Pandas 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
pd.__version__
```

```
'1.1.5'
```

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
```

```
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	0.459657	0.846327	0.003002	0.681628	0.489999
2	0.829306	0.996998	0.133635	0.622974	0.852390
3	0.121915	0.207136	0.199623	0.693491	0.947054
4	0.314747	0.648947	0.828961	0.027551	0.243747
5	0.531377	0.418425	0.087952	0.173818	0.676693

```
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
부산광역시	3404423	1668618	1735805
인천광역시	2947217	1476813	1470404
대구광역시	2427954	1198815	1229139
대전광역시	1471040	734441	736599
광주광역시	1455048	720060	734988

```
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
```

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

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
`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
```

```
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
서울특별시	9720846	4732275
부산광역시	3404423	1668618
인천광역시	2947217	1476813

```
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]
```

	인구수	남자인구수
서울특별시	9720846	4732275
부산광역시	3404423	1668618
인천광역시	2947217	1476813

데이터 연산

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

```
0    5
1    3
2    1
3    5
4    8
dtype: int64
```

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

	A	B	C
0	7	9	4
1	1	7	7
2	7	6	1

```
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	19	15	5
1	5	19	7
2	6	17	15

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

	B	A	E	C	D
0	14	16	10	1	10
1	2	16	3	11	10
2	12	16	7	2	17
3	5	13	2	4	12
4	3	1	6	3	14

```
df1 + df2
```

	A	B	C	D	E
0	35.0	NaN	16.0	15.0	NaN

연산자 범용 함수

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

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	9	1	6	0
4	0	5	6	3
1	7	5	3	2
3	6	3	1	3

```
df.sort_index()
```

	B	D	A	C
1	7	5	3	2
2	9	1	6	0
3	6	3	1	3
4	0	5	6	3

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

	A	B	C	D
2	6	9	0	1
4	6	0	3	5
1	3	7	2	5
3	1	6	3	3

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

	B	D	A	C
3	6	3	1	3
1	7	5	3	2
2	9	1	6	0
4	0	5	6	3

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

```
   B  D  A  C
```

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')
```

```
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
```

데이터 결합

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
2	임꺽정	컴퓨터학과	2019
3	김유신	통계학과	2020

```
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
3	임꺽정	컴퓨터학과	프로그래밍
4	임꺽정	컴퓨터학과	운영체제
5	김유신	통계학과	확률론

```
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
0	홍길동	경영학과	A
1	이순신	교육학과	A+
2	임꺽정	컴퓨터학과	B

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

```
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)
```

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

```
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
0	홍길동	햄버거	콜라
1	이순신	피자	사이다

```
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
2	임꺽정	3
3	김유신	2

```
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

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

```
df = pd.DataFrame([[-1.2, 1.2, 2.1],
                   [2.4, 5.5, -4.2],
                   [0.4, -4.3, -1.3],
                   [3.2, 3.1, -4.1]],
                  index=[1, 2, 3, 4],
                  columns=['A', 'B', 'C'])
df
```

	A	B	C
1	-1.2	1.2	2.1
2	2.4	5.5	-4.2
3	0.4	-4.3	-1.3
4	3.2	3.1	-4.1

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

	A	B	C
1	-1.2	1.2	2.1
2	2.4	5.5	-4.2

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

	A	B	C
3	0.4	-4.3	-1.3
4	3.2	3.1	-4.1

```
df.describe()
```

	A	B	C
count	4.000000e+00	4.000000	4.000000
mean	1.200000e+00	1.375000	-1.875000
std	1.986622e+00	4.172429	2.971391
min	-1.200000e+00	-4.300000	-4.200000
25%	5.551115e-17	-0.175000	-4.125000
50%	1.400000e+00	2.150000	-2.700000
75%	2.600000e+00	3.700000	-0.450000
max	3.200000e+00	5.500000	2.100000

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

```
     A    B    C
1 -1.2  1.2  2.1
2  2.4  5.5 -4.2
3  0.4 -4.3 -1.3
4  3.2  3.1 -4.1
7 4
```

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

```
     A    B    C
1 -1.2  1.2  2.1
2  2.4  5.5 -4.2
3  0.4 -4.3 -1.3
4  3.2  3.1 -4.1
A    1
B    3
C    2
dtype: int64
A    4
B    2
C    1
dtype: int64
```

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

```
     A    B    C
1 -1.2  1.2  2.1
2  2.4  5.5 -4.2
3  0.4 -4.3 -1.3
4  3.2  3.1 -4.1
A    1.986622
B    4.172429
```

```
C    2.971391
dtype: float64
A     3.946667
B    17.409167
C     8.829167
dtype: float64
```

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

```
     A    B    C
1 -1.2  1.2  2.1
2  2.4  5.5 -4.2
3  0.4 -4.3 -1.3
4  3.2  3.1 -4.1
A    4.8
B    5.5
C   -7.5
dtype: float64
     A    B    C
1 -1.2  1.2  2.1
2  1.2  6.7 -2.1
3  1.6  2.4 -3.4
4  4.8  5.5 -7.5
```

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

```
     A    B    C
1 -1.2  1.2  2.1
2  2.4  5.5 -4.2
3  0.4 -4.3 -1.3
4  3.2  3.1 -4.1
A    -3.6864
B   -87.9780
C   -47.0106
dtype: float64
        A       B        C
1 -1.2000   1.200   2.1000
2 -2.8800   6.600  -8.8200
3 -1.1520 -28.380  11.4660
4 -3.6864 -87.978 -47.0106
```

```
df.diff()
```

	A	B	C
1	NaN	NaN	NaN
2	3.6	4.3	-6.3
3	-2.0	-9.8	2.9
4	2.8	7.4	-2.8

```
df.corr()
```

	A	B	C
A	1.000000	0.537256	-0.973511
B	0.537256	1.000000	-0.467484
C	-0.973511	-0.467484	1.000000

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

```
A    0.537256
B    1.000000
C   -0.467484
dtype: float64
```

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	6	0.060141
1	a	B	6	0.915126
2	b	B	6	0.056689
3	b	A	6	0.692056
4	c	D	6	0.129076
5	d	C	6	0.932201
6	b	C	6	0.126522

```
df.dtypes
```

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

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

```
c1
```

```
a    6
b    6
c    6
d    6
Name: c3, dtype: int64
```

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

```
c2
A    0.446831
B    0.607007
C    0.569701
D         NaN
Name: c4, dtype: float64
```

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

```
c1  c2
a   A     0.060141
    B     0.915126
b   A     0.692056
    B     0.056689
    C     0.126522
c   D     0.129076
d   C     0.932201
Name: c4, dtype: float64
```

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

c2	A	B	C	D
c1				
a	0.060141	0.915126	NaN	NaN
b	0.692056	0.056689	0.126522	NaN
c	NaN	NaN	NaN	0.129076
d	NaN	NaN	0.932201	NaN

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

	c3	c4
c1		
a	6	0.487634
b	6	0.291756
c	6	0.129076
d	6	0.932201

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

		c3	c4
c1	**c2**		
a	**A**	6	0.060141
	B	6	0.915126
b	**A**	6	0.692056
	B	6	0.056689
	C	6	0.126522
c	**D**	6	0.129076
d	**C**	6	0.932201

```
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   6  0.060141
1  a  B   6  0.915126
b
  c1 c2  c3        c4
2  b  B   6  0.056689
3  b  A   6  0.692056
6  b  C   6  0.126522
c
  c1 c2  c3        c4
4  c  D   6  0.129076
d
  c1 c2  c3        c4
5  d  C   6  0.932201
```

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

```
('a', 'A')
  c1 c2  c3        c4
```

```
0  a  A   6  0.060141
('a', 'B')
  c1 c2  c3        c4
1  a  B   6  0.915126
('b', 'A')
  c1 c2  c3        c4
3  b  A   6  0.692056
('b', 'B')
  c1 c2  c3        c4
2  b  B   6  0.056689
('b', 'C')
  c1 c2  c3        c4
6  b  C   6  0.126522
('c', 'D')
  c1 c2  c3        c4
4  c  D   6  0.129076
('d', 'C')
  c1 c2  c3        c4
5  d  C   6  0.932201
```

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

		c4
c1	**c2**	
a	**A**	0.060141
	B	0.915126
b	**A**	0.692056
	B	0.056689
	C	0.126522
c	**D**	0.129076
d	**C**	0.932201

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

```
c1
a    6.0
b    6.0
c    6.0
d    6.0
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    0.487634
b    0.126522
c    0.129076
d    0.932201
Name: c4, dtype: float64
```

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

```
c1
a    0.604566
b    0.348424
c         NaN
d         NaN
Name: c4, dtype: float64
```

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

		mean	min	max
c1	**c2**			
a	**A**	0.060141	0.060141	0.060141
	B	0.915126	0.915126	0.915126
b	**A**	0.692056	0.692056	0.692056
	B	0.056689	0.056689	0.056689
	C	0.126522	0.126522	0.126522
c	**D**	0.129076	0.129076	0.129076
d	**C**	0.932201	0.932201	0.932201

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

	c1	c2	c4
0	a	A	0.060141
1	a	B	0.915126
2	b	A	0.692056
3	b	B	0.056689
4	b	C	0.126522
5	c	D	0.129076
6	d	C	0.932201

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

```
c1  c2
a   A     0.060141
    B     0.915126
b   A     0.692056
    B     0.056689
    C     0.126522
c   D     0.129076
d   C     0.932201
Name: c4, dtype: float64
```

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	6.0	6.0	NaN	NaN	0.060141	0.915126	NaN	NaN
b	6.0	6.0	6.0	NaN	0.692056	0.056689	0.126522	NaN
c	NaN	NaN	NaN	6.0	NaN	NaN	NaN	0.129076
d	NaN	NaN	6.0	NaN	NaN	NaN	0.932201	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	6.0	6.0	NaN	NaN	6	0.060141	0.915126	NaN	NaN	0.487634

```
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	6.0	6.0	NaN	NaN	12	0.060141	0.915126	NaN	NaN	0.975267
b	6.0	6.0	6.0	NaN	18	0.692056	0.056689	0.126522	NaN	0.875267
c	NaN	NaN	NaN	6.0	6	NaN	NaN	NaN	0.129076	0.129076
d	NaN	NaN	6.0	NaN	6	NaN	NaN	0.932201	NaN	0.932201
All	12.0	12.0	12.0	6.0	42	0.752197	0.971815	1.058723	0.129076	2.911811

```
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	6	6	0	0	12	0.060141	0.915126	0.000000	0.000000	0.975267
b	6	6	6	0	18	0.692056	0.056689	0.126522	0.000000	0.875267
c	0	0	0	6	6	0.000000	0.000000	0.000000	0.129076	0.129076
d	0	0	6	0	6	0.000000	0.000000	0.932201	0.000000	0.932201
All	12	12	12	6	42	0.752197	0.971815	1.058723	0.129076	2.911811

```
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

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

c2	A	B	C	D	All
c1					
a	6.0	6.0	NaN	NaN	12
b	6.0	6.0	6.0	NaN	18
c	NaN	NaN	NaN	6.0	6
d	NaN	NaN	6.0	NaN	6
All	12.0	12.0	12.0	6.0	42

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
```

```
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
```

```
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')
```

	c1	c2	c3	c4
0	a	A	6	0.060141
1	a	B	6	0.915126
2	b	B	6	0.056689
3	b	A	6	0.692056
4	c	D	6	0.129076
5	d	C	6	0.932201

```
df[3] = np.nan
df
```

	c1	c2	c3	c4	3
0	a	A	6	0.060141	NaN
1	a	B	6	0.915126	NaN
2	b	B	6	0.056689	NaN
3	b	A	6	0.692056	NaN
4	c	D	6	0.129076	NaN
5	d	C	6	0.932201	NaN
6	b	C	6	0.126522	NaN

```
df.dropna(axis='columns', how='all')
```

	c1	c2	c3	c4
0	a	A	6	0.060141
1	a	B	6	0.915126
2	b	B	6	0.056689
3	b	A	6	0.692056
4	c	D	6	0.129076
5	d	C	6	0.932201
6	b	C	6	0.126522

```
df.dropna(axis='rows', thresh=3)
```

	c1	c2	c3	c4	3
0	a	A	6	0.060141	NaN
1	a	B	6	0.915126	NaN
2	b	B	6	0.056689	NaN
3	b	A	6	0.692056	NaN

```
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
```

	c1	c2	c3	c4	3
0	a	A	6	0.060141	NaN
1	a	B	6	0.915126	NaN
2	b	B	6	0.056689	NaN

```
df.fillna(method='ffill', axis=0)
```

	c1	c2	c3	c4	3
0	a	A	6	0.060141	NaN
1	a	B	6	0.915126	NaN
2	b	B	6	0.056689	NaN
3	b	A	6	0.692056	NaN
4	c	D	6	0.129076	NaN
5	d	C	6	0.932201	NaN
6	b	C	6	0.126522	NaN

```
df.fillna(method='ffill', axis=1)
```

	c1	c2	c3	c4	3
0	a	A	6	0.0601408	0.0601408
1	a	B	6	0.915126	0.915126
2	b	B	6	0.0566891	0.0566891
3	b	A	6	0.692056	0.692056
4	c	D	6	0.129076	0.129076
5	d	C	6	0.932201	0.932201
6	b	C	6	0.126522	0.126522

```
df.fillna(method='bfill', axis=0)
```

	c1	c2	c3	c4	3

```
df.fillna(method='bfill', axis=1)
```

	c1	c2	c3	c4	3
0	a	A	6	0.0601408	NaN
1	a	B	6	0.915126	NaN
2	b	B	6	0.0566891	NaN
3	b	A	6	0.692056	NaN
4	c	D	6	0.129076	NaN
5	d	C	6	0.932201	NaN
6	b	C	6	0.126522	NaN

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