

# PANDAS DATAFRAME

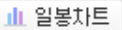
파이썬을 활용한 금융 데이터  
분석 기초 및 심화 과정

# DataFrame

## ■ Pandas DataFrame

- 행과 열로 구성된 2차원 데이터(엑셀 데이터)를 다루는데 효과적인 자료구조
- Series는 DataFrame에서 각 column을 구성하는데 사용함

DataFrame

일자별 주가 							자세히▶
일자	시가	고가	저가	종가	전일비	등락률	거래량
16,10,14	51,500	52,000	50,100	50,200	▼ 1,300	-2.52%	66,839
16,10,13	52,400	52,400	51,000	51,500	▼ 900	-1.72%	41,144
16,10,12	51,800	52,400	51,700	52,400	▲ 200	+0.38%	28,170
16,10,11	52,500	53,200	51,800	52,200	▼ 300	-0.57%	38,772
16,10,10	52,100	53,400	52,100	52,500	▼ 200	-0.38%	36,708
16,10,07	54,700	55,500	52,300	52,700	▼ 2,200	-4.01%	53,115
16,10,06	55,100	56,000	54,700	54,900	▼ 600	-1.08%	26,180
16,10,05	54,800	55,600	54,700	55,500	▲ 300	+0.54%	21,422
16,10,04	54,700	55,500	53,800	55,200	▲ 100	+0.18%	40,785
16,09,30	57,100	57,100	54,100	55,100	▼ 2,000	-3.50%	76,966

Series

# DataFrame 생성

- 파이썬 딕셔너리로 표현된 데이터에 대해 DataFrame 객체로 생성

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

```
raw_data = {'open': [100, 90, 80, 70],  
            'high': [110, 112, 115, 80],  
            'low': [90, 80, 70, 60],  
            'close': [105, 95, 85, 75]}
```

```
df = DataFrame(raw_data)  
print(df)
```

	close	high	low	open
0	105	110	90	100
1	95	112	80	90
2	85	115	70	80
3	75	80	60	70

# DataFrame 생성

- Column 순서 설정

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

```
raw_data = {'open': [100, 90, 80, 70],  
            'high': [110, 112, 115, 80],  
            'low': [90, 80, 70, 60],  
            'close': [105, 95, 85, 75]}
```

```
df = DataFrame(raw_data, columns=['open', 'high', 'low', 'close'])  
print(df)
```

	open	high	low	close
0	100	110	90	105
1	90	112	80	95
2	80	115	70	85
3	70	80	60	75

# DataFrame 객체의 구조 (1/2)

- DataFrame 객체에서 각 column은 Series 객체
  - 각 column의 인덱스는 동일

df →

	open	high	low	close
index	value	value	value	value
0	100	110	90	105
1	90	112	80	95
2	80	115	70	85
3	70	80	60	75

# DataFrame 객체의 구조 (2/2)

df

'open'
'high'
'low'
'close'

index	value
0	100
1	90
2	80
3	70

index	value
0	110
1	112
2	115
3	80

index	value
0	90
1	80
2	70
3	60

index	value
0	105
1	95
2	85
3	75

# DataFrame 인덱스 설정

- 미래에셋대우 일자별 주가 (5일치)
- 인덱스로 일자를 사용

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

```
date_list = pd.date_range(end='2017-06-02', periods=5).tolist()
date_list.reverse()
```

```
ohlc = {'open': [9320, 9450, 9660, 9640, 9890],
        'high': [9450, 9480, 9660, 9820, 9890],
        'low': [9320, 9240, 9500, 9550, 9600],
        'close': [9410, 9270, 9500, 9710, 9640]}
```

```
7 df = DataFrame(ohlc, columns=['open', 'high', 'low', 'close'], index=date_list)
   print(df)
```

일자별 주가 일봉차트

일자	시가	고가	저가	종가
17.06.02	9,320	9,450	9,320	9,410
17.06.01	9,450	9,480	9,240	9,270
17.05.31	9,660	9,660	9,500	9,500
17.05.30	9,640	9,820	9,550	9,710
17.05.29	9,890	9,890	9,600	9,640

# DataFrame 인덱스 설정

- 결괏값 (df)

	open	high	low	close
2017-06-02	9320	9450	9320	9410
2017-06-01	9450	9480	9240	9270
2017-05-31	9660	9660	9500	9500
2017-05-30	9640	9820	9550	9710
2017-05-29	9890	9890	9600	9640



# Column 선택하기

- 파이썬 딕셔너리에서 키를 통해 값에 접근했던 것과 같이 DataFrame 객체의 각 칼럼에 접근할 수 있음

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

```
raw_data = {'open': [100, 90, 80, 70],  
            'high': [110, 112, 115, 80],  
            'low': [90, 80, 70, 60],  
            'close': [105, 95, 85, 75]}
```

```
df = DataFrame(raw_data, columns=['open', 'high', 'low', 'close'])  
close = df['close']
```

```
print(close)
```

```
9 print(type(close))
```

```
0    105  
1     95  
2     85  
3     75  
Name: close, dtype: int64  
<class 'pandas.core.series.Series'>
```

# 여러 칼럼 선택하기

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

```
raw_data = {'open': [100, 90, 80, 70],  
            'high': [110, 112, 115, 80],  
            'low': [90, 80, 70, 60],  
            'close': [105, 95, 85, 75]}
```

```
df = DataFrame(raw_data, columns=['open', 'high', 'low', 'close'])  
print(df)
```

```
df2 = df[['open', 'close']]  
print(df2)
```

	open	high	low	close
0	100	110	90	105
1	90	112	80	95
2	80	115	70	85
3	70	80	60	75

	open	close
0	100	105
1	90	95
2	80	85
3	70	75

# row 선택하기

- row를 선택하는 세 가지 메서드

메서드	기능
loc	인덱스 라벨을 통해 로우를 선택
iloc	정수 인덱스를 통해서 로우를 선택
ix	인덱스 라벨 또는 정수 인덱스를 통해서 로우를 선택

# row 선택하기

- DataFrame

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

```
data = {'col0' : [1, 2, 3], 'col1': [10, 20, 30]}  
df = DataFrame(data, index=['a', 'b', 'c'])  
print(df)
```

```
   col0  col1  
a     1    10  
b     2    20  
c     3    30
```

# row 선택하기

- loc 메서드 사용

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

```
data = {'col0' : [1, 2, 3], 'col1': [10, 20, 30]}  
df = DataFrame(data, index=['a', 'b', 'c'])
```

```
print(df)  
s = df.loc['a']  
print(s, type(s))
```

	col0	col1
a	1	10
b	2	20
c	3	30

```
col0    1  
col1   10  
Name: a, dtype: int64 <class 'pandas.core.series.Series'>
```

# row 선택하기

- iloc 메서드 사용

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

```
data = {'col0' : [1, 2, 3], 'col1': [10, 20, 30]}  
df = DataFrame(data, index=['a', 'b', 'c'])
```

```
print(df)  
s = df.iloc[0]  
print(s, type(s))
```

	col0	col1
a	1	10
b	2	20
c	3	30

```
col0    1  
col1    10  
Name: a, dtype: int64 <class 'pandas.core.series.Series'>
```

# row 선택하기

- ix 메서드 사용

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

```
data = {'col0' : [1, 2, 3], 'col1': [10, 20, 30]}
```

```
df = DataFrame(data, index=['a', 'b', 'c'])
```

```
print(df)
```

```
s = df.ix[0]
```

```
print(s, type(s))
```

	col0	col1
a	1	10
b	2	20
c	3	30

```
col0    1  
col1    10  
Name: a, dtype: int64 <class 'pandas.core.series.Series'>
```

# row 선택하기

```
import pandas as pd
from pandas import Series, DataFrame
import datetime
```

```
date_list = pd.date_range(end='2017-06-02', periods=5).tolist()
date_list.reverse()
```

```
ohlc = {'open': [9320, 9450, 9660, 9640, 9890],
        'high': [9450, 9480, 9660, 9820, 9890],
        'low': [9320, 9240, 9500, 9550, 9600],
        'close': [9410, 9270, 9500, 9710, 9640]}
```

```
df = DataFrame(ohlc, columns=['open', 'high', 'low', 'close'], index=date_list)
print(df)
```

```
print(df.loc[datetime.datetime(2017, 6, 2)])
print(df.iloc[0])
print(df.ix[0])
```

DataFrame 인덱스로 DateTime 객체가  
사용된 경우



# Row/Column 삭제

## ■ drop 메서드

- Return new object with labels in requested axis removed (기존 DataFrame 유지)

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

```
data = {'col0': [1, 2, 3], 'col1': [10, 20, 30], 'col2': [100, 200, 300]}  
df = DataFrame(data)  
print(df)
```

```
df2 = df.drop('col0', axis=1)    # 칼럼 drop  
print(df2)
```

	col0	col1	col2
0	1	10	100
1	2	20	200
2	3	30	300

	col1	col2
0	10	100
1	20	200
2	30	300

# Row/Column 삭제

## ■ drop 메서드

- Return new object with labels in requested axis removed (기존 DataFrame 유지)

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

```
data = {'col0': [1, 2, 3], 'col1': [10, 20, 30], 'col2': [100, 200, 300]}
```

```
df = DataFrame(data)
```

```
print(df)
```

```
df2 = df.drop([0, 1])      # row drop
```

```
print(df2)
```

df			
	col0	col1	col2
0	1	10	100
1	2	20	200
2	3	30	300

df2			
	col0	col1	col2
2	3	30	300

# 연습 문제

- 문제-1 다음과 같은 데이터를 DataFrame 객체로 표현해 봅시다.

Columns →

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

Index →

A two-dimensional labeled data structure with columns of potentially different types

# 연습 문제

- 문제-2 read\_excel 함수를 사용해서 “성적표.xlsx”를 DataFrame 객체로 만든 후 이를 df라는 변수가 바인딩하게 만들어 봅시다.

[https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read\\_excel.html](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_excel.html)

```
pandas.read_excel(io, sheetname=0, header=0, skiprows=None, skip_footer=0, index_col=None, names=None,
parse_cols=None, parse_dates=False, date_parser=None, na_values=None, thousands=None, convert_float=True,
has_index_names=None, converters=None, dtype=None, true_values=None, false_values=None, engine=None,
squeeze=False, **kwargs) [source]
```

Read an Excel table into a pandas DataFrame

**io** : string, path object (pathlib.Path or py.\_path.local.LocalPath),  
file-like object, pandas ExcelFile, or xlrd workbook. The string could be a URL. Valid  
URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. For  
instance, a local file could be file://localhost/path/to/workbook.xlsx

# 연습 문제

- 문제-3 앞서 만든 df에 총점과 평균 칼럼을 추가해보세요.

## pandas.DataFrame.sum

`DataFrame.sum(axis=None, skipna=None, level=None, numeric_only=None, **kwargs)`

[\[source\]](#)

Return the sum of the values for the requested axis

### Parameters:

**axis** : *{index (0), columns (1)}*

**skipna** : *boolean, default True*

Exclude NA/null values. If an entire row/column is NA, the result will be NA

**level** : *int or level name, default None*

If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series

**numeric\_only** : *boolean, default None*

Include only float, int, boolean columns. If None, will attempt to use everything, then use only numeric data. Not implemented for Series.

### Returns:

**sum** : *Series or DataFrame (if level specified)*

# 연습 문제

- 문제-4 평균이 높은 순서대로 정렬하여 1등을 찾아 봅시다.

```
pandas.DataFrame.sort_values
```

`DataFrame.sort_values`(by, axis=0, ascending=True, inplace=False, kind='quicksort', na\_position='last')

[\[source\]](#)

Sort by the values along either axis

*New in version 0.17.0.*

# 더 읽어 보기

- <http://nbviewer.jupyter.org/gist/yong27/715c0ef9a09dd6eb37e9>