원본 링크 : https://velog.io/@suminwooo/파이썬-Pandas-활용

Pandas 활용

Pandas 란?

- 판다스(Pandas)는 Python에서 DB처럼 테이블 형식의 데이터를 쉽게 처리할 수 있는 라이브러리 입니다.
- 데이터가 테이블 형식(DB Table, csv 등)으로 이루어진 경우가 많아 데이터 분석 시 자주 사용하게 될 Python 패키지입니다.

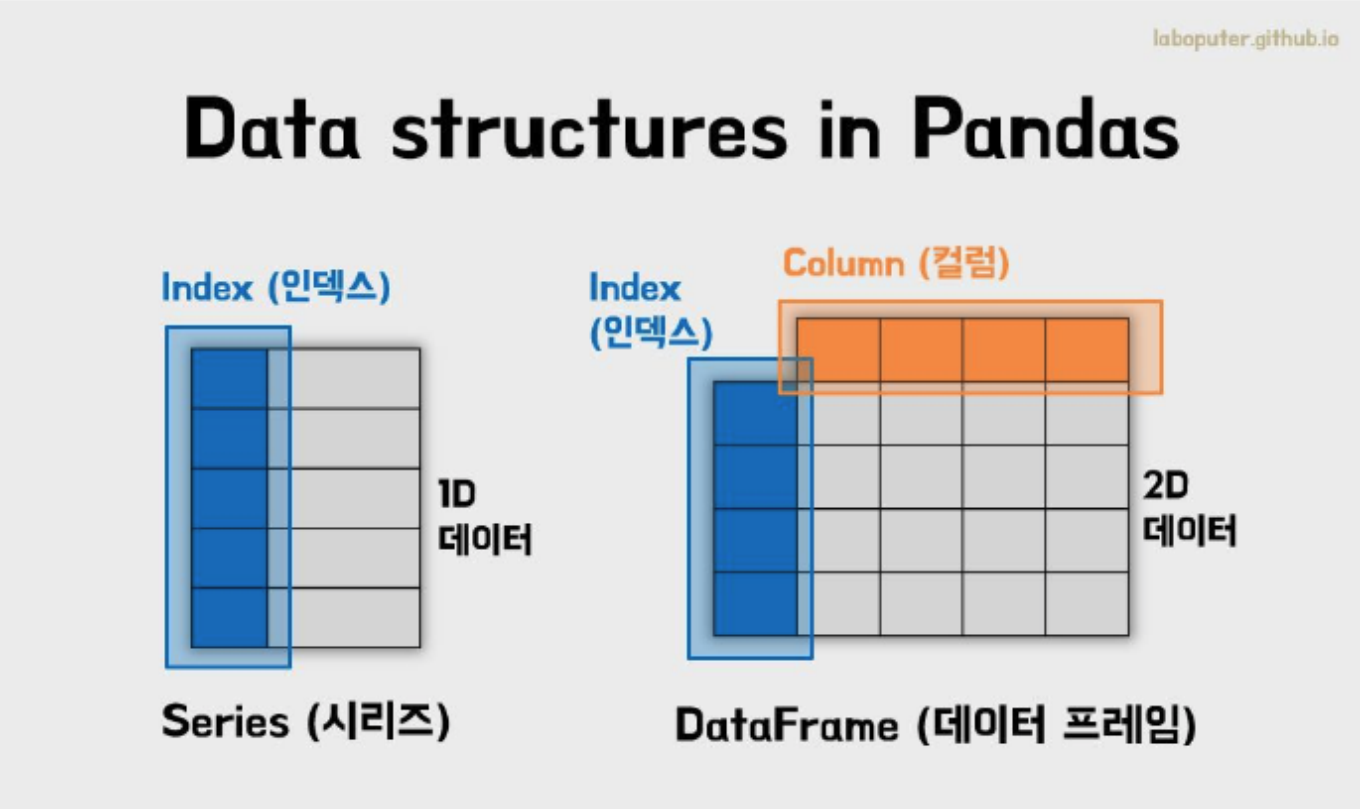
Pandas 사용 방법

- pandas 를 사용하기 위해서 다음과 같이 모듈을 import 합니다.
- 임포트를 할 때에는 pandas 라고 그대로 사용할 수 있지만, pd 라는 축약된 이름을 관례적으로 많이 사용합니다.

```
import pandas as pd # 설치시 pip install pandas
import numpy as np
import matplotlib.pyplot as plt
```

1. 데이터 오브젝트 생성

- 데이터 오브젝트는 '데이터를 담고 있는 그릇'이라고 생각하면 됩니다.
- 아래의 이미지처럼 pandas에서는 2가지 오브젝트 Series 와 DataFrame가 있습니다.
    - Series : 1차원 데이터와 각 데이터의 위치정보를 담는 인덱스로 구성
    - DataFrame : 2차원 데이터와 인덱스, 컬럼으로 구성(하나의 컬럼만 선택한다면 Series)




```
# Series 생성
s = pd.Series([1, 3, 5, np.nan, 6, 8])

# 0    1.0
# 1    3.0
# 2    5.0
# 3    NaN
# 4    6.0
# 5    8.0
# dtype: float64
```

- 위와 같이 Series() 안에 list로 1차원 데이터만 넘기면 됩니다. index는 입력하지 않아도 자동으로 0부터 입력됩니다.

```
# DataFrame 생성
dates = pd.date_range('20130101', periods=6)
# DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
#                '2013-01-05', '2013-01-06'],
#               dtype='datetime64[ns]', freq='D')

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

#                    A         B         C         D
# 2013-01-01  1.571507  0.160021 -0.015071 -0.118588
# 2013-01-02 -1.037697 -0.891196  0.495447  0.453095
# 2013-01-03 -1.682384 -0.026006 -0.152957 -0.212614
# 2013-01-04 -0.108757 -0.958267  0.407331  0.187037
```

```
# 2013-01-05  1.092380  2.841777 -0.125714 -0.760722
# 2013-01-06  1.638509 -0.601126 -1.043931 -1.330950
```

- pandas의 경우 리스트 이외에 딕셔너리 형식으로도 DataFrame을 만들 수 있습니다.
- 이 때에는 dict 의 key 값이 열을 정의하는 컬럼이 되며, 행을 정의하는 인덱스는 자동으로 0부터 시작하여 1씩 증가하는 정수 인덱스가 사용됩니다.

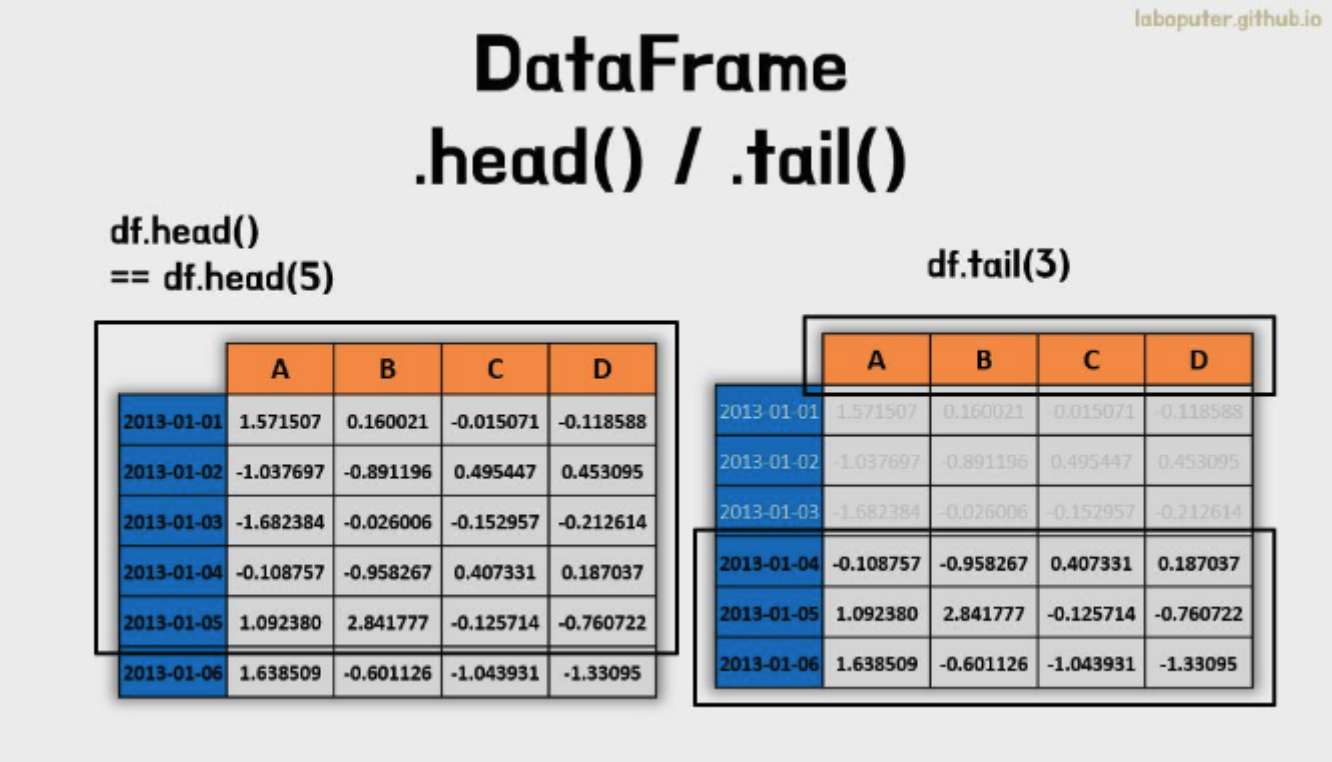
```
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20130102'),
                    'C': pd.Series(1, index=list(range(4)), dtype='float32'),
                    'D': np.array([3]*4, dtype='int32'),
                    'E': pd.Categorical(['test', 'train', 'test', 'train']),
                    'F': 'foo'})
#      A          B    C  D      E    F
# 0  1.0 2013-01-02  1.0  3   test  foo
# 1  1.0 2013-01-02  1.0  3  train  foo
# 2  1.0 2013-01-02  1.0  3   test  foo
# 3  1.0 2013-01-02  1.0  3  train  foo
```

- DataFrame의 .dtypes라는 값에는 각 컬럼이 어떤 데이터 형식인지가 저장되어 있습니다. 만약 섞여있을 경우 object가 됩니다.

```
df2.dtypes
# A           float64
# B     datetime64[ns]
# C            float32
# D              int32
# E           category
# F             object
# dtype: object
```

2. 데이터 확인하기

- DataFrame은 head(), tail()의 함수로 처음과 끝의 일부 데이터를 볼 수 있습니다.
- 데이터가 큰 경우에 데이터가 어떤식으로 구성되어 있는지 확인할 때 자주 사용합니다.



	A	B	C	D
2013-01-01	1.571507	0.160021	-0.015071	-0.118588
2013-01-02	-1.037697	-0.891196	0.495447	0.453095
2013-01-03	-1.682384	-0.026006	-0.152957	-0.212614
2013-01-04	-0.108757	-0.958267	0.407331	0.187037
2013-01-05	1.092380	2.841777	-0.125714	-0.760722
2013-01-06	1.638509	-0.601126	-1.043931	-1.33095

	A	B	C	D
2013-01-04	-0.108757	-0.958267	0.407331	0.187037
2013-01-05	1.092380	2.841777	-0.125714	-0.760722
2013-01-06	1.638509	-0.601126	-1.043931	-1.33095

jupyter Notebook 사용 팁

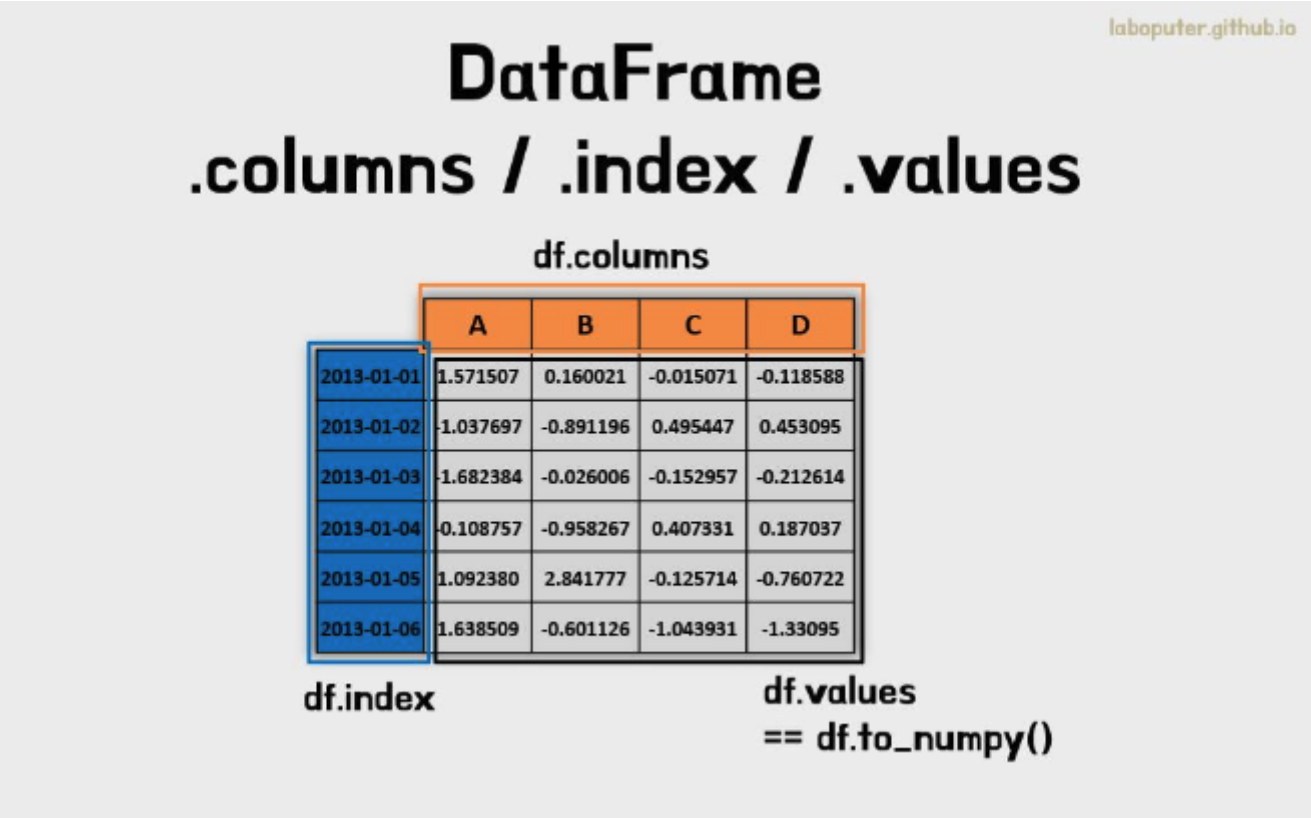
- head()와 같은 함수를 사용할 경우 jupyter notebook에서 함수의 기본값을 확인할 수 있습니다.
- shift+tab을 누를 경우 아래의 내용을 확인 할 수 있습니다. 이미지처럼 n: 'int' = 5 로 설정이 되어 있기 때문에 디폴트 값이 5입니다. 위의 이미지처럼 3개만 넣고 싶다면 3을 입력해줄 수 있습니다.

```
Signature: data.head(n: 'int' = 5) -> 'FrameOrSeries'
Docstring:
Return the first `n` rows.

This function returns the first `n` rows for the object based
on position. It is useful for quickly testing if your object
has the right type of data in it.

For negative values of `n`, this function returns all rows except
the last `n` rows, equivalent to ``df[:-n]``.
```

- DataFrame에서 인덱스를 확인하고 싶을 경우에는 .index, 컬럼은 .columns, 내부 데이터는 .values 속성을 통해 확인할 수 있습니다.

```
df.index
# DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
#                '2013-01-05', '2013-01-06'],
#               dtype='datetime64[ns]', freq='D')

df.columns
# Index(['A', 'B', 'C', 'D'], dtype='object')

df.values
# [[ 1.571507  0.160021 -0.015071 -0.118588]
```

```
#  [-1.037697 -0.891196  0.495447  0.453095]
#  [-1.682384 -0.026006 -0.152957 -0.212614]
#  [-0.108757 -0.958267  0.407331  0.187037]
#  [ 1.09238   2.841777 -0.125714 -0.760722]
#  [ 1.638509 -0.601126 -1.043931 -1.33095 ]]
```



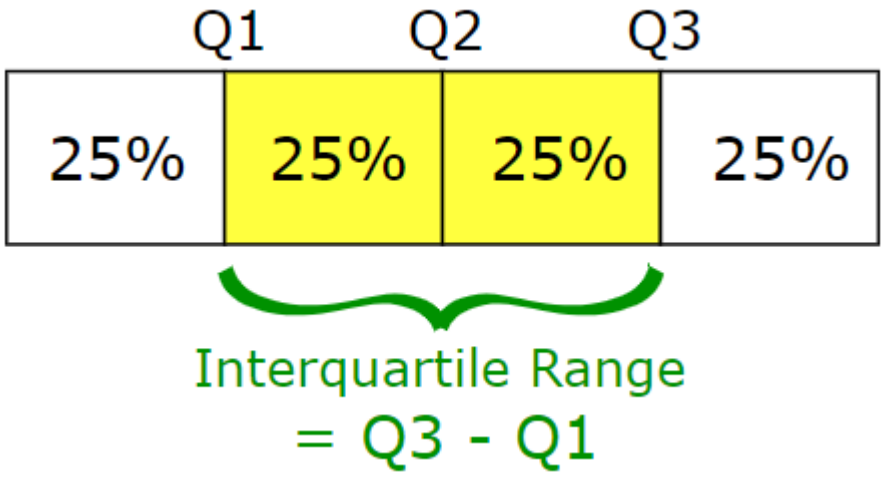
- DataFrame의 describe()를 통해 각 컬럼의 통계적인 수치를 확인할 수 있습니다.

1. count: 데이터 개수
2. mean: 평균값
3. std: 표준편차
4. min: 최소값
5. 25%: 1사분위값
6. 50%: 중앙값
7. 75%: 3사분위값
8. max: 최대값

```
df.describe()
#               A         B         C         D
# count  6.000000  6.000000  6.000000  6.000000
# mean   0.245593  0.087534 -0.072482 -0.297124
# std    1.407466  1.423367  0.549378  0.651149
# min   -1.682384 -0.958267 -1.043931 -1.330950
# 25%   -0.805462 -0.818679 -0.146146 -0.623695
# 50%    0.491811 -0.313566 -0.070392 -0.165601
# 75%    1.451725  0.113514  0.301730  0.110631
# max    1.638509  2.841777  0.495447  0.453095
```

분위수란?

1. 1사분위값 : 누적 확률이 0.25가 되는 곳의 확률 변호
2. 2사분위값 : 누적 확률이 0.5가 되는 곳의 확률 변호
3. 3사분위값 : 누적 확률이 0.75가 되는 곳의 확률 변호



- .T 속성은 DataFrame 에서 index 와 column 을 바꾼 형태의 DataFrame 입니다.

```
df.T
#    2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
# A    1.571507   -1.037697   -1.682384   -0.108757    1.092380    1.638509
# B    0.160021   -0.891196   -0.026006   -0.958267    2.841777   -0.601126
# C   -0.015071    0.495447   -0.152957    0.407331   -0.125714   -1.043931
# D   -0.118588    0.453095   -0.212614    0.187037   -0.760722   -1.330950
```

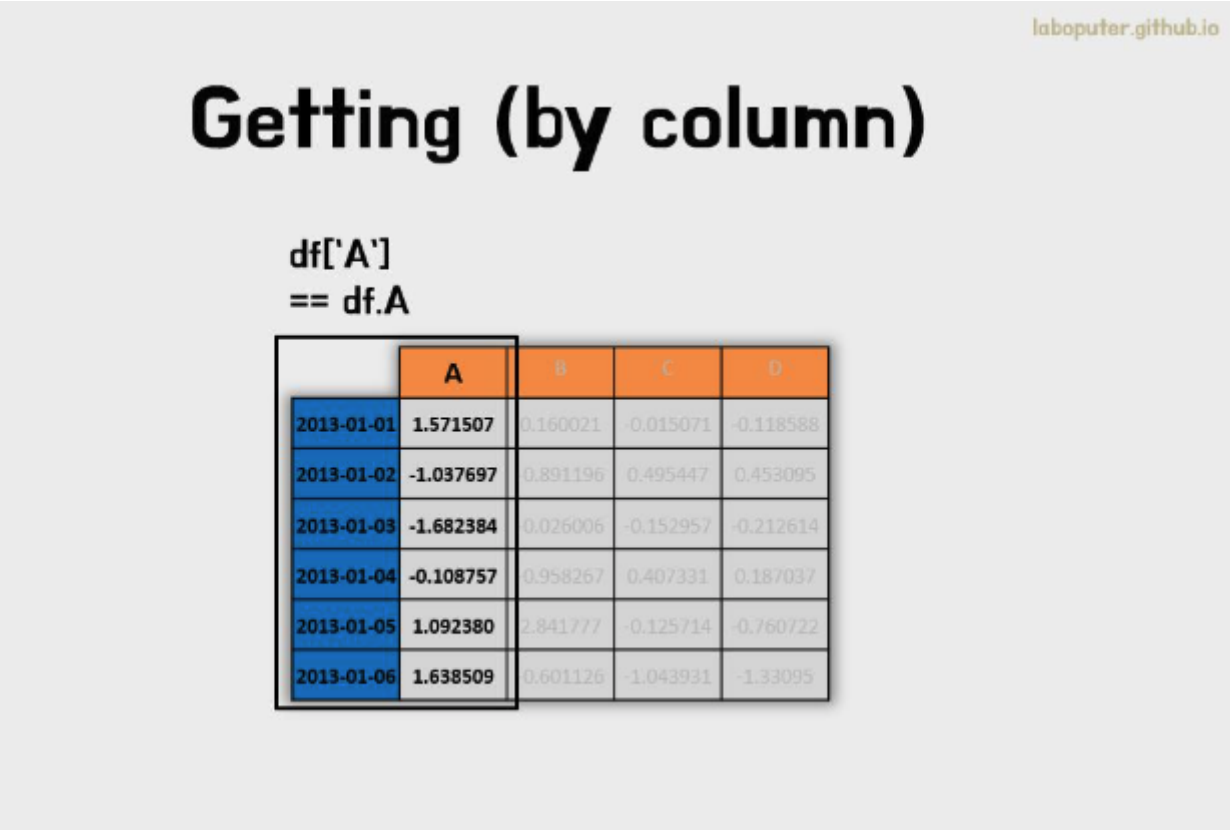
- .sort_index() 라는 메소드를 활용해 행과 열 이름을 정렬하여 나타낼 수도 있습니다.
  - axis: 축 기준 정보 (0: 인덱스 기준, 1: 컬럼 기준)
  - ascending: 정렬 방식 (false : 내림차순, true: 오름차순)

```
df.sort_index(axis=1, ascending=False)
#                    D         C         B         A
# 2013-01-01 -1.135632 -1.509059 -0.282863  0.469112
# 2013-01-02 -1.044236  0.119209 -0.173215  1.212112
# 2013-01-03  1.071804 -0.494929 -2.104569 -0.861849
# 2013-01-04  0.271860 -1.039575 -0.706771  0.721555
# 2013-01-05 -1.087401  0.276232  0.567020 -0.424972
# 2013-01-06  0.524988 -1.478427  0.113648 -0.673690

df.sort_values(by='B')
#                    A         B         C         D
# 2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
# 2013-01-04  0.721555 -0.706771 -1.039575  0.271860
# 2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
# 2013-01-02  1.212112 -0.173215  0.119209 -1.044236
# 2013-01-06 -0.673690  0.113648 -1.478427  0.524988
# 2013-01-05 -0.424972  0.567020  0.276232 -1.087401
```

3. 데이터 선택하기

- 데이터프레임 자체가 갖고 있는 인덱싱&슬라이싱 기능을 이용할 수 있습니다.
- 특정 컬럼의 값들만 가져오고 싶다면 df['A'](df.A와 동일)와 같은 형태로 입력합니다. 리턴되는 값은 Series의 자료구조를 갖고 있습니다.
- 단, 컬럼의 이름이 간혹 df.A로 쓰면 에러가 나는 경우가 발생하기 때문에 df['A']를 추천합니다.

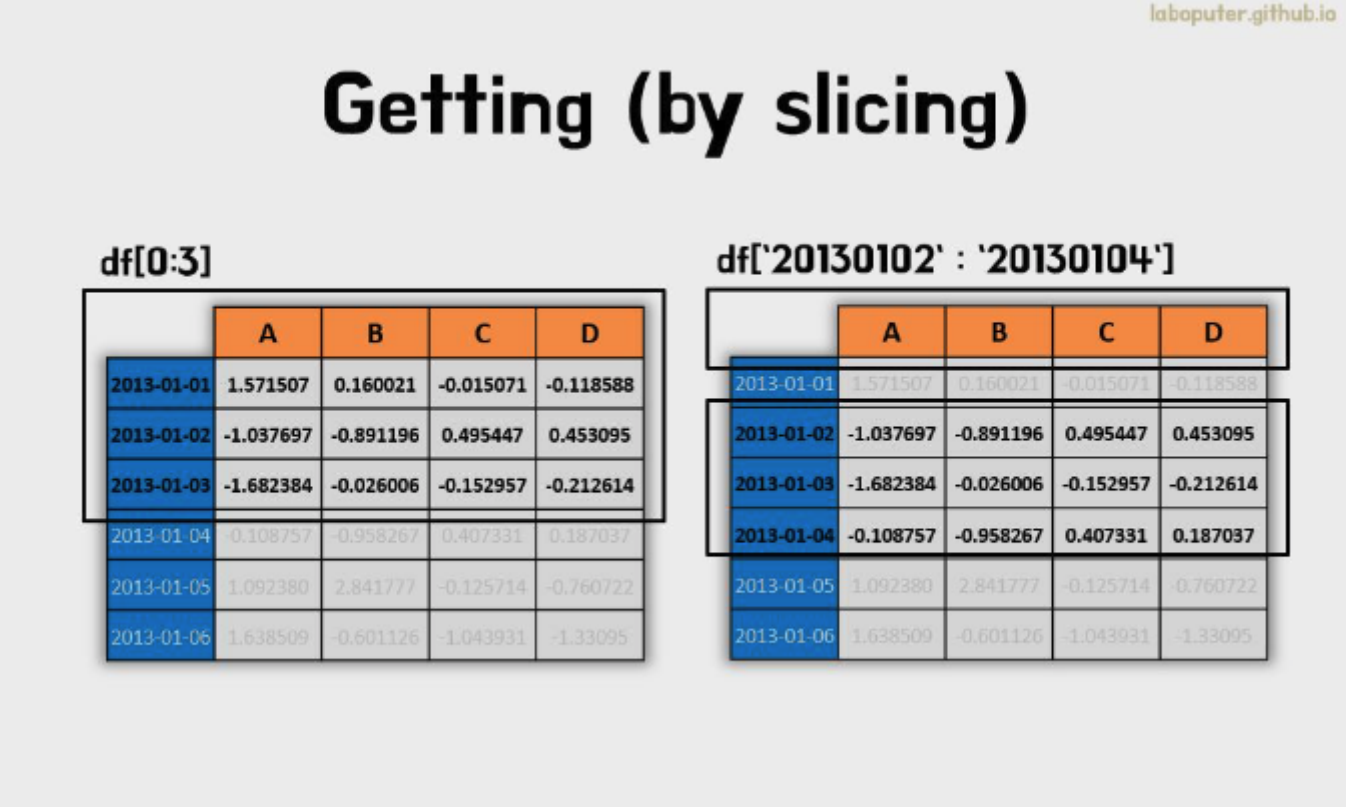
```
df['A']
# 2013-01-01    0.469112
# 2013-01-02    1.212112
# 2013-01-03   -0.861849
# 2013-01-04    0.721555
# 2013-01-05   -0.424972
# 2013-01-06   -0.673690
# req: D, Name: A, dtype: float64

type(df['A'])
# <class 'pandas.core.series.Series'>
```



- 특정 '행 범위'를 가져오고 싶다면 다음과 같이 리스트를 슬라이싱 할 때와 같이 동일하게 사용할 수 있습니다.
- df[0:3] 라고 하면 0, 1, 2번째 행을 가져옵니다.
- 또 다른 방법으로 df['20130102':'20130104'] 인덱스명을 직접 넣어서 해당하는 행 범위를 가져올 수도 있습니다.

- 파이썬에서 슬라이싱을 할 경우 경우에 따라 마지막 값이 포함되거나 포함되지 않을 수 있습니다. 다양한 경우를 외우기 보단 다양한 테스트를 통해 확인하는 방법을 추천합니다.

```
## 맨 처음 3개의 행
df[0:3]
#                   A         B         C         D
# 2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
# 2013-01-02  1.212112 -0.173215  0.119209 -1.044236
# 2013-01-03 -0.861849 -2.104569 -0.494929  1.071804

## 인덱스명에 해당하는 값
df['20130102':'20130104']
#                   A         B         C         D
# 2013-01-02  1.212112 -0.173215  0.119209 -1.044236
# 2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
# 2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

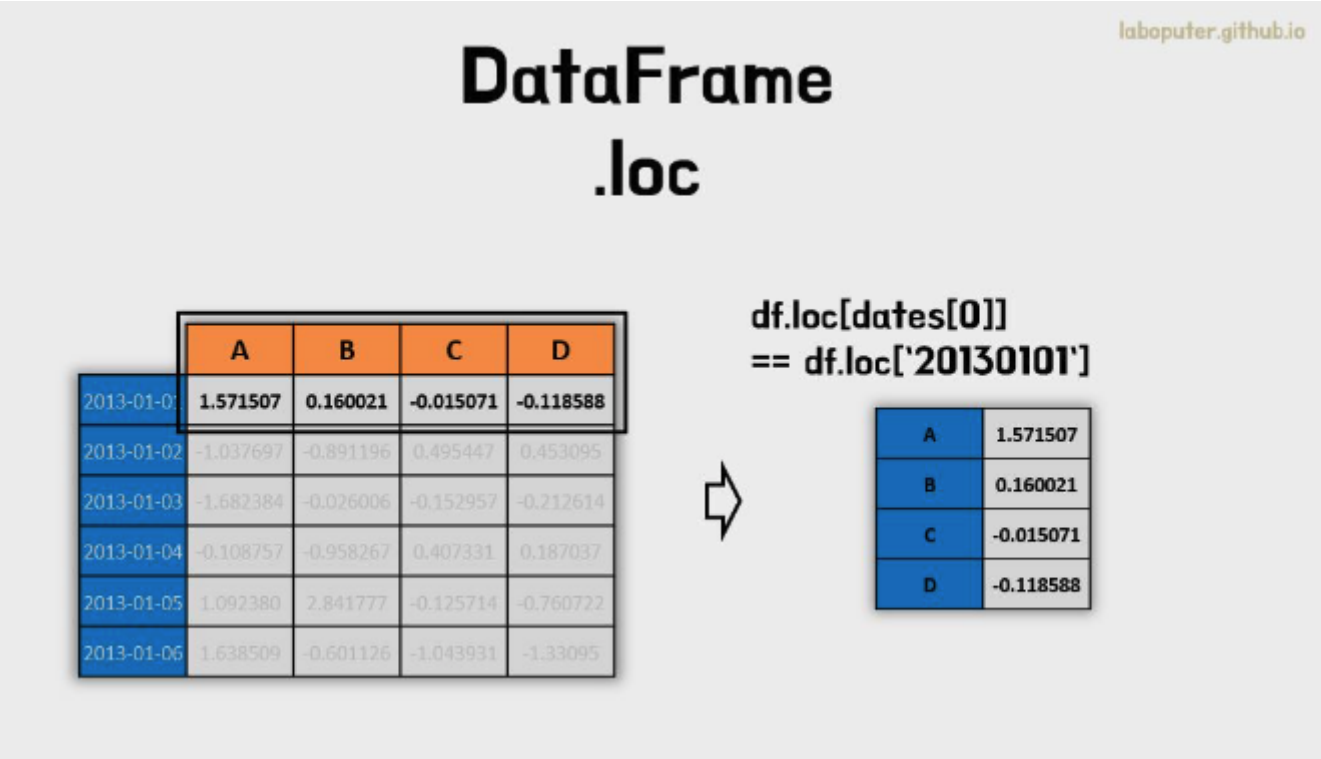


이름을 이용하여 선택하기: .loc

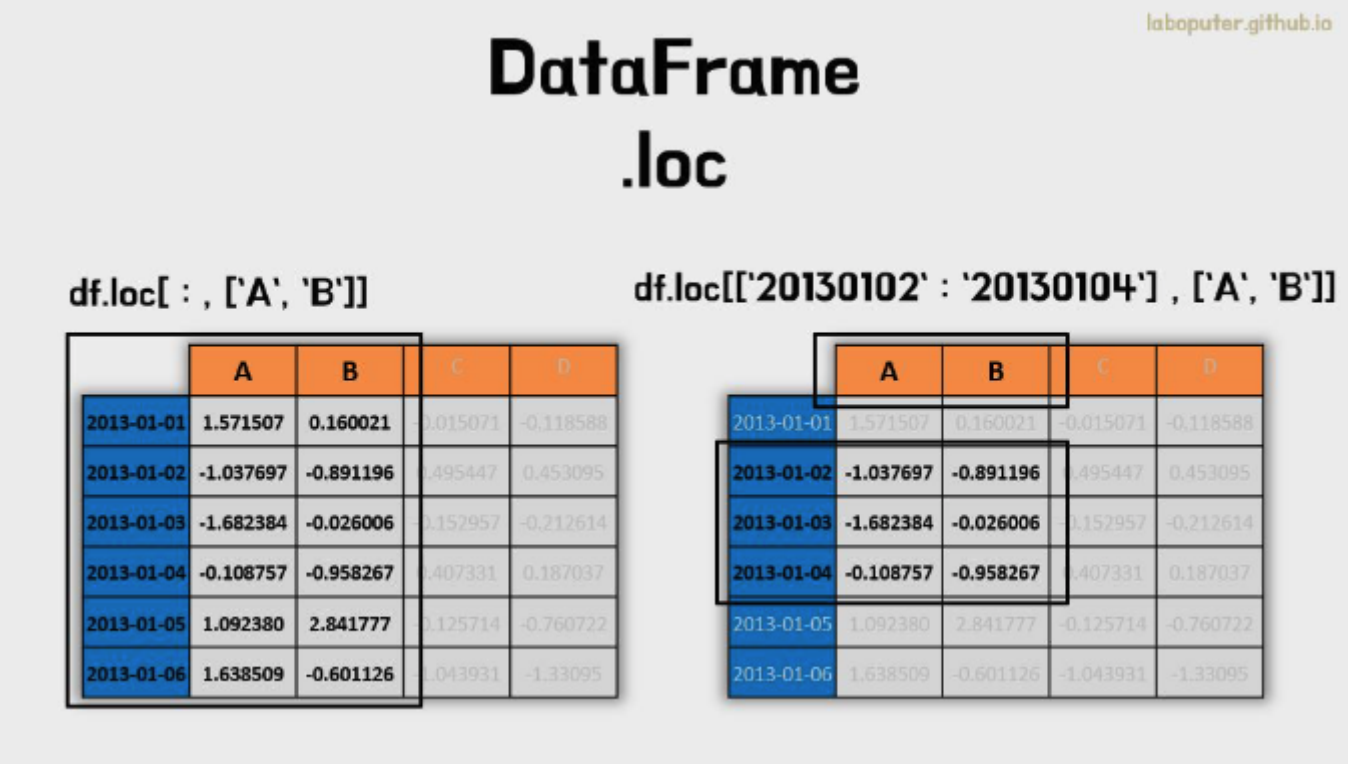
- 이름(Label)로 가져오는 것은 DataFrame의 .loc 속성을 이용합니다.
- .loc은 2차원으로 구성되어 있습니다. .loc[인덱스명, 컬럼명] 형식으로 접근가능 합니다.
- .loc[인덱스명]으로 입력하면 모든 행의 값으로 결과가 나옵니다. 여기에서는 .loc[인덱스명, :] 과 동일한 의미이며, :의 경우 모든 값을 의미합니다.
- .loc[선택 인덱스 리스트, 선택 컬럼 리스트]와 같은 리스트 형식으로 멀티인덱싱이 가능합니다.

```
df.loc[dates[0]]
# A    1.571507
```

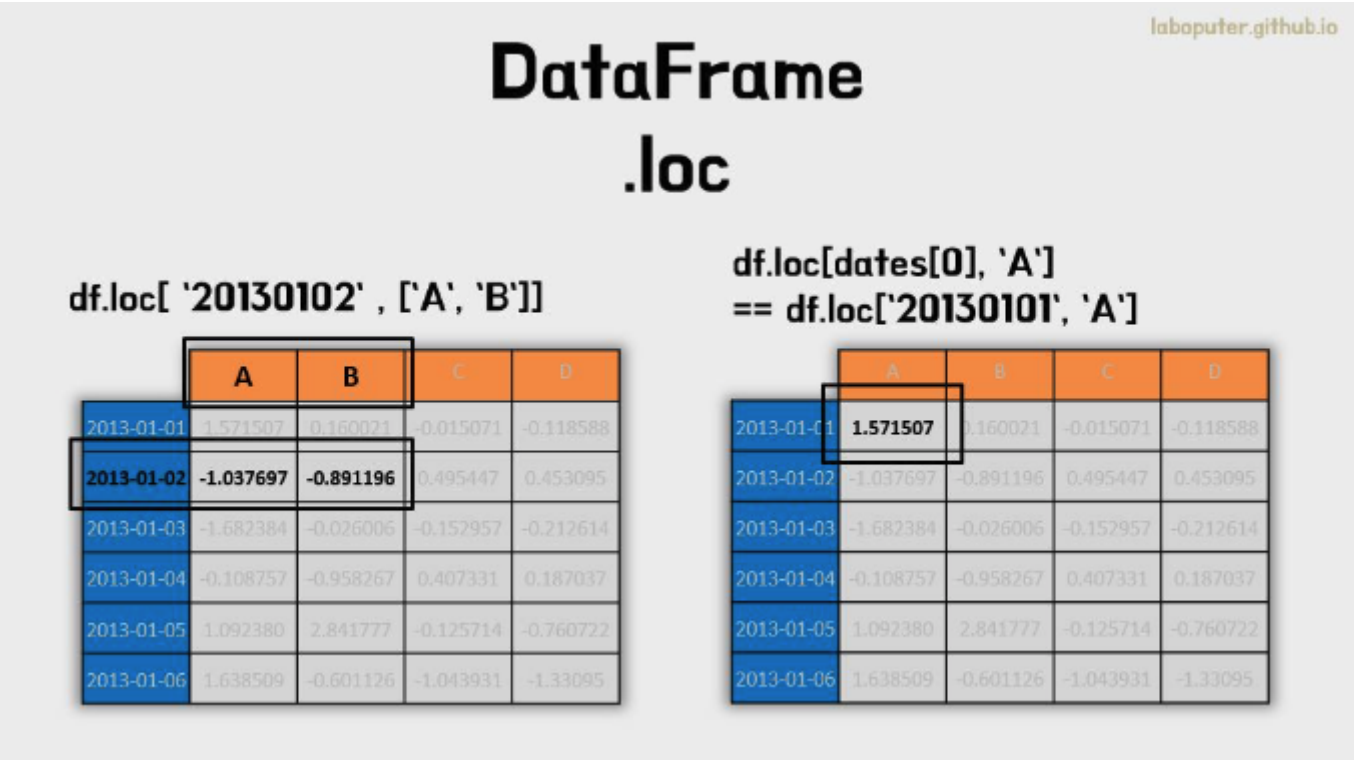
```
# B    0.160021
# C   -0.015071
# D   -0.118588
# Name: 2013-01-01 00:00:00, dtype: float64
```




```
df.loc[:,['A','B']]
#                    A         B
# 2013-01-01  1.571507  0.160021
# 2013-01-02 -1.037697 -0.891196
# 2013-01-03 -1.682384 -0.026006
# 2013-01-04 -0.108757 -0.958267
# 2013-01-05  1.092380  2.841777
# 2013-01-06  1.638509 -0.601126

df.loc['20130102':'20130104',['A','B']]
#                    A         B
# 2013-01-02 -1.037697 -0.891196
# 2013-01-03 -1.682384 -0.026006
# 2013-01-04 -0.108757 -0.958267
```



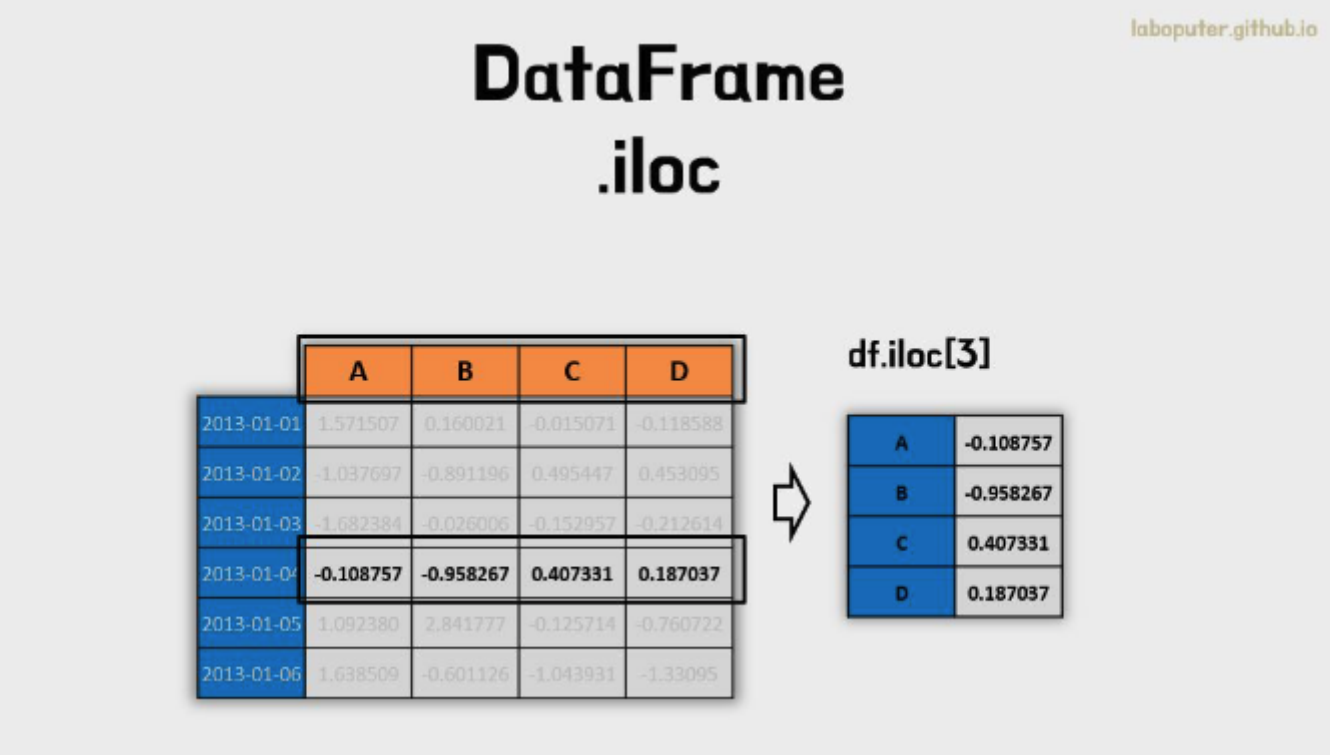

```
df.loc['20130102',['A','B']]
# A   -1.037697
# B   -0.891196
# Name: 2013-01-02 00:00:00, dtype: float64

df.loc[dates[0],'A']
# 1.571506676720408
```



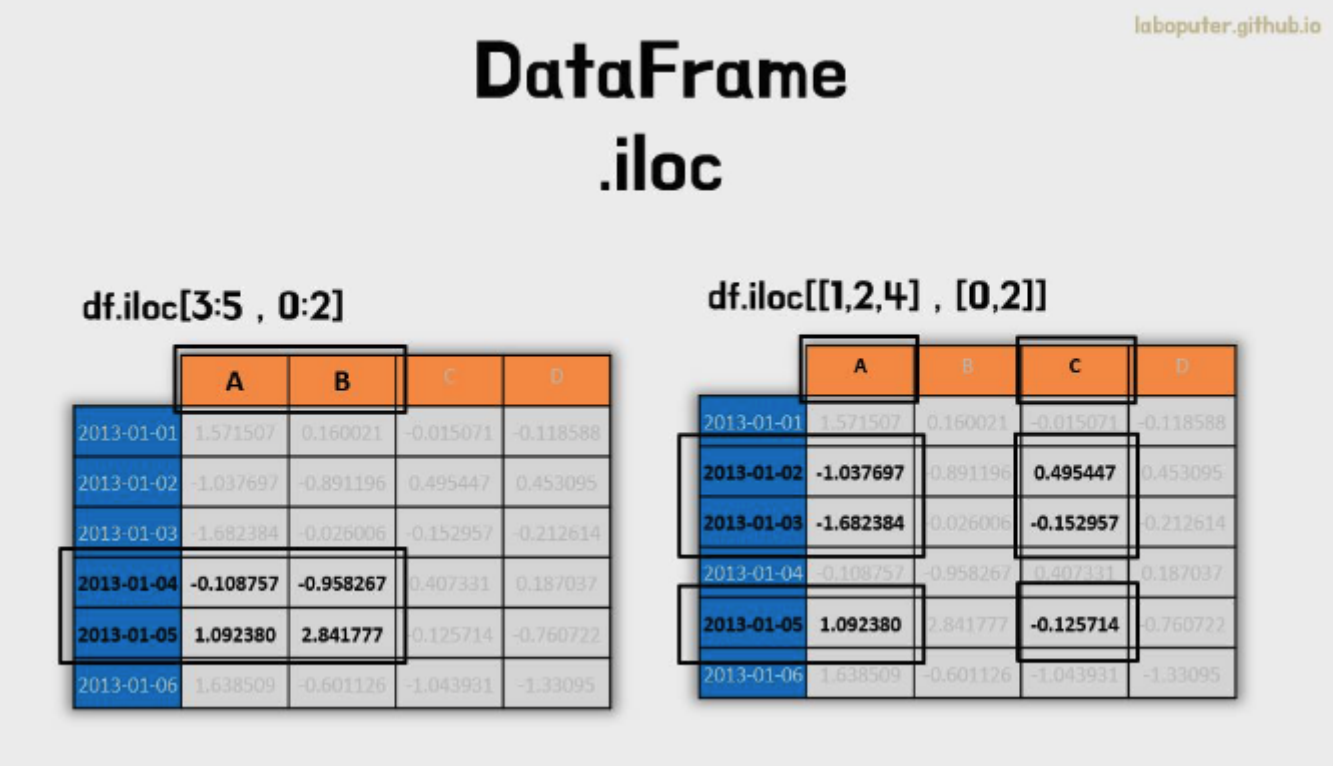
인덱스로 데이터 가져오기: .iloc

- 여기서 말하는 인덱스는 위치(숫자) 정보를 말합니다.
- .iloc도 .loc와 마찬가지로 2차원 형태로 구성되어 있어 1번째 인덱스는 행의 번호를, 2번째 인덱스는 컬럼의 번호를 의미합니다. 마찬가지로 멀티인덱싱도 가능합니다.

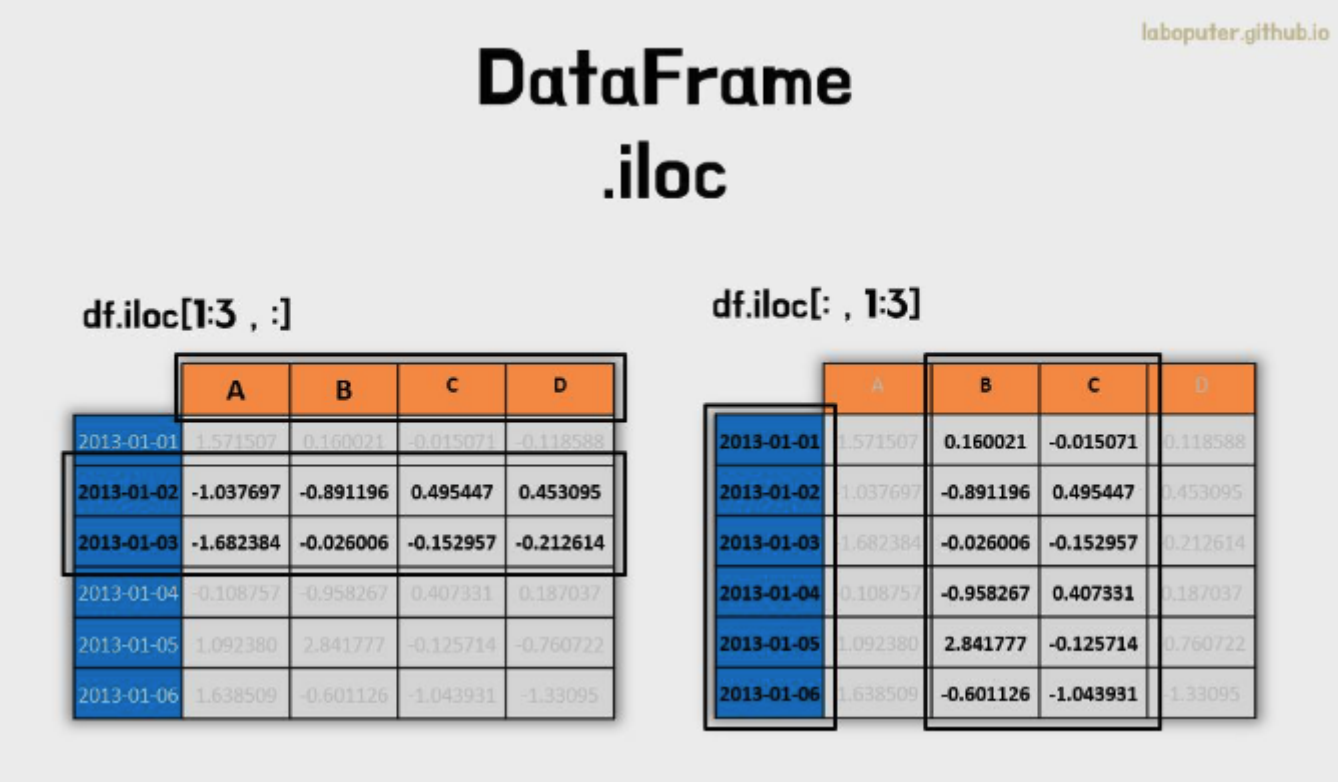
```
df.iloc[3]
# A   -0.108757
# B   -0.958267
# C    0.407331
# D    0.187037
# Name: 2013-01-04 00:00:00, dtype: float64
```




```
df.iloc[3:5,0:2]
#                    A         B
# 2013-01-04 -0.108757 -0.958267
# 2013-01-05  1.092380  2.841777


df.iloc[[1,2,4],[0,2]]
#                    A         C
# 2013-01-02 -1.037697  0.495447
# 2013-01-03 -1.682384 -0.152957
# 2013-01-05  1.092380 -0.125714
```




```
df.iloc[1:3,:]
#                    A         B         C         D
# 2013-01-02 -1.037697 -0.891196  0.495447  0.453095
# 2013-01-03 -1.682384 -0.026006 -0.152957 -0.212614

df.iloc[:,1:3]
#                    B         C
# 2013-01-01  0.160021 -0.015071
# 2013-01-02 -0.891196  0.495447
# 2013-01-03 -0.026006 -0.152957
# 2013-01-04 -0.958267  0.407331
# 2013-01-05  2.841777 -0.125714
# 2013-01-06 -0.601126 -1.043931
```

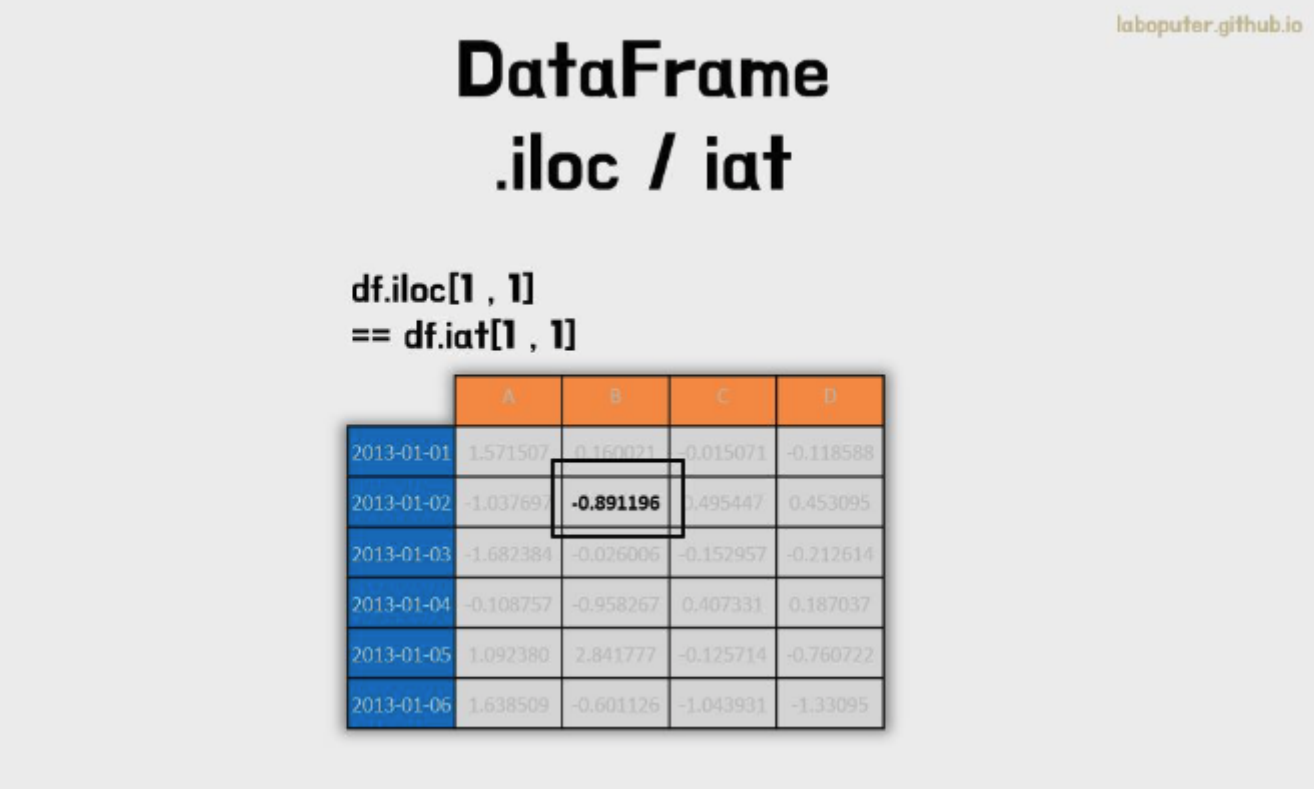


	A	B	C	D
2013-01-01	1.571507	0.160021	-0.015071	-0.118588
2013-01-02	-1.037697	-0.891196	0.495447	0.453095
2013-01-03	-1.682384	-0.026006	-0.152957	-0.212614
2013-01-04	-0.108757	-0.958267	0.407331	0.187037
2013-01-05	1.092380	2.841777	-0.125714	-0.760722
2013-01-06	1.638509	-0.601126	-1.043931	-1.33095

	A	B	C	D
2013-01-01	1.571507	0.160021	-0.015071	0.118588
2013-01-02	1.037697	-0.891196	0.495447	0.453095
2013-01-03	1.682384	-0.026006	-0.152957	0.212614
2013-01-04	0.108757	-0.958267	0.407331	0.187037
2013-01-05	1.092380	2.841777	-0.125714	0.760722
2013-01-06	1.638509	-0.601126	-1.043931	1.33095

- .iat의 경우 .iloc와 동일한 값을 가져 올 수 있습니다. 동일하지만 스칼라값을 가져오는 속도가 .iat이 빠릅니다.
- 헷갈릴 경우 하나만 써도 큰 문제는 없습니다.

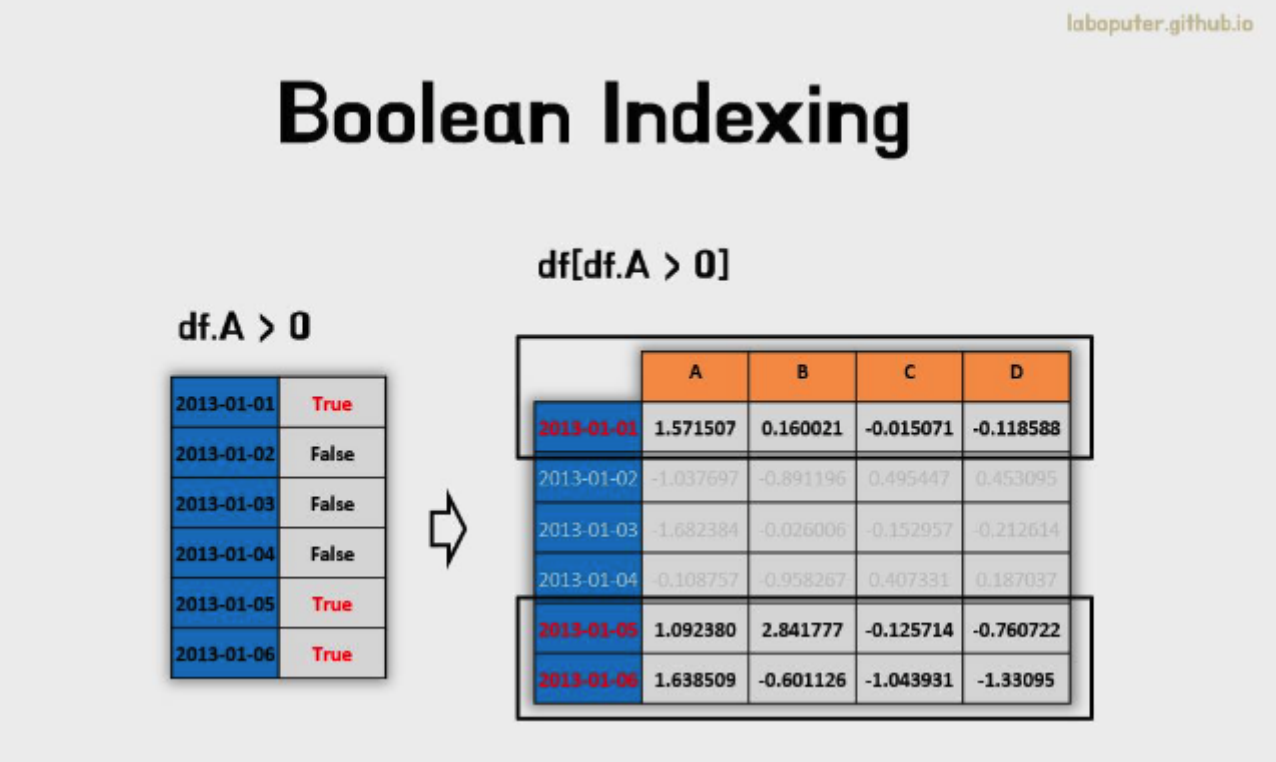
```
df.iloc[1,1]
# -0.89119558600132898

df.iat[1,1]
# 0.89119558600132898
```



조건으로 가져오기

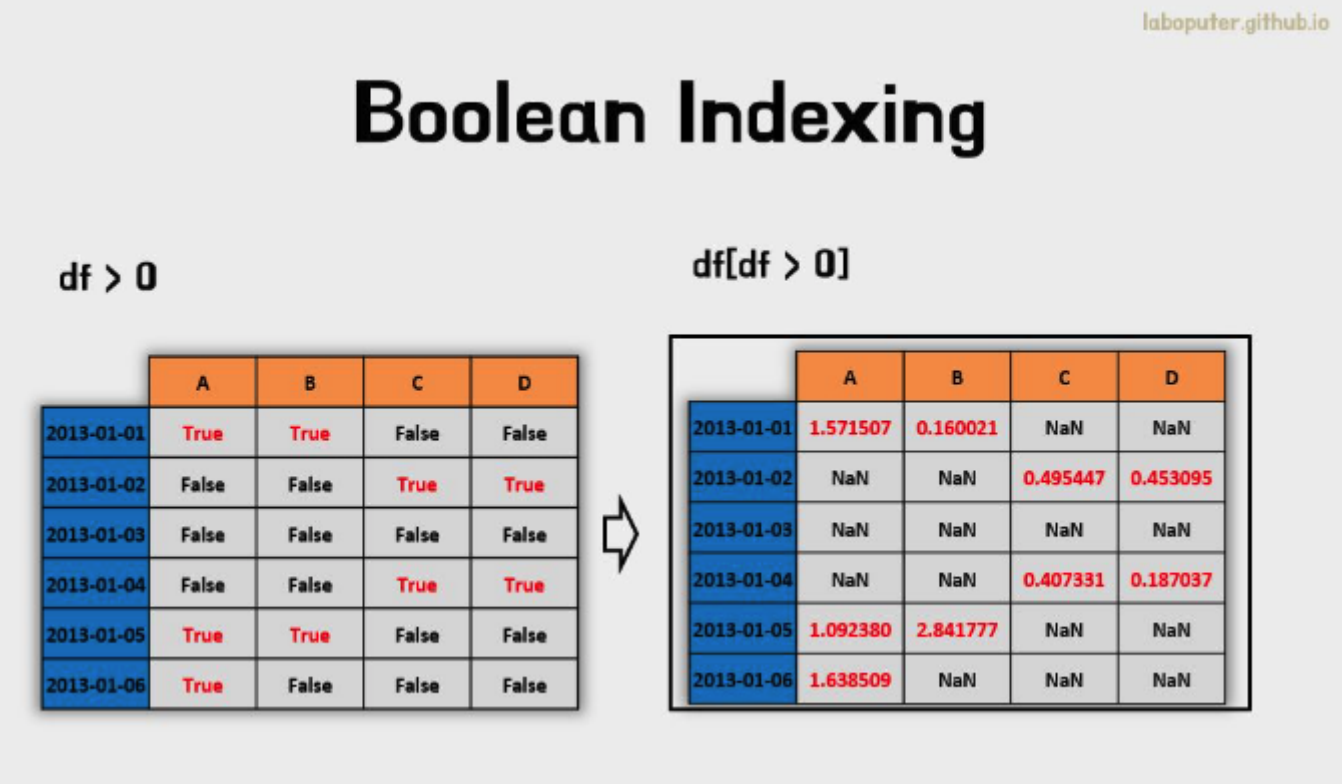
- 하나의 컬럼의 다양한 조건에 따라 행들을 선택할 수 있습니다.

```
df[df['A'] > 0]
#                    A         B         C         D
# 2013-01-01  1.571507  0.160021 -0.015071 -0.118588
# 2013-01-05  1.092380  2.841777 -0.125714 -0.760722
# 2013-01-06  1.638509 -0.601126 -1.043931 -1.330950
```



- DataFrame의 값 조건에 해당하는 것만 선택할 수도 있습니다.

```
df[df > 0]
#                    A         B         C         D
# 2013-01-01  1.571507  0.160021       NaN       NaN
# 2013-01-02       NaN       NaN  0.495447  0.453095
# 2013-01-03       NaN       NaN       NaN       NaN
# 2013-01-04       NaN       NaN  0.407331  0.187037
# 2013-01-05  1.092380  2.841777       NaN       NaN
# 2013-01-06  1.638509       NaN       NaN       NaN
```



- isin()을 이용하여 필터링을 할 수 있습니다.

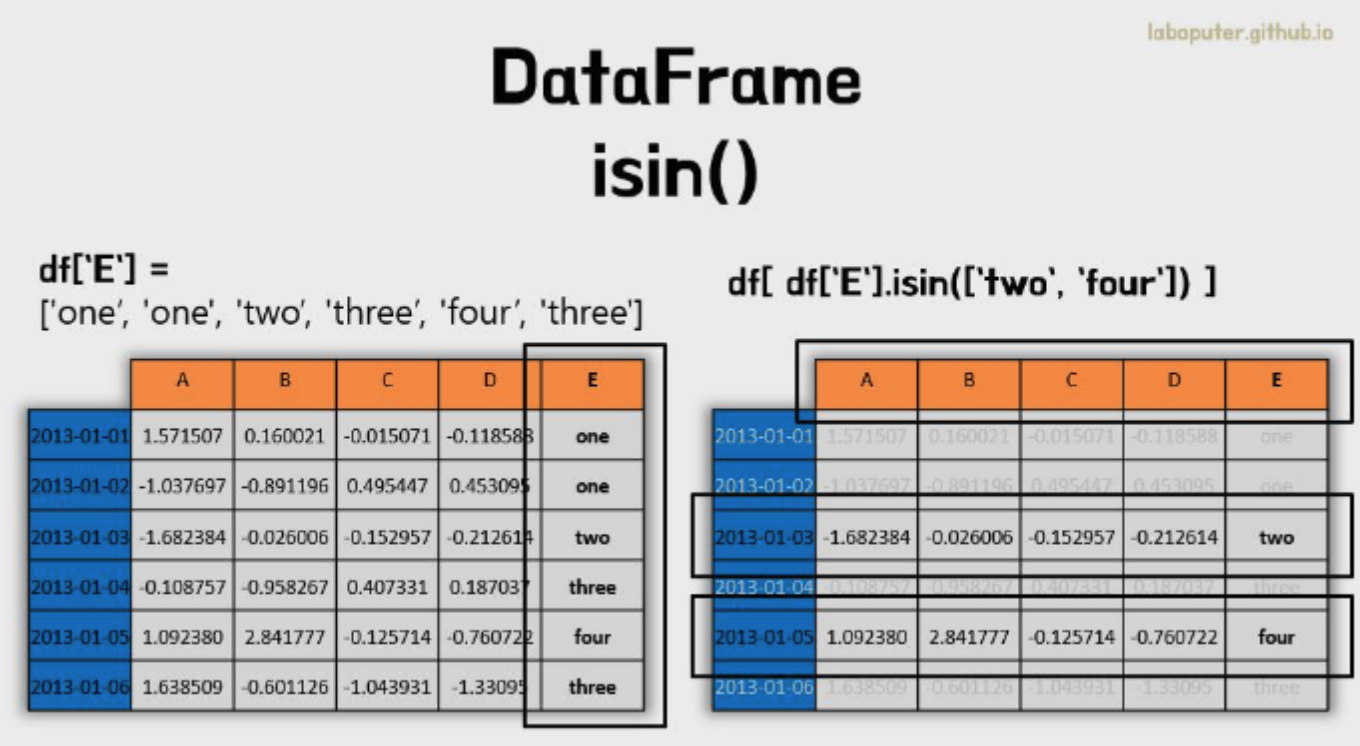
isin 함수

: 열이 list의 값들을 포함하고 있는 모든 행들을 골라낼 때 주로 사용한다.

```
df = DataFrame({'A': [1, 2, 3], 'B': ['a', 'b', 'f']})
df.isin([1, 3, 12, 'a'])

#        A      B
# 0   True   True
# 1  False  False
# 2   True  False
```

```
# 테이블 복사
df2 = df.copy()
# 새로운 컬럼 E에 값 추가
df2['E'] = ['one','one', 'two','three','four','three']
#                    A         B         C         D      E
# 2013-01-01  1.571507  0.160021 -0.015071 -0.118588    one
# 2013-01-02 -1.037697 -0.891196  0.495447  0.453095    one
# 2013-01-03 -1.682384 -0.026006 -0.152957 -0.212614    two
# 2013-01-04 -0.108757 -0.958267  0.407331  0.187037  three
# 2013-01-05  1.092380  2.841777 -0.125714 -0.760722   four
# 2013-01-06  1.638509 -0.601126 -1.043931 -1.330950  three

df2[df2['E'].isin(['two','four'])]
#                    A         B         C         D     E
```

```
# 2013-01-03 -1.682384 -0.026006 -0.152957 -0.212614   two
# 2013-01-05  1.092380  2.841777 -0.125714 -0.760722  four
```



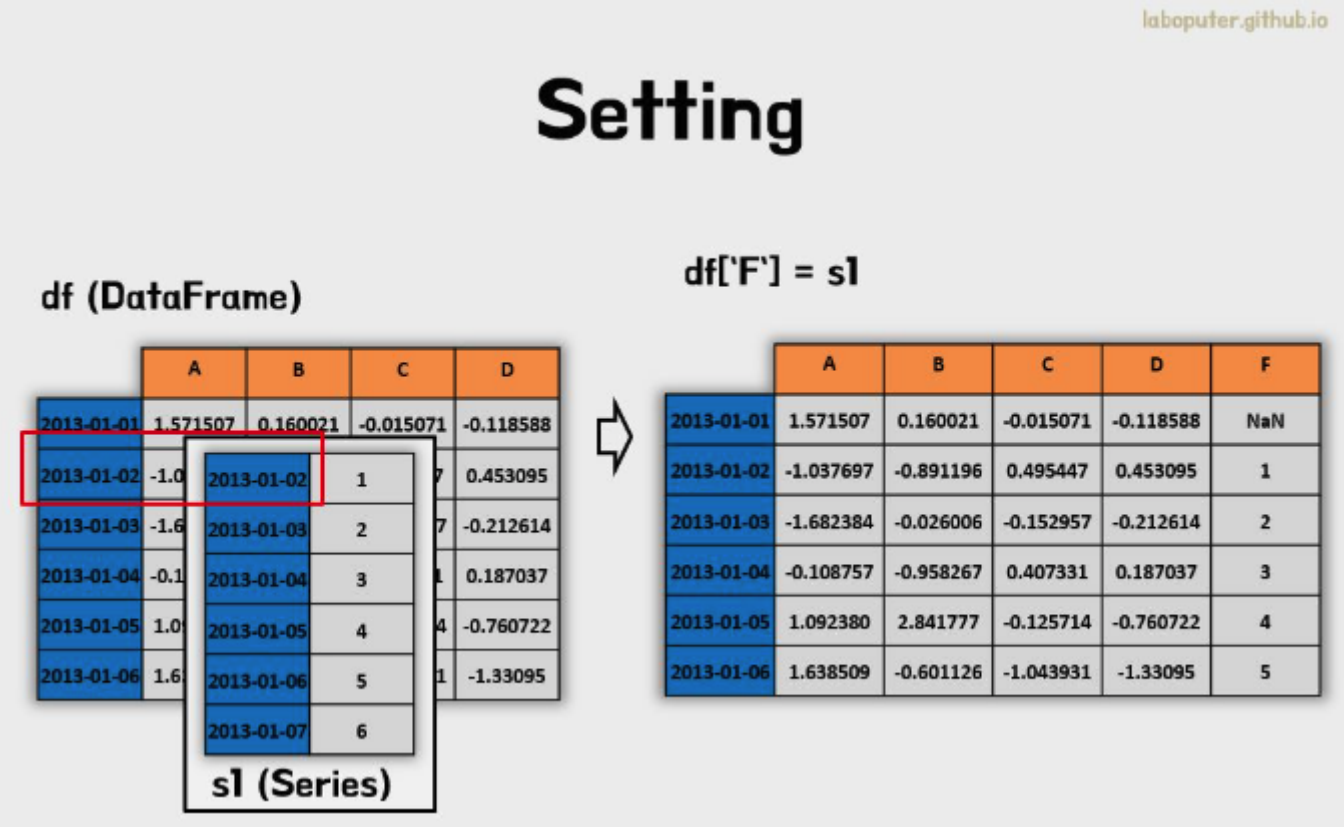
	A	B	C	D	E
2013-01-01	1.571507	0.160021	-0.015071	-0.118588	one
2013-01-02	-1.037697	-0.891196	0.495447	0.453095	one
2013-01-03	-1.682384	-0.026006	-0.152957	-0.212614	two
2013-01-04	-0.108757	-0.958267	0.407331	0.187037	three
2013-01-05	1.092380	2.841777	-0.125714	-0.760722	four
2013-01-06	1.638509	-0.601126	-1.043931	-1.33095	three

	A	B	C	D	E
2013-01-03	-1.682384	-0.026006	-0.152957	-0.212614	two
2013-01-05	1.092380	2.841777	-0.125714	-0.760722	four

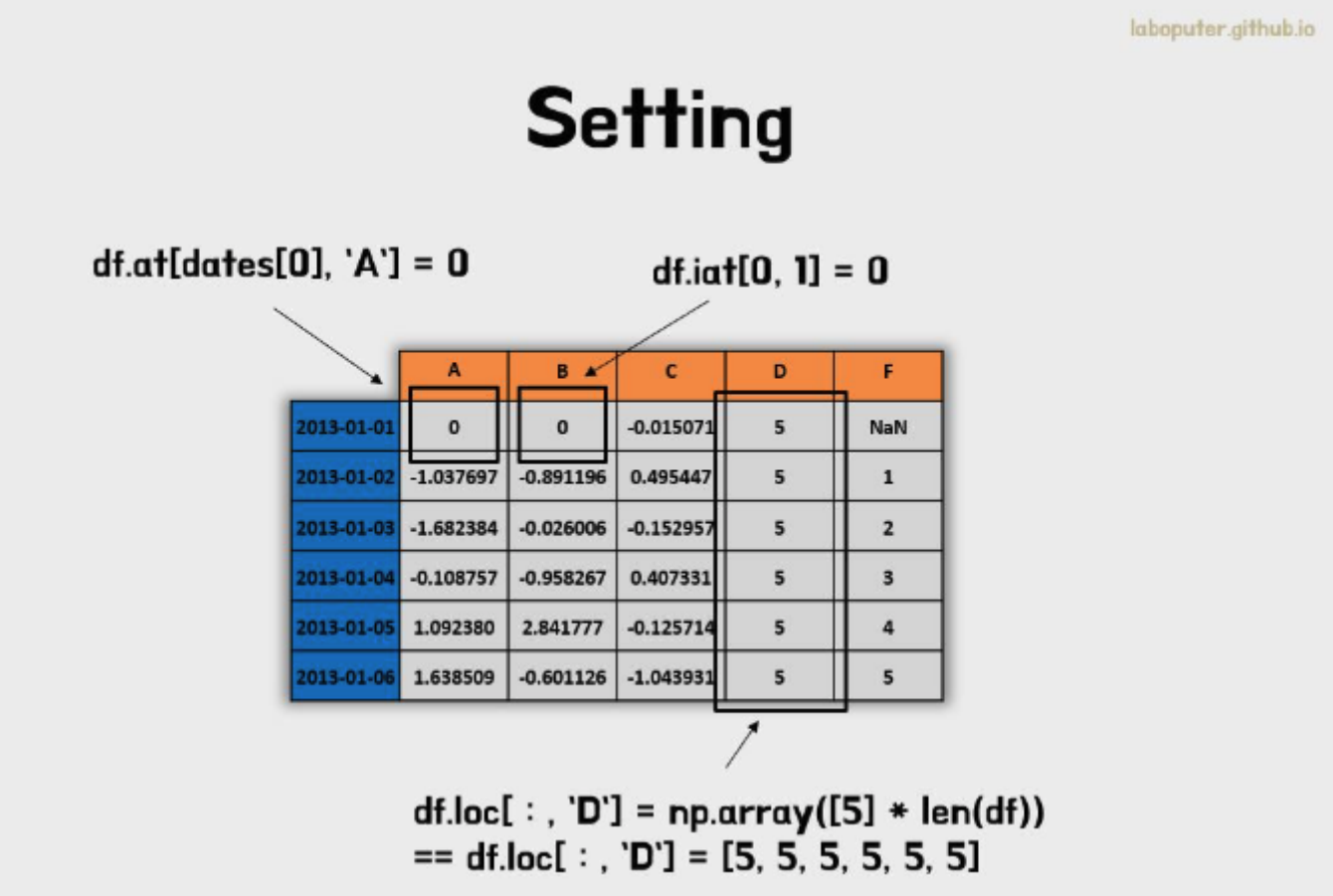
데이터 변경하기

- 데이터 프레임의 값들을 다른 값으로 변경할 수 있습니다.
- 기존 데이터 프레임에 새로운 열을 추가하고 싶을 때는 다음과 같이 같은 인덱스를 가진 시리즈나 리스트를 입력해줍니다.

```
s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('20130102',periods=6))
# 2013-01-02    1
# 2013-01-03    2
# 2013-01-04    3
# 2013-01-05    4
# 2013-01-06    5
# 2013-01-07    6
# Freq: D, dtype: int64

df['F'] = s1
```



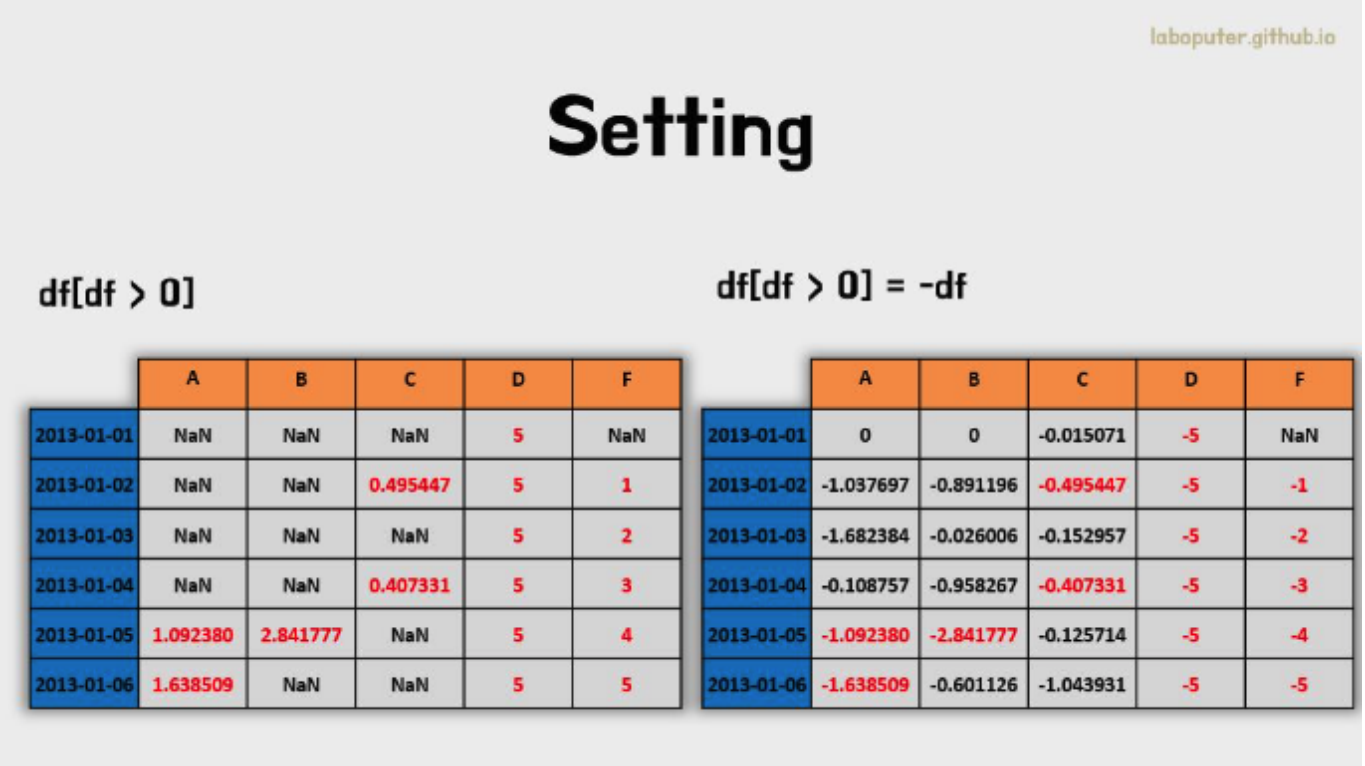

```
# 0번째 인덱스, 'A' 컬럼을 0으로 변경
df.loc[dates[0],'A'] = 0

# 0번째 인덱스, 1번째 컬럼을 0으로 변경
df.iloc[0,1] = 0

# 전체 인덱스, 'D' 컬럼 데이터를 변경
df.loc[:,'D'] = np.array([5] * len(df))

df
#                    A         B         C  D    F
# 2013-01-01  0.000000  0.000000 -0.015071  5  NaN
# 2013-01-02 -1.037697 -0.891196  0.495447  5  1.0
# 2013-01-03 -1.682384 -0.026006 -0.152957  5  2.0
# 2013-01-04 -0.108757 -0.958267  0.407331  5  3.0
# 2013-01-05  1.092380  2.841777 -0.125714  5  4.0
# 2013-01-06  1.638509 -0.601126 -1.043931  5  5.0
```



	A	B	C	D	F
2013-01-01	0	0	-0.015071	5	NaN
2013-01-02	-1.037697	-0.891196	0.495447	5	1
2013-01-03	-1.682384	-0.026006	-0.152957	5	2
2013-01-04	-0.108757	-0.958267	0.407331	5	3
2013-01-05	1.092380	2.841777	-0.125714	5	4
2013-01-06	1.638509	-0.601126	-1.043931	5	5

- 조건문(where)으로 선택하여 데이터를 변경할 수도 있습니다.

```
df2 = df.copy()

# 0보다 큰 데이터만 음수로 변경
df2[df2 > 0] = -df2

df2
#                   A         B         C  D    F
# 2013-01-01  0.000000  0.000000 -0.015071 -5  NaN
# 2013-01-02 -1.037697 -0.891196 -0.495447 -5 -1.0
# 2013-01-03 -1.682384 -0.026006 -0.152957 -5 -2.0
# 2013-01-04 -0.108757 -0.958267 -0.407331 -5 -3.0
# 2013-01-05 -1.092380 -2.841777 -0.125714 -5 -4.0
# 2013-01-06 -1.638509 -0.601126 -1.043931 -5 -5.0
```



	A	B	C	D	F
2013-01-01	NaN	NaN	NaN	5	NaN
2013-01-02	NaN	NaN	0.495447	5	1
2013-01-03	NaN	NaN	NaN	5	2
2013-01-04	NaN	NaN	0.407331	5	3
2013-01-05	1.092380	2.841777	NaN	5	4
2013-01-06	1.638509	NaN	NaN	5	5

	A	B	C	D	F
2013-01-01	0	0	-0.015071	-5	NaN
2013-01-02	-1.037697	-0.891196	-0.495447	-5	-1
2013-01-03	-1.682384	-0.026006	-0.152957	-5	-2
2013-01-04	-0.108757	-0.958267	-0.407331	-5	-3
2013-01-05	-1.092380	-2.841777	-0.125714	-5	-4
2013-01-06	-1.638509	-0.601126	-1.043931	-5	-5

4. 결측 데이터

- 여러가지 이유로 우리는 데이터를 전부 다 측정하지 못하는 경우가 종종 발생합니다.
- 이처럼 측정되지 못하여 비어있는 데이터를 '결측치'라고 합니다. - pandas 에서는 결측치를 np.nan 으로 나타냅니다.
- pandas 에서는 결측치를 기본적으로 연산에서 제외시키고 있습니다.
- 또한 머신러닝, 딥러닝의 경우 결측치가 존재한다면, 코드가 오류나는 경우도 존재하기 때문에 항상 데이터 분석을 하기 전에는 데이터 결측치를 확인하는 습관을 가지는 것이 중요합니다.
- reindex()을 통해 컬럼이나 인덱스를 추가, 삭제, 변경 등의 작업이 가능합니다. 결측 데이터를 만들기 위해 'E' 컬럼을 생성합니다.

```
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E'])
df1.loc[dates[0]:dates[1],'E'] = 1

df1
#                    A         B         C  D    F    E
# 2013-01-01  0.000000  0.000000 -0.015071  5  NaN  1.0
# 2013-01-02 -1.037697 -0.891196  0.495447  5  1.0  1.0
# 2013-01-03 -1.682384 -0.026006 -0.152957  5  2.0  NaN
# 2013-01-04 -0.108757 -0.958267  0.407331  5  3.0  NaN
```

- DataFrame의 dropna()를 통해 결측데이터를 삭제(drop)할 수 있습니다.

drop() 함수

- how='any'는 값들 중 하나라도 NaN인 경우 삭제, how='all'은 전체가 NaN인 경우 삭제

- axis=0 일 경우 NaN값이 있는 행 기준으로 삭제, axis=1 일 경우 열 기준으로 삭제

```
Signature:
data.dropna(
    axis: 'Axis' = 0,
    how: 'str' = 'any',
    thresh=None,
    subset=None,
    inplace: 'bool' = False,
)
Docstring:
Remove missing values.

See the :ref:`User Guide <missing_data>` for more on which values are
considered missing, and how to work with missing data.
```

```
df1.dropna(how='any')
#                    A         B         C  D    F    E
# 2013-01-02 -1.037697 -0.891196  0.495447  5  1.0  1.0
```

- DataFrame의 fillna()를 통해 결측데이터에 값을 넣을 수도 있습니다.

결측 데이터 채우기

- 결측치가 있다면 머신러닝 알고리즘이 학습과 예측을 할 수 없습니다.
- 결측치를 대체하는 방법으로는 여러가지가 있습니다.
- 평균, 중앙, 최빈값 등으로 채우기도 하며 그룹화된 값으로 대표값을 찾아 대체해 주기도 합니다.
- 결측치가 일부라면 제거하기도 합니다.
- 혹은, 머신러닝을 통해 예측해서 대체하기도 합니다.

```
df1.fillna(value=5)
#                    A         B         C  D    F    E
# 2013-01-01  0.000000  0.000000 -0.015071  5  5.0  1.0
# 2013-01-02 -1.037697 -0.891196  0.495447  5  1.0  1.0
# 2013-01-03 -1.682384 -0.026006 -0.152957  5  2.0  5.0
# 2013-01-04 -0.108757 -0.958267  0.407331  5  3.0  5.0
```

- 해당 값이 결측치인지 아닌지의 여부를 알고싶다면 isna() 메소드를 이용하면 됩니다.
- 결측치이면 True, 값이 있다면 False 로 나타납니다.
- 결측치의 전체 합계를 알고 싶다면 .isna()뒤에 .sum() 함수를 활용할 수 있습니다.

```
pd.isna(df1)
#                 A      B      C      D      F      E
# 2013-01-01  False  False  False  False   True  False
# 2013-01-02  False  False  False  False  False  False
# 2013-01-03  False  False  False  False  False   True
# 2013-01-04  False  False  False  False  False   True
```

4. 연산

- 통계적 지표가 계산이 가능합니다.
- 평균 구하기
    - axis = 1 : 인덱스 기준
    - axis = 0 : 칼럼 기준(default)

```
df.mean()
# A   -0.004474
# B   -0.383981
# C   -0.687758
# D    5.000000
# F    3.000000
# dtype: float64

df.mean(1)
# 2013-01-01    0.872735
# 2013-01-02    1.431621
# 2013-01-03    0.707731
# 2013-01-04    1.395042
# 2013-01-05    1.883656
# 2013-01-06    1.592306
# Freq: D, dtype: float64
```

- 함수 적용
    - 데이터에 대해 정의된 함수들이나 lamdba 식을 이용하여 새로운 함수도 적용할 수 있습니다.

```
df.apply(np.cumsum)
#                    A         B         C   D     F
# 2013-01-01  0.000000  0.000000 -1.509059   5   NaN
# 2013-01-02  1.212112 -0.173215 -1.389850  10   1.0
# 2013-01-03  0.350263 -2.277784 -1.884779  15   3.0
# 2013-01-04  1.071818 -2.984555 -2.924354  20   6.0
# 2013-01-05  0.646846 -2.417535 -2.648122  25  10.0
# 2013-01-06 -0.026844 -2.303886 -4.126549  30  15.0

df.apply(lambda x: x.max() - x.min())
# A    2.073961
# B    2.671590
# C    1.785291
# D    0.000000
# F    4.000000
# dtype: float64
```

5. 합치기

- 다양한 정보를 데이터가 있을 때 데이터들을 하나로 합쳐서 새로운 데이터로 만들어야 할 때가 있습니다.
- 같은 형태의 자료들을 이어 하나로 만들어주는 concat, 다른 형태의 자료들을 한 컬럼을 기준으로 합치는 merge를 활용할 수 있습니다.
- Concat
    - concat 을 이용하여 pandas 오브젝트들을 일렬로 연결시켜줍니다.

```
df = pd.DataFrame(np.random.randn(10, 4))
#           0         1         2         3
# 0 -0.548702  1.467327 -1.015962 -0.483075
# 1  1.637550 -1.217659 -0.291519 -1.745505
# 2 -0.263952  0.991460 -0.919069  0.266046
# 3 -0.709661  1.669052  1.037882 -1.705775
# 4 -0.919854 -0.042379  1.247642 -0.009920
# 5  0.290213  0.495767  0.362949  1.548106
# 6 -1.131345 -0.089329  0.337863 -0.945867
# 7 -0.932132  1.956030  0.017587 -0.016692
# 8 -0.575247  0.254161 -1.143704  0.215897
# 9  1.193555 -0.077118 -0.408530 -0.862495

# break it into pieces
pieces = [df[:3], df[3:7], df[7:]]

# concatenate agaain
pd.concat(pieces)
#           0         1         2         3
# 0 -0.548702  1.467327 -1.015962 -0.483075
# 1  1.637550 -1.217659 -0.291519 -1.745505
# 2 -0.263952  0.991460 -0.919069  0.266046
# 3 -0.709661  1.669052  1.037882 -1.705775
# 4 -0.919854 -0.042379  1.247642 -0.009920
# 5  0.290213  0.495767  0.362949  1.548106
# 6 -1.131345 -0.089329  0.337863 -0.945867
# 7 -0.932132  1.956030  0.017587 -0.016692
# 8 -0.575247  0.254161 -1.143704  0.215897
# 9  1.193555 -0.077118 -0.408530 -0.862495
```

- Merge
    - 데이터베이스에서 사용하는 SQL 스타일의 합치기 기능입니다. merge 메소드를 통해 이루어집니다.

```
# 1
left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
#    key  lval
# 0  foo     1
# 1  foo     2
```

```
right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})
#    key  rval
# 0  foo     4
# 1  foo     5

merged = pd.merge(left, right, on='key')
#    key  lval  rval
# 0  foo     1     4
# 1  foo     1     5
# 2  foo     2     4
# 3  foo     2     5

# 2
left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1, 2]})
#    key  lval
# 0  foo     1
# 1  bar     2

right = pd.DataFrame({'key': ['foo', 'bar'], 'rval': [4, 5]})
#    key  rval
# 0  foo     4
# 1  bar     5

merged = pd.merge(left, right, on='key')
#    key  lval  rval
# 0  foo     1     4
# 1  bar     2     5
```

6. 묶기

- SQL과 유사한 group by에 관련된 내용은 아래와 같은 과정을 말합니다.

    1. Splting : 어떠한 기준을 바탕으로 데이터를 나누는 일
    2. applying : 각 그룹에 어떤 함수를 독립적으로 적용시키는 일
    3. Combining : 적용되어 나온 결과들을 통합하는 일

- 아래의 예시처럼 같은 그룹의 합도 구할 수 있지만, .agg() 함수를 통해 여러가지 값을 확인할 수 있습니다. (ex. df.groupby('A').agg(['min', 'max']))

```
df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar',
                         'foo', 'bar', 'foo', 'foo'],
                   'B': ['one', 'one', 'two', 'three',
                         'two', 'two', 'one', 'three'],
                   'C': np.random.randn(8),
                   'D': np.random.randn(8)})

#      A      B         C         D
# 0  foo    one -1.202872 -0.055224
# 1  bar    one -1.814470  2.395985
# 2  foo    two  1.018601  1.552825
# 3  bar  three -0.595447  0.166599
```

```
# 4  foo    two  1.395433  0.047609
# 5  bar    two -0.392670 -0.136473
# 6  foo    one  0.007207 -0.561757
# 7  foo  three  1.928123 -1.623033

df.groupby('A').sum()
#             C        D
# A
# bar -2.802588  2.42611
# foo  3.146492 -0.63958

df.groupby(['A', 'B']).sum()
#                   C         D
# A   B
# bar one   -1.814470  2.395985
#     three -0.595447  0.166599
#     two   -0.392670 -0.136473
# foo one   -1.195665 -0.616981
#     three  1.928123 -1.623033
#     two    2.414034  1.600434
```

7. 데이터 구조 변경하기

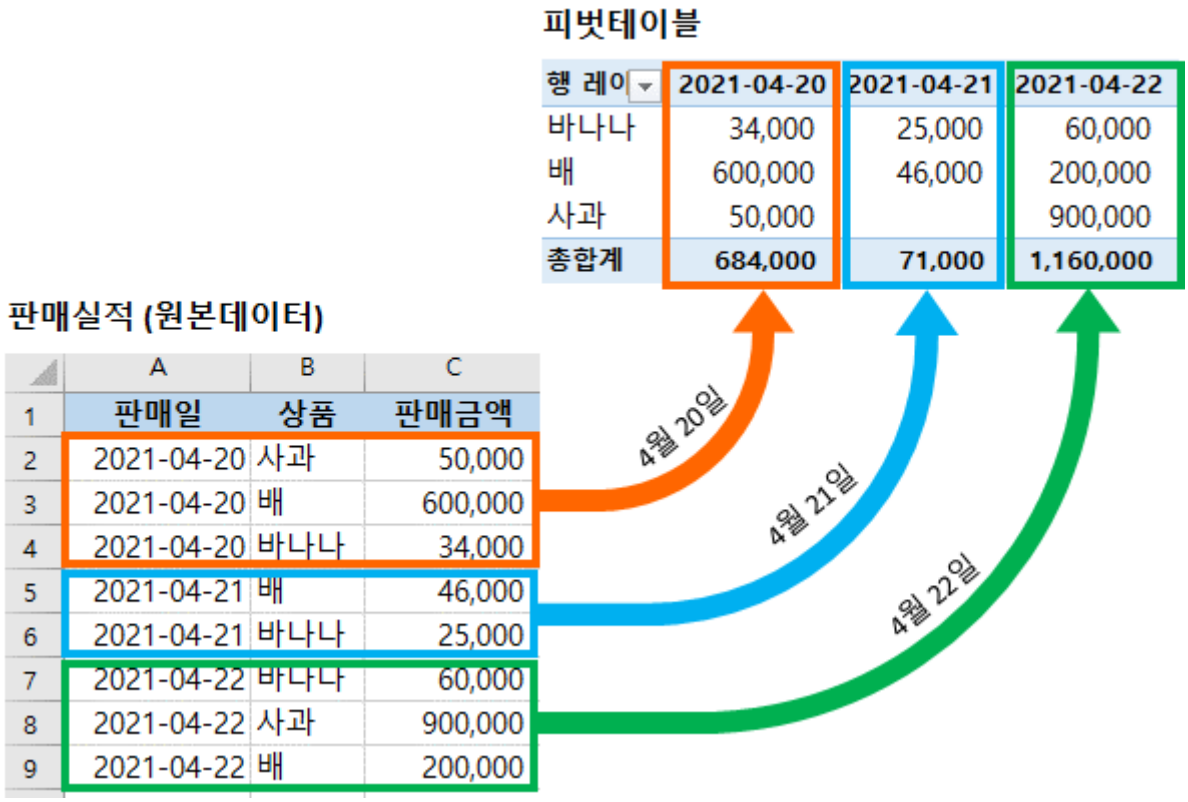
- 피봇 테이블

```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,
                   'B' : ['A', 'B', 'C'] * 4,
                   'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
                   'D' : np.random.randn(12),
                   'E' : np.random.randn(12)})
#         A  B    C         D         E
# 0     one  A  foo -0.268332 -1.378239
# 1     one  B  foo -1.168934  0.263587
# 2     two  C  foo  1.245084  0.882631
# 3   three  A  bar  1.339747  0.770703
# 4     one  B  bar  0.005996  0.501930
# 5     one  C  bar  0.083572 -0.151838
# 6     two  A  foo  1.172619  1.110582
# 7   three  B  foo -0.210904 -0.200479
# 8     one  C  foo  0.166766  0.308271
# 9     one  A  bar  0.516837  0.869884
# 10    two  B  bar -0.667602  0.584587
# 11  three  C  bar -0.848954  0.609278

pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
# C             bar       foo
# A     B
# one   A  0.516837 -0.268332
#       B  0.005996 -1.168934
#       C  0.083572  0.166766
# three A  1.339747       NaN
```

```
#       B       NaN -0.210904
#       C -0.848954       NaN
# two   A       NaN  1.172619
#       B -0.667602       NaN
#       C       NaN  1.245084
```

피봇 테이블이란?

- 피벗 테이블(pivot table)은 커다란 표(예: 데이터베이스, 스프레드시트, 비즈니스 인텔리전스 프로그램 등)의 데이터를 요약하는 통계표이다. 이 요약에는 합계, 평균, 기타 통계가 포함될 수 있으며 피벗 테이블이 이들을 함께 의미있는 방식으로 묶어준다.



8. 파일 입출력

- csv

```
# 저장
df.to_csv('foo.csv')

# 불러오기
pd.read_csv('foo.csv')
#      Unnamed: 0          A          B         C          D
# 0    2000-01-01   0.266457  -0.399641 -0.219582   1.186860
# 1    2000-01-02  -1.170732  -0.345873  1.653061  -0.282953
# 2    2000-01-03  -1.734933   0.530468  2.060811  -0.515536
# 3    2000-01-04  -1.555121   1.452620  0.239859  -1.156896
# 4    2000-01-05   0.578117   0.511371  0.103552  -2.428202
# 5    2000-01-06   0.478344   0.449933 -0.741620  -1.962409
# 6    2000-01-07   1.235339  -0.091757 -1.543861  -1.084753
```

```
# ..          ...        ...        ...       ...        ...
# 993  2002-09-20 -10.628548  -9.153563 -7.883146  28.313940
# 994  2002-09-21 -10.390377  -8.727491 -6.399645  30.914107
# 995  2002-09-22  -8.985362  -8.485624 -4.669462  31.367740
# 996  2002-09-23  -9.558560  -8.781216 -4.499815  30.518439
# 997  2002-09-24  -9.902058  -9.340490 -4.386639  30.105593
# 998  2002-09-25 -10.216020  -9.480682 -3.933802  29.758560
# 999  2002-09-26 -11.856774 -10.671012 -3.216025  29.369368
#
# [1000 rows x 5 columns]
```

- h5

```
# 저장
df.to_hdf('foo.h5', 'df')

# 불러오기
pd.read_hdf('foo.h5', 'df')
#                    A          B         C          D
# 2000-01-01   0.266457  -0.399641 -0.219582   1.186860
# 2000-01-02  -1.170732  -0.345873  1.653061  -0.282953
# 2000-01-03  -1.734933   0.530468  2.060811  -0.515536
# 2000-01-04  -1.555121   1.452620  0.239859  -1.156896
# 2000-01-05   0.578117   0.511371  0.103552  -2.428202
# 2000-01-06   0.478344   0.449933 -0.741620  -1.962409
# 2000-01-07   1.235339  -0.091757 -1.543861  -1.084753
# ...               ...        ...       ...        ...
# 2002-09-20 -10.628548  -9.153563 -7.883146  28.313940
# 2002-09-21 -10.390377  -8.727491 -6.399645  30.914107
# 2002-09-22  -8.985362  -8.485624 -4.669462  31.367740
# 2002-09-23  -9.558560  -8.781216 -4.499815  30.518439
# 2002-09-24  -9.902058  -9.340490 -4.386639  30.105593
# 2002-09-25 -10.216020  -9.480682 -3.933802  29.758560
# 2002-09-26 -11.856774 -10.671012 -3.216025  29.369368
#
# [1000 rows x 4 columns]
```

- excel

```
# 저장
df.to_excel('foo.xlsx', sheet_name='Sheet1')

# 불러오기
pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
#                    A          B         C          D
# 2000-01-01   0.266457  -0.399641 -0.219582   1.186860
# 2000-01-02  -1.170732  -0.345873  1.653061  -0.282953
# 2000-01-03  -1.734933   0.530468  2.060811  -0.515536
# 2000-01-04  -1.555121   1.452620  0.239859  -1.156896
```

```
# 2000-01-05   0.578117   0.511371  0.103552  -2.428202
# 2000-01-06   0.478344   0.449933 -0.741620  -1.962409
# 2000-01-07   1.235339  -0.091757 -1.543861  -1.084753
# ...               ...        ...       ...        ...
# 2002-09-20 -10.628548  -9.153563 -7.883146  28.313940
# 2002-09-21 -10.390377  -8.727491 -6.399645  30.914107
# 2002-09-22  -8.985362  -8.485624 -4.669462  31.367740
# 2002-09-23  -9.558560  -8.781216 -4.499815  30.518439
# 2002-09-24  -9.902058  -9.340490 -4.386639  30.105593
# 2002-09-25 -10.216020  -9.480682 -3.933802  29.758560
# 2002-09-26 -11.856774 -10.671012 -3.216025  29.369368
#
# [1000 rows x 4 columns]
```

- 참고 : https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html