

# Pandas

---



# Pandas



- 1~2차원의 표 형태의 데이터를 아주 잘 다룹니다.
- 행과 열에 이름을 붙일 수 있습니다. (*index*와 *columns*)

pandas → 데이터 표를 다루는 도구

	A	D	E	F	I	J
	Pas	Name	Sex	Age	Ticket	Fare
1	1	Braund, Mr. Owen Harris	male	22	A/5 21171	7.25
2	2	Cumings, Mrs. John Bradley (Florence Briggs)	female	38	PC 17599	71.2833
3	3	Heikkinen, Miss. Laina	female	26	STON/O2.	7.925
4	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	113803	53.1
5	5	Allen, Mr. William Henry	male	35	373450	8.05
6	6	Moran, Mr. James	male		330877	8.4583
7	7	McCarthy, Mr. Timothy J	male	54	17463	51.8625
8	8	Palsson, Master. Gosta Leonard	male	2	349909	21.075
9	9	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina)	female	27	347742	11.1333
10	10	Nasser, Mrs. Nicholas (Adele Achem)	female	14	237736	30.0708
11	11	Sandstrom, Miss. Marguerite Rut	female	4	PP 9549	16.7
12	12	Bonnell, Miss. Elizabeth	female	58	113783	26.55

# Pandas



- “Python Data Analysis Library” 전세계에서 가장 인기있는 데이터분석 패키지
- Row (행)과 Column (열) 로 이루어진 2차원 테이블 형태를 매우 잘 다룰 수 있음

행 (날짜) x  
값 형태의  
Series

```
sr0 = pd.Series(np.random.randn(5).round(2),
                 index=pd.date_range(start=dt(2020,10,1), end=dt(2020,10,5)))
sr0
```

2020-10-01	0.83
2020-10-02	0.13
2020-10-03	0.22
2020-10-04	-0.54
2020-10-05	1.25

Freq: D, dtype: float64

행 (날짜) x  
열 형태의  
DataFrame

```
pd.DataFrame(np.random.randn(5, 3).round(2),
              index=pd.date_range(start=dt(2020,10,1), end=dt(2020,10,5)),
              columns=['열1', '열2', '열3'])
```

	열1	열2	열3
2020-10-01	-0.26	-1.34	-0.57
2020-10-02	1.36	0.05	1.95
2020-10-03	-0.57	0.80	-1.65
2020-10-04	1.36	-0.88	-0.44
2020-10-05	0.47	-0.54	-0.89

# pandas 의 등장. 데이터에 라벨을 붙이고 싶어요

## ▪ pandas가 등장한 이유

- 주가 데이터를 2차원 numpy 배열로 살펴보자 (???)

	0	1	2	3	4	5	6	7
0	array([[58400., 59200., 59000., 60400., 61000., 61000., 59500., 59300.],							
1	[ 9190., 9280., 9270., 9370., 9470., 9530., 9320., 9320.],							
2	[ 9080., 9280., 9360., 9400., 9390., 9080., 8880., 9070.],							
3	[11000., 11150., 11050., 11200., 11350., 11250., 11000., 10850.],							
4	[ 671., 671., 671., 671., 671., 671., 671., 671.],							
5	[ 7990., 7960., 8120., 8200., 8180., 8180., 8000., 8020.]]							

- 한 종목의 주가 뽑아내기

```
주가_데이터_넘파이_배열[0]
array([58400., 59200., 59000., 60400., 61000., 61000., 59500., 59300.])
```

그래서 도.대.체. 이 종목은 무슨 종목이고, 주가의 날짜가 언제인가요?

0번 종목의 0~7번 주가입니다. 자세한 건 설명서를 찾아보세요 ...

# pandas 기본 1. 『 Series 』 행×값 데이터

- 1차원 데이터를 잘 표현하기 위해 만든, Series
  - 데이터 배열에 이름과 각 데이터의 라벨(인덱스)를 붙임

		이름	A005930
			+
2020-09-09	+		58400
2020-09-10			59200
2020-09-11			59000
2020-09-14			60400
2020-09-15			61000
2020-09-16			61000
2020-09-17			59500
2020-09-18			59300
인덱스			데이터 배열

# pandas 기본 1. 『 Series 』 행×값 데이터

- pandas 임포트

```
import pandas as pd
```

- Series 출력

my_series	# Series
2020-09-09	58400.0
2020-09-10	59200.0
2020-09-11	59000.0
2020-09-14	60400.0
2020-09-15	61000.0
2020-09-16	61000.0
2020-09-17	59500.0
2020-09-18	59300.0

Name: A005930, dtype: float64

# pandas 기본 1. 『 Series 』 행 × 값 데이터

## ■ 인덱스와 이름 보기

```
my_series.index    # Series의 인덱스 보기
```

```
Index(['2020-09-09', '2020-09-10', '2020-09-11',  
      '2020-09-14', '2020-09-15', '2020-09-16',  
      '2020-09-17', '2020-09-18'],  
      dtype='object')
```

```
my_series.name     # Series의 이름 보기
```

```
'A005930'
```

## pandas 기본 2. 『 Series 』 접근

```
my_series.loc[ 인덱스 (or 인덱스slice) ]  
my_series.iloc[ 배열번호 (or 배열번호slice) ]
```

.loc 는 Series에서 인덱스를 이용, 일부를 추출한다. (slice는 끝을 포함)

.iloc는 Series에서 배열번호를 이용, 일부를 추출한다. (slice는 끝을 포함 안함)

인자는 또한 slice가 될 수 있다. (슬라이스: ‘처음:끝:간격’ 형태)

- indexing을 통해 series의 원소를 가져올 수 있습니다.

```
my_series.loc['2020-09-15'] # 인덱스로 접근
```

61000.0

```
my_series.iloc[4] # 배열 번호로 접근
```

61000.0



## pandas 기본 2. 『 Series 』 접근

- slicing을 통해 series의 부분을 가져올 수 있습니다.

```
my_series.loc['2020-09-15':'2020-09-17'] # 인덱스로 Slicing
```

```
2020-09-15    61000.0  
2020-09-16    61000.0  
2020-09-17    59500.0  
Name: A005930, dtype: float64
```

```
my_series.iloc[4:7] # 배열 번호로 Slicing (끝 포함 X)
```

```
2020-09-15    61000.0  
2020-09-16    61000.0  
2020-09-17    59500.0  
Name: A005930, dtype: float64
```

## pandas 기본 2. 『 Series 』 연산

### ▪ Broadcasting

```
my_series.iloc[4:] + my_series.iloc[4:]
```

```
2020-09-15    122000.0  
2020-09-16    122000.0  
2020-09-17    119000.0  
2020-09-18    118600.0  
Name: A005930, dtype: float64
```

```
my_series.loc['2020-09-15'::2] + 30000 # Scalar Broadcasting
```

```
2020-09-15     91000.0  
2020-09-17     89500.0  
Name: A005930, dtype: float64
```

## pandas 기본 2. 『 Series 』 연산

- Aggregation

4일간의 주가 평균! (9/15 ~ 9/18)

```
my_series.iloc[4:].mean()    # 평균 aggregation
```

60200.0

## pandas 기본 2. 『 Series 』 생성

- `pd.Series()`를 통해 새로운 Series를 생성합니다.
- 리스트는 데이터로 입력됩니다.

```
new_sr = pd.Series([11,22,3,45], name='apple',  
                    index=['a','b','xs','e11'])
```

new\_sr

a        11

b        22

xs       3

e11      45

Name: apple, dtype: int64

\* 인자 `name`과 `index`는 생략 가능함

- `name`은 기본 값, `None`이 됨
- `index`는 0, 1, 2, 3, ... 으로 자동 지정 됨

## pandas 기본 3. 『 DataFrame 』 행 × 열 데이터

- 2차원 데이터를 잘 다루기 위해 만든, DataFrame
  - 여러 개의 Series를 묶어서 만든 형태 (즉, 하나 씩 떼어놓으면 개는 Series임)
  - 각각 Series의 name은 DataFrame에서는 열(Column)이 된다

pd.Series

	name
	A005930
2020-09-09	58400
2020-09-10	59200
2020-09-11	59000
2020-09-14	60400
2020-09-15	61000
2020-09-16	61000
2020-09-17	59500
2020-09-18	59300

index

data

+

pd.Series

	name
	A005940
2020-09-09	9190
2020-09-10	9280
2020-09-11	9270
2020-09-14	9370
2020-09-15	9470
2020-09-16	9530
2020-09-17	9320
2020-09-18	9320

index

data

=

pd.DataFrame

	columns	
	A005930	A005940
2020-09-09	58400	9190
2020-09-10	59200	9280
2020-09-11	59000	9270
2020-09-14	60400	9370
2020-09-15	61000	9470
2020-09-16	61000	9530
2020-09-17	59500	9320
2020-09-18	59300	9320

index

data

## pandas 기본 3. 『 DataFrame 』 행 × 열 데이터

- 2차원 데이터를 잘 다루기 위해 만든, DataFrame
  - 여러 개의 Series를 묶어서 만든 형태 (즉, 하나 씩 떼어놓으면 개는 Series임)
  - 각각 Series의 name은 DataFrame에서는 열(Column)이 된다

my\_df

Symbol	A005930	A005940	A005950	A005960	A005980	A005990
2020-09-09	58400.0	9190.0	9080.0	11000.0	671.0	7990.0
2020-09-10	59200.0	9280.0	9280.0	11150.0	671.0	7960.0
2020-09-11	59000.0	9270.0	9360.0	11050.0	671.0	8120.0
2020-09-14	60400.0	9370.0	9400.0	11200.0	671.0	8200.0
2020-09-15	61000.0	9470.0	9390.0	11350.0	671.0	8180.0
2020-09-16	61000.0	9530.0	9080.0	11250.0	671.0	8180.0
2020-09-17	59500.0	9320.0	8880.0	11000.0	671.0	8000.0
2020-09-18	59300.0	9320.0	9070.0	10850.0	671.0	8020.0

## pandas 기본 4. 『 DataFrame 』 접근

```
my_df.loc[ index (or slice) , column (or slice) ]
my_df.iloc[ 인덱스번호 (or slice) , 컬럼번호 (or slice) ]
```

.loc 는 DataFrame에서 인덱스와 컬럼(열 이름)을 이용, 일부를 추출한다. (slice 끝 포함)

.iloc는 DataFrame에서 인덱스번호와 컬럼 번호를 이용, 일부를 추출한다. (slice 끝 포함 X)

- 모든 인자는 slice가 될 수 있다. (슬라이스: ‘처음:끝:간격’ 형태)
- 추출할 shape가 (1, m) 혹은 (n, 1)일 경우 Series가 반환되며, 그 외 DataFrame을 반환한다.

- indexing을 통해 dataframe의 원소를 가져올 수 있습니다.

```
my_df.loc[ '2020-09-15' , 'A005930' ]    # 단일 원소 접근
```

61000.0

## pandas 기본 4. 『 DataFrame 』 접근

- indexing + slicing으로 접근 시, 접근 결과가 1차원이면 series가 반환됩니다.

```
▼ # [ 인덱스, 컬럼 slice ] 로 접근 → Series 반환  
my_df.loc['2020-09-15', 'A005930':'A005950']
```

```
Symbol  
A005930    61000.0  
A005940     9470.0  
A005950     9390.0  
Name: 2020-09-15, dtype: float64
```

```
▼ # [ 인덱스 slice, 컬럼 이름 ] 으로 접근 → Series 반환  
my_df.loc['2020-09-15':'2020-09-18', 'A005930']
```

```
2020-09-15    61000.0  
2020-09-16    61000.0  
2020-09-17    59500.0  
2020-09-18    59300.0  
Name: A005930, dtype: float64
```



## pandas 기본 4. 『 DataFrame 』 접근

- indexing + slicing으로 접근 시, 접근 결과가 2차원이면 dataframe이 반환됩니다.

```
▼ # .loc [ index slice, column slice ] → DataFrame 반환  
my_df.loc[ '2020-09-14': '2020-09-18':2, 'A005940': 'A005980' ]
```

Symbol	A005940	A005950	A005960	A005980
2020-09-14	9370.0	9400.0	11200.0	671.0
2020-09-16	9530.0	9080.0	11250.0	671.0
2020-09-18	9320.0	9070.0	10850.0	671.0

shape (3, 4)

## pandas 기본 4. 『 DataFrame 』 접근

### ▪ iloc를 이용한 접근

▼ *# .iloc [ index, column slice ] 으로 접근 → Series 반환*  
`my_df.iloc[0, 3:]`

```
Symbol
A005960    11000.0
A005980     671.0
A005990     7990.0
Name: 2020-09-09, dtype: float64
```

shape (3, )

▼ *# .iloc [ index slice, column slice ] → DataFrame 반환*  
`my_df.iloc[4:7:2, 1:4]`

	Symbol	A005940	A005950	A005960
	2020-09-15	9470.0	9390.0	11350.0
	2020-09-17	9320.0	8880.0	11000.0

shape (2, 3)

## pandas 기본 4. 『 DataFrame 』 접근 2

- :으로 모든 원소를 지칭하기

```
▼ # ::으로 모든 원소 지칭
my_df.loc[:, 'A005930']
```

```
2020-09-09    58400.0
2020-09-10    59200.0
2020-09-11    59000.0
2020-09-14    60400.0
2020-09-15    61000.0
2020-09-16    61000.0
2020-09-17    59500.0
2020-09-18    59300.0
Name: A005930, dtype: float64
```

```
▼ # : 한번만 사용 가능
my_df.iloc[5, :]
```

```
Symbol
A005930    61000.0
A005940     9530.0
A005950     9080.0
A005960    11250.0
A005980     671.0
A005990     8180.0
Name: 2020-09-16, dtype: float64
```

```
▼ # shape 보기
my_df.iloc[5, :].shape
```

```
(6,)
```

## pandas 기본 4. 『 DataFrame 』 접근 2

- slicing 응용

```
my_df.iloc[5:, ::3]
```

Symbol	A005930	A005960
2020-09-16	61000.0	11250.0
2020-09-17	59500.0	11000.0
2020-09-18	59300.0	10850.0

```
my_df.iloc[5:, ::3].shape
```

(3, 2)

## pandas 기본 4. 『 DataFrame 』 접근 2

- 응용의 응용

```
my_df.iloc[::4].loc[:, 'A005950':]
```

Symbol	A005950	A005960	A005980	A005990
2020-09-09	9080.0	11000.0	671.0	7990.0
2020-09-15	9390.0	11350.0	671.0	8180.0

```
my_df.loc['2020-09-09'::4].iloc[:, 2:]
```

Symbol	A005950	A005960	A005980	A005990
2020-09-09	9080.0	11000.0	671.0	7990.0
2020-09-15	9390.0	11350.0	671.0	8180.0

## pandas 기본 4. 『 DataFrame 』 연산

### ▪ Broadcasting

```
my_df_2 = my_df.loc[:, '2020-09-11', 'A005960':]
```

```
my_df_2
```

df 일부를 추출,  
새로운 df 만들기

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

```
my_df_2 * 1.30 # Broadcasting
```

30% 상승  
(상한가 가격)

Symbol	A005960	A005980	A005990
2020-09-09	14300.0	872.3	10387.0
2020-09-10	14495.0	872.3	10348.0
2020-09-11	14365.0	872.3	10556.0

## pandas 기본 5. 『 DataFrame 』 연산

### ▪ Aggregation

```
my_df_2.mean(axis=0) # 평균 Agg
```

```
Symbol  
A005960    11066.666667  
A005980     671.000000  
A005990     8023.333333  
dtype: float64
```

종목별 3일 주가 평균

## pandas 기본 6. 『 DataFrame 』 연산 2

### ■ DataFrame 연산 총 정리

#### - 단일 연산

<b>.abs()</b>	<b>.isna()</b>	<b>.notna()</b>	<b>.pow()</b>
절댓값	na여부	유효여부	거듭제곱

```
my_df_2.notna()
```

Symbol	A005960	A005980	A005990
2020-09-09	True	True	True
2020-09-10	True	True	True
2020-09-11	True	True	True

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

my\_df\_2



## pandas 기본 6. 『 DataFrame 』 연산 2

### ■ DataFrame 연산 총 정리

#### - 축 방향 연산 (axis= 0 or 1)

**.mean()**      **.median()**      **.max()**      **.min()**      **.sum()**  
 평균              중앙값              최댓값              최솟값              더하기

**.prod()**      **.idxmax()**      **.idxmin()**  
 곱하기      최대원소의 인덱스      최소원소의 인덱스

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

my\_df\_2

```
my_df_2.sum(axis=0)
```

```
Symbol
A005960    33200.0
A005980     2013.0
A005990    24070.0
dtype: float64
```

```
my_df_2.median(axis=1)
```

```
2020-09-09    7990.0
2020-09-10    7960.0
2020-09-11    8120.0
dtype: float64
```

## pandas 기본 6. 『 DataFrame 』 연산 2

### ■ DataFrame 연산 총 정리

#### - 누적 축 방향 연산 (axis=0 or 1)

**.cummax()**

누적최댓값

**.cummin()**

누적최솟값

**.cumprod()**

누적곱셈

**.cumsum()**

누적덧셈

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

my\_df\_2

```
my_df_2.cumsum(axis=0)
```

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	22150.0	1342.0	15950.0
2020-09-11	33200.0	2013.0	24070.0

## pandas 기본 6. 『 DataFrame 』 정렬

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

my\_df\_2

- 특정 axis를 기준으로 정렬하기

```
my_df.sort_values(정렬기준, axis=축, ascending=True)
```

정렬기준의 행 또는 열 값들을 기준으로 axis 방향 정렬을 수행한다.

[인자 설명]

- axis: 정렬할 축, 0 또는 'index'는 행을 정렬하고 1 또는 'columns'은 열을 정렬한다.
- ascending: 오름차순 여부를 지정함. 기본 값은 True, 내림차순은 False

```
my_df_2.sort_values('2020-09-11', axis='columns', ascending=True)
```

Symbol	A005980	A005990	A005960
2020-09-09	671.0	7990.0	11000.0
2020-09-10	671.0	7960.0	11150.0
2020-09-11	671.0	8120.0	11050.0

'2020-09-11' 인덱스를 기준,  
컬럼 값으로 오름차순 정렬

## pandas 기본 6. 『 DataFrame 』 정렬

Symbol	A005960	A005980	A005990
2020-09-09	11000.0	671.0	7990.0
2020-09-10	11150.0	671.0	7960.0
2020-09-11	11050.0	671.0	8120.0

my\_df\_2

- 특정 axis를 기준으로 랭킹 매기기

```
my_df.rank(axis=축, ascending=True)
```

축의 방향으로 정렬, 각각의 행 또는 열들의 순위를 매긴다. 축과 오름차순 여부 지정 가능.

```
my_df_2.rank(axis=0, ascending=False)
```

Symbol	A005960	A005980	A005990
2020-09-09	3.0	2.0	2.0
2020-09-10	1.0	2.0	3.0
2020-09-11	2.0	2.0	1.0

각각의 컬럼 별, 컬럼 내 값으로  
내림차순 순위를 매김

(같은 값의 경우, 순위의 평균)

## pandas 기본 7. 『 DataFrame 』 만들기

- `pd.DataFrame()`을 통해 새로운 DataFrame을 생성합니다.
- 2차원 리스트는 데이터로 입력됩니다.

```

# 데이터 인덱스, 컬럼으로 DataFrame 생성하기
pd.DataFrame([[0,1,2],[3,4,5]],
              index=['00','11'],
              columns=['a','b','c'])

```

	a	b	c
00	0	1	2
11	3	4	5

```

# 넘파이 배열로 DataFrame 생성하기
pd.DataFrame(np.ones((3,5)))

```

	0	1	2	3	4
0	1.0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0	1.0

## pandas 기본 7. 『 DataFrame 』 만들기

- 행과 열을 추가/수정할 때 사용할 실습 DataFrame 입니다.

```
▼ traffic_data = pd.DataFrame(  
    [[9800,5500],[10200,6600]],  
    index=['10/22','10/23'],  
    columns=['원티드','사람인'])  
traffic_data    # traffic_data 라는 DataFrame 생성
```

	원티드	사람인
10/22	9800	5500
10/23	10200	6600

## pandas 기본 7. 『 DataFrame 』 행 열 추가하기

### ■ 새로운 열 추가하기

```
# 새로운 '잡플래닛' 열 (빈 데이터) 추가하기
traffic_data.loc[:, '잡플래닛'] = np.nan
traffic_data
```

numpy와 pandas에서는  
값이 없는 빈 값을  
NaN 이라 정의

	원티드	사람인	잡플래닛
10/22	9800	5500	NaN
10/23	10200	6600	NaN

```
# 새로운 '잡코리아' 열 데이터 추가하기
traffic_data.loc[:, '잡코리아'] = [980, 1020]
traffic_data
```

	원티드	사람인	잡플래닛	잡코리아
10/22	9800	5500	NaN	980
10/23	10200	6600	NaN	1020

## pandas 기본 7. 『 DataFrame 』 행 열 추가하기

### ■ 새로운 열 추가하기

```
▼ # 숫자를 하나만 주면 broadcast 된다.  
traffic_data.loc[:, '스펙업'] = 20000  
traffic_data
```

	원티드	사람인	잡플래닛	잡코리아	스펙업
10/22	9800	5500	NaN	980	20000
10/23	10200	6600	NaN	1020	20000



## pandas 기본 7. 『 DataFrame 』 행 열 추가하기

### ■ 새로운 행 추가하기

```

# 새로운 10/24 행 데이터 추가하기
traffic_data.loc['10/24'] = [12500, 4520, 3000,
                             780, 21000]
traffic_data

```

	원티드	사람인	잡플래닛	잡코리아	스펙업
10/22	9800	5500	NaN	980	20000
10/23	10200	6600	NaN	1020	20000
10/24	12500	4520	3000.0	780	21000

```

# 값을 하나만 지정하면, broadcast 된다
traffic_data.loc['10/25'] = 2000
traffic_data

```

	원티드	사람인	잡플래닛	잡코리아	스펙업
10/22	9800	5500	NaN	980	20000
10/23	10200	6600	NaN	1020	20000
10/24	12500	4520	3000.0	780	21000
10/25	2000	2000	2000.0	2000	2000

## pandas 기본 7. 『 DataFrame 』 값 수정하기

- 행과 열을 함께 지정하며, 데이터를 수정합니다.

```
traffic_data.loc['10/22', '잡플래닛'] = 99  
traffic_data
```

	원티드	사람인	잡플래닛	잡코리아	스펙업
10/22	9800	5500	99.0	980	20000
10/23	10200	6600	NaN	1020	20000
10/24	12500	4520	3000.0	780	21000
10/25	2000	2000	2000.0	2000	2000

# pandas 고급: concat() 으로 병합하기

- concat은 index를 기준으로 여러 개의 dataframe을 병합하는 함수입니다
- 복수 개의 dataframe을 병합할 수 있습니다.
  - 값을 찾을 수 없는 경우에는 NaN 값으로 채워집니다
- df\_a와 df\_b를 병합 재료로 준비합니다

## df\_a 준비

```
df_a = stock_info.iloc[:14]
df_a
```

	itemname	Sector
Symbol		
A000020	동화약품	제약_및_바이오
A000030	우리은행	NaN
A000040	KR모터스	자동차_및_부품
A000050	경방	내구_소비재_및_의류
A000060	메리츠화재	보험
A000070	삼양홀딩스	소재
A000080	하이트진로	음식료_및_담배
A000100	유한양행	제약_및_바이오
A000120	CJ대한통운	운송
A000140	하이트진로홀딩스	음식료_및_담배
A000150	두산	자본재
A000180	성창기업지주	소재
A000210	대림산업	자본재
A000220	유유제약	제약_및_바이오

## df\_b 준비

```
df_b = d['adj_close'].loc[:, '2020-10-13:'].iloc[:14]
df_b
```

추후, query문을 사용하기 위해 컬럼 명 변경

```
df_b = df_b.rename(columns={
    '2020-10-13': '주가_10_13',
    '2020-10-14': '주가_10_14'})
df_b
```

	주가_10_13	주가_10_14
Symbol		
A000020	24300.0	23850.0
A000030	14800.0	14800.0
A000040	850.0	874.0
A000050	10850.0	10800.0
A000060	13700.0	13550.0
A000070	63000.0	61800.0
A000080	38700.0	38500.0
A000100	64100.0	63600.0
A000120	186500.0	180500.0
A000140	17300.0	17250.0
A000150	47500.0	47700.0
A000180	1715.0	1825.0
A000210	78300.0	76600.0
A000220	16600.0	16250.0

# pandas 고급: `concat()` 으로 병합하기

- `axis=` 키워드 인자로 병합하는 방향을 지정
  - 0 (또는 'index') 은 행 방향 ↓ 으로 병합
  - 1 (또는 'column') 은 열 방향 → 으로 병합
- `df.concat([df_a, df_b])` 로 병합하면 의도와 다르게 행 방향으로 병합됨
  - 병합 시, 찾을 수 없는 값은 NaN으로 채워집니다.

```
pd.concat([df_a, df_b])
```

	itemname	Sector	2020-10-13	2020-10-14
Symbol				
A000020	동화약품	제약_및_바이오	NaN	NaN
A000030	우리은행	NaN	NaN	NaN
A000040	KR모터스	자동차_및_부품	NaN	NaN
A000050	경방	내구_소비재_및_의류	NaN	NaN
A000060	메리츠화재	보험	NaN	NaN
A000070	삼양홀딩스	소재	NaN	NaN
A000080	하이트진로	음식료_및_담배	NaN	NaN
A000100	유한양행	제약_및_바이오	NaN	NaN
A000120	CJ대한통운	운송	NaN	NaN
A000140	하이트진로홀딩스	음식료_및_담배	NaN	NaN
A000150	두산	자본재	NaN	NaN
A000180	성장기업지주	소재	NaN	NaN
A000210	대림산업	자본재	NaN	NaN
A000220	유유제약	제약_및_바이오	NaN	NaN
A000020	NaN	NaN	24300.0	23850.0
A000030	NaN	NaN	14800.0	14800.0
A000040	NaN	NaN	850.0	874.0
A000050	NaN	NaN	10850.0	10800.0
A000060	NaN	NaN	13700.0	13550.0
A000070	NaN	NaN	63000.0	61800.0
A000080	NaN	NaN	38700.0	38500.0
A000100	NaN	NaN	64100.0	63600.0
A000120	NaN	NaN	186500.0	180500.0
A000140	NaN	NaN	17300.0	17250.0
A000150	NaN	NaN	47500.0	47700.0
A000180	NaN	NaN	1715.0	1825.0
A000210	NaN	NaN	78300.0	76600.0
A000220	NaN	NaN	16600.0	16250.0

df\_a

df\_b

# pandas 고급: concat() 으로 병합하기

- df.concat([df\_a, df\_b], axis=1)  
로 열 방향 병합을 수행하면,  
우측의 결과와 같이 의도한  
병합이 이루어짐

```
my_concat_df = pd.concat([df_a, df_b], axis=1)  
my_concat_df
```

	itemname	Sector	주가 _10_13	주가 _10_14
Symbol	df_a		df_b	
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

# pandas 고급: **groupby()** 그룹으로 묶어서 Aggregation 하기

## ■ groupby(by=집계대상, axis=집계축)

- 특정 인덱스나 컬럼의 값 별로 그룹핑하여 Aggregation 합니다 (axis는 생략 가능, 기본 값은 index 행 방향 집계)
- 아래의 예제는 by='Sector'로, 종목의 업종 별로 집계를 수행한 것을 볼 수 있습니다

	itemname	Sector	주가 _10_13	주가 _10_14
Symbol				
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Aggregation

```
my_concat_df.groupby('Sector').sum()
```

	주가_10_13	주가_10_14
Sector		
내구_소비재_및_의류	10850.0	10800.0
보험	13700.0	13550.0
소재	64715.0	63625.0
운송	186500.0	180500.0
음식료_및_담배	56000.0	55750.0
자동차_및_부품	850.0	874.0
자본재	125800.0	124300.0
제약_및_바이오	105000.0	103700.0

그룹 별 '덧셈'

# pandas 고급: **groupby()** 그룹으로 묶어서 Aggregation 하기

## ■ groupby(by=집계대상, axis=집계축)

- 특정 인덱스나 컬럼의 값 별로 그룹핑하여 Aggregation 합니다 (axis는 생략 가능, 기본 값은 index 행 방향 집계)
- 아래의 예제는 by='Sector'로, 종목의 업종 별로 집계를 수행한 것을 볼 수 있습니다

	itemname	Sector	주가 _10_13	주가 _10_14
Symbol				
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Aggregation

```
my_concat_df.groupby('Sector').mean()
```

	주가_10_13	주가_10_14
Sector		
내구_소비재_및_의류	10850.0	10800.000000
보험	13700.0	13550.000000
소재	32357.5	31812.500000
운송	186500.0	180500.000000
음식료_및_담배	28000.0	27875.000000
자동차_및_부품	850.0	874.000000
자본재	62900.0	62150.000000
제약_및_바이오	35000.0	34566.666667

그룹 별 '평균'

# pandas 고급: `groupby()` 그룹으로 묶어서 Aggregation 하기

- (Aggregation 예시) 그룹 별 '값 개수 세기'

Symbol	itemname	Sector	주가 _10_13	주가 _10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Aggregation

```
my_concat_df.groupby('Sector').count()
```

	itemname	주가_10_13	주가_10_14
Sector			
내구_소비재_및_의류	1	1	1
보험	1	1	1
소재	2	2	2
운송	1	1	1
음식료_및_담배	2	2	2
자동차_및_부품	1	1	1
자본재	2	2	2
제약_및_바이오	3	3	3



# pandas 고급: `groupby()` 그룹으로 묶어서 Aggregation 하기

- (Aggregation 예시) 그룹 별 '최댓값'

Symbol	itemname	Sector	주가 _10_13	주가 _10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Aggregation

```
my_concat_df.groupby('Sector').max()
```

itemname	주가_10_13	주가_10_14
Sector		
내구_소비재_및_의류	경방	10850.0 10800.0
보험	메리츠화재	13700.0 13550.0
소재	성창기업지주	63000.0 61800.0
운송	CJ대한통운	186500.0 180500.0
음식료_및_담배	하이트진로홀딩스	38700.0 38500.0
자동차_및_부품	KR모터스	850.0 874.0
자본재	두산	78300.0 76600.0
제약_및_바이오	유한양행	64100.0 63600.0

# pandas 고급: `groupby()` 그룹으로 묶어서 Aggregation 하기

- (Aggregation 예시) 그룹 별 '최댓값'

Symbol	itemname	Sector	주가 _10_13	주가 _10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Aggregation

```
my_concat_df.groupby('Sector').max()
```

itemname	주가_10_13	주가_10_14
Sector		
내구_소비재_및_의류	경방	10850.0 10800.0
보험	메리츠화재	13700.0 13550.0
소재	성창기업지주	63000.0 61800.0
운송	CJ대한통운	186500.0 180500.0
음식료_및_담배	하이트진로홀딩스	38700.0 38500.0
자동차_및_부품	KR모터스	850.0 874.0
자본재	두산	78300.0 76600.0
제약_및_바이오	유한양행	64100.0 63600.0

# pandas 고급: query() DataFrame 필터링하기

## ▪ query(작성한\_쿼리문)

- dataframe의 컬럼을 대상, 작성한 쿼리문으로 dataframe을 필터링하여 추출합니다
- 컬럼은 큰 따옴표 없이 작성하며, 값을 작성할 경우 숫자는 그대로 작성, 문자열은 “ ” 큰 따옴표로 묶어줍니다.
- 쿼리 문은 and, or 등으로 여러 개를 중첩할 수 있습니다

	itemname	Sector	주가 _10_13	주가 _10_14
Symbol				
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

“운송” 섹터인 종목

```
my_concat_df.query('Sector == "운송"')
```

	itemname	Sector	주가_10_13	주가_10_14
Symbol				
A000120	CJ대한통운	운송	186500.0	180500.0

Sector의 값으로  
필터링하는 쿼리문

# pandas 고급: query() DataFrame 필터링하기

## ■ query(작성한\_쿼리문)

- dataframe의 컬럼을 대상, 작성한 쿼리문으로 dataframe을 필터링하여 추출합니다
- 컬럼은 큰 따옴표 없이 작성하며, 값을 작성할 경우 숫자는 그대로 작성, 문자열은 “ ” 큰 따옴표로 묶어줍니다.
- 쿼리 문은 **and, or** 등으로 여러 개를 중첩할 수 있습니다

	itemname	Sector	주가 _10_13	주가 _10_14
Symbol				
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로를 딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성창기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

“제약\_및\_바이오” 섹터인 종목

```
my_concat_df.query('Sector == "제약_및_바이오"')
```

	itemname	Sector	주가_10_13	주가_10_14
Symbol				
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

Sector의 값으로  
필터링하는 쿼리문

# pandas 고급: query() DataFrame 필터링하기

- (query문 작성 예시) 특정 일자의 주가를 기준으로 필터링

Symbol	itemname	Sector	주가_10_13	주가_10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성장기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

```
my_concat_df.query('주가_10_13 > 20000')
```

Symbol	itemname	Sector	주가_10_13	주가_10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000150	두산	자본재	47500.0	47700.0
A000210	대림산업	자본재	78300.0	76600.0

# pandas 고급: query() DataFrame 필터링하기

- (query문 작성 예시) 주가와 섹터에 대한 필터를 **and**로 동시에 적용하여 필터링

Symbol	itemname	Sector	주가_10_13	주가_10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000030	우리은행	NaN	14800.0	14800.0
A000040	KR모터스	자동차_및_부품	850.0	874.0
A000050	경방	내구_소비재_및_의류	10850.0	10800.0
A000060	메리츠화재	보험	13700.0	13550.0
A000070	삼양홀딩스	소재	63000.0	61800.0
A000080	하이트진로	음식료_및_담배	38700.0	38500.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0
A000120	CJ대한통운	운송	186500.0	180500.0
A000140	하이트진로홀딩스	음식료_및_담배	17300.0	17250.0
A000150	두산	자본재	47500.0	47700.0
A000180	성장기업지주	소재	1715.0	1825.0
A000210	대림산업	자본재	78300.0	76600.0
A000220	유유제약	제약_및_바이오	16600.0	16250.0

```
my_concat_df.query('주가_10_13 > 20000 and '
                    'Sector == "제약_및_바이오"')
```

Symbol	itemname	Sector	주가_10_13	주가_10_14
A000020	동화약품	제약_및_바이오	24300.0	23850.0
A000100	유한양행	제약_및_바이오	64100.0	63600.0

## pandas 고급2: 인덱스를 잡는 `.set_index()`, 인덱스를 푸는 `.reset_index()`

- `set_index()` 함수로 특정 컬럼을 인덱스로 만들 수 있습니다

```
df = pd.DataFrame({'month': [1, 4, 7, 10],
                   'year': [2012, 2014, 2013, 2014],
                   'sale': [55, 40, 84, 31],
                   'code': ['a', 'b', np.nan, np.nan]})
```

df

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	NaN
3	10	2014	31	NaN

```
# year를 인덱스로 잡기
df.set_index('year')
```

	month	sale	code
year			
2012	1	55	a
2014	4	40	b
2013	7	84	NaN
2014	10	31	NaN

## pandas 고급2: 인덱스를 잡는 `.set_index()`, 인덱스를 푸는 `.reset_index()`

- reset\_index() 함수로 이미 잡힌 인덱스를 풀 수 있습니다 (컬럼화)

```
# month를 인덱스로 잡기
df.set_index('month')
```

	year	sale	code
month			
1	2012	55	a
4	2014	40	b
7	2013	84	NaN
10	2014	31	NaN

```
# 멀티 인덱스
df.set_index(['year', 'month'])
```

	year	month	sale	code
year				
2012		1	55	a
2014		4	40	b
2013		7	84	NaN
2014		10	31	NaN

```
# 인덱스 잡아서 새로운 df 만들기
idx_df = df.set_index(['year', 'month'])
idx_df
```

	year	month	sale	code
year				
2012		1	55	a
2014		4	40	b
2013		7	84	NaN
2014		10	31	NaN

```
# .reset_index()로 잡힌 인덱스 풀기
idx_df.reset_index()
```

	year	month	sale	code
year				
0	2012	1	55	a
1	2014	4	40	b
2	2013	7	84	NaN
3	2014	10	31	NaN



## pandas 고급2: NaN 결측치를 채우는 .fillna() 함수

- fillna() 함수로 dataframe 내 빈 값 NaN을 특정 값으로 채울 수 있습니다

```
df
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	NaN
3	10	2014	31	NaN

```
df.fillna('')
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	
3	10	2014	31	

```
df.fillna('a')
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	a
3	10	2014	31	a

```
df.fillna('bbb')
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	bbb
3	10	2014	31	bbb

## pandas 고급2: NaN 결측치를 채우는 .fillna() 함수

- fillna() 함수의 method 인자에 입력을 넣어, 특정 방법으로 채우기를 할 수 있습니다

method 인자의 입력:

- pad / ffill: propagate last valid observation forward to next valid
- backfill / bfill: use next valid observation to fill gap.

▼ # ffill은 이전 관측치로 채우는 방법입니다  
df.fillna(method='ffill', axis=0)

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	b
3	10	2014	31	b

▼ # ffill은 이전 관측치로 채우는 방법입니다  
df.fillna(method='ffill', axis=1)

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	84
3	10	2014	31	31

## pandas 고급2: 특정 row나 column을 지우는 drop 함수

- `.drop(index=[지울 인덱스], columns=[지울 컬럼])` : dataframe의 row나 column을 삭제합니다
- `.dropna(axis=0, how='any')` : dataframe에서 NaN이 포함된 항목을 방법에 따라 삭제합니다

df

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	NaN
3	10	2014	31	NaN

▼ # row를 지울 경우, drop() 함수에 index를 명시  
df.drop(index=[0,3])

	month	year	sale	code
1	4	2014	40	b
2	7	2013	84	NaN

▼ # column을 지울 경우, drop() 함수에 컬럼을 명시  
df.drop(columns=['year'])

	month	sale	code
0	1	55	a
1	4	40	b
2	7	84	NaN
3	10	31	NaN

▼ # dropna() 함수는 nan이 포함된 row나 column을 지웁니다  
df.dropna(axis=0)

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b

## pandas 고급2: 컬럼을 명시하는 같은 문법

- 아래와 같이 다양한 방법으로 컬럼을 명시할 수 있습니다 (모두 같은 결과)
  - 단, 이중 `.month`와 같이 `attribute`로 접근하는 경우, 컬럼명이 파이썬 변수 네이밍 규약을 따라야합니다

```
df
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	NaN
3	10	2014	31	NaN

```
df.loc[:, 'month']
```

```
0    1
1    4
2    7
3   10
Name: month, dtype: int64
```

```
df['month']
```

```
0    1
1    4
2    7
3   10
Name: month, dtype: int64
```

```
df.month
```

```
0    1
1    4
2    7
3   10
Name: month, dtype: int64
```

## pandas 고급2: dataframe을 필터링하는 또다른 방법

- `df[ 특정 조건에 대한 df의 bool 결과 ]`의 형태로, dataframe을 필터링 할 수 있습니다

```
df
```

	month	year	sale	code
0	1	2012	55	a
1	4	2014	40	b
2	7	2013	84	NaN
3	10	2014	31	NaN

```
df.sale < 50
```

```
0    False
1     True
2    False
3     True
Name: sale, dtype: bool
```

```
df[df.sale < 50]
```

	month	year	sale	code
1	4	2014	40	b
3	10	2014	31	NaN

```
df[df['code'].isna()]
```

	month	year	sale	code
2	7	2013	84	NaN
3	10	2014	31	NaN

Q & A



THANK YOU :)