

## 02. pandas 시작하기

1. pandas 자료구조 소개
2. 핵심기능

# pandas 란?

---

- 표 형식의 데이터나 다양한 형태의 데이터를 다루는 데 초점을 맞춰 설계된 라이브러리
- NumPy 배열 기반 계산 스타일을 많이 차용
  - ▣ NumPy는 단일 산술 배열 데이터를 다루는 데 특화됨
  - ▣ Pandas는 표 형식 데이터를 다루는데 특화됨

# pandas 기본설정

---

## □ 기본설정

- ▣ 아래 설정은 본 자료의 마지막까지 적용됨

```
import pandas as pd
```

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

## ▣ 본 자료의 예제 작성 시 주의사항

- jupyter notebook에서 하나의 파일에 작성하기
- 각 코드 셀의 실행 결과가 다르게 나오는 경우
  - Cell>Run All을 실행하여 본 자료의 실행 순서와 일치시킬 것
  - 혹은 처음부터 해당 셀까지 선택하여 Ctrl+Enter

# 1. pandas 자료구조 소개

Series

DataFrame

# Series

---

- 일련의 객체를 담을 수 있는 1차원 배열 같은 자료구조
- 어떤 NumPy 자료형이라도 담을 수 있음
- 색인 - 배열 의 연관 구조

# Series

- 기본 색인으로 Series 객체 생성

```
obj = pd.Series([4, 7, -5, 3])  
obj
```

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

```
obj.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj.index # like range(4)
```

기본 색인 : 0 ~ N-1 까지의 정수  
(N은 데이터 길이)

```
RangeIndex(start=0, stop=4, step=1)
```

# Series

- 색인을 지정하여 Series 객체 생성

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])  
obj2
```

```
d    4  
b    7  
a   -5  
c    3  
dtype: int64
```

```
obj2.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
obj2.index
```

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

# Series

## □ 파이선의 사전과 유사

```
obj2['a']
```

-5

```
obj2['d'] = 6
```

```
obj2[['c', 'a', 'd']]
```

색인의 배열

```
c    3  
a   -5  
d    6  
dtype: int64
```

```
obj2[obj2 > 0]
```

NumPy 배열 연산

```
d    6  
b    7  
c    3  
dtype: int64
```



# Series

```
obj2 * 2
```

```
d    12  
b    14  
a   -10  
c     6  
dtype: int64
```

```
np.exp(obj2)
```

```
d    403.428793  
b   1096.633158  
a     0.006738  
c    20.085537  
dtype: float64
```

```
'b' in obj2
```

```
True
```

```
'e' in obj2
```

```
False
```

# Series

## □ 파이선 사전 객체로부터 Series 객체 생성

```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
```

obj3                      사전                      사전의 키값이 순서대로 들어감

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

```
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
```

obj4                      사전                      색인                      색인 순서를 직접 지정

```
California    NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

NaN : Not a Number  
누락된 값 표시

# Series

## □ 누락된 값 찾기

```
pd.isnull(obj4)
```

함수

```
California    True
Ohio         False
Oregon       False
Texas        False
dtype: bool
```

```
pd.notnull(obj4)
```

```
California    False
Ohio         True
Oregon       True
Texas        True
dtype: bool
```

```
obj4.isnull()
```

인스턴스 메소드

```
California    True
Ohio         False
Oregon       False
Texas        False
dtype: bool
```

# Series

## □ 산술연산에서 색인과 라벨로 자동 정렬

```
obj3
```

```
Ohio      35000  
Texas     71000  
Oregon    16000  
Utah       5000  
dtype: int64
```

```
obj4
```

```
California    NaN  
Ohio          35000.0  
Oregon        16000.0  
Texas         71000.0  
dtype: float64
```

```
obj3 + obj4
```

```
California    NaN  
Ohio          70000.0  
Oregon        32000.0  
Texas        142000.0  
Utah          NaN  
dtype: float64
```

# Series

- Series 객체와 Series의 색인의 name 속성 지정

```
obj4.name = 'population'  
obj4.index.name = 'state'  
obj4
```

```
state  
California      NaN  
Ohio            35000.0  
Oregon          16000.0  
Texas           71000.0  
Name: population, dtype: float64
```

# Series

- 할당문으로 색인 변경

```
obj
```

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

```
obj
```

```
obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
```

```
obj
```

```
Bob    4
Steve  7
Jeff   -5
Ryan   3
dtype: int64
```

# DataFrame

---

- 표 같은 스프레드시트 형식의 자료구조
- R의 데이터프레임에서 유래
- 행과 열에 대한 색인을 가짐
  - ▣ 색인은 같은 Series 객체를 담고 있는 파이선 사전으로 간주

# DataFrame

## □ 사전을 이용한 DataFrame 객체 생성하기

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],  
        'year': [2000, 2001, 2002, 2001, 2002, 2003],  
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}  
frame = pd.DataFrame(data)
```

동일 길이의 리스트

frame

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

색인 Series



# DataFrame

```
frame.head()
```

처음 5개 행 출력

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9

열 순서 지정하여 DataFrame 생성

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

# DataFrame

```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],  
                       index=['one', 'two', 'three', 'four', 'five', 'six'])  
frame2
```

	year	state	pop	debt
<b>one</b>	2000	Ohio	1.5	NaN
<b>two</b>	2001	Ohio	1.7	NaN
<b>three</b>	2002	Ohio	3.6	NaN
<b>four</b>	2001	Nevada	2.4	NaN
<b>five</b>	2002	Nevada	2.9	NaN
<b>six</b>	2003	Nevada	3.2	NaN

사전(data)에 없는 값  
은 결측치로 저장됨

```
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

# DataFrame

## □ 열 접근하기

```
frame2['state']
```

one	Ohio
two	Ohio
three	Ohio
four	Nevada
five	Nevada
six	Nevada

Name: state, dtype: object

```
frame2.year
```

one	2000
two	2001
three	2002
four	2001
five	2002
six	2003

Name: year, dtype: int64

## □ 행 접근하기

```
frame2.loc['three']
```

year	2002
state	Ohio
pop	3.6
debt	NaN

Name: three, dtype: object

# DataFrame

## □ 열 대입하기

- ▣ 스칼라나 배열로 대입 가능
- ▣ 리스트나 배열 대입 시 길이가 DataFrame 열 크기와 동일해야 함

```
frame2['debt'] = 16.5  
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
frame2['debt'] = np.arange(6.)  
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

# DataFrame

## □ 열 대입하기

- ▣ Series 대입 시 DataFrame의 색인에 따라 값이 대입됨
  - 존재하지 않는 색인에는 결측치 대입

```
val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])  
frame2['debt'] = val  
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

# DataFrame

- 열 생성 및 삭제
  - ▣ 존재하지 않는 열에 값 대입하면 새로운 열 생성

```
frame2['eastern'] = frame2.state == 'Ohio'  
frame2
```

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

```
del frame2['eastern']  
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

# DataFrame

---

## □ 주의

- DataFrame의 색인을 이용해서 얻은 열은 내부 데이터에 대한 뷰이며 복사가 이루어지지 않는다. 따라서, 이렇게 얻은 Series 객체에 대한 변경은 실제 DataFrame에 반영된다.
- 복사본이 필요할 때는 Series의 copy 메소드를 이용하라.

# DataFrame

## □ 중첩된 사전을 이용한 DataFrame 생성

pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}

열 색인      행 색인

```
frame3 = pd.DataFrame(pop)  
frame3
```

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

```
frame3.T
```

	2000	2001	2002
Nevada	NaN	2.4	2.9
Ohio	1.5	1.7	3.6

색인 지정하여 생성

```
pd.DataFrame(pop, index=[2001, 2002, 2003])
```

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN



# DataFrame

- Series 객체를 담고 있는 사전을 이용한 DataFrame 생성

```
pdata = {'Ohio': frame3['Ohio'][:-1],  
         'Nevada': frame3['Nevada'][:2]}  
pd.DataFrame(pdata)
```

	Ohio	Nevada
2000	1.5	NaN
2001	1.7	2.4

```
frame3.index.name = 'year'; frame3.columns.name = 'state'  
frame3
```

색인과 열 각각  
name 속성 지정

state	Nevada	Ohio
year		
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

# DataFrame

- values 속성
  - ▣ 저장된 데이터를 2차원 배열로 반환

```
frame3.values
```

```
array([[nan, 1.5],  
       [2.4, 1.7],  
       [2.9, 3.6]])
```

```
frame2.values
```

```
array([[2000, 'Ohio', 1.5, nan],  
       [2001, 'Ohio', 1.7, -1.2],  
       [2002, 'Ohio', 3.6, nan],  
       [2001, 'Nevada', 2.4, -1.5],  
       [2002, 'Nevada', 2.9, -1.7],  
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

←←←  
각 열이 다른 dtype을 가짐

# DataFrame

## □ DataFrame 생성을 위한 입력데이터의 종류

형	설명
2차원 ndarray	데이터를 담고 있는 행렬. 선택적으로 행(로우)과 열(컬럼)의 이름을 전달할 수 있다.
배열, 리스트, 튜플의 사전	사전의 모든 항목은 같은 길이를 가져야 하며, 각 항목의 내용이 DataFrame의 컬럼이 된다.
NumPy의 구조화 배열	배열의 사전과 같은 방식으로 취급된다.
Series의 사전	Series의 각 값이 컬럼이 된다. 명시적으로 색인을 넘겨주지 않으면 각 Series의 색인이 하나로 합쳐져서 로우의 색인이 된다.
사전의 사전	내부에 있는 사전이 컬럼이 된다. 키값은 'Series의 사전'과 마찬가지로 합쳐져서 로우의 색인이 된다.
사전이나 Series의 리스트	리스트의 각 항목이 DataFrame의 로우가 된다. 합쳐진 사전의 키값이나 Series의 색인이 DataFrame의 컬럼 이름이 된다.
리스트나 튜플의 리스트	'2차원 ndarray'의 경우와 같은 방식으로 취급된다.
다른 DataFrame	색인을 따로 지정하지 않으면 DataFrame의 색인이 그대로 사용된다.
NumPy MaskedArray	'2차원 ndarray'의 경우와 같은 방식으로 취급되지만 마스크값은 반환되는 DataFrame에서 NA 값이 된다.

# DataFrame

- 색인 객체(index objects)
  - ▣ 각 행과 열에 대한 이름과 다른 메타데이터(축 이름 등)를 저장
  - ▣ Series나 DataFrame 객체를 생성할 때 사용되는 배열이나 다른 순차적인 이름은 내부적으로 색인으로 변환됨

```
obj = pd.Series(range(3), index=['a', 'b', 'c'])  
index = obj.index  
index
```

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

```
index[1:]
```

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

```
index[1] = 'd' # TypeError
```

색인 객체는 변경 불가

# DataFrame

- ▣ 색인은 자료구조 사이에서 안전하게 공유 가능

```
labels = pd.Index(np.arange(3))  
labels
```

```
Int64Index([0, 1, 2], dtype='int64')
```

```
obj2 = pd.Series([1.5, -2.5, 0], index=labels)  
obj2
```

```
0    1.5  
1   -2.5  
2    0.0  
dtype: float64
```

```
obj2.index is labels
```

```
True
```

# DataFrame

- 배열과 유사하게 Index 객체도 고정 크기로 동작함

```
frame3
```

state	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

```
frame3.columns
```

```
Index(['Nevada', 'Ohio'], dtype='object', name='state')
```

```
'Ohio' in frame3.columns
```

```
True
```

```
2003 in frame3.index
```

```
False
```

# DataFrame

---

- ▣ Python의 집합과 달리, pandas의 인덱스는 중복된 값을 허용함
  - 중복되는 값으로 선택하면 해당 값을 가진 모든 항목이 선택됨

```
dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])  
dup_labels
```

```
Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

# DataFrame

## ▣ 색인 메서드와 속성

메서드	설명
append	추가적인 색인 객체를 덧붙여 새로운 색인을 반환한다.
difference	색인의 차집합을 반환한다.
intersection	색인의 교집합을 반환한다.
union	색인의 합집합을 반환한다.
isin	색인이 넘겨받은 색인에 존재하는지 알려주는 불리언 배열을 반환한다.
delete	i 위치의 색인이 삭제된 새로운 색인을 반환한다.
drop	넘겨받은 값이 삭제된 새로운 색인을 반환한다.
insert	i 위치에 색인이 추가된 새로운 색인을 반환한다.
is_monotonic	색인이 단조성을 가진다면 True를 반환한다.
is_unique	중복되는 색인이 없다면 True를 반환한다.
unique	색인에서 중복되는 요소를 제거하고 유일한 값만 반환한다.



## 2. 핵심기능

재색인  
하나의 행, 열 삭제하기  
색인하기, 선택하기, 거르기  
정수 색인  
산술 연산과 데이터 정렬

# 재색인(Reindexing)

## □ Series 재색인

```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])  
obj
```

```
d    4.5  
b    7.2  
a   -5.3  
c    3.6  
dtype: float64
```

```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])  
obj2
```

존재하지 않는 색인에 대해 NaN 추가

```
a   -5.3  
b    7.2  
c    3.6  
d    4.5  
e    NaN  
dtype: float64
```

# 재색인(Reindexing)

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])  
obj3
```

```
0    blue  
2  purple  
4  yellow  
dtype: object
```

```
obj3.reindex(range(6), method='ffill') 누락된 값은 직전의 값으로 채움
```

```
0    blue  
1    blue  
2  purple  
3    purple  
4  yellow  
5    yellow  
dtype: object
```

# 재색인(Reindexing)

## □ DataFrame - 행 재색인

```
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),  
                      index=['a', 'c', 'd'],  
                      columns=['Ohio', 'Texas', 'California'])  
frame
```

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```
frame2 = frame.reindex(['a', 'b', 'c', 'd'])  
frame2
```

	Ohio	Texas	California
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

# 재색인(Reindexing)

## □ DataFrame - 열 재색인

```
states = ['Texas', 'Utah', 'California']  
frame.reindex(columns=states)
```

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

# 재색인(Reindexing)

## □ 재색인 함수 인자

인자	설명
tolerance	전/후 보간 시에 사용할 최대 갭 크기(값의 차이)
level	MultilIndex의 단계(level)에 단순 색인을 맞춘다. 그렇지 않으면 MultilIndex의 하위집합에 맞춘다.
copy	True인 경우 새로운 색인이 이전 색인과 동일하더라도 데이터를 복사한다. False인 경우 새로운 색인이 이전 색인과 동일할 경우 복사하지 않는다.
index	색인으로 사용할 새로운 순서. Index 인스턴스나 다른 순차적인 자료구조가 사용 가능하다. Index는 복사가 이루어지지 않고 그대로 사용된다.
method	채움 메서드. ffill은 직전 값을 채워 넣고 bfill은 다음 값을 채워 넣는다.
fill_value	재색인 과정 중에 새롭게 나타나는 비어 있는 데이터를 채우기 위한 값
limit	전/후 보간 시에 사용할 최대 갭 크기(채워넣을 원소의 수)

## 재색인(Reindexing)

- loc을 이용해서 라벨로 색인 하기

```
rows = ['a', 'b', 'c', 'd']  
states = ['Texas', 'Utah', 'California']  
frame.loc[rows, states]
```

	Texas	Utah	California
a	1.0	NaN	2.0
b	NaN	NaN	NaN
c	4.0	NaN	5.0
d	7.0	NaN	8.0

# 축에서 항목 삭제하기

- drop 메서드를 사용한 삭제
  - ▣ 기존 객체는 그대로 유지, 삭제된 새로운 객체를 얻음
- Series

```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])  
obj
```

```
a    0.0  
b    1.0  
c    2.0  
d    3.0  
e    4.0  
dtype: float64
```

```
new_obj = obj.drop('c')  
new_obj
```

```
a    0.0  
b    1.0  
d    3.0  
e    4.0  
dtype: float64
```



## 축에서 항목 삭제하기

```
obj.drop(['d', 'c'])
```

```
a    0.0  
b    1.0  
e    4.0  
dtype: float64
```

```
obj
```

```
a    0.0  
b    1.0  
c    2.0  
d    3.0  
e    4.0  
dtype: float64
```

# 축에서 항목 삭제하기

## ▣ 원본 객체의 항목 삭제

```
obj
```

```
a    0.0  
b    1.0  
c    2.0  
d    3.0  
e    4.0  
dtype: float64
```

```
obj.drop('c', inplace=True)  
obj
```

```
a    0.0  
b    1.0  
d    3.0  
e    4.0  
dtype: float64
```

# 축에서 항목 삭제하기

## □ DataFrame

### ▣ 행 삭제

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])  
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data.drop(['Colorado', 'Ohio']) axis 0에서 해당 항목 삭제
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

# 축에서 항목 삭제하기

## ▣ 열 삭제

```
data.drop('two', axis=1)
```

 axis 1 에서 해당 항목 삭제

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

```
data.drop(['two', 'four'], axis='columns')
```

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

# 색인하기, 선택하기, 거르기(filtering)

## □ Series

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])  
obj
```

```
a    0.0  
b    1.0  
c    2.0  
d    3.0  
dtype: float64
```

```
obj['b']
```

```
1.0
```

```
obj[1]
```

```
1.0
```

```
obj[2:4]
```

```
c    2.0  
d    3.0  
dtype: float64
```

정수로 슬라이싱  
→ 끝점 미포함

```
obj[['b', 'a', 'd']]
```

```
b    1.0  
a    0.0  
d    3.0  
dtype: float64
```

```
obj[[1, 3]]
```

```
b    1.0  
d    3.0  
dtype: float64
```

```
obj[obj < 2]
```

```
a    0.0  
b    1.0  
dtype: float64
```

라벨 이름으로 슬라이싱  
→ 끝점 포함

```
obj['b':'c']
```

```
b    1.0  
c    2.0  
dtype: float64
```

```
obj['b':'c'] = 5  
obj
```

```
a    0.0  
b    5.0  
c    5.0  
d    3.0  
dtype: float64
```

# 색인하기, 선택하기, 거르기(filtering)

## □ DataFrame

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),  
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],  
                    columns=['one', 'two', 'three', 'four'])  
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

# 색인하기, 선택하기, 거르기(filtering)

- 열 선택하기

```
data['two']
```

```
Ohio      1
Colorado  5
Utah       9
New York  13
Name: two, dtype: int32
```

```
data[['three', 'one']]
```

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

- 행 선택하기

슬라이싱으로 선택

```
data[:2]
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

불리언 배열로 선택

```
data[data['three'] > 5]
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

불리언 DataFrame으로 값 선택

```
data < 5
```

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
data[data < 5] = 0
data
```

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

# 색인하기, 선택하기, 거르기(filtering)

## □ DataFrame을 라벨로 선택하기

- ▣ **loc** : 축 이름으로 선택, **iloc** : 정수 색인으로 선택

data				
	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data.loc['Colorado', ['two', 'three']]
```

```
two      5
three    6
Name: Colorado, dtype: int32
```



# 색인하기, 선택하기, 거르기(filtering)

data

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

data.iloc[2, [3, 0, 1]]

```
four    11
one      8
two      9
Name: Utah, dtype: int32
```

data.iloc[2]

```
one      8
two      9
three    10
four     11
Name: Utah, dtype: int32
```

data.iloc[[1, 2], [3, 0, 1]]

	four	one	two
Colorado	7	0	5
Utah	11	8	9

# 색인하기, 선택하기, 거르기(filtering)

data

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

data.loc[:, 'Utah', 'two']

Ohio 0  
Colorado 5  
Utah 9  
Name: two, dtype: int32

data.iloc[:, :3][data.three > 5]

	one	two	three
Colorado	0	5	6
Utah	8	9	10
New York	12	13	14

# 색인하기, 선택하기, 거르기(filtering)

## □ DataFrame의 값 선택하기

방식	설명
<code>df[val]</code>	DataFrame에서 하나의 컬럼 또는 여러 컬럼을 선택한다. 편의를 위해 불리언 배열, 슬라이스, 불리언 DataFrame(어떤 기준에 근거해서 값을 대입해야 할 때)을 사용할 수 있다.
<code>df.loc[val]</code>	DataFrame에서 라벨값으로 로우의 부분집합을 선택한다.
<code>df.loc[:, val]</code>	DataFrame에서 라벨값으로 컬럼의 부분집합을 선택한다.
<code>df.loc[val1, val2]</code>	DataFrame에서 라벨값으로 로우와 컬럼의 부분집합을 선택한다.
<code>df.iloc[where]</code>	DataFrame에서 정수 색인으로 로우의 부분집합을 선택한다.
<code>df.iloc[:, where]</code>	DataFrame에서 정수 색인으로 컬럼의 부분집합을 선택한다.
<code>df.iloc[where_i, where_j]</code>	DataFrame에서 정수 색인으로 로우와 컬럼의 부분집합을 선택한다.
<code>df.at[label_i, label_j]</code>	로우와 컬럼의 라벨로 단일 값을 선택한다.
<code>df.iat[i, j]</code>	로우와 컬럼의 정수 색인으로 단일 값을 선택한다.
reindex 메서드	하나 이상의 축을 새로운 색인으로 맞춘다.
get_value, set_value 메서드	로우와 컬럼 이름으로 DataFrame의 값을 선택한다.

## 정수 색인

```
ser = pd.Series(np.arange(3.))  기본 라벨 색인 (라벨 0, 1, 2)
ser
```

```
0    0.0
1    1.0
2    2.0
dtype: float64
```

```
ser.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
ser[-1]  라벨 색인 2 인지 정수 색인 2인지 모호함으로 에러 발생
          파이썬 리스트 색인 방식
```

-----  
-----  
**KeyError**

Traceback (m

## 정수 색인

```
ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
ser2
```

라벨 색인 (라벨 'a', 'b', 'c')

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

```
ser2.index
```

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

```
ser2[-1]
```

라벨 색인 2가 아니라 정수 색인 2로 명확함

```
2.0
```

# 정수 색인

```
ser
0    0.0
1    1.0
2    2.0
dtype: float64
```

```
ser[:1]
0    0.0
dtype: float64

ser.loc[:1]
0    0.0
1    1.0
dtype: float64

ser.iloc[:1]
0    0.0
dtype: float64
```

정수의 축 색인이 있다면  
우선적으로 라벨 색인으로 선택

라벨 색인

정수 색인

# 산술 연산과 데이터 정렬

- 다른 색인을 가지는 객체 간의 산술 연산
  - ▣ 두 색인이 색인 통합됨
- Series

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
               index=['a', 'c', 'e', 'f', 'g'])
```

s1	
a	7.3
c	-2.5
d	3.4
e	1.5
dtype: float64	

s2	
a	-2.1
c	3.6
e	-1.5
f	4.0
g	3.1
dtype: float64	

s1 + s2	
a	5.2
c	1.1
d	NaN
e	0.0
f	NaN
g	NaN
dtype: float64	

겹치는 색인이  
없는 경우 NaN

# 산술 연산과 데이터 정렬

## □ DataFrame

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),  
                    index=['Ohio', 'Texas', 'Colorado'])  
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),  
                    index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

df1				
	b	c	d	
Ohio	0.0	1.0	2.0	
Texas	3.0	4.0	5.0	
Colorado	6.0	7.0	8.0	

df2				
	b	d	e	
Utah	0.0	1.0	2.0	
Ohio	3.0	4.0	5.0	
Texas	6.0	7.0	8.0	
Oregon	9.0	10.0	11.0	

df1 + df2					
	b	c	d	e	
Colorado	NaN	NaN	NaN	NaN	
Ohio	3.0	NaN	6.0	NaN	
Oregon	NaN	NaN	NaN	NaN	
Texas	9.0	NaN	12.0	NaN	
Utah	NaN	NaN	NaN	NaN	



# 산술 연산과 데이터 정렬

```
df1 = pd.DataFrame({'A': [1, 2]})  
df2 = pd.DataFrame({'B': [3, 4]})
```

df1	
A	
0	1
1	2

df2	
B	
0	3
1	4

df1 - df2		
A B		
0	NaN	NaN
1	NaN	NaN

# 산술 연산과 데이터 정렬

## □ 산술 연산 메서드에 채워 넣을 값 지정하기

```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),  
                    columns=list('abcd'))  
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),  
                    columns=list('abcde'))  
df2.loc[1, 'b'] = np.nan
```

df1				
	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

df2					
	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

df1 + df2					
	a	b	c	d	e
0	0.0	2.0	4.0	6.0	NaN
1	9.0	NaN	13.0	15.0	NaN
2	18.0	20.0	22.0	24.0	NaN
3	NaN	NaN	NaN	NaN	NaN

# 산술 연산과 데이터 정렬

- 존재하지 않는 축의 값을 (0과 같은) 특수한 값으로 지정하여 연산

```
df1.add(df2, fill_value=0)
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

- 계산 인자를 뒤집어 계산하는 짝궁 메서드-r로 시작

```
1 / df1
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250	0.200000	0.166667	0.142857
2	0.125	0.111111	0.100000	0.090909

```
df1.rdiv(1)
```

	a	b	c	d
0	inf	1.000000	0.500000	0.333333
1	0.250	0.200000	0.166667	0.142857
2	0.125	0.111111	0.100000	0.090909

# 산술 연산과 데이터 정렬

- Series나 DataFrame 재색인 시 fill\_value 지정

```
df1.reindex(columns=df2.columns, fill_value=0)
```

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	0
1	4.0	5.0	6.0	7.0	0
2	8.0	9.0	10.0	11.0	0

- 산술 연산 메서드

메서드	설명
add, radd	덧셈(+)을 위한 메서드
sub, rsub	뺄셈(-)을 위한 메서드
div, rdiv	나눗셈(/)을 위한 메서드
floordiv, rfloordiv	소수점 내림(//) 연산을 위한 메서드
mul, rmul	곱셈(*)을 위한 메서드
pow, rpow	멱승(**)을 위한 메서드

# 산술 연산과 데이터 정렬

## ▣ 2차원 배열과 그 배열의 한 행에 대한 연산

```
arr = np.arange(12.).reshape((3, 4))  
arr
```

```
array([[ 0.,  1.,  2.,  3.],  
       [ 4.,  5.,  6.,  7.],  
       [ 8.,  9., 10., 11.]])
```

```
arr[0]
```

```
array([0., 1., 2., 3.])
```

```
arr - arr[0]
```

모든 행에 대해 연산 (브로드캐스팅)

```
array([[0., 0., 0., 0.],  
       [4., 4., 4., 4.],  
       [8., 8., 8., 8.]])
```

# 산술 연산과 데이터 정렬

## ▣ DataFrame과 Series 간의 연산

```
frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),  
                      columns=list('bde'),  
                      index=['Utah', 'Ohio', 'Texas', 'Oregon'])  
series = frame.iloc[0]
```

frame	series				frame - series
	b	d	e		
Utah	0.0	1.0	2.0	b 0.0 d 1.0 e 2.0 Name: Utah, dtype: float64	Utah 0.0 0.0 0.0
Ohio	3.0	4.0	5.0		Ohio 3.0 3.0 3.0
Texas	6.0	7.0	8.0		Texas 6.0 6.0 6.0
Oregon	9.0	10.0	11.0		Oregon 9.0 9.0 9.0

## 산술 연산과 데이터 정렬

- 색인값을 DataFrame의 열이나 Series의 색인에서 찾을 수 없다면 형식을 맞추기 위해 **재색인**됨

```
series2 = pd.Series(range(3), index=['b', 'e', 'f'])
```

frame				series2		frame + series2				
	b	d	e	b	e		b	d	e	f
Utah	0.0	1.0	2.0	f	2	Utah	0.0	NaN	3.0	NaN
Ohio	3.0	4.0	5.0	dtype: int64		Ohio	3.0	NaN	6.0	NaN
Texas	6.0	7.0	8.0			Texas	6.0	NaN	9.0	NaN
Oregon	9.0	10.0	11.0			Oregon	9.0	NaN	12.0	NaN

# 산술 연산과 데이터 정렬

- 각 행 방향으로 연산을 수행하고 싶다면 산술 연산 메서드를 사용

```
series3 = frame['d']
```

frame	series3	frame.sub(series3, <u>axis='index'</u> )
	Utah 1.0	(axis=0) 연산을 적용할 축 번호
<b>b</b> <b>d</b> <b>e</b>	Ohio 4.0	<b>b</b> <b>d</b> <b>e</b>
<b>Utah</b> 0.0 1.0 2.0	Texas 7.0	<b>Utah</b> -1.0 0.0 1.0
<b>Ohio</b> 3.0 4.0 5.0	Oregon 10.0	<b>Ohio</b> -1.0 0.0 1.0
<b>Texas</b> 6.0 7.0 8.0	Name: d, dtype: float64	<b>Texas</b> -1.0 0.0 1.0
<b>Oregon</b> 9.0 10.0 11.0		<b>Oregon</b> -1.0 0.0 1.0



# Reference

---

- 참고도서

- ▣ Python for Data Analysis, 웨스 펙키니, 한빛미디어
  - Chapter 5. Pandas 시작하기

- 남은 내용들

- 5.2.6 함수 적용과 매핑
- 5.2.7 정렬과 순위
- 5.2.8 중복 색인
- 5.3 기술 통계 계산과 요약