

# Chapter01.

## Pandas

# 목차

1. Pandan 패키지 – p.3
2. Series 문법 – p.4
3. DataFrame 문법 – p.9
4. groupby – p.45
5. apply – p.51
6. pivot table – p.53

# 1. Pandas 패키지

- 데이터 분석을 위한 고수준의 자료구조와 데이터분석도구 제공
- 고수준 자료구조 : 벡터 또는 행렬 자료를 효과적으로 처리할 수 있는 자료구조
- 주요 데이터분석도구 : 색인기능, 자료구조변환, 수학 및 통계 연산 등
- Series : 벡터 자료를 다루는 데 효과적인 자료구조와 분석도구 제공 클래스
- DataFrame : 행렬 자료를 다루는 데 효과적인 자료구조와 분석도구 제공 클래스

## 2. Series 문법

- Series 특징

- ✓ 리스트를 가지고 생성하고, 기본적으로는 리스트처럼 정수 인덱스를 이용해서 순서대로 저장
- ✓ values 속성을 호출하면 데이터를 배열 원소로 반환
- ✓ Index 속성을 호출하면 인덱스 정보를 반환
- ✓ 각각의 데이터는 [인덱스]를 이용해서 접근 가능

```
price = Series([4000, 3000, 3500, 2000])
```

```
print(price)
```

```
0    4000
```

```
1    3000
```

```
2    3500
```

```
3    2000
```

```
print(price[0]) # 4000
```

## ● Series 기능

- ✓ 인덱스를 리스트의 형태로 대입하면 인덱스에 해당하는 데이터 반환
- ✓ 인덱스로 조건을 입력하면 조건에 맞는 데이터만 반환
- ✓ 정수 데이터와 산술 연산 가능
- ✓ numpy 함수 사용 가능
- ✓ 연산을 할 때는 values의 값을 가지고 연산
- ✓ dict 객체로 생성 가능

```

price = Series([4000, 3000, 3500, 2000])
print(price)
print(price.index)
print(price.values)
print('=====')
fruit = Series([4000, 3000, 3500, 2000],
               index=['apple', 'mellon', 'orange', 'kiwi'])
print(fruit)
print(fruit[0]) # 순번 이용 데이터 접근
print(fruit['apple']) # 인덱스 이용 데이터 접근

print(fruit[fruit>3000]) # 부울리언 식
print("=====")

```

```

0    4000
1    3000
2    3500
3    2000
dtype: int64
RangeIndex(start=0, stop=4,
step=1)
[4000 3000 3500 2000]
=====
apple    4000
mellon   3000
orange   3500
kiwi     2000
dtype: int64
4000
4000
apple    4000
orange   3500
dtype: int64
=====

```

- Series 기능

- ✓ `pandas.isnull(Series객체)`: 데이터가 없는 경우 True 있는 경우는 False 반환
- ✓ `pandas.isnotnull(Series객체)`: 데이터가 있는 경우는 True 없는 경우는 False 반환
- ✓ Series 끼리의 산술 연산은 인덱스가 동일한 데이터끼리 연산
- ✓ 한쪽에만 존재하거나 None 데이터와의 연산은 NaN

```
from pandas import Series, DataFrame
import numpy as np
import pandas as pd
goods1 = Series([4000, 3500, None, 2000],
                 index=['apple', 'mango', 'orange', 'kiwi'])
goods2 = Series([3000, 3000, 3500, 2000],
                 index=['apple', 'banana', 'mango', 'kiwi'])
print(pd.isnull(goods1))
print(goods1+goods2)
```

```
# pd.isnull(goods1)
apple    False
mango    False
orange   True
kiwi     False
dtype: bool
# goods1 + goods2
apple    7000.0
banana   NaN
kiwi     4000.0
mango    7000.0
orange   NaN
dtype: float64
```



### 3. DataFrame 문법

- DataFrame

✓ DataFrame은 여러 개의 칼럼(Column)으로 구성된 2차원 행렬 자료구조

일자	시가	고가	저가	종가	전일비	등락률	거래량
16.02.29	11,650	12,100	11,600	11,900	▲300	+2.59%	225,844
16.02.26	11,100	11,800	11,050	11,600	▲600	+5.45%	385,241
16.02.25	11,200	11,200	10,900	11,000	▼100	-0.90%	161,214
16.02.24	11,100	11,100	10,950	11,100	▲50	+0.45%	77,201
16.02.23	11,000	11,150	10,900	11,050	▲100	+0.91%	113,131
16.02.22	10,950	11,050	10,850	10,950	▼100	-0.90%	138,387
16.02.19	10,950	11,100	10,800	11,050	-0	0.00%	76,105
16.02.18	11,050	11,200	10,950	11,050	▲250	+2.31%	83,611
16.02.17	11,150	11,300	10,800	10,800	▼350	-3.14%	189,480
16.02.16	10,950	11,200	10,850	11,150	▲300	+2.76%	133,359

- DataFrame 특징

- ✓ DataFrame에서 사용 가능한 데이터
  - 리스트, 튜플, 딕셔너리, Series, 2차원 ndarray
- ✓ 일반적으로 딕셔너리를 이용해서 생성
  - ✓ `{'eno': [1,2,3], 'name':['홍길동','이순신','강감찬']}`
- ✓ 딕셔너리의 키는 오름차순 정렬되어서 배치됨
- ✓ 정렬순서를 변경하고자 하면 columns 매개변수에 순서를 리스트로 대입
- ✓ DataFrame의 각 컬럼의 데이터는 사전처럼['컬럼이름'] 또는 .컬럼이름으로 접근하면 Series 객체로 반환
- ✓ 특정 행에 접근하기 위해서는 iloc[인덱스]를 이용하며, 해당 자료를 Series 반환

```
from pandas import Series, DataFrame
```

```
items = {'code': [1,2,3,4,5,6],  
         'name': ['apple','watermelon','oriental  
melon', 'banana', 'lemon', 'mango'],  
         'manufacture': ['korea', 'korea',  
'korea','philippines','korea', 'taiwan'],  
         'price':[1500, 15000,1000,500,1500,700]}
```

```
data = DataFrame(items)  
print(data)
```

	code	manufacture	name	price
0	1	korea	apple	1500
1	2	korea	watermelon	15000
2	3	korea	oriental melon	1000
3	4	philippines	banana	500
4	5	korea	lemon	1500
5	6	taiwan	mango	700

```
from pandas import Series, DataFrame
```

```
items = {'code': [1,2,3,4,5,6],  
         'name': ['apple','watermelon','oriental melon',  
                 'banana', 'lemon', 'mango'],  
         'manufacture': ['korea', 'korea',  
                         'korea','philippines','korea', 'taiwan'],  
         'price':[1500, 15000,1000,500,1500,700]}
```

```
data = DataFrame(items, columns=['code', 'name',  
                                'manufacture', 'price'])  
print(data['name']) #name 컬럼의 모든 값을 가져오기  
print('=====')  
print(data.iloc[0]) #0번째 데이터를 가져오기  
print('=====')  
print(data['name'][0])# name값의 0번째 가져오기
```

```
# name 컬럼의 모든 값
```

```
0      apple
```

```
1    watermelon
```

```
2  oriental melon
```

```
3      banana
```

```
4      lemon
```

```
5      mango
```

```
Name: name, dtype: object
```

```
=====
```

```
# 0번째 데이터 가져오기
```

```
code      1
```

```
name      apple
```

```
manufacture  korea
```

```
price      1500
```

```
Name: 0, dtype: object
```

```
=====
```

```
# name값의 0번째 가져오기
```

```
apple
```

- DataFrame의 index와 values 속성

- ✓ 기본적으로 인덱스는 0부터 시작하는 숫자이지만 index 속성을 이용해서 데이터프레임을 생성할 때 index 지정이 가능
- ✓ index 속성은 index의 값들을 반환하고 values 속성은 데이터의 모임을 2차원 행렬 형태로 반환
- ✓ 특정 컬럼을 삭제할 때는 del 객체['컬럼이름']
- ✓ 컬럼의 데이터 전체를 변경할 때는 Series 객체 또는 리스트 및 튜플을 이용, 길이는 DataFrame의 행 길이와 동일
- ✓ Series 객체를 대입할 때는 index에 해당하는 데이터가 수정되는데 없는 index에는 NaN 값이 대입
- ✓ 데이터프레임 객체에 없는 컬럼의 이름을 이용해서 대입하면 컬럼 추가
- ✓ T 속성을 이용해서 index와 column을 변경한 전치행렬 객체를 반환

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
items = {'code': [1,2,3,4,5,6],
        'name': ['apple','watermelon','oriental melon', 'banana', 'lemon', 'mango'],
        'manufacture': ['korea', 'korea', 'korea','philippines','korea', 'taiwan'],
        'price':[1500, 15000,1000,500,1500,700]}
```

```
data = DataFrame(items, columns=['code', 'name', 'manufacture', 'price'])
```

# 색인 추출 & 수정

```
Print(data.index)
```

```
Data.index = np.arrange(1,7,1)
```

```
Print(data.index)
```

# 칼럼 값 수정

```
data.price = [1500, 10000, 500, 1200, 300, 5000]
```

```
print(data)
```

```
data.price = Series([3000, 20000, 300, 1000 ], index=[1,2,3,4])
```

```
print(data)
```

# 색인 추출 & 수정

```
RangeIndex(start=0, stop=6, step=1)
```

```
Int64Index([1, 2, 3, 4, 5, 6], dtype='int64')
```

# 칼럼 값 수정

	code	name	manufacture	price
1	1	apple	korea	1500
2	2	watermelon	korea	10000
3	3	oriental melon	korea	500
4	4	banana	philippines	1200
5	5	lemon	korea	300
6	6	mango	taiwan	5000

# index 없는 경우 NaN

	code	name	manufacture	price
1	1	apple	korea	3000.0
2	2	watermelon	korea	20000.0
3	3	oriental melon	korea	300.0
4	4	banana	philippines	1000.0
5	5	lemon	korea	NaN
6	6	mango	taiwan	NaN

- 중첩 dict도 DataFrame 생성

- ✓ 외부에 있는 키가 컬럼의 이름이 되고 내부에 있는 키가 인덱스
- ✓ 인덱스는 index 속성을 이용해서 변경이 가능

```
from pandas import Series, DataFrame
```

```
items = {'1':{'name':'apple', 'manufacture':'korea', 'price':1500},  
        '2': {'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},  
        '3': {'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},  
        '4': {'name': 'banana', 'manufacture': 'philippines', 'price': 500},  
        '5': {'name': 'lemon', 'manufacture': 'korea', 'price': 1500},  
        '6': {'name': 'mango', 'manufacture': 'korea', 'price': 700}}
```

```
data = DataFrame(items)
```

```
print(data)
```

```
# 전치행렬
```

```
data = data.T
```

```
print(data)
```

	1	2	3	4	5	6
name	apple	watermelon	oriental melon	banana	lemon	mango
manufacture	korea	korea	korea	philippines	korea	korea
price	1500	15000	1000	500	1500	700

```
# 전치행렬
```

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
6	mango	korea	700

- Series, DataFrame의 인덱스 재구성

- ✓ **reindex** 속성을 이용해서 index를 재배치하거나 추가 또는 삭제하는 가능
- ✓ 인덱스 자체의 값을 변경하는 것은 안됨
- ✓ 인덱스를 변경할 때 fill\_value 파라미터에 값을 대입하면 누락된 인덱스에 값을 대입
- ✓ method 파라미터에 ffill 또는 bfill을 대입하면 이전 데이터나 다음 데이터를 대입
- ✓ DataFrame의 경우는 index와 columns 매개변수를 이용해서 index와 column을 재배치 할 수 있다.



```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np

items = {'1':{'name':'apple', 'manufacture':'korea', 'price':1500},
        '2':{'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},
        '3':{'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},
        '4':{'name': 'banana', 'manufacture': 'philippines', 'price': 500},
        '5':{'name': 'lemon', 'manufacture': 'korea', 'price': 1500},
        '6':{'name': 'mango', 'manufacture': 'korea', 'price': 700}}

data = DataFrame(items)
print(data)
dataT = data.T # 전치행렬

# 전치행렬
print(dataT)

data = dataT.reindex(['1','2','3','4','5','7']) # 재 인덱싱 - 6번은 없어지고 7번은 추가되는데 데이터는 NaN
print(data);
print("=====")
data = dataT.reindex(['1','2','3','4','5','7'], fill_value=0) # 재 인덱싱 - 6번은 없어지고 7번은 추가 모든 값 0
print(data);
print("=====")
data = dataT.reindex(['1','2','3','4','5','7'], method='ffill', limit=2) # 재 인덱싱 - 6번은 없어지고 7번은 추가
되는데 값은 이전데이터와 동일
print(data);

```

## # 전치행렬

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
6	mango	korea	700

## # 재 인덱싱

# 6번 제외, 7번 추가

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
7	NaN	NaN	NaN

# 6번 제외, 7번 추가 모든 값 0

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
7	0	0	0

# 6번 제외, 7번 추가 이전 데이터와 동일

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
7	mango	korea	700

- Series, DataFrame의 데이터 삭제

- ✓ 인덱스 이름을 이용해서 행을 삭제할 때 drop 메서드에 인덱스 또는 인덱스의 리스트 넘김
- ✓ 열을 삭제할 때는 열이름 또는 열이름의 리스트를 넘겨주고, axis 파라미터에 1 대입

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
items = {'1': {'name': 'apple', 'manufacture': 'korea', 'price': 1500},
        '2': {'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},
        '3': {'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},
        '4': {'name': 'banana', 'manufacture': 'philippines', 'price': 500},
        '5': {'name': 'lemon', 'manufacture': 'korea', 'price': 1500},
        '6': {'name': 'mango', 'manufacture': 'korea', 'price': 700}}
```

```
data = DataFrame(items)
data = data.T
```

```
data = dataT.drop("1") #인덱스가 1인 행 삭제
print(data)
print("=====")
data = dataT.drop(["3", "5"]) #인덱스가 3, 5 인 데이터 삭제
print(data)
print("=====")
data = dataT.drop("price", axis=1) #price 컬럼 삭제
print(data)
```

	manufacture	name	price
2	korea	watermelon	15000
3	korea	oriental melon	1000
4	philippines	banana	500
5	korea	lemon	1500
6	korea	mango	700

	manufacture	name	price
2	korea	watermelon	15000
4	philippines	banana	500
6	korea	mango	700

	manufacture	name
2	korea	watermelon
4	philippines	banana
6	korea	mango

- Series, DataFrame의 색인 및 선택 또는 필터링
  - ✓ 색인에 컬럼 이름을 대입해서 조회가능
  - ✓ 컬럼 이름의 범위를 대입해서 데이터를 컬럼 단위로 추출 가능
  - ✓ 컬럼 이름을 이용한 조건을 대입해서 컬럼 단위 추출 가능
  - ✓ **iloc[행번호 또는 인덱스]**를 대입하면 행번호 또는 인덱스에 해당하는 데이터 반환
  - ✓ **iloc[행번호 또는 인덱스, [컬럼이름]]**를 대입하면 행번호 또는 인덱스에 해당하는 데이터의 컬럼 값만 반환
  - ✓ 행번호 또는 인덱스 자리에 컬럼을 이용한 **조건 입력** 가능

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
items = {'1':{'name':'apple', 'manufacture':'korea', 'price':1500},
        '2': {'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},
        '3': {'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},
        '4': {'name': 'banana', 'manufacture': 'philippines', 'price': 500},
        '5': {'name': 'lemon', 'manufacture': 'korea', 'price': 1500},
        '6': {'name': 'mango', 'manufacture': 'korea', 'price': 700}}
```

```
data = DataFrame(items)
```

```
print(data)
```

```
print('=====')
```

```
print(data['1']) #key가 1번인 데이터
```

```
print('=====')
```

```
print(data['1':'3']) # 오류 발생
```

```
print('=====')
```

```
print(data[['1','3']]) #key가 1,3번인 데이터
```

	1	2	3	4	5	6
name	apple	watermelon	oriental melon	banana	lemon	mango
manufacture	korea	korea	korea	philippines	korea	korea
price	1500	15000	1000	500	1500	700

```
=====
```

```
name      apple
manufacture korea
price      1500
Name: 1, dtype: object
```

```
=====
```

```
KeyError: '1'
```

```
=====
```

	1	3
name	apple	oriental melon
manufacture	korea	korea
price	1500	1000

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np

items = {'1':{'name':'apple', 'manufacture':'korea', 'price':1500},
        '2': {'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},
        '3': {'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},
        '4': {'name': 'banana', 'manufacture': 'philippines', 'price': 500},
        '5': {'name': 'lemon', 'manufacture': 'korea', 'price': 1500},
        '6': {'name': 'mango', 'manufacture': 'korea', 'price': 700}}

data = DataFrame(items)
dataT = data.T
print(dataT)
print('=====')
print(dataT[0:3]) #1-3행까지 추출
print('=====')
print(dataT['price'] > 1000) #price 컬럼의 값이 1000 초과 여부 확인
print('=====')
print(dataT[dataT['price'] > 1000]) #price 컬럼의 값이 1000 초과된 행 만 출력

```

## # 원본행렬

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000
4	banana	philippines	500
5	lemon	korea	1500
6	mango	korea	700

## # 행과 열 색인

### # 1~3행 선택

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
3	oriental melon	korea	1000

### # price가 1000 초과 여부

1	True
2	True
3	False
4	False
5	True
6	False

Name: price, dtype: bool

### # price가 1000 초과된 행 추출

	name	manufacture	price
1	apple	korea	1500
2	watermelon	korea	15000
5	lemon	korea	1500



```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
items = {'1':{'name':'apple', 'manufacture':'korea', 'price':1500},
        '2': {'name': 'watermelon', 'manufacture': 'korea', 'price': 15000},
        '3': {'name': 'oriental melon', 'manufacture': 'korea', 'price': 1000},
        '4': {'name': 'banana', 'manufacture': 'philippines', 'price': 500},
        '5': {'name': 'lemon', 'manufacture': 'korea', 'price': 1500},
        '6': {'name': 'mango', 'manufacture': 'korea', 'price': 700}}
data = DataFrame(items)
data = data.T
print(data.ix[0])#0번행의 데이터를 출력
print('=====')
print(data.iloc[0, ['name']])#0번행의 name 데이터를 출력
print('=====')
print(data.iloc[0, ['name', 'price']])#0번행의 name과 price 데이터를 출력
print('=====')
print(data.iloc[:3, ['name', 'price']])#3번행 까지의 name과 price 데이터를 출력
print('=====')
print(data.iloc[data.price>1000, ['name', 'price']])#price가 1000이 넘는 데이터의 name과
price 출력 name과 price 데이터를 출력

```

```
manufacture korea
name       apple
price      1500
Name: 1, dtype: object
```

```
=====
name  apple
Name: 1, dtype: object
```

```
=====
name  apple
price 1500
Name: 1, dtype: object
```

```
=====
      name price
1    apple 1500
2  watermelon 15000
3 oriental melon 1000
```

```
=====
      name price
1    apple 1500
2  watermelon 15000
5    lemon 1500
```

## ● DataFrame의 연산

- ✓ 산술 연산은 동일한 인덱스 값을 찾아서 연산 수행
- ✓ 어느 한쪽에만 존재하는 인덱스의 연산은 NaN
- ✓ 산술연산을 add, sub, div, mul 메서드를 이용해서 수행할 수 있는데 이 때 fill\_value를 이용해서 한쪽에만 존재하는 인덱스에 기본 값을 삽입할 수 있음
- ✓ Series와의 산술 연산 가능
  - Series의 인덱스를 DataFrame의 컬럼 이름과 매핑해서 연산을 수행하는데 동일한 값이 없으면 NaN
  - DataFrame의 모든 행에 대해서 브로드캐스팅 연산
  - 행단위로 연산을 하고자 하는 경우는 add, sub, div, mul 메서드를 호출해서 첫번째 매개변수로 Series 객체 대입하고 axis에 0 대입

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
items1 = {'1':{'price':1500},
          '2':{'price': 15000},
          '3':{'price': 1000}}
items2 = {'1':{'price':1500},
          '2':{'price': 15000},
          '4':{'price': 1000}}
```

```
data1 = DataFrame(items1)
data1 = data1.T
```

```
data2 = DataFrame(items2)
data2 = data2.T
```

```
print(data1 + data2)
print("=====")
print(data1.add(data2, fill_value=0))
```

```
price
1  3000.0
2 30000.0
3    NaN
4    NaN
=====
price
1  3000.0
2 30000.0
3  1000.0
4  1000.0
```

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np

items1 = {'1':{'price':1500},
          '2':{'price': 15000},
          '3':{'price': 1000}}
items2 = Series([300, 200, 100], index=["1", "2", "4"])

data1 = DataFrame(items1)
print(data1+items2)
print('=====')
data1 = data1.T
print(data1.add(items2, axis=0))

```

```

1      2 3 4
price 1800.0 15200.0 NaN NaN
=====
      price
1  1800.0
2 15200.0
3     NaN
4     NaN

```

## ● DataFrame의 정렬과 순위

### ✓ 정렬

- `sort_index()`를 호출하면 인덱스가 정렬하며 `axis`에 1을 대입하면 컬럼의 이름이 정렬
- 기본은 오름차순 정렬이며 내림차순 정렬을 하고자 하는 경우에는 `ascending`의 값을 `False`로 대입
- `Series` 객체의 정렬은 `sort_values()`
- `DataFrame`에서 특정 컬럼을 기준으로 정렬을 하고자 하면 `sort_values` 메서드에 `by` 매개변수로 컬럼의 이름이나 컬럼의 이름 리스트를 대입
- 데이터가 `NaN` 인 경우는 정렬을 하는 경우 가장 마지막에 위치

### ✓ 순위

- 데이터의 순위는 `rank()`를 이용하는데 기본적으로는 오름차순으로 순위를 설정
- `ascending`의 값을 `False`로 대입하면 내림차순 순위
- `axis` 매개변수를 이용하면 축을 설정할 수 있으며 기본적으로는 컬럼 단위이며 `axis`에 1을 대입하면 행 단위 순위
- 동점은 순위의 평균을 출력하는데 `method` 매개변수에 `max`를 대입하면 큰 순위를 출력하고 `min`을 대입하면 작은 순위 `first`를 대입하면 먼저 등장한 데이터가 작은 순위를 갖는다

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

```
items = {'apple':{'count':10,'price':1500},
        'banana':{'count':5, 'price': 15000},
        'melon': { 'count':7,'price': 1000},
        'kiwi': {'count':20,'price': 500},
        'mango': {'count':30,'price': 1500},
        'orange': { 'count':4,'price': 700}}
```

```
data = DataFrame(items)
```

```
data = data.T
```

```
print(data.sort_values(ascending=[False,True], by=["price", "count"]))
```

	count	price
banana	5	15000
apple	10	1500
mango	30	1500
melon	7	1000
orange	4	700
kiwi	20	500

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
items = {'apple':{'count':10,'price':1500},
        'banana':{'count':5, 'price': 15000},
        'melon': { 'count':7,'price': 1000},
        'kiwi': {'count':20,'price': 500},
        'mango': {'count':30,'price': 1500},
        'orange': { 'count':4,'price': 700}}
data = DataFrame(items)
data = data.T
print(data.rank())
print("=====")
print(data.rank(ascending=False, method='min'))
print("=====")

```

	count	price
apple	4.0	4.5
banana	2.0	6.0
kiwi	5.0	1.0
mango	6.0	4.5
melon	3.0	3.0
orange	1.0	2.0

=====

	count	price
apple	3.0	2.0
banana	5.0	1.0
kiwi	2.0	6.0
mango	1.0	2.0
melon	4.0	4.0
orange	6.0	5.0

=====



## ● 통계 메서드

- ✓ axis는 계산 방향으로 0은 행 축이고 1은 열 축
- ✓ skipna는 NaN값이 있는 경우 제외여부로 True로 설정하면 제외하고 False이면 포함
- ✓ count, min, max, sum, mean, median, var, std
- ✓ argmin(최소값 위치), argmax, idxmin(최소값 색인), idxmax, quantile(분위수)
- ✓ describe(요약)
- ✓ cumsum(누적합), cummin, cummax, cumprod
- ✓ diff(산술 차)
- ✓ pct\_change
- ✓ unique(): 동일한 값을 제외한 배열 리턴 - Series에만 사용
- ✓ value\_counts(): 빈도수를 반환하는데 기본적으로 내림차순 정렬을 수행하며 sort 속성에 False를 대입하면 정렬하지 않는다.
  - Series에만 사용

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
items = {'apple':{'count':10,'price':1500},
        'banana':{'count':5, 'price': 15000},
        'melon': { 'count':7,'price': 1000},
        'kiwi': {'count':20,'price': 500},
        'mango': {'count':30,'price': 1500},
        'orange': { 'count':4,'price': 700}}
data = DataFrame(items)
data = data.T
print(data.describe())
print("=====")

```

	count	price
count	6.000000	6.000000
mean	12.666667	3366.666667
std	10.269697	5713.726163
min	4.000000	500.000000
25%	5.500000	775.000000
50%	8.500000	1250.000000
75%	17.500000	1500.000000
max	30.000000	15000.000000

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
stocks = {'2017-02-19':{'다음':50300,"네이버": 51100},
"2017-02-22":{"다음":50300, '네이버': 50800},
'2016-02-23':{'다음':50800,'네이버': 53000}}
data = DataFrame(stocks)
data = data.T
print(data.diff())
print("=====")
print(data.pct_change())
print("=====")

```

```

네이버  다음
2016-02-23  NaN  NaN
2017-02-19 -1900.0 -500.0
2017-02-22 -300.0  0.0
=====
네이버  다음
2016-02-23  NaN  NaN
2017-02-19 -0.035849 -0.009843
2017-02-22 -0.005871 0.000000
=====

```

- DataFrame의 결측치(NaN) 처리

- ✓ isnull(): NaN 이나 None은 True, 그렇지 않은 경우는 False 반환
- ✓ notnull(): isnull()의 반대
- ✓ dropna(): NaN 인 값을 소유한 행 제외하는데 how 파라미터에 all을 대입하면 컬럼의 모든 값이 NaN인 경우만 제외하며 thresh 파라미터에 정수를 대입하면 그 정수 값 이상의 값을 소유한 컬럼만 반환
- ✓ fillna(): NaN을 소유한 데이터의 값을 설정할 때 사용하는 메서드로 특정한 값으로 변경할 수 있고 method 파라미터를 이용해서 이전 값이나 이후 값으로 채울 수 있으며 limit를 이용해서 채울 개수를 지정할 수 있음

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
stocks = {'2017-02-19':{'다음':50300,"네이버": 51100, "넥슨":None, "NC":None},
"2017-02-22":{"다음':50300, '네이버': 50800, "넥슨":35000, "NC":None},
'2017-02-23':{'다음':50800,'네이버': 53000, "넥슨":37000, "NC":8000}}
data = DataFrame(stocks)
data = data.T
print(data.dropna())
print("=====")
print(data.dropna(how='all'))
print("=====")
print(data.diff().fillna(0))
print("=====")

```

```

NC   네이버   넥슨   다음
2017-02-23  8000.0  53000.0  37000.0  50800.0
=====

```

```

NC   네이버   넥슨   다음
2017-02-19   NaN  51100.0   NaN  50300.0
2017-02-22   NaN  50800.0  35000.0  50300.0
2017-02-23  8000.0  53000.0  37000.0  50800.0
=====

```

```

NC   네이버   넥슨   다음
2017-02-19  0.0   0.0   0.0   0.0
2017-02-22  0.0 -300.0   0.0   0.0
2017-02-23  0.0 2200.0 2000.0  500.0
=====

```

- DataFrame 상관관계

- ✓ `corr()`을 이용하면 상관관계를 알아볼 수 있음
- ✓ `cov()`는 공분산
- ✓ Series 사이의 상관관계나 공분산을 알고자 할 때는 Series 객체를 매개변수로 넘겨준다.
- ✓ DataFrame이 호출하면 모든 상관관계나 공분산 리턴
- ✓ DataFrame에서 `corrwith` 메서드를 이용하면 Series 나 DataFrame 과의 상관관계를 리턴

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
stocks = {'2017-02-19':{'다음':50300,"네이버": 51100, "넥슨":32000, "NC":4000},
"2017-02-22":{"다음":50300, '네이버': 50800, "넥슨":35000, "NC":6500},
'2017-02-23':{'다음':50800,'네이버': 50700, "넥슨":37000, "NC":8000}}
data = DataFrame(stocks)
data = data.T
d = data.pct_change().fillna(0)
print(d)
print('=====')
print('넥슨과 네이버', d.넥슨.corr(d.네이버))
print('=====')
print('넥슨과 NC',d.넥슨.corr(d.NC))
print('=====')
print(d.corr())
print('=====')
print(d.corrwith(d.NC))

```

```

NC    네이버    넥슨    다음
2017-02-19  0.000000  0.000000  0.000000  0.000000
2017-02-22  0.625000 -0.005871  0.093750  0.000000
2017-02-23  0.230769 -0.001969  0.057143  0.00994
=====
넥슨과 네이버 -0.951188402706
=====
넥슨과 NC 0.962244003854
=====
          NC    네이버    넥슨    다음
NC  1.000000 -0.999276  0.962244 -0.149307
네이버 -0.999276  1.000000 -0.951188  0.186829
넥슨  0.962244 -0.951188  1.000000  0.125468
다음 -0.149307  0.186829  0.125468  1.000000
=====
NC  1.000000
네이버 -0.999276
넥슨  0.962244
다음 -0.149307
dtype: float64

```

- DataFrame 계층적 색인
  - ✓ index나 컬럼이 2 level 이상으로 이루어진 경우
  - ✓ 그룹화 연산을 할 때 유용
  - ✓ 컬럼의 값들을 가져오는 방법은 이전과 동일
  - ✓ 집계함수를 이용할 때 level에 인덱스나 컬럼의 이름을 대입하고 axis에 축 방향을 대입하면 그 레벨에 맞는 집계를 구하는 것이 가능



```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np
li = [50300, 51100, 32000, 4000, 50300, 50800, 35000, 6500, 50800, 50700, 50500, 37500, 8200]
ar = np.array(li)
ar = ar.reshape(4,4)
stocks = DataFrame(ar, index=['다음', '네이버', '넥슨', 'NC'],
                    columns=['3월', '3월', '4월', '4월'], ['11일', '12일', '11일', '12일'])
print(stocks)
print("=====")
print(stocks['3월']) #3월의 데이터 가져오기
print("=====")
print(stocks['3월']['12일']) #3월 11일의 데이터 가져오기
print("=====")
print(stocks.ix['다음']) #다음의 데이터 가져오기

```

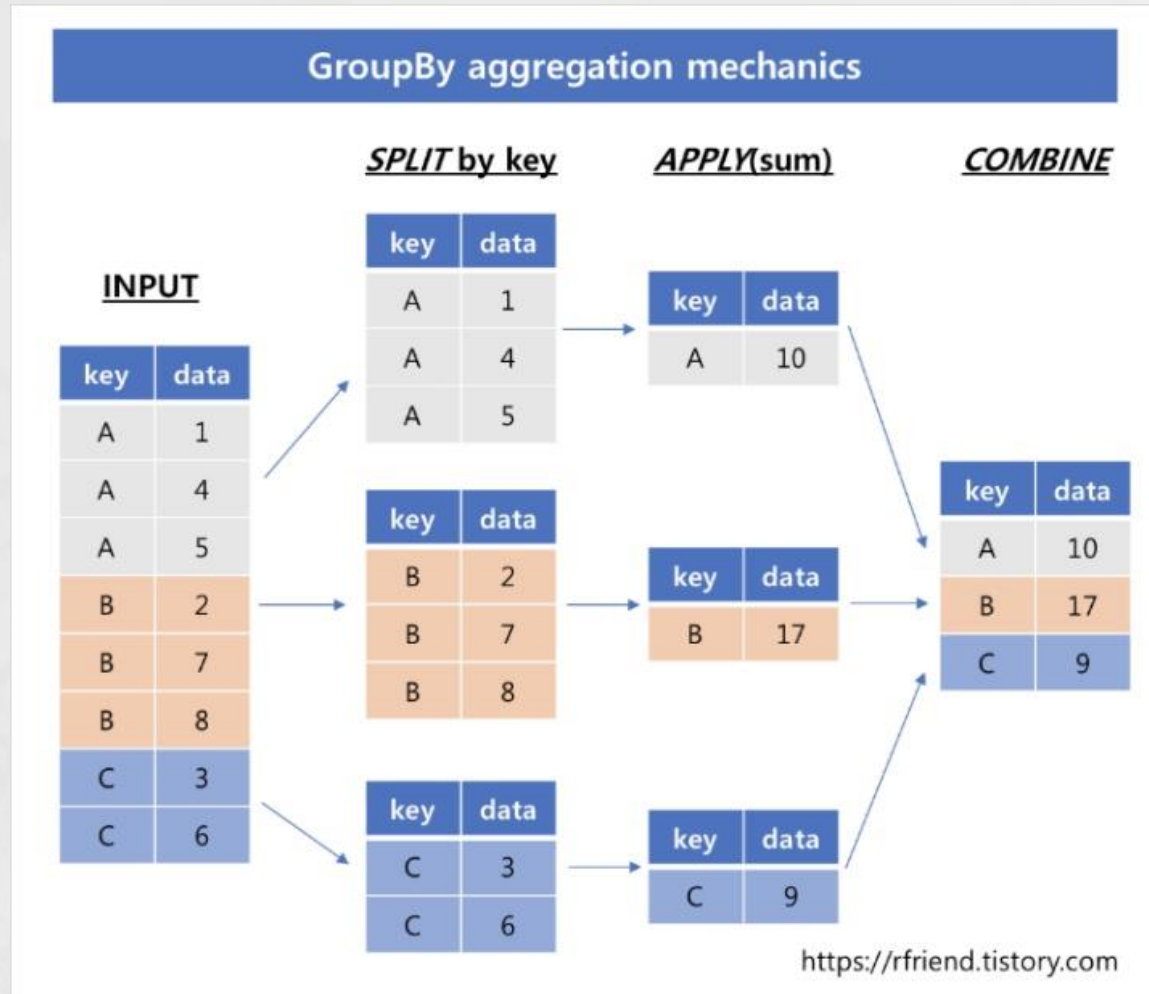
```

3월      4월
11일    12일  11일  12일
다음  50300  51100  32000  4000
네이버 50300  50800  35000  6500
넥슨   50800  50700  37000  8000
NC     51800  50500  37500  8200
=====
11일    12일
다음  50300  51100
네이버 50300  50800
넥슨   50800  50700
NC     51800  50500
=====
다음    51100
네이버   50800
넥슨    50700
NC       50500
Name: 12일, dtype: int32
=====
3월 11일   50300
    12일   51100
4월 11일   32000
    12일   4000
Name: 다음, dtype: int32

```

## 4. Group by

- ✓ 범주형 변수를 대상으로 각 집단별 그룹 객체를 생성하여 자료처리에 이용





## 파이썬 GroupBy를 사용해 그룹별 가중 평균 구하기 (Group Weighted Average by GroupBy Operation)

### Original Dataset

Group	Value	Weight
A	1	0.0
A	2	0.1
A	3	0.2
A	4	0.3
A	5	0.4
B	6	0.0
B	7	0.1
B	8	0.2
B	9	0.3
B	10	0.4

### 1 Split

Group	Value	Weight
A	1	0.0
A	2	0.1
A	3	0.2
A	4	0.3
A	5	0.4

Group	Value	Weight
B	6	0.0
B	7	0.1
B	8	0.2
B	9	0.3
B	10	0.4

### 2 Apply

Weighted Average of Group A

$$= \frac{1 \cdot 0.0 + 2 \cdot 0.1 + 3 \cdot 0.2 + 4 \cdot 0.3 + 5 \cdot 0.4}{0.0 + 0.1 + 0.2 + 0.3 + 0.4}$$
$$= \frac{4}{1} = 4$$

Weighted Average of Group B

$$= \frac{6 \cdot 0.0 + 7 \cdot 0.1 + 8 \cdot 0.2 + 9 \cdot 0.3 + 10 \cdot 0.4}{0.0 + 0.1 + 0.2 + 0.3 + 0.4}$$
$$= \frac{9}{1} = 9$$

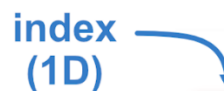
### 3 Combine

Group	Weighted Average
A	4
B	9

R, Python 분석과 프로그래밍의 친구 <http://rfriend.tistory.com>

```
# group weighted average by category
grouped = df.groupby('grp_col')
weighted_avg_func = lambda g: np.average(g['val'], weights=g['weight'])
grouped.apply(weighted_avg_func)
```

## Group by example



The diagram illustrates a 1D index array used for grouping data. A blue arrow points from the index value '0' to the first row of the dataset. Another blue arrow points from the index value '50' to the sixth row of the dataset. The index values are: 0, 1, 2, 3, 4, 50, 51, 52, 53, 54, 100, 101, 102, 103, 104.

index (1D)	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	5.1	3.5	1.4	0.2
1	setosa	4.9	3.0	1.4	0.2
2	setosa	4.7	3.2	1.3	0.2
3	setosa	4.6	3.1	1.5	0.2
4	setosa	5.0	3.6	1.4	0.2
50	versicolor	7.0	3.2	4.7	1.4
51	versicolor	6.4	3.2	4.5	1.5
52	versicolor	6.9	3.1	4.9	1.5
53	versicolor	5.5	2.3	4.0	1.3
54	versicolor	6.5	2.8	4.6	1.5
100	virginica	6.3	3.3	6.0	2.5
101	virginica	5.8	2.7	5.1	1.9
102	virginica	7.1	3.0	5.9	2.1
103	virginica	6.3	2.9	5.6	1.8
104	virginica	6.5	3.0	5.8	2.2

- 집단변수 1개를 이용하여 전체 칼럼 그룹화  
`df.groupby('species').sum()`

	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	5.1	3.5	1.4	0.2
1	setosa	4.9	3.0	1.4	0.2
2	setosa	4.7	3.2	1.3	0.2
3	setosa	4.6	3.1	1.5	0.2
4	setosa	5.0	3.6	1.4	0.2
50	versicolor	7.0	3.2	4.7	1.4
51	versicolor	6.4	3.2	4.5	1.5
52	versicolor	6.9	3.1	4.9	1.5
53	versicolor	5.5	2.3	4.0	1.3
54	versicolor	6.5	2.8	4.6	1.5
100	virginica	6.3	3.3	6.0	2.5
101	virginica	5.8	2.7	5.1	1.9
102	virginica	7.1	3.0	5.9	2.1
103	virginica	6.3	2.9	5.6	1.8
104	virginica	6.5	3.0	5.8	2.2

	species	sepal_length	sepal_width	petal_length	petal_width
	setosa	24.3	16.4	7.0	1.0
	versicolor	32.3	14.6	22.7	7.2
	virginica	32.0	14.9	28.4	10.5

- 집단변수 1개를 이용하여 1개 칼럼 그룹화

```
df['sepal.width'].groupby(df['species']).sum()
```

sepal_width	
species	
setosa	16.4
versicolor	14.6
virginica	14.9

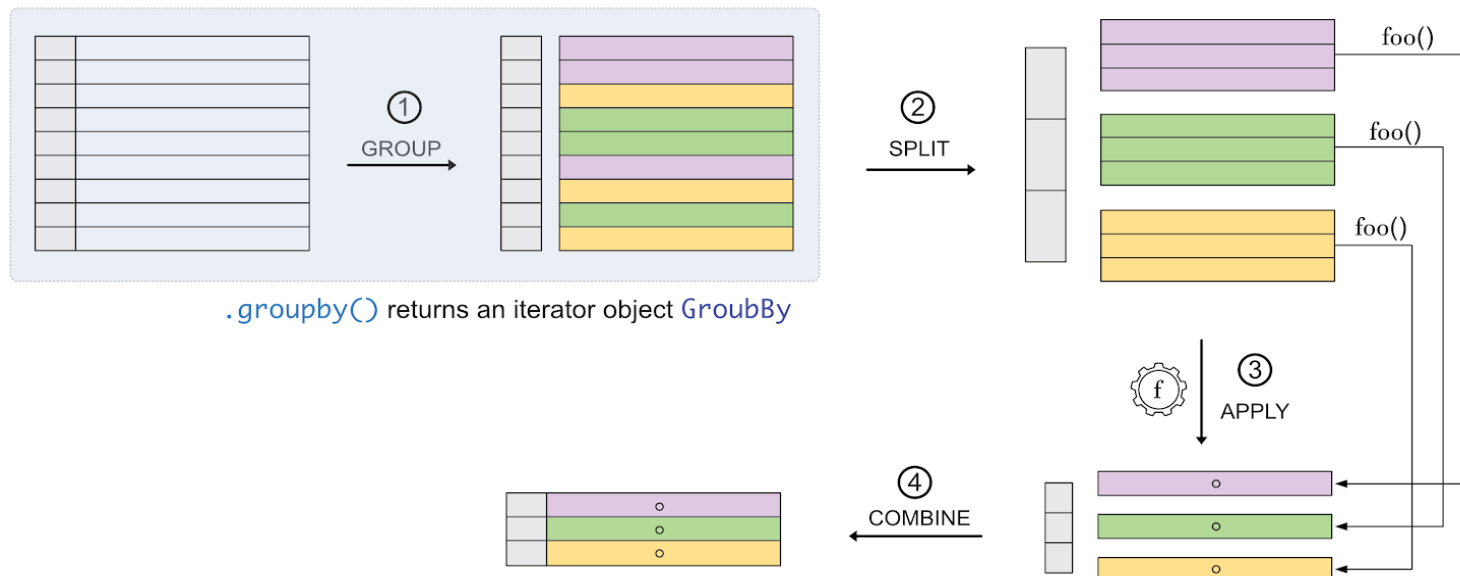
- 2개 변수를 이용하여 나머지 칼럼 그룹화

```
multicol_sum = df.groupby(['species', 'petal_width']).sum()
```

		sepal_length	sepal_width	petal_length
species	petal_width			
setosa	0.2	24.3	16.4	7.0
	1.3	5.5	2.3	4.0
	1.4	7.0	3.2	4.7
versicolor	1.5	19.8	9.1	14.0
	1.8	6.3	2.9	5.6
	1.9	5.8	2.7	5.1
virginica	2.1	7.1	3.0	5.9
	2.2	6.5	3.0	5.8
	2.5	6.3	3.3	6.0

# 5. Apply

✓ 그룹 객체 또는 pandas 객체에 외부함수를 적용하는 역할





## ❖ Apply example

# 사용자 함수 정의

```
def my_sum(gr):  
    return gr.sum()
```

```
df.groupby('species').apply(my_sum)
```

		species	sepal_length	sepal_width	petal_length	petal_width
species						
setosa	setosasetosasetosasetosasetosa		24.3	16.4	7.0	1.0
versicolor	versicolorversicolorversicolorversicolorversic...		32.3	14.6	22.7	7.2
virginica	virginicavirginicavirginicavirginicavirginica		32.0	14.9	28.4	10.5

- DataFrame에 함수 적용

- ✓ apply 메서드를 이용하면 행이나 열 단위로 함수를 적용할 수 있음
- ✓ apply()의 첫번째 매개변수는 함수이며 axis의 값을 생략하면 컬럼 단위로 함수를 수행하고 1을 대입하면 행 단위로 함수를 수행
- ✓ 데이터의 각각에 함수를 적용하고자 하는 경우는 applymap을 이용
- ✓ Series에 적용하고자 하는 경우는 map을 이용

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np

def func(x):
    return x.sum()
#f = lambda x: x.sum()

items = {'apple':{'count':10,'price':1500},
        'banana':{'count':5, 'price': 15000},
        'melon': { 'count':7,'price': 1000},
        'kiwi': {'count':20,'price': 500},
        'mango': {'count':30,'price': 1500},
        'orange': { 'count':4,'price': 700}}
data = DataFrame(items)
data = data.T
print(data.apply(func))
print("=====")
print(data.apply(func, axis=1))

```

```

count    76
price  20200
dtype: int64
=====
apple    1510
banana   15005
kiwi     520
mango    1530
melon    1007
orange    704
dtype: int64

```

```

from pandas import Series, DataFrame
import pandas as pd
import numpy as np

def func(x):
    return x+10
#f = lambda x: x+10

items = {'apple':{'count':10,'price':1500},
        'banana': {'count':5, 'price': 15000},
        'melon': { 'count':7,'price': 1000},
        'kiwi': {'count':20,'price': 500},
        'mango': {'count':30,'price': 1500},
        'orange': { 'count':4,'price': 700}}
data = DataFrame(items)
data = data.T
print(data.applymap(func))
print("=====")
print(data["count"].map(func))

```

```

count price
apple    20  1510
banana   15 15010
kiwi     30   510
mango    40  1510
melon    17  1010
orange   14   710
=====
apple    20
banana   15
kiwi     30
mango    40
melon    17
orange   14
Name: count, dtype: int64

```

# 6. Pivot table

- ✓ 사용자가 직접 변수(칼럼)를 지정하여 테이블을 생성하는 기능

## Pivot

df

	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t



```
df.pivot(index='foo',  
          columns='bar',  
          values='baz')
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

# Pivot table example

```
In [1]: df
```

```
Out[1]:
```

	date	variable	value
0	2000-01-03	A	0.469112
1	2000-01-04	A	-0.282863
2	2000-01-05	A	-1.509059
3	2000-01-03	B	-1.135632
4	2000-01-04	B	1.212112
5	2000-01-05	B	-0.173215
6	2000-01-03	C	0.119209
7	2000-01-04	C	-1.044236
8	2000-01-05	C	-0.861849
9	2000-01-03	D	-2.104569
10	2000-01-04	D	-0.494929
11	2000-01-05	D	1.071804

```
pd.pivot_table(df, index='date', columns='variable', values='value')
```

variable	A	B	C	D
date				
2000-01-03	0.469112	-1.135632	0.119209	-2.104569
2000-01-04	-0.282863	1.212112	-1.044236	-0.494929
2000-01-05	-1.509059	-0.173215	-0.861849	1.071804

# Stack

df2

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

└──────────┘  
MultiIndex



stacked = df2.stack()

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

└──────────┘  
MultiIndex



# Unstack

stacked

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultIndex



stacked.unstack()

		A	B
first	second		
bar	one	1	2
	two	3	4
baz	one	5	6
	two	7	8

MultIndex