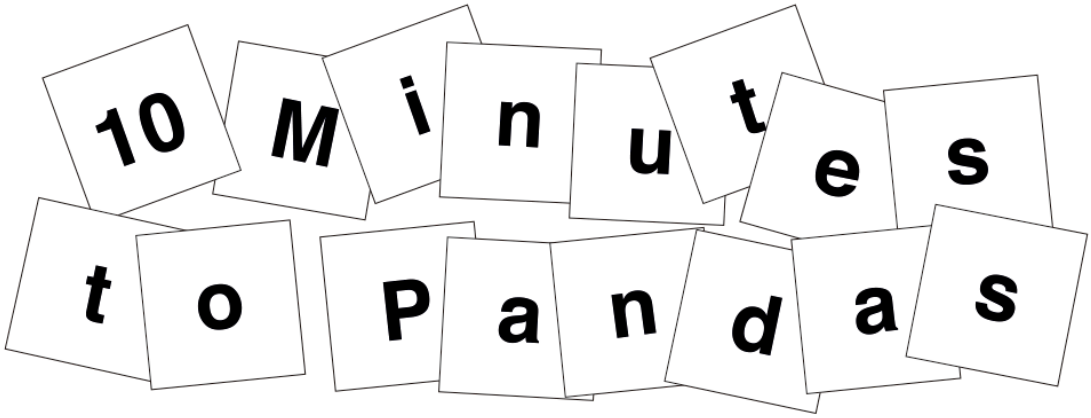
판다스 10분 완성
10 Minutes to Pandas



데잇걸즈2

Pandas 10분 완성

역자 주 : 본 자료는 10 Minutes to Pandas (하단 원문 링크 참조)의 한글 번역 자료로, 번역은 데잇걸즈2 프로그램 교육생 모두가 함께 진행하였습니다. 데잇걸즈2는 과학기술정보통신부와 한국정보화진흥원이 주관하는 SW여성인재 빅데이터 분석 교육과정으로, 상세한 소개는 페이스북 페이지를 참조 부탁 드립니다.

역자 주 (참조 자료) : 10 Minuts to Pandas 원문, 판다스 개발자의 PyCon Korea 2016 발표 : Keynote, Pandas 10분 완성 원문의 인터넷 강의 영상, Pandas Cheat Sheet

---

이 소개서는 주로 신규 사용자를 대상으로 한 판다스에 대한 간략한 소개로, 아래와 같이 구성되어 있습니다. 더 자세한 방법은 Cookbook에서 볼 수 있습니다.

1. Object Creation (객체 생성)
2. Viewing Data (데이터 확인하기)
3. Selection (선택)
4. Missing Data (결측치)
5. Operation (연산)

6. Merge (병합)
7. Grouping (그룹화)
8. Reshaping (변형)
9. Time Series (시계열)
10. Categoricals (범주화)
11. Plotting (그래프)
12. Getting Data In / Out (데이터 입 / 출력)
13. Gotchas (잡았다!)

일반적으로 각 패캐지는 pd, np, plt라는 이름으로 불러옵니다.

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

1. Object Creation (객체 생성)

데이터 구조 소개 섹션을 참조하세요.

Pandas는 값을 가지고 있는 리스트를 통해 Series를 만들고, 정수로 만들어진 인덱스를 기본값으로 불러올 것입니다.

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

s
```

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

datetime 인덱스와 레이블이 있는 열을 가지고 있는 numpy 배열을 전달하여 데이터프레임을 만듭니다.

```
dates = pd.date_range('20130101', periods=6)
```

```
dates
```

```
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'))
```

```
df
```

	A	B	C	D
2013-01-01	1.203664	0.035199	-0.516512	-1.651954
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447
2013-01-03	-2.364223	-2.187468	1.018928	1.252907
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004
2013-01-05	1.387345	-0.443100	-0.540677	-0.370186
2013-01-06	0.222998	-1.308863	0.433432	0.409407

Series와 같은 것으로 변환될 수 있는 객체들의 dict로 구성된 데이터프레임을 만듭니다.

```
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' })
```

```
df2
```

	A	B	C	D	E	F

	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

데이터프레임 결과물의 열은 다양한 데이터 타입 (dtypes)으로 구성됩니다.

```
df2.dtypes
```

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

IPython을 이용하고 계시다면 (공용 속성을 포함한) 열 이름에 대한 Tap 자동완성 기능이 자동으로 활성화 됩니다. 다음은 완성될 속성에 대한 부분집합 (subset)입니다.

역자 주 : 아래 제시된 코드의 경우, IPython이 아닌 환경 (Google Colaboratory, Jupyter 등)에서는 사용이 불가능한 코드인 점에 주의하세요.

```
# df2.<TAB>
```

역자 주 : IPython에서 실행하면 다음과 같은 결과값이 나옵니다.

```
df2.A                  df2.bool
df2.abs                df2.boxplot
df2.add                df2.C
df2.add_prefix         df2.clip
df2.add_suffix         df2.clip_lower
df2.align              df2.clip_upper
df2.all                df2.columns
df2.any                df2.combine
```

```
df2.append              df2.combine_first
df2.apply               df2.compound
df2.applymap            df2.consolidate
df2.D
```

보시다시피 A, B, C, D열이 탭 자동완성 기능으로 실행됩니다. 물론 E도 있습니다. 나머지 속성들은 간결하게 잘라 버렸습니다.

2. Viewing Data (데이터 확인하기)

Basic Section을 참조하세요.

데이터프레임의 가장 윗 줄과 마지막 줄을 확인하고 싶을 때에 사용하는 방법은 다음과 같습니다.

역자 주: 괄호() 안에는 숫자가 들어갈 수도 있고 안 들어갈 수도 있습니다. 숫자가 들어간다면, 윗 / 마지막 줄의 특정 줄을 불러올 수 있습니다. 숫자가 들어가지 않다면, 기본값인 5로 처리됩니다.

예시

```
df.tail(3)  # 끝에서 마지막 3줄을 불러옴
df.tail()  # 끝에서 마지막 5줄 불러옴
```

```
df.head()
```

	A	B	C	D
2013-01-01	1.203664	0.035199	-0.516512	-1.651954
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447
2013-01-03	-2.364223	-2.187468	1.018928	1.252907
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004
2013-01-05	1.387345	-0.443100	-0.540677	-0.370186

```
df.tail(3)
```

	A	B	C	D

	A	B	C	D
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004
2013-01-05	1.387345	-0.443100	-0.540677	-0.370186
2013-01-06	0.222998	-1.308863	0.433432	0.409407

인덱스 (index), 열 (column) 그리고 numpy 데이터에 대한 세부 정보를 봅니다.

```
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
```

```
array([[ 1.20366414,  0.03519932, -0.51651206, -1.65195383],
       [-0.93589333,  0.85494382, -0.81497074, -0.33344655],
       [-2.36422326, -2.18746816,  1.01892836,  1.25290739],
       [-2.21401998,  0.36188549, -0.390074  , -0.49700376],
       [ 1.38734459, -0.44310022, -0.54067692, -0.37018639],
       [ 0.22299798, -1.30886252,  0.43343249,  0.40940659]])
```

describe()는 데이터의 대략적인 통계적 정보 요약을 보여줍니다.

```
df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.450022	-0.447900	-0.134979	-0.198379
std	1.647755	1.127290	0.706005	0.972158

	A	B	C	D
min	-2.364223	-2.187468	-0.814971	-1.651954
25%	-1.894488	-1.092422	-0.534636	-0.465299
50%	-0.356448	-0.203950	-0.453293	-0.351816
75%	0.958498	0.280214	0.227556	0.223693
max	1.387345	0.854944	1.018928	1.252907

데이터를 전치합니다.

```
df.T
```

	2013-01-01 00:00:00	2013-01-02 00:00:00	2013-01-03 00:00:00	2013-01-04 00
A	1.203664	-0.935893	-2.364223	-2.
B	0.035199	0.854944	-2.187468	0.
C	-0.516512	-0.814971	1.018928	-0.
D	-1.651954	-0.333447	1.252907	-0.

축 별로 정렬합니다.

```
df.sort_index(axis=1, ascending=False)
```

	D	C	B	A
2013-01-01	-1.651954	-0.516512	0.035199	1.203664
2013-01-02	-0.333447	-0.814971	0.854944	-0.935893
2013-01-03	1.252907	1.018928	-2.187468	-2.364223
2013-01-04	-0.497004	-0.390074	0.361885	-2.214020
2013-01-05	-0.370186	-0.540677	-0.443100	1.387345
2013-01-06	0.409407	0.433432	-1.308863	0.222998

값 별로 정렬합니다.

```
df.sort_values(by='B')
```

	A	B	C	D
2013-01-03	-2.364223	-2.187468	1.018928	1.252907
2013-01-06	0.222998	-1.308863	0.433432	0.409407
2013-01-05	1.387345	-0.443100	-0.540677	-0.370186
2013-01-01	1.203664	0.035199	-0.516512	-1.651954
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447

3. Selection (선택)

주석 (Note) : 선택과 설정을 위한 Python / Numpy의 표준화된 표현들이 직관적이며, 코드 작성을 위한 양방향 작업에 유용하지만 우리는 Pandas에 최적화된 데이터 접근 방법인 .at, .iat, .loc 및 .iloc 을 추천합니다.

데이터 인덱싱 및 선택 문서와 다중 인덱싱 / 심화 인덱싱 문서를 참조하세요.

Getting (데이터 얻기)

df.A 와 동일한 Series를 생성하는 단일 열을 선택합니다.

```
df['A']
```

```
2013-01-01    1.203664
2013-01-02   -0.935893
2013-01-03   -2.364223
2013-01-04   -2.214020
2013-01-05    1.387345
2013-01-06    0.222998
Freq: D, Name: A, dtype: float64
```

행을 분할하는 [ ]를 통해 선택합니다.

```
df[0:3]
```

	A	B	C	D
2013-01-01	1.203664	0.035199	-0.516512	-1.651954
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447
2013-01-03	-2.364223	-2.187468	1.018928	1.252907

```
df['20130102':'20130104']
```

	A	B	C	D
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447
2013-01-03	-2.364223	-2.187468	1.018928	1.252907
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004

Selection by Label (Label 을 통한 선택)

Label을 통한 선택에서 더 많은 내용을 확인하세요.

라벨을 사용하여 횡단면을 얻습니다.

```
df.loc[dates[0]]
```

```
A    1.203664
B    0.035199
C   -0.516512
D   -1.651954
Name: 2013-01-01 00:00:00, dtype: float64
```

라벨을 사용하여 여러 축 (의 데이터)을 얻습니다.

```
df.loc[:,['A','B']]
```

	A	B
2013-01-01	1.203664	0.035199
2013-01-02	-0.935893	0.854944
2013-01-03	-2.364223	-2.187468
2013-01-04	-2.214020	0.361885
2013-01-05	1.387345	-0.443100
2013-01-06	0.222998	-1.308863

양쪽 종단점을 포함한 라벨 슬라이싱을 봅니다.

```
df.loc['20130102':'20130104', ['A','B']]
```

	A	B
2013-01-02	-0.935893	0.854944
2013-01-03	-2.364223	-2.187468
2013-01-04	-2.214020	0.361885

반환되는 객체의 차원를 줄입니다.

```
df.loc['20130102',['A','B']]
```

```
A   -0.935893
B    0.854944
Name: 2013-01-02 00:00:00, dtype: float64
```

스칼라 값을 얻습니다.

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

```
1.2036641391265706
```

스칼라 값을 더 빠르게 구하는 방법입니다 (앞선 메소드와 동일합니다).

```
df.at[dates[0],'A']
```

```
1.2036641391265706
```

Selection by Position (위치로 선택하기)

자세한 내용은 위치로 선택하기를 참고해주세요.

넘겨받은 정수의 위치를 기준으로 선택합니다.

```
df.iloc[3]
```

```
A    0.834505
B    0.029459
C    0.543112
D   -1.471167
Name: 2013-01-04 00:00:00, dtype: float64
```

정수로 표기된 슬라이스들을 통해, numpy / python과 유사하게 작동합니다.

```
df.iloc[3:5,0:2]
```

	A	B
2013-01-04	0.834505	0.029459
2013-01-05	0.646337	2.139297

정수로 표기된 위치값의 리스트들을 통해, numpy / python의 스타일과 유사해집니다.

```
df.iloc[[1,2,4],[0,2]]
```

	A	C
2013-01-02	-1.403231	0.791482
2013-01-03	0.812184	-0.394338
2013-01-05	0.646337	-0.839317

명시적으로 행을 나누고자 하는 경우입니다.

```
df.iloc[1:3,:]
```

	A	B	C	D
2013-01-02	-1.403231	-0.316784	0.791482	-0.699104
2013-01-03	0.812184	-0.117943	-0.394338	1.669255

명시적으로 열을 나누고자 하는 경우입니다.

```
df.iloc[:,1:3]
```

	B	C
2013-01-01	-0.228990	-0.412877
2013-01-02	-0.316784	0.791482
2013-01-03	-0.117943	-0.394338
2013-01-04	0.029459	0.543112
2013-01-05	2.139297	-0.839317
2013-01-06	-0.026487	0.471119

명시적으로 (특정한) 값을 얻고자 하는 경우입니다.

```
df.iloc[1,1]
```

```
-0.31678422882681939
```

스칼라 값을 빠르게 얻는 방법입니다 (위의 방식과 동일합니다).

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

```
-0.31678422882681939
```

Boolean Indexing

데이터를 선택하기 위해 단일 열의 값을 사용합니다.

```
df[df.A > 0]
```

	A	B	C	D
2013-01-03	0.812184	-0.117943	-0.394338	1.669255
2013-01-04	0.834505	0.029459	0.543112	-1.471167
2013-01-05	0.646337	2.139297	-0.839317	0.107340
2013-01-06	1.766095	-0.026487	0.471119	0.227956

Boolean 조건을 충족하는 데이터프레임에서 값을 선택합니다.

```
df[df > 0]
```

	A	B	C	D
2013-01-01	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	0.791482	NaN
2013-01-03	0.812184	NaN	NaN	1.669255
2013-01-04	0.834505	0.029459	0.543112	NaN
2013-01-05	0.646337	2.139297	NaN	0.107340
2013-01-06	1.766095	NaN	0.471119	0.227956

필터링을 위한 메소드 isin()을 사용합니다.

```
df2 = df.copy()
```

```
df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
```

```
df2
```

	A	B	C	D	E

	A	B	C	D	E
2013-01-01	-1.285004	-0.228990	-0.412877	-0.801001	one
2013-01-02	-1.403231	-0.316784	0.791482	-0.699104	one
2013-01-03	0.812184	-0.117943	-0.394338	1.669255	two
2013-01-04	0.834505	0.029459	0.543112	-1.471167	three
2013-01-05	0.646337	2.139297	-0.839317	0.107340	four
2013-01-06	1.766095	-0.026487	0.471119	0.227956	three

```
df2[df2['E'].isin(['two','four'])]
```

	A	B	C	D	E
2013-01-03	0.812184	-0.117943	-0.394338	1.669255	two
2013-01-05	0.646337	2.139297	-0.839317	0.107340	four

Setting (설정)

새 열을 설정하면 데이터가 인덱스 별로 자동 정렬됩니다.

```
s1 = pd.Series([1,2,3,4,5,6], index=pd.date_range('20130102', periods=6))
```

```
s1
```

```
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
```

라벨에 의해 값을 설정합니다.

```
df.at[dates[0],'A'] = 0
```

위치에 의해 값을 설정합니다.

```
df.iat[0,1] = 0
```

Numpy 배열을 사용한 할당에 의해 값을 설정합니다.

```
df.loc[:,'D'] = np.array([5] * len(df))
```

위 설정대로 작동한 결과입니다.

```
df
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.412877	5	NaN
2013-01-02	-1.403231	-0.316784	0.791482	5	1.0
2013-01-03	0.812184	-0.117943	-0.394338	5	2.0
2013-01-04	0.834505	0.029459	0.543112	5	3.0
2013-01-05	0.646337	2.139297	-0.839317	5	4.0
2013-01-06	1.766095	-0.026487	0.471119	5	5.0

where 연산을 설정합니다.

```
df2 = df.copy()
```

```
df2[df2 > 0] = -df2
```

```
df2
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.412877	-5	NaN
2013-01-02	-1.403231	-0.316784	-0.791482	-5	-1.0

	A	B	C	D	F
2013-01-03	-0.812184	-0.117943	-0.394338	-5	-2.0
2013-01-04	-0.834505	-0.029459	-0.543112	-5	-3.0
2013-01-05	-0.646337	-2.139297	-0.839317	-5	-4.0
2013-01-06	-1.766095	-0.026487	-0.471119	-5	-5.0

4. Missing Data (결측치)

Pandas는 결측치를 표현하기 위해 주로 np.nan 값을 사용합니다. 이 방법은 기본 설정값이지만 계산에는 포함되지 않습니다. Missing data section을 참조하세요.

Reindexing으로 지정된 축 상의 인덱스를 변경 / 추가 / 삭제할 수 있습니다. Reindexing은 데이터의 복사본을 반환합니다.

```
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	E
2013-01-01	1.203664	0.035199	-0.516512	-1.651954	1.0
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447	1.0
2013-01-03	-2.364223	-2.187468	1.018928	1.252907	NaN
2013-01-04	-2.214020	0.361885	-0.390074	-0.497004	NaN

결측치를 가지고 있는 행들을 지웁니다.

```
df1.dropna(how='any')
```

	A	B	C	D	E

	A	B	C	D	E
2013-01-01	1.203664	0.035199	-0.516512	-1.651954	1.0
2013-01-02	-0.935893	0.854944	-0.814971	-0.333447	1.0

결측치를 채워 넣습니다.

```
df1.fillna(value=5)
```

	A	B	C	D	F	E
2013-01-01	0.000000	0.000000	-0.412877	5	5.0	1.0
2013-01-02	-1.403231	-0.316784	0.791482	5	1.0	1.0
2013-01-03	0.812184	-0.117943	-0.394338	5	2.0	5.0
2013-01-04	0.834505	0.029459	0.543112	5	3.0	5.0

nan인 값에 boolean을 통한 표식을 얻습니다.

역자 주 : 데이터프레임의 모든 값이 boolean 형태로 표시되도록 하며, nan인 값에만 True가 표시되게 하는 함수입니다.

```
pd.isna(df1)
```

	A	B	C	D	E
2013-01-01	False	False	False	False	False
2013-01-02	False	False	False	False	False
2013-01-03	False	False	False	False	True
2013-01-04	False	False	False	False	True

5. Operation (연산)

이진 (Binary) 연산의 기본 섹션을 참조하세요.

Stats (통계)

일반적으로 결측치를 제외한 후 연산됩니다.

기술통계를 수행합니다.

```
df.mean()
```

```
A    0.442648
B    0.284590
C    0.026530
D    5.000000
F    3.000000
dtype: float64
```

다른 축에서 동일한 연산을 수행합니다.

```
df.mean(1)
```

```
2013-01-01    1.146781
2013-01-02    1.014293
2013-01-03    1.459981
2013-01-04    1.881415
2013-01-05    2.189263
2013-01-06    2.442146
Freq: D, dtype: float64
```

정렬이 필요하며, 차원이 다른 객체로 연산해보겠습니다. 또한, pandas는 지정된 차원을 따라 자동으로 브로드 캐스팅됩니다.

역자 주 : broadcast란 numpy에서 유래한 용어로, n차원이나 스칼라 값으로 연산을 수행할 때 도출되는 결과의 규칙을 설명하는 것을 의미합니다.

```
s = pd.Series([1,3,5,np.nan,6,8], index=dates).shift(2)
```

```
s
```

```
2013-01-01    NaN
2013-01-02    NaN
```

```
2013-01-03    1.0
2013-01-04    3.0
2013-01-05    5.0
2013-01-06    NaN
Freq: D, dtype: float64
```

```
df.sub(s, axis='index')
```

	A	B	C	D	F
2013-01-01	NaN	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	NaN	NaN	NaN
2013-01-03	-0.187816	-1.117943	-1.394338	4.0	1.0
2013-01-04	-2.165495	-2.970541	-2.456888	2.0	0.0
2013-01-05	-4.353663	-2.860703	-5.839317	0.0	-1.0
2013-01-06	NaN	NaN	NaN	NaN	NaN

Apply (적용)

데이터에 함수를 적용합니다.

```
df.apply(np.cumsum)
```

	A	B	C	D	F
2013-01-01	0.000000	0.000000	-0.412877	5	NaN
2013-01-02	-1.403231	-0.316784	0.378605	10	1.0
2013-01-03	-0.591046	-0.434727	-0.015733	15	3.0
2013-01-04	0.243458	-0.405268	0.527380	20	6.0
2013-01-05	0.889795	1.734029	-0.311938	25	10.0
2013-01-06	2.655891	1.707541	0.159182	30	15.0

```
df.apply(lambda x: x.max() - x.min())
```

```
A    3.169326
B    2.456081
C    1.630800
D    0.000000
F    4.000000
dtype: float64
```

Histogramming (히스토그래밍)

더 많은 내용은 Histogramming and Discretization (히스토그래밍과 이산화) 항목을 참조하세요.

```
s = pd.Series(np.random.randint(0, 7, size=10))
```

```
s
```

```
0    0
1    0
2    3
3    0
4    3
5    2
6    5
7    5
8    2
9    3
dtype: int32
```

```
s.value_counts()
```

```
3    3
0    3
5    2
2    2
dtype: int64
```

String Methods (문자열 메소드)

Series는 다음의 코드와 같이 문자열 처리 메소드 모음 (set)을 가지고 있습니다.
이 모음은 배열의 각 요소를 쉽게 조작할 수 있도록 만들어주는 문자열의 속성에 포함되어 있습니다.

문자열의 패턴 일치 확인은 기본적으로 정규 표현식을 사용하며, 몇몇 경우에는 항상 정규 표현식을 사용함에 유의하십시오.

좀 더 자세한 내용은 벡터화된 문자열 메소드 부분에서 확인할 수 있습니다.

```
s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])
```

```
s.str.lower()
```

```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

6. Merge (병합)

Concat (연결)

결합 (join) / 병합 (merge) 형태의 연산에 대한 인덱스, 관계 대수 기능을 위한 다양한 형태의 논리를 포함한 Series, 데이터프레임, Panel 객체를 손쉽게 결합할 수 있도록 하는 다양한 기능을 pandas 에서 제공합니다.

Merging 부분을 참조하세요.

concat()으로 pandas 객체를 연결합니다.

```
df = pd.DataFrame(np.random.randn(10, 4))

df
```

	0	1	2	3
0	-0.619545	0.438321	0.045161	-0.090580
1	-0.607351	-0.460920	1.086252	0.069311
2	-1.505874	-0.147020	0.762800	-1.948289
3	0.893628	-1.387833	1.010362	1.073543
4	0.007528	-0.380234	0.466893	0.189073
5	1.173880	-0.164525	1.020937	0.641751
6	-0.550514	-0.796966	-0.071519	-0.493431
7	0.250619	-1.676189	-1.722703	-0.639210
8	-0.119734	-0.599197	2.282847	0.403409
9	0.233205	-0.569511	-0.780681	0.654899

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

pd.concat(pieces)
```

	0	1	2	3
0	-0.619545	0.438321	0.045161	-0.090580
1	-0.607351	-0.460920	1.086252	0.069311
2	-1.505874	-0.147020	0.762800	-1.948289
3	0.893628	-1.387833	1.010362	1.073543
4	0.007528	-0.380234	0.466893	0.189073
5	1.173880	-0.164525	1.020937	0.641751
6	-0.550514	-0.796966	-0.071519	-0.493431
7	0.250619	-1.676189	-1.722703	-0.639210
8	-0.119734	-0.599197	2.282847	0.403409

	0	1	2	3
9	0.233205	-0.569511	-0.780681	0.654899

Join (결합)

SQL 방식으로 병합합니다. 데이터베이스 스타일 결합 부분을 참고하세요.

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

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

```
left
```

	key	lval
0	foo	1
1	foo	2

```
right
```

	key	rval
0	foo	4
1	foo	5

```
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

다른 예시입니다.

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

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

```
left
```

	key	lval
0	foo	1
1	bar	2

```
right
```

	key	rval
0	foo	4
1	bar	5

```
pd.merge(left, right, on= 'key')
```

	key	lval	rval
0	foo	1	4
1	bar	2	5

Append (추가)

데이터프레임에 행을 추가합니다. Appending 부분을 참조하세요.

```
df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
```

```
df
```

	A	B	C	D

	A	B	C	D
0	1.299919	-1.472544	-0.707566	-0.365660
1	0.187241	1.968653	0.824469	0.358518
2	-0.034656	0.829071	-0.378907	0.293894
3	-0.414106	2.328096	-1.367931	0.228907
4	0.914727	-0.052547	-0.583842	-0.231221
5	0.178399	-1.257682	0.560755	0.005913
6	0.507263	0.446625	-0.014416	0.345235
7	1.367934	0.292150	1.821888	-2.561016

```
s = df.iloc[3]

df.append(s, ignore_index=True)
```

	A	B	C	D
0	1.299919	-1.472544	-0.707566	-0.365660
1	0.187241	1.968653	0.824469	0.358518
2	-0.034656	0.829071	-0.378907	0.293894
3	-0.414106	2.328096	-1.367931	0.228907
4	0.914727	-0.052547	-0.583842	-0.231221
5	0.178399	-1.257682	0.560755	0.005913
6	0.507263	0.446625	-0.014416	0.345235
7	1.367934	0.292150	1.821888	-2.561016
8	-0.414106	2.328096	-1.367931	0.228907

7. Grouping (그룹화)

그룹화는 다음 단계 중 하나 이상을 포함하는 과정을 가리킵니다.

- 몇몇 기준에 따라 여러 그룹으로 데이터를 **분할 (splitting)**
- 각 그룹에 독립적으로 함수를 **적용 (applying)**
- 결과물들을 하나의 데이터 구조로 **결합 (combining)**

자세한 내용은 그룹화 부분을 참조하세요.

```
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)
    })

df
```

	A	B	C	D
0	foo	one	0.297352	-1.025981
1	bar	one	0.796546	1.127086
2	foo	two	1.645520	-0.659429
3	bar	three	1.328364	0.263898
4	foo	two	0.531388	0.859260
5	bar	two	2.099372	-0.720175
6	foo	one	-0.038247	-0.295680
7	foo	three	0.351615	0.543172

생성된 데이터프레임을 그룹화한 후 각 그룹에 sum() 함수를 적용합니다.

```
df.groupby('A').sum()
```

	C	D
A		
bar	4.224282	0.670809
foo	2.787628	-0.578657

여러 열을 기준으로 그룹화하면 계층적 인덱스가 형성됩니다. 여기에도 sum 함수를 적용할 수 있습니다.

```
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

8. Reshaping (변형)

계층적 인덱싱 및 변형 부분을 참조하세요.

Stack (스택)

```
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz',
                     'foo', 'foo', 'qux', 'qux'],
                    ['one', 'two', 'one', 'two',
                     'one', 'two', 'one', 'two']]))

index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])

df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=['A', 'B'])

df2  =  df[:4]
```

```
df2
```

		A	B
first	**second**		
bar	**one**	-0.726072	-1.436126
	two	0.211388	1.305562
baz	**one**	0.399729	-1.519716
	two	-0.278913	0.079106

stack() 메소드는 데이터프레임 열들의 계층을 "압축"합니다.

```
stacked = df2.stack()
```

```
stacked
```

```
first  second
bar    one     A   -0.726072
               B   -1.436126
       two     A    0.211388
               B    1.305562
baz    one     A    0.399729
               B   -1.519716
       two     A   -0.278913
               B    0.079106
dtype: float64
```

"Stack된" 데이터프레임 또는 (MultiIndex를 인덱스로 사용하는) Series인 경우, stack()의 역 연산은 unstack()이며, 기본적으로 **마지막 계층**을 unstack합니다.

```
stacked.unstack()
```

		A	B
first	**second**		
bar	**one**	-0.726072	-1.436126

		A	B
first	second		
	two	0.211388	1.305562
baz	one	0.399729	-1.519716
	two	-0.278913	0.079106

```
stacked.unstack(1)
```

	second	one	two
first			
bar	A	-0.726072	0.211388
	B	-1.436126	1.305562
baz	A	0.399729	-0.278913
	B	-1.519716	0.079106

```
stacked.unstack(0)
```

	first	bar	baz
second			
one	A	-0.726072	0.399729
	B	-1.436126	-1.519716
two	A	0.211388	-0.278913
	B	1.305562	0.079106

Pivot Tables (피봇 테이블)

피봇 테이블 부분을 참조하세요.

```
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)})
```

```
df
```

	A	B	C	D	E
0	one	A	foo	-0.776195	1.198841
1	one	B	foo	-0.317653	-1.110124
2	two	C	foo	1.848317	0.050875
3	three	A	bar	1.678460	-0.206626
4	one	B	bar	-0.509394	0.740372
5	one	C	bar	0.128912	-0.491783
6	two	A	foo	1.251120	-1.181534
7	three	B	foo	-0.292120	0.299805
8	one	C	foo	1.371375	-0.603625
9	one	A	bar	1.291114	-1.712893
10	two	B	bar	0.897307	-0.651877
11	three	C	bar	0.082510	-0.336216

이 데이터로부터 피봇 테이블을 매우 쉽게 생성할 수 있습니다.

```
pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
```

	C	bar	foo
A	**B**		
one	**A**	1.291114	-0.776195
	B	-0.509394	-0.317653
	C	0.128912	1.371375
three	**A**	1.678460	NaN
	B	NaN	-0.292120
	C	0.082510	NaN

	C	bar	foo
A	B		
two	A	NaN	1.251120
	B	0.897307	NaN
	C	NaN	1.848317

9. Time Series (시계열)

Pandas는 자주 일어나는 변환 (예시 : 5분마다 일어나는 데이터에 대한 2차 데이터 변환) 사이에 수행하는 리샘플링 연산을 위한 간단하고, 강력하며, 효율적인 함수를 제공합니다. 이는 재무 (금융) 응용에서 매우 일반적이지만 이에 국한되지는 않습니다. 시계열 부분을 참고하세요.

```
rng = pd.date_range('1/1/2012', periods=100, freq='S')

ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)

ts.resample('5Min').sum()
```

```
2012-01-01    25654
Freq: 5T, dtype: int32
```

시간대를 표현합니다.

```
rng = pd.date_range('3/6/2012 00:00', periods=5, freq='D')

ts = pd.Series(np.random.randn(len(rng)), rng)

ts
```

```
2012-03-06   -1.170229
2012-03-07    0.995390
2012-03-08   -2.433136
2012-03-09   -1.579099
```

```
2012-03-10   -0.139682
Freq: D, dtype: float64
```

```
ts_utc = ts.tz_localize('UTC')
```

```
ts_utc
```

```
2012-03-06 00:00:00+00:00   -1.170229
2012-03-07 00:00:00+00:00    0.995390
2012-03-08 00:00:00+00:00   -2.433136
2012-03-09 00:00:00+00:00   -1.579099
2012-03-10 00:00:00+00:00   -0.139682
Freq: D, dtype: float64
```

다른 시간대로 변환합니다.

```
ts_utc.tz_convert('US/Eastern')
```

```
2012-03-05 19:00:00-05:00   -1.170229
2012-03-06 19:00:00-05:00    0.995390
2012-03-07 19:00:00-05:00   -2.433136
2012-03-08 19:00:00-05:00   -1.579099
2012-03-09 19:00:00-05:00   -0.139682
Freq: D, dtype: float64
```

시간 표현 ↔ 기간 표현으로 변환합니다.

```
rng = pd.date_range('1/1/2012', periods=5, freq='M')
```

```
ts = pd.Series(np.random.randn(len(rng)), index=rng)
```

```
ts
```

```
2012-01-31    0.080979
2012-02-29    0.075085
2012-03-31   -0.076771
2012-04-30    0.819286
```

```
2012-05-31   -0.542812
Freq: M, dtype: float64
```

```
ps = ts.to_period()
```

```
ps
```

```
2012-01    0.080979
2012-02    0.075085
2012-03   -0.076771
2012-04    0.819286
2012-05   -0.542812
Freq: M, dtype: float64
```

```
ps.to_timestamp()
```

```
2012-01-01    0.080979
2012-02-01    0.075085
2012-03-01   -0.076771
2012-04-01    0.819286
2012-05-01   -0.542812
Freq: MS, dtype: float64
```

기간 ↔ 시간 변환은 편리한 산술 기능들을 사용할 수 있도록 만들어줍니다. 다음 예제에서,
우리는 11월에 끝나는 연말 결산의 분기별 빈도를 분기말 익월의 월말일 오전 9시로 변환합니다.

```
prng = pd.period_range('1990Q1', '2000Q4', freq='Q-NOV')
```

```
ts = pd.Series(np.random.randn(len(prng)), prng)
```

```
ts.index = (prng.asfreq('M', 'e') + 1).asfreq('H', 's') + 9
```

```
ts.head()
```

```
1990-03-01 09:00    0.018325
1990-06-01 09:00    1.330483
1990-09-01 09:00    1.122604
1990-12-01 09:00    0.288536
```

```
1991-03-01 09:00    1.161760
Freq: H, dtype: float64
```

10. Categoricals (범주화)

Pandas는 데이터프레임 내에 범주형 데이터를 포함할 수 있습니다. 범주형 소개 와 API 문서 부분을 참조하세요.

```
df = pd.DataFrame({"id":[1,2,3,4,5,6], "raw_grade":['a', 'b', 'b', 'a', 'a', '
```

가공하지 않은 성적을 범주형 데이터로 변환합니다.

```
df["grade"] = df["raw_grade"].astype("category")
```

```
df["grade"]
```

```
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): [a, b, e]
```

범주에 더 의미 있는 이름을 붙여주세요 (Series.cat.categories로 할당하는 것이 적합합니다).

```
df["grade"].cat.categories = ["very good", "good", "very bad"]
```

범주의 순서를 바꾸고 동시에 누락된 범주를 추가합니다 (Series.cat에 속하는 메소드는 기본적으로 새로운 Series를 반환합니다).

```
df["grade"] = df["grade"].cat.set_categories(["very bad", "bad", "medium", "go
```

```
df["grade"]
```

```
0    very good
1         good
2         good
3    very good
4    very good
5     very bad
Name: grade, dtype: category
Categories (5, object): [very bad, bad, medium, good, very good]
```

정렬은 사전 순서가 아닌, 해당 범주에서 지정된 순서대로 배열합니다.

역자 주 : 131번에서 very bad, bad, medium, good, very good 의 순서로 기재되어 있기 때문에 정렬 결과도 해당 순서대로 배열됩니다.

```
df.sort_values(by="grade")
```

	id	raw_grade	grade
5	6	e	very bad
1	2	b	good
2	3	b	good
0	1	a	very good
3	4	a	very good
4	5	a	very good

범주의 열을 기준으로 그룹화하면 빈 범주도 표시됩니다.

```
df.groupby("grade").size()
```

```
grade
very bad     1
bad          0
medium       0
good         2
```

```
very good    3
dtype: int64
```

11. Plotting (그래프)

Plotting 부분을 참조하세요.

```
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=
```

```
ts = ts.cumsum()
```

```
ts.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x9974fd0>
```

데이터프레임에서 plot() 메소드는 라벨이 존재하는 모든 열을 그릴 때 편리합니다.

```
df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
                  columns=['A', 'B', 'C', 'D'])
```

```
df = df.cumsum()
```

```
plt.figure(); df.plot(); plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x9b10208>
```

12. Getting Data In / Out (데이터 입 / 출력)

CSV

csv 파일에 씁니다.

```
df.to_csv('foo.csv')
```

csv 파일을 읽습니다.

```
pd.read_csv('foo.csv')
```

	Unnamed: 0	A	B	C	D
0	2000-01-01	-1.170941	0.688051	-0.383810	0.837035
1	2000-01-02	-1.325416	0.061442	-1.080497	0.281412
2	2000-01-03	-0.687276	0.916830	-2.839985	1.852432
3	2000-01-04	-1.288728	-0.242376	-3.791390	1.309750
4	2000-01-05	-0.937522	-0.779122	-5.202554	2.219908
5	2000-01-06	-2.136242	-0.693236	-6.256821	3.015780
6	2000-01-07	-1.412520	-2.517668	-5.712015	3.805923
7	2000-01-08	-0.049283	-1.716615	-7.405345	4.770589
8	2000-01-09	0.592184	-0.617583	-7.339519	3.694828
9	2000-01-10	1.378037	0.110647	-5.297688	3.467191
10	2000-01-11	2.474387	-0.083230	-4.153509	3.091714
11	2000-01-12	1.621882	0.016834	-7.112481	1.237205
12	2000-01-13	1.246822	-1.368471	-5.531885	0.338128
13	2000-01-14	1.978975	-0.527194	-5.112244	1.466986
14	2000-01-15	1.450342	0.450082	-4.731543	3.673504
15	2000-01-16	1.948955	1.059912	-6.297084	4.191426
16	2000-01-17	2.035009	2.273572	-6.637326	3.746256
17	2000-01-18	1.550255	3.503449	-6.578056	4.099211
18	2000-01-19	1.864008	4.212954	-7.183693	2.888867
19	2000-01-20	-0.075277	5.007933	-7.093309	4.735946
20	2000-01-21	-1.140021	6.814976	-6.716311	5.306338
21	2000-01-22	1.162872	6.534765	-6.110653	5.233384

	Unnamed: 0	A	B	C	D
22	2000-01-23	1.760717	5.714708	-4.054118	4.785703
23	2000-01-24	0.536450	5.849652	-3.765441	3.989357
24	2000-01-25	0.506539	6.692062	-4.821839	3.023168
25	2000-01-26	0.958889	6.568955	-4.259598	2.421934
26	2000-01-27	-0.172696	4.921587	-4.413335	2.314551
27	2000-01-28	0.165740	5.134014	-3.341849	1.133864
28	2000-01-29	0.093579	4.991880	-4.020020	2.081721
29	2000-01-30	0.424503	4.328352	-3.503704	2.303425
...	...	...	...	...	...
970	2002-08-28	-9.441308	-10.407354	60.908649	-3.372065
971	2002-08-29	-10.022290	-10.300304	59.022868	-4.729307
972	2002-08-30	-10.771862	-10.121161	59.190720	-4.776657
973	2002-08-31	-11.397452	-10.721580	59.955429	-4.837409
974	2002-09-01	-12.606183	-12.794658	60.073882	-6.897691
975	2002-09-02	-11.490124	-12.763907	60.835452	-6.677909
976	2002-09-03	-11.985902	-12.161442	60.848448	-6.248310
977	2002-09-04	-12.581613	-11.248297	60.504354	-6.408271
978	2002-09-05	-13.733525	-10.625722	58.903688	-5.788621
979	2002-09-06	-13.257015	-10.091945	58.625660	-4.776391
980	2002-09-07	-12.855639	-8.795421	59.073399	-4.017372
981	2002-09-08	-11.482862	-9.402805	58.042510	-3.530595
982	2002-09-09	-10.850022	-9.553852	57.214538	-4.053349
983	2002-09-10	-12.208049	-9.259484	58.237309	-3.971102
984	2002-09-11	-12.401630	-9.367988	58.999006	-3.615675
985	2002-09-12	-14.382630	-7.615701	61.633138	-2.822245
986	2002-09-13	-14.385503	-5.825456	62.643005	-2.631831
987	2002-09-14	-14.670608	-6.534945	63.046983	-2.521697
988	2002-09-15	-15.424981	-6.552120	64.461886	-3.493400

	Unnamed: 0	A	B	C	D
989	2002-09-16	-13.875303	-7.511547	64.741750	-4.255253
990	2002-09-17	-13.574444	-7.407093	64.003745	-3.096605
991	2002-09-18	-13.843896	-7.287694	64.860323	-3.211695
992	2002-09-19	-13.444606	-8.069938	66.156664	-3.679680
993	2002-09-20	-14.319578	-6.771972	64.871045	-4.633304
994	2002-09-21	-15.126463	-7.993281	65.080881	-3.497950
995	2002-09-22	-14.717619	-8.359075	65.765170	-5.577461
996	2002-09-23	-13.763743	-8.046417	66.821624	-5.256422
997	2002-09-24	-15.111257	-5.814779	66.104899	-6.185853
998	2002-09-25	-14.890142	-5.402545	65.420458	-5.578971
999	2002-09-26	-14.917314	-5.732310	63.944766	-6.181776

1000 rows × 5 columns

HDF5

HDFStores에 읽고 씁니다.

HDF5 Store에 씁니다.

```
df.to_hdf('foo.h5','df')
```

HDF5 Store에서 읽어옵니다.

```
pd.read_hdf('foo.h5','df')
```

	A	B	C	D
2000-01-01	-1.170941	0.688051	-0.383810	0.837035
2000-01-02	-1.325416	0.061442	-1.080497	0.281412
2000-01-03	-0.687276	0.916830	-2.839985	1.852432
2000-01-04	-1.288728	-0.242376	-3.791390	1.309750

	A	B	C	D
2000-01-05	-0.937522	-0.779122	-5.202554	2.219908
2000-01-06	-2.136242	-0.693236	-6.256821	3.015780
2000-01-07	-1.412520	-2.517668	-5.712015	3.805923
2000-01-08	-0.049283	-1.716615	-7.405345	4.770589
2000-01-09	0.592184	-0.617583	-7.339519	3.694828
2000-01-10	1.378037	0.110647	-5.297688	3.467191
2000-01-11	2.474387	-0.083230	-4.153509	3.091714
2000-01-12	1.621882	0.016834	-7.112481	1.237205
2000-01-13	1.246822	-1.368471	-5.531885	0.338128
2000-01-14	1.978975	-0.527194	-5.112244	1.466986
2000-01-15	1.450342	0.450082	-4.731543	3.673504
2000-01-16	1.948955	1.059912	-6.297084	4.191426
2000-01-17	2.035009	2.273572	-6.637326	3.746256
2000-01-18	1.550255	3.503449	-6.578056	4.099211
2000-01-19	1.864008	4.212954	-7.183693	2.888867
2000-01-20	-0.075277	5.007933	-7.093309	4.735946
2000-01-21	-1.140021	6.814976	-6.716311	5.306338
2000-01-22	1.162872	6.534765	-6.110653	5.233384
2000-01-23	1.760717	5.714708	-4.054118	4.785703
2000-01-24	0.536450	5.849652	-3.765441	3.989357
2000-01-25	0.506539	6.692062	-4.821839	3.023168
2000-01-26	0.958889	6.568955	-4.259598	2.421934
2000-01-27	-0.172696	4.921587	-4.413335	2.314551
2000-01-28	0.165740	5.134014	-3.341849	1.133864
2000-01-29	0.093579	4.991880	-4.020020	2.081721
2000-01-30	0.424503	4.328352	-3.503704	2.303425
...	...	...	...	...
2002-08-28	-9.441308	-10.407354	60.908649	-3.372065

	A	B	C	D
2002-08-29	-10.022290	-10.300304	59.022868	-4.729307
2002-08-30	-10.771862	-10.121161	59.190720	-4.776657
2002-08-31	-11.397452	-10.721580	59.955429	-4.837409
2002-09-01	-12.606183	-12.794658	60.073882	-6.897691
2002-09-02	-11.490124	-12.763907	60.835452	-6.677909
2002-09-03	-11.985902	-12.161442	60.848448	-6.248310
2002-09-04	-12.581613	-11.248297	60.504354	-6.408271
2002-09-05	-13.733525	-10.625722	58.903688	-5.788621
2002-09-06	-13.257015	-10.091945	58.625660	-4.776391
2002-09-07	-12.855639	-8.795421	59.073399	-4.017372
2002-09-08	-11.482862	-9.402805	58.042510	-3.530595
2002-09-09	-10.850022	-9.553852	57.214538	-4.053349
2002-09-10	-12.208049	-9.259484	58.237309	-3.971102
2002-09-11	-12.401630	-9.367988	58.999006	-3.615675
2002-09-12	-14.382630	-7.615701	61.633138	-2.822245
2002-09-13	-14.385503	-5.825456	62.643005	-2.631831
2002-09-14	-14.670608	-6.534945	63.046983	-2.521697
2002-09-15	-15.424981	-6.552120	64.461886	-3.493400
2002-09-16	-13.875303	-7.511547	64.741750	-4.255253
2002-09-17	-13.574444	-7.407093	64.003745	-3.096605
2002-09-18	-13.843896	-7.287694	64.860323	-3.211695
2002-09-19	-13.444606	-8.069938	66.156664	-3.679680
2002-09-20	-14.319578	-6.771972	64.871045	-4.633304
2002-09-21	-15.126463	-7.993281	65.080881	-3.497950
2002-09-22	-14.717619	-8.359075	65.765170	-5.577461
2002-09-23	-13.763743	-8.046417	66.821624	-5.256422
2002-09-24	-15.111257	-5.814779	66.104899	-6.185853
2002-09-25	-14.890142	-5.402545	65.420458	-5.578971

	A	B	C	D
2002-09-26	-14.917314	-5.732310	63.944766	-6.181776

1000 rows × 4 columns

Excel

MS Excel에 읽고 씁니다.

엑셀 파일에 씁니다.

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

엑셀 파일을 읽어옵니다.

```
pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
```

	A	B	C	D
2000-01-01	-1.170941	0.688051	-0.383810	0.837035
2000-01-02	-1.325416	0.061442	-1.080497	0.281412
2000-01-03	-0.687276	0.916830	-2.839985	1.852432
2000-01-04	-1.288728	-0.242376	-3.791390	1.309750
2000-01-05	-0.937522	-0.779122	-5.202554	2.219908
2000-01-06	-2.136242	-0.693236	-6.256821	3.015780
2000-01-07	-1.412520	-2.517668	-5.712015	3.805923
2000-01-08	-0.049283	-1.716615	-7.405345	4.770589
2000-01-09	0.592184	-0.617583	-7.339519	3.694828
2000-01-10	1.378037	0.110647	-5.297688	3.467191
2000-01-11	2.474387	-0.083230	-4.153509	3.091714
2000-01-12	1.621882	0.016834	-7.112481	1.237205
2000-01-13	1.246822	-1.368471	-5.531885	0.338128
2000-01-14	1.978975	-0.527194	-5.112244	1.466986

	A	B	C	D
2000-01-15	1.450342	0.450082	-4.731543	3.673504
2000-01-16	1.948955	1.059912	-6.297084	4.191426
2000-01-17	2.035009	2.273572	-6.637326	3.746256
2000-01-18	1.550255	3.503449	-6.578056	4.099211
2000-01-19	1.864008	4.212954	-7.183693	2.888867
2000-01-20	-0.075277	5.007933	-7.093309	4.735946
2000-01-21	-1.140021	6.814976	-6.716311	5.306338
2000-01-22	1.162872	6.534765	-6.110653	5.233384
2000-01-23	1.760717	5.714708	-4.054118	4.785703
2000-01-24	0.536450	5.849652	-3.765441	3.989357
2000-01-25	0.506539	6.692062	-4.821839	3.023168
2000-01-26	0.958889	6.568955	-4.259598	2.421934
2000-01-27	-0.172696	4.921587	-4.413335	2.314551
2000-01-28	0.165740	5.134014	-3.341849	1.133864
2000-01-29	0.093579	4.991880	-4.020020	2.081721
2000-01-30	0.424503	4.328352	-3.503704	2.303425
...	...	...	...	...
2002-08-28	-9.441308	-10.407354	60.908649	-3.372065
2002-08-29	-10.022290	-10.300304	59.022868	-4.729307
2002-08-30	-10.771862	-10.121161	59.190720	-4.776657
2002-08-31	-11.397452	-10.721580	59.955429	-4.837409
2002-09-01	-12.606183	-12.794658	60.073882	-6.897691
2002-09-02	-11.490124	-12.763907	60.835452	-6.677909
2002-09-03	-11.985902	-12.161442	60.848448	-6.248310
2002-09-04	-12.581613	-11.248297	60.504354	-6.408271
2002-09-05	-13.733525	-10.625722	58.903688	-5.788621
2002-09-06	-13.257015	-10.091945	58.625660	-4.776391
2002-09-07	-12.855639	-8.795421	59.073399	-4.017372

	A	B	C	D
2002-09-08	-11.482862	-9.402805	58.042510	-3.530595
2002-09-09	-10.850022	-9.553852	57.214538	-4.053349
2002-09-10	-12.208049	-9.259484	58.237309	-3.971102
2002-09-11	-12.401630	-9.367988	58.999006	-3.615675
2002-09-12	-14.382630	-7.615701	61.633138	-2.822245
2002-09-13	-14.385503	-5.825456	62.643005	-2.631831
2002-09-14	-14.670608	-6.534945	63.046983	-2.521697
2002-09-15	-15.424981	-6.552120	64.461886	-3.493400
2002-09-16	-13.875303	-7.511547	64.741750	-4.255253
2002-09-17	-13.574444	-7.407093	64.003745	-3.096605
2002-09-18	-13.843896	-7.287694	64.860323	-3.211695
2002-09-19	-13.444606	-8.069938	66.156664	-3.679680
2002-09-20	-14.319578	-6.771972	64.871045	-4.633304
2002-09-21	-15.126463	-7.993281	65.080881	-3.497950
2002-09-22	-14.717619	-8.359075	65.765170	-5.577461
2002-09-23	-13.763743	-8.046417	66.821624	-5.256422
2002-09-24	-15.111257	-5.814779	66.104899	-6.185853
2002-09-25	-14.890142	-5.402545	65.420458	-5.578971
2002-09-26	-14.917314	-5.732310	63.944766	-6.181776

1000 rows × 4 columns

13. Gotchas (잡았다!)

연산 수행 시 다음과 같은 예외 상황을 볼 수도 있습니다.

```
if pd.Series([False, True, False]):
    print("I was true")
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-153-9cae3ab0f79f> in <module>()
----> 1 if pd.Series([False, True, False]):
      2     print("I was true")


C:\Users\Admin\Anaconda3\lib\site-packages\pandas\core\generic.py in __nonzero
    951         raise ValueError("The truth value of a {0} is ambiguous. "
    952                          "Use a.empty, a.bool(), a.item(), a.any() or
--> 953                          .format(self.__class__.__name__))
    954
    955     __bool__ = __nonzero__


ValueError: The truth value of a Series is ambiguous. Use a.empty, a.bool(), a
```

이러한 경우에는 any(), all(), empty 등을 사용해서 무엇을 원하는지를 선택 (반영)해주어야 합니다.

```
if pd.Series([False, True, False])is not None:
      print("I was not None")
```

위에 대한 설명과 자세한 내용은 비교 부분을 참조하세요.

Gotchas 부분도 참조하세요.
