020-01-02 00:00:00'] + [None])
idx
```

```
DatetimeIndex(['2020-01-01 12:00:00', '2020-01-02 00:00:00', 'NaT'], dtype='datetime64[ns]',
```

```
idx[2]
```

```
NaT
```

```
pd.isnull(idx)
```

```
array([False, False,  True])
```


시계열 기본

```
dates = [datetime(2020, 1, 1), datetime(2020, 1, 2), datetime(2020, 1, 4), datetime(2020, 1, 7),
         datetime(2020, 1, 10), datetime(2020, 1, 11), datetime(2020, 1, 15)]
dates
```

```
[datetime.datetime(2020, 1, 1, 0, 0),
 datetime.datetime(2020, 1, 2, 0, 0),
 datetime.datetime(2020, 1, 4, 0, 0),
 datetime.datetime(2020, 1, 7, 0, 0),
 datetime.datetime(2020, 1, 10, 0, 0),
 datetime.datetime(2020, 1, 11, 0, 0),
 datetime.datetime(2020, 1, 15, 0, 0)]
```

```
ts = pd.Series(np.random.randn(7), index=dates)
ts
```

```
2020-01-01    0.828153
2020-01-02    0.668588
2020-01-04    0.075077
2020-01-07    0.459817
2020-01-10   -1.729824
2020-01-11   -0.403687
2020-01-15    2.096749
dtype: float64
```

```
ts.index
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-04', '2020-01-07',
               '2020-01-10', '2020-01-11', '2020-01-15'],
              dtype='datetime64[ns]', freq=None)
```

```
ts.index[0]
```

```
Timestamp('2020-01-01 00:00:00')
```

```
ts[ts.index[2]]
```

```
0.07507698686343409
```

```
ts['20200104']
```

```
0.07507698686343409
```

```
ts['1/4/2020']
```

```
0.07507698686343409
```

```
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('2017-10-01', periods=1000))
ts
```

```
2017-10-01   -1.420342
2017-10-02   -0.770328
2017-10-03    0.674475
2017-10-04   -0.026037
2017-10-05   -0.136982
                ...
2020-06-22   -1.014287
2020-06-23    0.254001
2020-06-24   -1.124229
2020-06-25   -1.412706
2020-06-26    2.196948
Freq: D, Length: 1000, dtype: float64
```

```
ts['2020']
```

```
2020-01-01    0.197849
2020-01-02   -1.077263
2020-01-03   -0.505628
2020-01-04   -0.524031
2020-01-05   -0.655873
                ...
2020-06-22   -1.014287
2020-06-23    0.254001
2020-06-24   -1.124229
2020-06-25   -1.412706
2020-06-26    2.196948
Freq: D, Length: 178, dtype: float64
```

```
ts['2020-06']
```

```
2020-06-01   -0.493871
2020-06-02    0.735484
2020-06-03   -0.314472
2020-06-04   -1.768392
2020-06-05    0.325760
2020-06-06    0.781031
2020-06-07    1.006581
2020-06-08    1.483781
2020-06-09    0.667332
2020-06-10   -0.512309
2020-06-11    0.378903
2020-06-12   -2.377225
2020-06-13   -1.686634
2020-06-14   -1.699558
2020-06-15   -0.149238
2020-06-16   -0.265368
2020-06-17   -0.028276
2020-06-18    3.959999
2020-06-19    2.236131
2020-06-20    0.252964
2020-06-21    0.293082
2020-06-22   -1.014287
2020-06-23    0.254001
2020-06-24   -1.124229
2020-06-25   -1.412706
2020-06-26    2.196948
Freq: D, dtype: float64
```



```
ts[datetime(2020, 6, 20):]
```

```
2020-06-20    0.252964
2020-06-21    0.293082
2020-06-22   -1.014287
2020-06-23    0.254001
2020-06-24   -1.124229
2020-06-25   -1.412706
2020-06-26    2.196948
Freq: D, dtype: float64
```

```
ts['2020-06-10':'2020-06-20']
```

```
2020-06-10   -0.512309
2020-06-11    0.378903
2020-06-12   -2.377225
2020-06-13   -1.686634
2020-06-14   -1.699558
2020-06-15   -0.149238
2020-06-16   -0.265368
2020-06-17   -0.028276
2020-06-18    3.959999
2020-06-19    2.236131
2020-06-20    0.252964
Freq: D, dtype: float64
```

```
tdf = pd.DataFrame(np.random.randn(1000, 4),
                   index=pd.date_range('2017-10-01', periods=1000),
                   columns=['A', 'B', 'C', 'D'])
```

```
tdf
```

	A	B	C	D
2017-10-01	1.609879	-1.136145	0.643200	0.227646
2017-10-02	-1.885087	-0.269613	2.125796	-1.386158
2017-10-03	-1.061120	0.748388	-1.964428	1.343831
2017-10-04	0.088057	-0.295152	0.159467	0.525364
2017-10-05	-0.763533	-1.066217	-1.579313	0.010115
...	...	...	...	...
2020-06-22	-0.173846	1.029804	1.236924	-0.329191
2020-06-23	-1.102508	-0.128789	0.675384	1.341087
2020-06-24	0.390441	-0.247763	0.294704	0.811862
2020-06-25	1.045906	1.897151	0.214847	-0.322413
2020-06-26	-1.311100	-0.379670	-0.407460	1.644619

1000 rows × 4 columns


```
tdf['2020']
```

	A	B	C	D
2020-01-01	-0.403012	0.641469	-0.120983	0.577990
2020-01-02	0.100218	-0.513692	-1.325447	-0.553030
2020-01-03	0.400035	-0.221343	0.293143	0.173345
2020-01-04	-0.586777	0.025463	0.441146	1.464452
2020-01-05	-0.072576	1.324623	-1.295825	-0.729552
...	...	...	...	...
2020-06-22	-0.173846	1.029804	1.236924	-0.329191
2020-06-23	-1.102508	-0.128789	0.675384	1.341087
2020-06-24	0.390441	-0.247763	0.294704	0.811862
2020-06-25	1.045906	1.897151	0.214847	-0.322413
2020-06-26	-1.311100	-0.379670	-0.407460	1.644619

178 rows × 4 columns

```
tdf.loc['2020-06']
```

	A	B	C	D
2020-06-01	-0.338590	1.051900	1.157340	0.413740
2020-06-02	1.318181	-0.389031	-0.223880	-1.750814
2020-06-03	-0.755378	0.650233	0.120089	0.329027
2020-06-04	1.263764	-0.704872	0.634886	-0.623752
2020-06-05	1.758623	0.055859	0.242151	1.570657
2020-06-06	-0.029935	2.195807	1.199621	1.355318
2020-06-07	0.350829	0.887965	-0.660216	-0.032210
2020-06-08	0.746787	-0.394732	-1.435143	0.524777
2020-06-09	1.241311	1.619016	-0.382691	-0.411006
2020-06-10	0.040276	-0.382559	-0.720537	1.681073
2020-06-11	-0.616292	-0.742009	-1.265899	-1.255333
2020-06-12	-0.070184	0.346207	-2.034462	-0.621509
2020-06-13	0.043446	-1.254183	0.972193	-1.035266
2020-06-14	1.019597	-0.581715	-2.526763	-0.567033
2020-06-15	0.346268	0.294683	0.340597	0.493301
2020-06-16	-0.263815	-0.147701	0.499732	-0.878572
2020-06-17	1.763087	0.475867	-0.666925	1.641322
2020-06-18	0.344728	1.919335	-0.419496	-0.608727
2020-06-19	-1.218446	-0.466427	-1.197872	-0.236982
2020-06-20	-0.045209	0.778766	0.384635	-0.247412
2020-06-21	-2.073453	-0.559049	1.266168	0.605200
2020-06-22	-0.173846	1.029804	1.236924	-0.329191
2020-06-23	-1.102508	-0.128789	0.675384	1.341087
2020-06-24	0.390441	-0.247763	0.294704	0.811862


```
tdf['2020-06-20':]
```

	A	B	C	D

```
tdf['C']
```

```
2017-10-01    0.643200
2017-10-02    2.125796
2017-10-03   -1.964428
2017-10-04    0.159467
2017-10-05   -1.579313
                ...
2020-06-22    1.236924
2020-06-23    0.675384
2020-06-24    0.294704
2020-06-25    0.214847
2020-06-26   -0.407460
Freq: D, Name: C, Length: 1000, dtype: float64
```

```
ts = pd.Series(np.random.randn(10),
               index=pd.DatetimeIndex(['2020-01-01', '2020-01-01', '2020-01-02', '2020-01-02', '202
                                       '2020-01-04', '2020-01-05', '2020-01-05', '2020-01-06', '202
ts
```

```
2020-01-01   -0.344128
2020-01-01   -2.441707
2020-01-02    0.710743
2020-01-02   -1.085734
2020-01-03    1.205633
2020-01-04    0.589756
2020-01-05    0.009542
2020-01-05   -1.248808
2020-01-06    0.760588
2020-01-07   -0.770594
dtype: float64
```



```
ts.index.is_unique
```

```
False
```

```
ts['2020-01-01']
```

```
2020-01-01   -0.344128
2020-01-01   -2.441707
dtype: float64
```

```
ts.groupby(level=0).mean()
```

```
2020-01-01   -1.392917
2020-01-02   -0.187495
2020-01-03    1.205633
2020-01-04    0.589756
2020-01-05   -0.619633
2020-01-06    0.760588
2020-01-07   -0.770594
dtype: float64
```

```
pd.date_range('2020-01-01', '2020-07-01')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10',
               ...
               '2020-06-22', '2020-06-23', '2020-06-24', '2020-06-25',
               '2020-06-26', '2020-06-27', '2020-06-28', '2020-06-29',
               '2020-06-30', '2020-07-01'],
              dtype='datetime64[ns]', length=183, freq='D')
```

```
pd.date_range(start='2020-01-01', periods=10)
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10'],
              dtype='datetime64[ns]', freq='D')
```

```
pd.date_range(end='2020-07-01', periods=10)
```

```
DatetimeIndex(['2020-06-22', '2020-06-23', '2020-06-24', '2020-06-25',
               '2020-06-26', '2020-06-27', '2020-06-28', '2020-06-29',
               '2020-06-30', '2020-07-01'],
              dtype='datetime64[ns]', freq='D')
```

```
pd.date_range('2020-07-01', '2020-07-7', freq='B')
```

```
DatetimeIndex(['2020-07-01', '2020-07-02', '2020-07-03', '2020-07-06',
               '2020-07-07'],
              dtype='datetime64[ns]', freq='B')
```


주기와 오프셋

- 주기코드

코드	오프셋	설명
D	Day	달력상 일
B	BusinessDay	영업일
W-MON, W-TUE, ...	Week	주
WON-MON, WON-2MON, ...	WeekOfMonth	월별 주차와 요일
MS	MonthBegin	월 시작일
BMS	BusinessMonthBegin	영업일 기준 월 시작일
M	MonthEnd	월 마지막일
BM	BusinessMonthEnd	영업일 기준 월 마지막일
QS-JAN, QS-FEB, ...	QuarterBegin	분기 시작
BQS-JAN, BQS-FEB, ...	BusinessQuarterBegin	영업일 기준 분기 시작
Q-JAN, Q-FEB, ...	QuarterEnd	분기 마지막
BQ-JAN, BQ-FEB, ...	BusinessQuarterEnd	영업일 기준 분기 마지막
AS-JAN, AS-FEB, ...	YearBegin	연초
BAS-JAN, BAS-FEB, ...	BusinessYearBegin	영업일 기준 연초

코드	오프셋	설명
A-JAN, A-FEB,...	YearEnd	연말
BA-JAN, BA-FEB,...	BusinessYearEnd	영업일 기준 연말
H	Hour	시간
BH	BusinessHour	영업 시간
T 또는 min	Minute	분
S	Second	초
L 또는 ms	Milli	밀리초
U	Micro	마이크로초
N	Nano	나노초

```
pd.timedelta_range(0, periods=12, freq='H')
```

```
TimedeltaIndex(['00:00:00', '01:00:00', '02:00:00', '03:00:00', '04:00:00',
                '05:00:00', '06:00:00', '07:00:00', '08:00:00', '09:00:00',
                '10:00:00', '11:00:00'],
               dtype='timedelta64[ns]', freq='H')
```

```
pd.timedelta_range(0, periods=60, freq='T')
```

```
TimedeltaIndex(['00:00:00', '00:01:00', '00:02:00', '00:03:00', '00:04:00',
                '00:05:00', '00:06:00', '00:07:00', '00:08:00', '00:09:00',
                '00:10:00', '00:11:00', '00:12:00', '00:13:00', '00:14:00',
                '00:15:00', '00:16:00', '00:17:00', '00:18:00', '00:19:00',
                '00:20:00', '00:21:00', '00:22:00', '00:23:00', '00:24:00',
                '00:25:00', '00:26:00', '00:27:00', '00:28:00', '00:29:00',
                '00:30:00', '00:31:00', '00:32:00', '00:33:00', '00:34:00',
                '00:35:00', '00:36:00', '00:37:00', '00:38:00', '00:39:00',
                '00:40:00', '00:41:00', '00:42:00', '00:43:00', '00:44:00',
                '00:45:00', '00:46:00', '00:47:00', '00:48:00', '00:49:00',
                '00:50:00', '00:51:00', '00:52:00', '00:53:00', '00:54:00',
                '00:55:00', '00:56:00', '00:57:00', '00:58:00', '00:59:00'],
               dtype='timedelta64[ns]', freq='T')
```

```
pd.timedelta_range(0, periods=10, freq='1H30T')
```

```
TimedeltaIndex(['00:00:00', '01:30:00', '03:00:00', '04:30:00', '06:00:00',
                '07:30:00', '09:00:00', '10:30:00', '12:00:00', '13:30:00'],
               dtype='timedelta64[ns]', freq='90T')
```

```
pd.date_range('2020-01-01', periods=20, freq='B')
```

```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-06',
               '2020-01-07', '2020-01-08', '2020-01-09', '2020-01-10',
               '2020-01-13', '2020-01-14', '2020-01-15', '2020-01-16',
               '2020-01-17', '2020-01-20', '2020-01-21', '2020-01-22',
               '2020-01-23', '2020-01-24', '2020-01-27', '2020-01-28'],
              dtype='datetime64[ns]', freq='B')
```

```
pd.date_range('2020-01-01', periods=30, freq='2H')
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 02:00:00',
```

```
                '2020-01-01 04:00:00', '2020-01-01 06:00:00',
                '2020-01-01 08:00:00', '2020-01-01 10:00:00',
                '2020-01-01 12:00:00', '2020-01-01 14:00:00',
                '2020-01-01 16:00:00', '2020-01-01 18:00:00',
                '2020-01-01 20:00:00', '2020-01-01 22:00:00',
                '2020-01-02 00:00:00', '2020-01-02 02:00:00',
                '2020-01-02 04:00:00', '2020-01-02 06:00:00',
                '2020-01-02 08:00:00', '2020-01-02 10:00:00',
                '2020-01-02 12:00:00', '2020-01-02 14:00:00',
                '2020-01-02 16:00:00', '2020-01-02 18:00:00',
                '2020-01-02 20:00:00', '2020-01-02 22:00:00',
                '2020-01-03 00:00:00', '2020-01-03 02:00:00',
                '2020-01-03 04:00:00', '2020-01-03 06:00:00',
                '2020-01-03 08:00:00', '2020-01-03 10:00:00'],
               dtype='datetime64[ns]', freq='2H')
```

```
pd.date_range('2020-01-01', periods=20, freq='S')
```

```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 00:00:01',
               '2020-01-01 00:00:02', '2020-01-01 00:00:03',
               '2020-01-01 00:00:04', '2020-01-01 00:00:05',
               '2020-01-01 00:00:06', '2020-01-01 00:00:07',
               '2020-01-01 00:00:08', '2020-01-01 00:00:09',
               '2020-01-01 00:00:10', '2020-01-01 00:00:11',
               '2020-01-01 00:00:12', '2020-01-01 00:00:13',
               '2020-01-01 00:00:14', '2020-01-01 00:00:15',
               '2020-01-01 00:00:16', '2020-01-01 00:00:17',
               '2020-01-01 00:00:18', '2020-01-01 00:00:19'],
              dtype='datetime64[ns]', freq='S')
```


시프트(Shift)

```
ts = pd.Series(np.random.randn(5),
               index=pd.date_range('2020-01-01', periods=5, freq='B'))
ts
```

```
2020-01-01   -1.502628
2020-01-02   -1.059176
2020-01-03   -1.437863
2020-01-06   -1.129893
2020-01-07   -0.118184
Freq: B, dtype: float64
```

```
ts.shift(1)
```

```
2020-01-01         NaN
2020-01-02   -1.502628
2020-01-03   -1.059176
2020-01-06   -1.437863
2020-01-07   -1.129893
Freq: B, dtype: float64
```

```
ts.shift(3)
```

```
2020-01-01         NaN
```

```
2020-01-02         NaN
2020-01-03         NaN
2020-01-06   -1.502628
2020-01-07   -1.059176
Freq: B, dtype: float64
```

```
ts.shift(-2)
```

```
2020-01-01   -1.437863
2020-01-02   -1.129893
2020-01-03   -0.118184
2020-01-06         NaN
2020-01-07         NaN
Freq: B, dtype: float64
```

```
ts.shift(3, freq='B')
```

```
2020-01-06   -1.502628
2020-01-07   -1.059176
2020-01-08   -1.437863
2020-01-09   -1.129893
2020-01-10   -0.118184
Freq: B, dtype: float64
```

```
ts.shift(2, freq='W')
```

```
2020-01-12   -1.502628
2020-01-12   -1.059176
2020-01-12   -1.437863
2020-01-19   -1.129893
2020-01-19   -0.118184
dtype: float64
```


시간대 처리

- 국제표준시(Coordinated Universal Time, UTC)를 기준으로 떨어진 거리만큼 오프셋으로 시간대 처리
- 전 세계의 시간대 정보를 모아놓은 올슨 데이터베이스를 활용한 라이브러리인 `pytz` 사용

```
import pytz
pytz.common_timezones
```

```
['Africa/Abidjan', 'Africa/Accra', 'Africa/Addis_Ababa', 'Africa/Algiers', 'Africa/Asmara',
```

```
tz = pytz.timezone('Asia/Seoul')
```

```
dinx = pd.date_range('2020-01-01 09:00', periods=7, freq='B')
ts = pd.Series(np.random.randn(len(dinx)), index=dinx)
ts
```

```
2020-01-01 09:00:00   -0.605082
```

```
2020-01-02 09:00:00   -0.554809
2020-01-03 09:00:00    0.545710
2020-01-06 09:00:00    0.129023
2020-01-07 09:00:00    0.494613
2020-01-08 09:00:00    0.444270
2020-01-09 09:00:00   -0.040237
Freq: B, dtype: float64
```

```
pd.date_range('2020-01-01 09:00', periods=7, freq='B', tz='UTC')
```

```
DatetimeIndex(['2020-01-01 09:00:00+00:00', '2020-01-02 09:00:00+00:00',
               '2020-01-03 09:00:00+00:00', '2020-01-06 09:00:00+00:00',
               '2020-01-07 09:00:00+00:00', '2020-01-08 09:00:00+00:00',
               '2020-01-09 09:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='B')
```

```
ts_utc = ts.tz_localize('UTC')
ts_utc
```

```
2020-01-01 09:00:00+00:00   -0.605082
2020-01-02 09:00:00+00:00   -0.554809
2020-01-03 09:00:00+00:00    0.545710
2020-01-06 09:00:00+00:00    0.129023
2020-01-07 09:00:00+00:00    0.494613
2020-01-08 09:00:00+00:00    0.444270
2020-01-09 09:00:00+00:00   -0.040237
Freq: B, dtype: float64
```



```
ts_utc.index
```

```
DatetimeIndex(['2020-01-01 09:00:00+00:00', '2020-01-02 09:00:00+00:00',
               '2020-01-03 09:00:00+00:00', '2020-01-06 09:00:00+00:00',
               '2020-01-07 09:00:00+00:00', '2020-01-08 09:00:00+00:00',
               '2020-01-09 09:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='B')
```

```
ts_utc.tz_convert('Asia/Seoul')
```

```
2020-01-01 18:00:00+09:00   -0.605082
2020-01-02 18:00:00+09:00   -0.554809
2020-01-03 18:00:00+09:00    0.545710
2020-01-06 18:00:00+09:00    0.129023
2020-01-07 18:00:00+09:00    0.494613
2020-01-08 18:00:00+09:00    0.444270
2020-01-09 18:00:00+09:00   -0.040237
Freq: B, dtype: float64
```

```
ts_seoul = ts.tz_localize('Asia/Seoul')
ts_seoul
```

```
2020-01-01 09:00:00+09:00   -0.605082
2020-01-02 09:00:00+09:00   -0.554809
2020-01-03 09:00:00+09:00    0.545710
2020-01-06 09:00:00+09:00    0.129023
2020-01-07 09:00:00+09:00    0.494613
2020-01-08 09:00:00+09:00    0.444270
```

```
2020-01-09 09:00:00+09:00   -0.040237
Freq: B, dtype: float64
```

```
ts_seoul.tz_convert('UTC')
```

```
2020-01-01 00:00:00+00:00   -0.605082
2020-01-02 00:00:00+00:00   -0.554809
2020-01-03 00:00:00+00:00    0.545710
2020-01-06 00:00:00+00:00    0.129023
2020-01-07 00:00:00+00:00    0.494613
2020-01-08 00:00:00+00:00    0.444270
2020-01-09 00:00:00+00:00   -0.040237
Freq: B, dtype: float64
```

```
ts_seoul.tz_convert('Europe/Berlin')
```

```
2020-01-01 01:00:00+01:00   -0.605082
2020-01-02 01:00:00+01:00   -0.554809
2020-01-03 01:00:00+01:00    0.545710
2020-01-06 01:00:00+01:00    0.129023
2020-01-07 01:00:00+01:00    0.494613
2020-01-08 01:00:00+01:00    0.444270
2020-01-09 01:00:00+01:00   -0.040237
Freq: B, dtype: float64
```



```
ts.index.tz_localize('America/New_York')
```

```
DatetimeIndex(['2020-01-01 09:00:00-05:00', '2020-01-02 09:00:00-05:00',
               '2020-01-03 09:00:00-05:00', '2020-01-06 09:00:00-05:00',
               '2020-01-07 09:00:00-05:00', '2020-01-08 09:00:00-05:00',
               '2020-01-09 09:00:00-05:00'],
              dtype='datetime64[ns, America/New_York]', freq='B')
```

```
stamp = pd.Timestamp('2020-01-01 12:00')
stamp_utc = stamp.tz_localize('UTC')
stamp_utc
```

```
Timestamp('2020-01-01 12:00:00+0000', tz='UTC')
```

```
stamp_utc.value
```

```
1577880000000000000
```

```
stamp_utc.tz_convert('Asia/Seoul')
```

```
Timestamp('2020-01-01 21:00:00+0900', tz='Asia/Seoul')
```

```
stamp_utc.tz_convert('Asia/Seoul').value
```

```
1577880000000000000
```

```
stamp_ny = pd.Timestamp('2020-01-01 12:00', tz='America/New_York')
```

```
stamp_ny
```

```
Timestamp('2020-01-01 12:00:00-0500', tz='America/New_York')
```

```
stamp_ny.value
```

```
1577898000000000000
```

```
stamp_utc.tz_convert('Asia/Shanghai')
```

```
Timestamp('2020-01-01 20:00:00+0800', tz='Asia/Shanghai')
```

```
stamp = pd.Timestamp('2020-01-01 12:00', tz='Asia/Seoul')
stamp
```

```
Timestamp('2020-01-01 12:00:00+0900', tz='Asia/Seoul')
```

```
from pandas.tseries.offsets import Hour
stamp + Hour()
```

```
Timestamp('2020-01-01 13:00:00+0900', tz='Asia/Seoul')
```



```
stamp + 3 * Hour()
```

```
Timestamp('2020-01-01 15:00:00+0900', tz='Asia/Seoul')
```

```
ts_utc
```

```
2020-01-01 09:00:00+00:00   -0.605082
2020-01-02 09:00:00+00:00   -0.554809
2020-01-03 09:00:00+00:00    0.545710
2020-01-06 09:00:00+00:00    0.129023
2020-01-07 09:00:00+00:00    0.494613
2020-01-08 09:00:00+00:00    0.444270
2020-01-09 09:00:00+00:00   -0.040237
Freq: B, dtype: float64
```

```
ts1 = ts_utc[:5].tz_convert('Europe/Berlin')
ts2 = ts_utc[2:].tz_convert('America/New_York')
ts = ts1 + ts2
```

```
ts.index
```

```
DatetimeIndex(['2020-01-01 09:00:00+00:00', '2020-01-02 09:00:00+00:00',
               '2020-01-03 09:00:00+00:00', '2020-01-06 09:00:00+00:00',
               '2020-01-07 09:00:00+00:00', '2020-01-08 09:00:00+00:00',
               '2020-01-09 09:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq='B')
```

기간과 기간 연산

```
p = pd.Period(2020, freq='A-JAN')
p
```

```
Period('2020', 'A-JAN')
```

```
p + 2
```

```
Period('2022', 'A-JAN')
```

```
p - 3
```

```
Period('2017', 'A-JAN')
```

```
p1 = pd.Period(2010, freq='A-JAN')
p2 = pd.Period(2020, freq='A-JAN')
p2 - p1
```

```
<10 * YearEnds: month=1>
```

```
pr = pd.period_range('2020-01-01', '2020-06-30', freq='M')
```

```
pd.Series(np.random.randn(6), index=pr)
```

```
2020-01    1.795191
2020-02   -1.805473
2020-03    0.921910
2020-04    0.624512
2020-05    1.180537
2020-06   -0.712804
Freq: M, dtype: float64
```



```
pidx = pd.PeriodIndex(['2020-1', '2020-2', '2020-4'], freq='M')
pidx
```

```
PeriodIndex(['2020-01', '2020-02', '2020-04'], dtype='period[M]', freq='M')
```

```
p = pd.Period('2020', freq='A-FEB')
p
```

```
Period('2020', 'A-FEB')
```

```
p.asfreq('M', how='start')
```

```
Period('2019-03', 'M')
```

```
p.asfreq('M', how='end')
```

```
Period('2020-02', 'M')
```

```
p = pd.Period('2020', freq='A-OCT')
p
```

```
Period('2020', 'A-OCT')
```

```
p.asfreq('M', how='start')
```

```
Period('2019-11', 'M')
```

```
p.asfreq('M', how='end')
```

```
Period('2020-10', 'M')
```

```
pr = pd.period_range('2010', '2020', freq='A-JAN')
ts = pd.Series(np.random.randn(len(pr)), index=pr)
ts
```

```
2010    0.151803
2011    0.657910
2012    2.358392
2013    0.105678
2014   -0.111647
2015    0.609567
2016   -0.844259
2017   -0.012856
2018   -0.301620
2019    0.086615
2020    0.360138
Freq: A-JAN, dtype: float64
```



```
ts.asfreq('M', how='start')
```

```
2009-02    0.151803
2010-02    0.657910
2011-02    2.358392
2012-02    0.105678
2013-02   -0.111647
2014-02    0.609567
2015-02   -0.844259
2016-02   -0.012856
2017-02   -0.301620
2018-02    0.086615
2019-02    0.360138
Freq: M, dtype: float64
```

```
ts.asfreq('B', how='end')
```

```
2010-01-29    0.151803
2011-01-31    0.657910
2012-01-31    2.358392
2013-01-31    0.105678
2014-01-31   -0.111647
2015-01-30    0.609567
2016-01-29   -0.844259
2017-01-31   -0.012856
```

```
2018-01-31   -0.301620
2019-01-31    0.086615
2020-01-31    0.360138
Freq: B, dtype: float64
```

```
p = pd.Period('2020Q2', freq='Q-JAN')
p
```

```
Period('2020Q2', 'Q-JAN')
```

```
p.asfreq('D', 'start')
```

```
Period('2019-05-01', 'D')
```

```
p.asfreq('D', 'end')
```

```
Period('2019-07-31', 'D')
```

```
pr = pd.period_range('2019Q3', '2020Q3', freq='Q-JAN')
ts = pd.Series(np.arange(len(pr)), index=pr)
ts
```

```
2019Q3    0
2019Q4    1
2020Q1    2
2020Q2    3
2020Q3    4
Freq: Q-JAN, dtype: int64
```



```
pr = pd.date_range('2020-01-01', periods=5, freq='Q-JAN')
ts = pd.Series(np.random.randn(5), index=pr)
ts
```

```
2020-01-31   -1.051255
2020-04-30   -0.517585
2020-07-31   -0.940126
2020-10-31    0.482459
2021-01-31   -0.652840
Freq: Q-JAN, dtype: float64
```

```
ts.to_period()
```

```
2020Q4   -1.051255
2021Q1   -0.517585
2021Q2   -0.940126
2021Q3    0.482459
2021Q4   -0.652840
Freq: Q-JAN, dtype: float64
```

```
pr = pd.date_range('2020-01-01', periods=5, freq='D')
ts = pd.Series(np.random.randn(5), index=pr)
ts
```

```
2020-01-01   -0.551289
2020-01-02   -0.264403
2020-01-03   -1.003318
2020-01-04    0.601887
2020-01-05   -0.203059
Freq: D, dtype: float64
```

```
p = ts.to_period('M')
p
```

```
2020-01   -0.551289
2020-01   -0.264403
2020-01   -1.003318
2020-01    0.601887
2020-01   -0.203059
Freq: M, dtype: float64
```

```
p.to_timestamp(how='start')
```

```
2020-01-01   -0.551289
2020-01-01   -0.264403
2020-01-01   -1.003318
2020-01-01    0.601887
2020-01-01   -0.203059
dtype: float64
```


리샘플링(Resampling)

- 리샘플링(Resampling): 시계열의 빈도 변환
- 다운샘플링(Down sampling): 상위 빈도 데이터를 하위 빈도 데이터로 집계
- 업샘플링(Up sampling): 하위 빈도 데이터를 상위 빈도 데이터로 집계
- `resample` 메소드

인자	설명
`freq`	리샘플링 빈도
`axis`	리샘플링 축 (기본값 `axis=0`)
`fill_method`	업샘플링시 보간 수행 (`None`, `ffill`, `bfill`)
`closed`	다운샘플링 시 각 간격의 포함 위치 (`right`, `left`)
`label`	다운샘플링 시 집계된 결과 라벨 결정 (`right`, `left`)
`loffset`	나뉜 그룹의 라벨을 맞추기 위한 오프셋
`limit`	보간법을 사용할 때 보간을 적용할 최대 기간
`kind`	기간(`period`) 또는 타임스탬프(`timestamp`) 집계 구분
`convention`	기간을 리샘플링할 때 하위 빈도 기간에서 상위 빈도로 변환 시 방식 (`start` 또는 `end`)

```
dr = pd.date_range('2020-01-01', periods=200, freq='D')
ts = pd.Series(np.random.randn(len(dr)), index=dr)
ts
```

```
2020-01-01   -1.056421
2020-01-02   -0.094451
2020-01-03    0.344301
```

```
2020-01-04   -1.525271
2020-01-05   -1.491888
                ...
2020-07-14    0.494544
2020-07-15    0.544288
2020-07-16    0.533294
2020-07-17   -0.565676
2020-07-18   -0.065898
Freq: D, Length: 200, dtype: float64
```

```
ts.resample('M').mean()
```

```
2020-01-31   -0.191458
2020-02-29    0.153107
2020-03-31    0.139791
2020-04-30   -0.267224
2020-05-31    0.259763
2020-06-30   -0.017521
2020-07-31    0.365229
Freq: M, dtype: float64
```

```
ts.resample('M', kind='period').mean()
```

```
2020-01   -0.191458
2020-02    0.153107
2020-03    0.139791
2020-04   -0.267224
2020-05    0.259763
2020-06   -0.017521
2020-07    0.365229
Freq: M, dtype: float64
```



```
dr = pd.date_range('2020-01-01', periods=10, freq='T')
ts = pd.Series(np.arange(10), index=dr)
ts
```

```
2020-01-01 00:00:00    0
2020-01-01 00:01:00    1
2020-01-01 00:02:00    2
2020-01-01 00:03:00    3
2020-01-01 00:04:00    4
2020-01-01 00:05:00    5
2020-01-01 00:06:00    6
2020-01-01 00:07:00    7
2020-01-01 00:08:00    8
2020-01-01 00:09:00    9
Freq: T, dtype: int64
```

```
ts.resample('2T', closed='left').sum()
```

```
2020-01-01 00:00:00     1
2020-01-01 00:02:00     5
2020-01-01 00:04:00     9
2020-01-01 00:06:00    13
2020-01-01 00:08:00    17
Freq: 2T, dtype: int64
```

```
ts.resample('2T', closed='right').sum()
```

```
2019-12-31 23:58:00     0
2020-01-01 00:00:00     3
2020-01-01 00:02:00     7
2020-01-01 00:04:00    11
2020-01-01 00:06:00    15
2020-01-01 00:08:00     9
Freq: 2T, dtype: int64
```

```
ts.resample('2T', closed='right', label='right').sum()
```

```
2020-01-01 00:00:00     0
2020-01-01 00:02:00     3
2020-01-01 00:04:00     7
2020-01-01 00:06:00    11
2020-01-01 00:08:00    15
2020-01-01 00:10:00     9
Freq: 2T, dtype: int64
```

```
ts.resample('2T', closed='right', label='right', loffset='-1s').sum()
```

```
2019-12-31 23:59:59     0
2020-01-01 00:01:59     3
2020-01-01 00:03:59     7
2020-01-01 00:05:59    11
2020-01-01 00:07:59    15
2020-01-01 00:09:59     9
Freq: 2T, dtype: int64
```



```
ts.resample('2T').ohlc()
```

	open	high	low	close
2020-01-01 00:00:00	0	1	0	1
2020-01-01 00:02:00	2	3	2	3
2020-01-01 00:04:00	4	5	4	5
2020-01-01 00:06:00	6	7	6	7
2020-01-01 00:08:00	8	9	8	9

```
df = pd.DataFrame(np.random.randn(10, 4),
                  index=pd.date_range('2019-10-01', periods=10, freq='M'),
                  columns=['C1', 'C2', 'C3', 'C4'])
df
```

	C1	C2	C3	C4
2019-10-31	-1.942787	1.377695	-0.425404	-0.445326
2019-11-30	0.560426	0.665310	-0.228888	-1.205160
2019-12-31	0.887624	-1.821641	-0.613763	-1.333571
2020-01-31	1.157783	-1.730131	1.686589	0.718557
2020-02-29	-0.199500	-0.977584	-0.412742	1.807881
2020-03-31	-0.244514	-0.388788	-1.052303	0.586320
2020-04-30	0.395351	0.337783	0.270425	0.822531

```
df.resample('Y').asfreq()
```

	C1	C2	C3	C4
2019-12-31	0.887624	-1.821641	-0.613763	-1.333571
2020-12-31	NaN	NaN	NaN	NaN

```
df.resample('W-FRI').asfreq()
```

	C1	C2	C3	C4
2019-11-01	NaN	NaN	NaN	NaN
2019-11-08	NaN	NaN	NaN	NaN
2019-11-15	NaN	NaN	NaN	NaN
2019-11-22	NaN	NaN	NaN	NaN
2019-11-29	NaN	NaN	NaN	NaN
2019-12-06	NaN	NaN	NaN	NaN
2019-12-13	NaN	NaN	NaN	NaN
2019-12-20	NaN	NaN	NaN	NaN
2019-12-27	NaN	NaN	NaN	NaN
2020-01-03	NaN	NaN	NaN	NaN
2020-01-10	NaN	NaN	NaN	NaN
2020-01-17	NaN	NaN	NaN	NaN

```
df.resample('H').asfreq()
```


	C1	C2	C3	C4
2019-10-31 00:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 01:00:00	NaN	NaN	NaN	NaN
2019-10-31 02:00:00	NaN	NaN	NaN	NaN
2019-10-31 03:00:00	NaN	NaN	NaN	NaN
2019-10-31 04:00:00	NaN	NaN	NaN	NaN
...	...	...	...	...
2020-07-30 20:00:00	NaN	NaN	NaN	NaN
2020-07-30 21:00:00	NaN	NaN	NaN	NaN
2020-07-30 22:00:00	NaN	NaN	NaN	NaN
2020-07-30 23:00:00	NaN	NaN	NaN	NaN
2020-07-31 00:00:00	-2.055520	0.843720	-0.299770	1.656547

6577 rows × 4 columns

```
df.resample('H').ffill()
```

	C1	C2	C3	C4
2019-10-31 00:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 01:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 02:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 03:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 04:00:00	-1.942787	1.377695	-0.425404	-0.445326
...	...	...	...	...
2020-07-30 20:00:00	0.394069	-1.670554	-0.886881	-1.174432
2020-07-30 21:00:00	0.394069	-1.670554	-0.886881	-1.174432

```
df.resample('H').ffill(limit=2)
```

	C1	C2	C3	C4
2019-10-31 00:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 01:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 02:00:00	-1.942787	1.377695	-0.425404	-0.445326
2019-10-31 03:00:00	NaN	NaN	NaN	NaN
2019-10-31 04:00:00	NaN	NaN	NaN	NaN
...	...	...	...	...
2020-07-30 20:00:00	NaN	NaN	NaN	NaN
2020-07-30 21:00:00	NaN	NaN	NaN	NaN
2020-07-30 22:00:00	NaN	NaN	NaN	NaN
2020-07-30 23:00:00	NaN	NaN	NaN	NaN
2020-07-31 00:00:00	-2.055520	0.843720	-0.299770	1.656547

6577 rows × 4 columns


```
df.resample('Q-DEC').mean()
```

	C1	C2	C3	C4
2019-12-31	-0.164912	0.073788	-0.422685	-0.994686
2020-03-31	0.237923	-1.032168	0.073848	1.037586
2020-06-30	0.379846	-0.471367	-0.192646	-0.360235
2020-09-30	-2.055520	0.843720	-0.299770	1.656547

```
df.resample('Y').mean()
```

	C1	C2	C3	C4
2019-12-31	-0.164912	0.073788	-0.422685	-0.994686
2020-12-31	-0.028887	-0.523840	-0.093738	0.526943

무빙 윈도우(Moving Window)

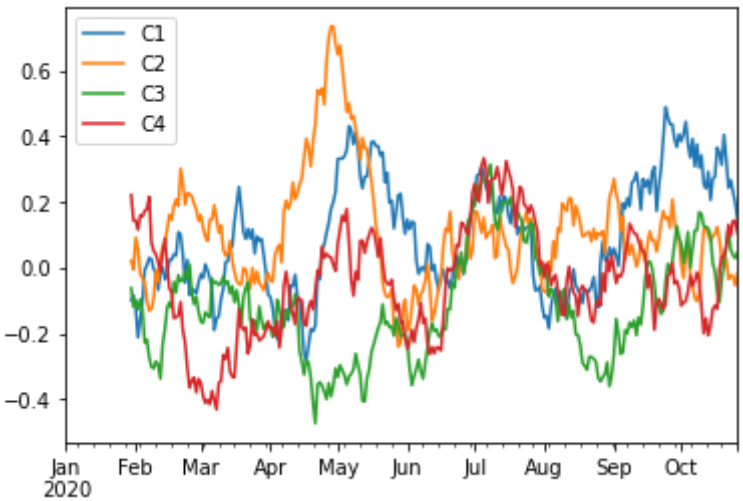
```
df = pd.DataFrame(np.random.randn(300, 4),
                  index=pd.date_range('2020-01-01', periods=300, freq='D'),
                  columns=['C1', 'C2', 'C3', 'C4'])
df
```

	C1	C2	C3	C4
2020-01-01	0.558996	-0.317758	1.276503	2.713060
2020-01-02	-0.127756	-1.450317	-0.599810	0.064586
2020-01-03	2.057265	2.112478	0.135372	1.108773
2020-01-04	0.330530	1.539590	-0.603751	-1.221122
2020-01-05	-0.967048	-0.647928	0.195913	-0.989493
...	...	...	...	...
2020-10-22	-0.752786	0.792270	0.752290	0.626881
2020-10-23	1.769088	0.594816	-0.880872	-0.235238
2020-10-24	0.799666	0.373848	0.437941	1.862933
2020-10-25	-0.349840	-1.329729	0.322836	1.290295
2020-10-26	-1.245695	-0.339869	-0.340915	-0.020850

300 rows × 4 columns

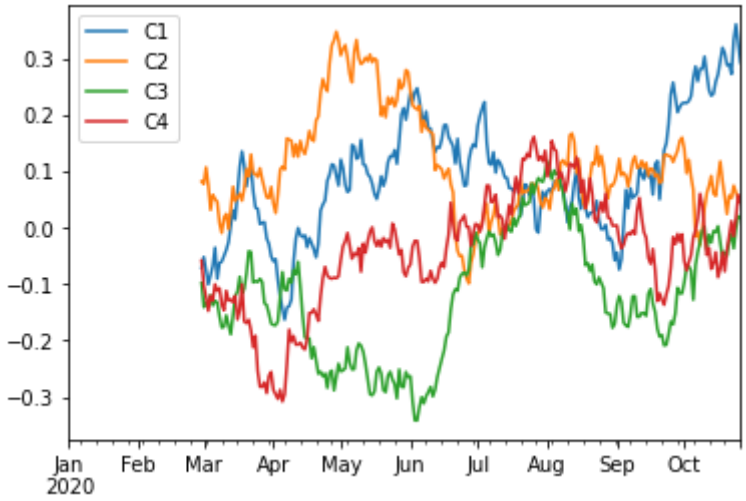

```
df.rolling(30).mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286ec72ac8>
```



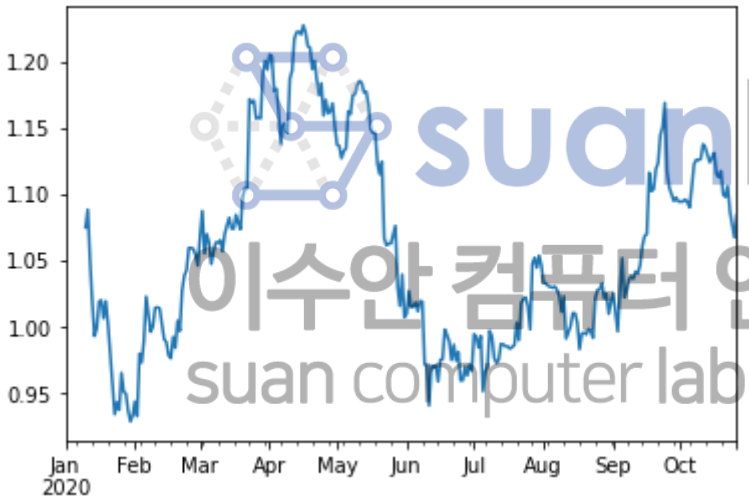
```
df.rolling(60).mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286d19bb00>
```



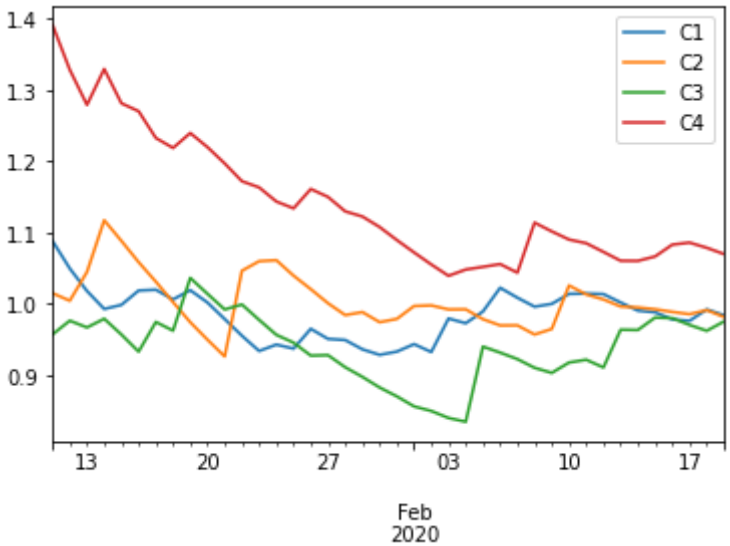

```
df.C1.rolling(60, min_periods=10).std().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286cc24048>
```



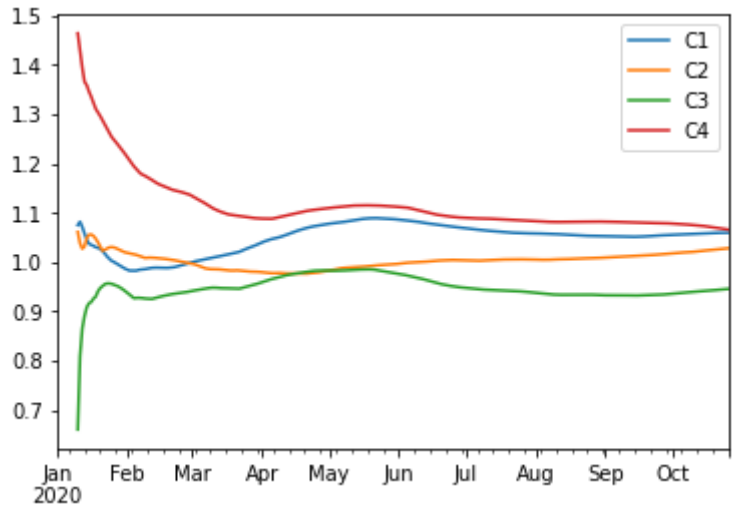

```
df.rolling(60, min_periods=10).std()[10:50].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286cb440f0>
```



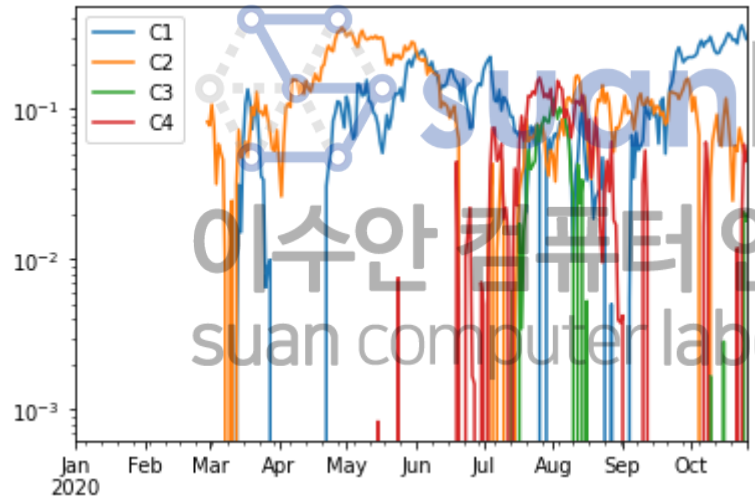
```
df.rolling(60, min_periods=10).std().expanding().mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286d5b8550>
```



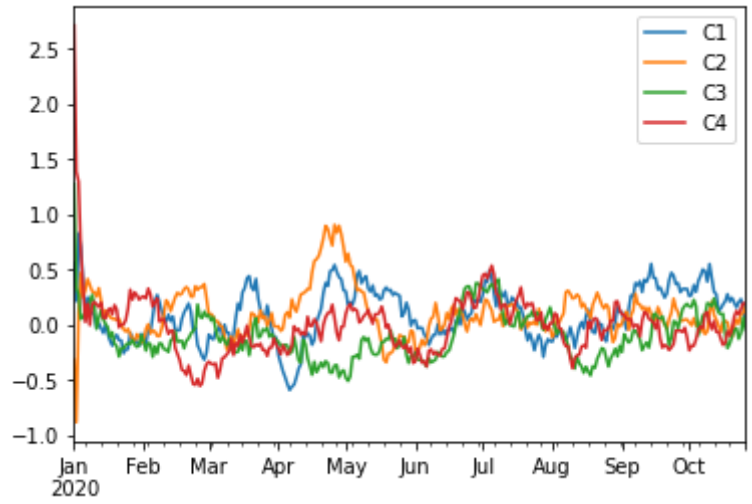

```
df.rolling(60).mean().plot(logy=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2870c48b70>
```



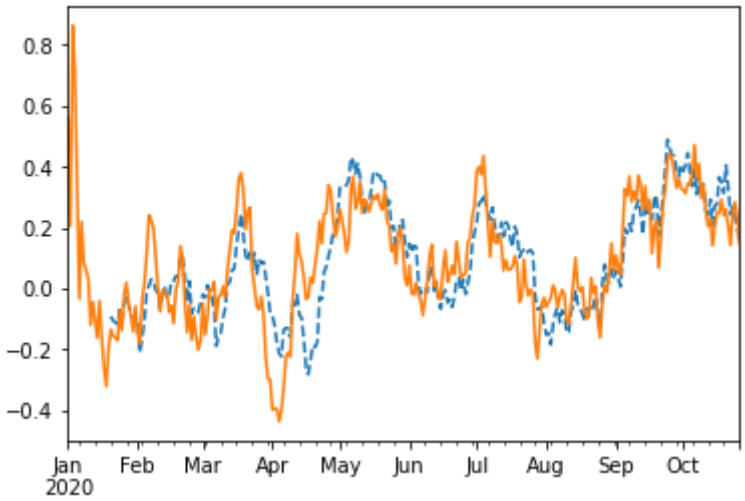

```
df.rolling('20D').mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286c80f978>
```



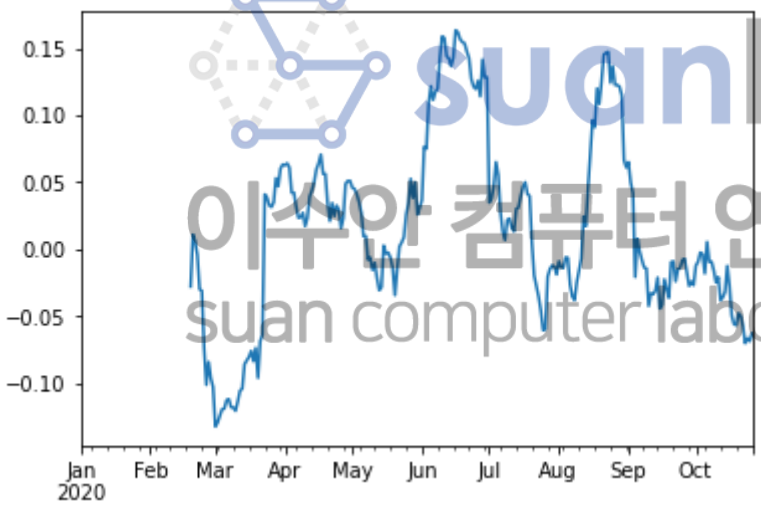
```
df.C1.rolling(30, min_periods=20).mean().plot(style='--', label='Simple MA')
df.C1.ewm(span=30).mean().plot(style='-', label='EWMA')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286c93bda0>
```




```
df.C1.rolling(100, min_periods=50).corr(df.C3).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286c935eb8>
```




```
df.C2.rolling(100, min_periods=50).corr(df.C4).plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f286c38e470>
```

데이터 읽기 및 저장

함수	설명
read_csv	파일, URL, 객체로부터 구분된 데이터 읽기 (기본 구분자: ',')
read_table	파일, URL, 객체로부터 구분된 데이터 읽기 (기본 구분자: '\t')
read_fwf	고정폭 컬럼 형식에서 데이터 읽기 (구분자 없는 데이터)
read_clipboard	클립보드에 있는 데이터 읽기. 웹페이지에 있는 표를 읽어올 때 유용
read_excel	엑셀 파일(xls, xlsx)에서 표 형식 데이터 읽기
read_hdf	Pandas에서 저장한 HDFS 파일의 데이터 읽기
read_html	HTML 문서 내의 모든 테이블 데이터 읽기
read_json	JSON에서 데이터 읽기
read_msgpack	메시지팩 바이너리 포맷으로 인코딩된 pandas 데이터 읽기
read_pickle	파이썬 피클 포맷으로 저장된 객체 읽기
read_sas	SAS 시스템의 사용자 정의 저장 포맷 데이터 읽기
read_sql	SQL 질의 결과를 DataFrame 형식으로 읽기
read_stata	Stata 파일에서 데이터 읽기
read_feather	Feather 바이너리 파일 포맷의 데이터 읽기

텍스트 파일 읽기/쓰기

```
%%writefile example1.csv
a, b, c, d, e, text
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
```

```
Writing example1.csv
```

```
!ls
```

```
example1.csv  sample_data
```

```
pd.read_csv('example1.csv')
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv

```
%%writefile example2.csv
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
```

```
11, 12, 13, 14, 15, csv
```

```
Writing example2.csv
```

```
pd.read_csv('example2.csv', header=None)
```

	0	1	2	3	4	5
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv

```
pd.read_csv('example2.csv', names=['a', 'b', 'c', 'd', 'e', 'text'])
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv

```
pd.read_csv('example2.csv', names=['a', 'b', 'c', 'd', 'e', 'text'], index_col='text')
```

	a	b	c	d	e
text					
hi	1	2	3	4	5
pandas	6	7	8	9	10
csv	11	12	13	14	15


```
%%writefile example3.txt
     a     b     c
1  0.1   0.2   0.3
2  0.4   0.5   0.6
3  0.7   0.8   0.9
```

```
Writing example3.txt
```

```
pd.read_table('example3.txt', sep='Ws+')
```

	a	b	c
1	0.1	0.2	0.3
2	0.4	0.5	0.6
3	0.7	0.8	0.9

```
%%writefile example4.csv
```

```
%%writefile example4.csv
# 파일 설명
a, b, c, d, e, text
# 컬럼은 a, b, c, d, e와 text가 있음
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
```

```
Writing example4.csv
```

```
pd.read_csv('example4.csv', skiprows=[0, 2])
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv

```
%%writefile example5.csv
a, b, c, d, e, text
1, 2, NA, 4, 5, hi
6, 7, 8, NULL, 10, pandas
11, NA, 13, 14, 15, csv
```

```
Writing example5.csv
```



```
pd.read_csv('example5.csv')
```

	a	b	c	d	e	text
0	1	2	NA	4	5	hi
1	6	7	8	NULL	10	pandas
2	11	NA	13	14	15	csv

```
%%writefile example6.csv
a, b, c, d, e, text
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
11, 12, 13, 14, 15, csv
1, 2, 3, 4, 5, hi
6, 7, 8, 9, 10, pandas
```

```
11, 12, 13, 14, 15, csv
```

```
Writing example6.csv
```

```
pd.read_csv('example6.csv', nrows=5)
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv
3	1	2	3	4	5	hi
4	6	7	8	9	10	pandas

```
df = pd.read_csv('example6.csv')
df
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv
3	1	2	3	4	5	hi
4	6	7	8	9	10	pandas
5	11	12	13	14	15	csv
6	1	2	3	4	5	hi
7	6	7	8	9	10	pandas
8	11	12	13	14	15	csv
9	1	2	3	4	5	hi
10	6	7	8	9	10	pandas
11	11	12	13	14	15	csv
12	1	2	3	4	5	hi
13	6	7	8	9	10	pandas
14	11	12	13	14	15	csv


```
df.to_csv('output.csv')
```

```
!cat output.csv
```

```
,a, b, c, d, e, text
0,1,2,3,4,5, hi
```

```
1,6,7,8,9,10, pandas
2,11,12,13,14,15, csv
3,1,2,3,4,5, hi
4,6,7,8,9,10, pandas
5,11,12,13,14,15, csv
6,1,2,3,4,5, hi
7,6,7,8,9,10, pandas
8,11,12,13,14,15, csv
9,1,2,3,4,5, hi
10,6,7,8,9,10, pandas
11,11,12,13,14,15, csv
12,1,2,3,4,5, hi
13,6,7,8,9,10, pandas
14,11,12,13,14,15, csv
```

```
dr = pd.date_range('2020-01-01', periods=10)
ts = pd.Series(np.arange(10), index=dr)
ts
```

```
2020-01-01    0
2020-01-02    1
2020-01-03    2
2020-01-04    3
2020-01-05    4
2020-01-06    5
2020-01-07    6
2020-01-08    7
2020-01-09    8
2020-01-10    9
Freq: D, dtype: int64
```



```
ts.to_csv('ts.csv', header=['value'])
```

```
!cat ts.csv
```

```
,value
2020-01-01,0
2020-01-02,1
2020-01-03,2
2020-01-04,3
2020-01-05,4
2020-01-06,5
2020-01-07,6
2020-01-08,7
2020-01-09,8
2020-01-10,9
```

```
%%writefile example.json
[{"a":1, "b":2, "c":3, "d":4, "e":5},
 {"a":6, "b":7, "c":8, "d":9, "e":10},
 {"a":11, "b":12, "c":13, "d":14, "e":15}]
```

```
Writing example.json
```

```
!cat example.json
```

```
[{"a":1, "b":2, "c":3, "d":4, "e":5},
 {"a":6, "b":7, "c":8, "d":9, "e":10},
 {"a":11, "b":12, "c":13, "d":14, "e":15}]
```

```
pd.read_json('example.json')
```

	a	b	c	d	e
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

```
ts.to_json("output.json")
```

```
!cat output.json
```

```
{"1577836800000":0,"1577923200000":1,"1578009600000":2,"1578096000000":3,"1578182400000":4,"
```

```
df.to_json("output.json")
```

```
!cat output.json
```

```
{"a":{"0":1,"1":6,"2":11,"3":1,"4":6,"5":11,"6":1,"7":6,"8":11,"9":1,"10":6,"11":11,"12":1,"
```


이진 데이터 파일 읽기/쓰기

```
df = pd.read_csv('example1.csv')
df
```

	a	b	c	d	e	text
0	1	2	3	4	5	hi
1	6	7	8	9	10	pandas
2	11	12	13	14	15	csv

```
df.to_pickle('df_pickle')
pd.read_pickle('df_pickle')
```

```
          a    b    c    d    e    text
```

```
df = pd.DataFrame({'a': np.random.randn(100),
                   'b': np.random.randn(100),
                   'c': np.random.randn(100)})
df
```

	a	b	c
0	-1.261616	1.382712	-0.426104
1	1.057429	-2.146040	-0.838184
2	0.610527	0.014495	1.148332
3	-0.035792	0.354276	-1.433421
4	-0.170993	0.382314	0.532502
...	...	...	...
95	1.270977	-0.844213	0.704958
96	-0.306064	1.620980	0.367980
97	-0.298462	-0.510507	1.416835
98	-0.633801	-1.552177	0.594091
99	0.647711	0.881097	1.686910

100 rows × 3 columns


```
h = pd.HDFStore('date.h5')
h['obj1'] = df
h['obj1_col1'] = df['a']
h['obj1_col2'] = df['b']
h['obj1_col3'] = df['c']
h
```

```
<class 'pandas.io.pytables.HDFStore'>
File path: date.h5
```

```
h['obj1']
```

	a	b	c
0	-1.261616	1.382712	-0.426104
1	1.057429	-2.146040	-0.838184
2	0.610527	0.014495	1.148332
3	-0.035792	0.354276	-1.433421
4	-0.170993	0.382314	0.532502

```
h.put('obj2', df, format='table')
```

```
h.select('obj2', where=['index > 50 and index <= 60'])
```

	a	b	c
51	0.335940	0.990830	1.328594
52	-0.546832	0.713767	0.155539
53	-0.154082	-1.657008	-0.500032
54	2.422587	-0.254522	-0.355217
55	-0.468801	-0.768493	0.505517
56	0.667608	1.713248	-1.843119
57	1.883762	0.199250	1.296229
58	0.621591	-0.108326	-0.650358
59	0.650828	-0.873431	1.738071
60	0.299669	-0.867400	-0.440471


```
h.close()
```

```
df.to_hdf('data.h5', 'obj3', format='table')
```

```
pd.read_hdf('data.h5', 'obj3', where=['index < 10'])
```

	a	b	c
0	-1.261616	1.382712	-0.426104
1	1.057429	-2.146040	-0.838184

```
df.to_excel('example.xlsx', 'Sheet1')
```

3	-0.035792	0.354276	-1.433421

```
!ls
```

```
data.h5    example1.csv  example4.csv  example.json  output.csv   ts.csv
date.h5    example2.csv  example5.csv  example.xlsx  output.json
df_pickle  example3.txt  example6.csv  exmaple3.txt  sample_data
```

7	1.645011	0.846613	0.593269

```
pd.read_excel('example.xlsx', 'Sheet1')
```

	Unnamed: 0	a	b	c
0	0	-1.261616	1.382712	-0.426104
1	1	1.057429	-2.146040	-0.838184
2	2	0.610527	0.014495	1.148332
3	3	-0.035792	0.354276	-1.433421
4	4	-0.170993	0.382314	0.532502
...	...	...	...	...
95	95	1.270977	-0.844213	0.704958
96	96	-0.306064	1.620980	0.367980
97	97	-0.298462	-0.510507	1.416835
98	98	-0.633801	-1.552177	0.594091
99	99	0.647711	0.881097	1.686910

100 rows × 4 columns

데이터 정제

누락값 처리

- 대부분의 실제 데이터들은 정제되지 않고 누락값들이 존재
- 서로 다른 데이터들은 다른 형태의 결측을 가짐
- 결측 데이터는 `null`, `NaN`, `NA` 로 표기

None: 파이썬 누락 데이터

```
a = np.array([1, 2, None, 4, 5])
```

```
a = np.array([1, 2, None, 4, 5])
a
```

```
array([1, 2, None, 4, 5], dtype=object)
```

```
#a.sum()
```

NaN: 누락된 수치 데이터

```
a = np.array([1, 2, np.nan, 4, 5])
a.dtype
```

```
dtype('float64')
```

```
0 + np.nan
```

```
nan
```

```
np.nan + np.nan
```

```
nan
```



```
a.sum(), a.min(), a.max()
```

```
(nan, nan, nan)
```

```
np.nansum(a), np.nanmin(a), np.nanmax(a)
```

```
(12.0, 1.0, 5.0)
```

```
pd.Series([1, 2, np.nan, 4, None])
```

```
0    1.0
1    2.0
2    NaN
3    4.0
4    NaN
dtype: float64
```

```
s = pd.Series(range(5), dtype=int)
s
```

```
0    0
1    1
2    2
3    3
4    4
dtype: int64
```

```
s[0] = None
s
```

```
s
```

```
0    NaN
1    1.0
2    2.0
3    3.0
4    4.0
dtype: float64
```

```
s[3] = np.nan
```

```
s = pd.Series([True, False, None, np.nan])
s
```

```
0     True
1    False
2     None
3      NaN
dtype: object
```

Null 값 처리

인자	설명
isnull()	누락되거나 NA인 값을 불리언 값으로 반환
notnull()	issull()의 반대
dropna()	누락된 데이터가 있는 축 제외
fillna()	누락된 값을 대체하거나 ffill 이나 bfill 로 보간 메소드 적용


```
s = pd.Series([1, 2, np.nan, 'String', None])
s
```

```
0         1
1         2
2       NaN
3    String
4      None
dtype: object
```

```
s.isnull()
```

```
0    False
1    False
2     True
3    False
4     True
dtype: bool
```

```
s[s.notnull()]
```

```
0         1
1         2
3    String
dtype: object
```

```
s.dropna()
```

```
0         1
1         2
3    String
dtype: object
```

```
df.dropna(axis='columns')
```

	a	b	c
0	-1.261616	1.382712	-0.426104
1	1.057429	-2.146040	-0.838184
2	0.610527	0.014495	1.148332
3	-0.035792	0.354276	-1.433421
4	-0.170993	0.382314	0.532502
...	...	...	...
95	1.270977	-0.844213	0.704958
96	-0.306064	1.620980	0.367980
97	-0.298462	-0.510507	1.416835
98	-0.633801	-1.552177	0.594091
99	0.647711	0.881097	1.686910

100 rows × 3 columns


```
df[3] = np.nan
df
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN

```
df.dropna(axis='columns', how='all')
```

	a	b	c
0	-1.261616	1.382712	-0.426104
1	1.057429	-2.146040	-0.838184
2	0.610527	0.014495	1.148332
3	-0.035792	0.354276	-1.433421
4	-0.170993	0.382314	0.532502
...	...	...	...
95	1.270977	-0.844213	0.704958
96	-0.306064	1.620980	0.367980
97	-0.298462	-0.510507	1.416835
98	-0.633801	-1.552177	0.594091
99	0.647711	0.881097	1.686910

100 rows × 3 columns


```
df.dropna(axis='rows', thresh=3)
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN
1	1.057429	-2.146040	-0.838184	NaN
2	0.610527	0.014495	1.148332	NaN
3	-0.035792	0.354276	-1.433421	NaN
4	-0.170993	0.382314	0.532502	NaN
...	...	...	...	...
95	1.270977	-0.844213	0.704958	NaN
96	-0.306064	1.620980	0.367980	NaN
97	-0.298462	-0.510507	1.416835	NaN
98	-0.633801	-1.552177	0.594091	NaN
99	0.647711	0.881097	1.686910	NaN

100 rows × 4 columns

```
s
```

```
0         1
1         2
2       NaN
3    String
4      None
dtype: object
```

```
s.fillna(0)
```

```
0         1
1         2
2         0
3    String
4         0
dtype: object
```

```
s.fillna(method='ffill')
```

```
0         1
1         2
2         2
3    String
4    String
dtype: object
```

```
s.fillna(method='bfill')
```

```
0         1
1         2
2    String
3    String
4      None
dtype: object
```



```
df
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN

```
df.fillna(method='ffill', axis=0)
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN
1	1.057429	-2.146040	-0.838184	NaN
2	0.610527	0.014495	1.148332	NaN
3	-0.035792	0.354276	-1.433421	NaN
4	-0.170993	0.382314	0.532502	NaN
...	...	...	...	...
95	1.270977	-0.844213	0.704958	NaN
96	-0.306064	1.620980	0.367980	NaN
97	-0.298462	-0.510507	1.416835	NaN
98	-0.633801	-1.552177	0.594091	NaN
99	0.647711	0.881097	1.686910	NaN

100 rows × 4 columns


```
df.fillna(method='ffill', axis=1)
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	-0.426104
1	1.057429	-2.146040	-0.838184	-0.838184
2	0.610527	0.014495	1.148332	1.148332
3	-0.035792	0.354276	-1.433421	-1.433421
4	-0.170993	0.382314	0.532502	0.532502
...	...	...	...	...
95	1.270977	-0.844213	0.704958	0.704958
96	-0.306064	1.620980	0.367980	0.367980
97	-0.298462	-0.510507	1.416835	1.416835
98	-0.633801	-1.552177	0.594091	0.594091
99	0.647711	0.881097	1.686910	1.686910

100 rows × 4 columns

```
df.fillna(method='bfill', axis=0)
```

	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN
1	1.057429	-2.146040	-0.838184	NaN
2	0.610527	0.014495	1.148332	NaN
3	-0.035792	0.354276	-1.433421	NaN
4	-0.170993	0.382314	0.532502	NaN
...	...	...	...	...
95	1.270977	-0.844213	0.704958	NaN
96	-0.306064	1.620980	0.367980	NaN
97	-0.298462	-0.510507	1.416835	NaN
98	-0.633801	-1.552177	0.594091	NaN
99	0.647711	0.881097	1.686910	NaN

100 rows × 4 columns

```
df.fillna(method='bfill', axis=1)
```


	a	b	c	3
0	-1.261616	1.382712	-0.426104	NaN
1	1.057429	-2.146040	-0.838184	NaN
2	0.610527	0.014495	1.148332	NaN
3	-0.035792	0.354276	-1.433421	NaN
4	-0.170993	0.382314	0.532502	NaN
...	...	...	...	...
95	1.270977	-0.844213	0.704958	NaN
96	-0.306064	1.620980	0.367980	NaN
97	-0.298462	-0.510507	1.416835	NaN
98	-0.633801	-1.552177	0.594091	NaN
99	0.647711	0.881097	1.686910	NaN

100 rows × 4 columns

중복 제거

```
df = pd.DataFrame({'c1': ['a', 'b', 'c'] * 2 + ['b'] + ['c'],
                   'c2': [1, 2, 1, 1, 2, 3, 3, 4]})
df
```

	c1	c2
0	a	1
1	b	2
2	c	1
3	a	1
4	b	2
5	c	3
6	b	3
7	c	4

```
df.duplicated()
```

```
0    False
1    False
2    False
3     True
4     True
5    False
6    False
7    False
dtype: bool
```



```
df.drop_duplicates()
```

	c1	c2
0	a	1
1	b	2
2	c	1
5	c	3
6	b	3
7	c	4

값 치환

```
s = pd.Series([1., 2., -999., 3., -1000., 4.])
s
```

```
0       1.0
1       2.0
2    -999.0
3       3.0
4   -1000.0
5       4.0
dtype: float64
```

```
s.replace(-999, np.nan)
```

```
0       1.0
1       2.0
2       NaN
3       3.0
4   -1000.0
5       4.0
dtype: float64
```

```
s.replace([-999, -1000], np.nan)
```

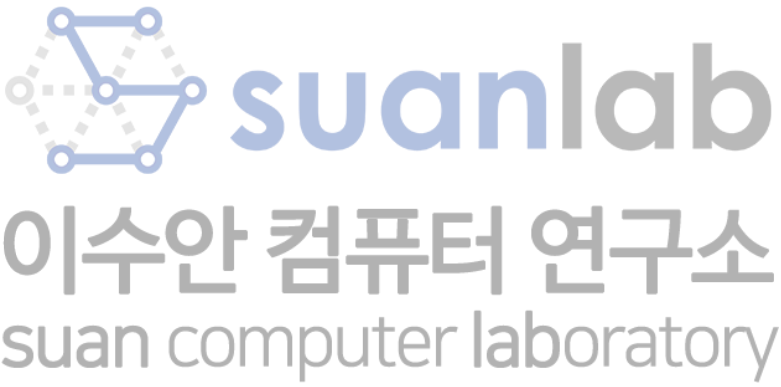
```
0    1.0
1    2.0
2    NaN
3    3.0
4    NaN
5    4.0
dtype: float64
```

```
s.replace([-999, -1000], [np.nan, 0])
```

```
0    1.0
1    2.0
2    NaN
3    3.0
4    0.0
5    4.0
dtype: float64
```


참고문헌

- Pandas 사이트: https://pandas.pydata.org/
- Jake VanderPlas, "Python Data Science Handbook", O'Reilly
- Wes Mckinney, "Python for Data Analysis", O'Reilly


suanlab
이수안 컴퓨터 연구소
suan computer laboratory