Pandas1

September 27, 2020

아래 강의 노트는 Python for data Analysis 책4장을 기반으로 번역 및 편집하여 페이지 구성함

1 pandas 시작하기

- **공식홈페이지 :** https://pandas.pydata.org/
- **문서** https://pandas.pydata.org/

`scipy` , `numpy`, `statsmodels`, `scikit-learn`와 `matplotlib`와 함께 사용 하는 경우가 흔함

- `numpy` : 단일 산술 배열 데이터를 다루는데 특화
- `pandas` : 표 형식의 데이터나 다양한 형태의 데이터를 다루는데 초점

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.

[1]:
```
import pandas as pd
```

`series`와 `Dataframe`은 로컬 네임스페이스로 임포트 하는 것이 편하기 때문에 다음과 같이 사용

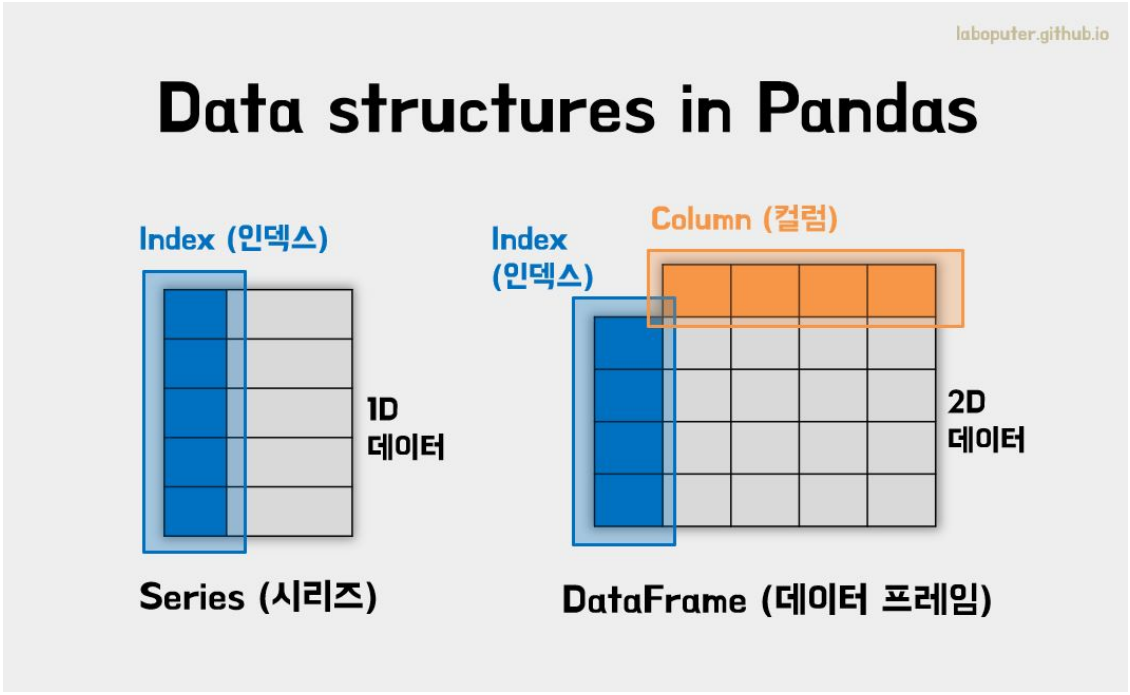
[2]:
```
from pandas import Series, DataFrame
```

[3]:
```
import numpy as np
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
np.set_printoptions(precision=4, suppress=True)
```

1.1 pandas 자료구조

1.1.1 Series

series 는 객체를 담을 수 있는 1차원 배열같은 자료구조 **색인** 이라고하는 배열의 데이터와 연관된 이



름을 갖음

출처 :https://laboputer.github.io/assets/img/ml/python/pandas/1.JPG

[4]:
```
import numpy as np
import pandas as pd
obj = pd.Series([4, 7, -5, 3])
print(obj)
nparr = np.array([[4, 7, -5, 3]])
nparr
```

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

[4]:
```
array([[ 4,  7, -5,  3]])
```

Series의 배열과 색인 객체는 각각 value, index 속성을 통해 얻기 가능

[5]:
```
print(obj.values)
print(obj.index)  # like range(4)
```

```
[ 4  7 -5  3]
RangeIndex(start=0, stop=4, step=1)
```

색인을 지정하여 Series객체를 생성할 때

[6]:
```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(obj2)
print(obj2.index)
```

```
d    4
b    7
a   -5
c    3
dtype: int64
Index(['d', 'b', 'a', 'c'], dtype='object')
```

[7]:
```
obj2['a']
obj2['d'] = 6
OB=obj2[['c', 'a', 'd']]
OB
obj2
```

[7]:
```
d    6
b    7
a   -5
c    3
dtype: int64
```

Series는 고정길이의 정렬된 사전형 색인값에 데이터 값을 매핑 하고 있기때문에, 파이썬의 사전형과 비슷

[8]:
```
import numpy as np
print(obj2[obj2 > 0])
print(obj2 * 2)
np.exp(obj2)
```

```
d    6
b    7
c    3
dtype: int64
d    12
b    14
a   -10
c     6
dtype: int64
```

[8]:
```
d     403.428793
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

[9]:
```
'b' in obj2
'e' in obj2
```

[9]:
```
False
```

파이썬의 Dictionary 타입을 Series로 변형가능

[10]:
```
sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
obj3
```

[10]:
```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

사전 객체만으로 Series객체를 생성하면 생성된 Series객체의 색인에는 사전의 키 값의 순서대로 입력

색인을 지정하고 싶다면 원하는 순서대로 색인을 직접 넘기기 가능 * 아래의 예는 sdata에 있는 값중 3개만 확인 가능 * California는 값을 찾을수 없기 때문에 NaN 표시 * utah는 색인에 없기 때문에 실행결과에서 빠짐

[11]:
```
states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

[11]:
```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

isnull,notnull누락된 데이터를 찾을 때 사용

[12]:
```
pd.isnull(obj4)
pd.notnull(obj4)
```

[12]:
```
California    False
Ohio           True
Oregon         True
Texas          True
dtype: bool
```

[13]:
```
obj4.isnull()
```

[13]:
```
California     True
Ohio          False
Oregon        False
```

```
Texas         False
dtype: bool
```

Series에서 산술 기능은 색인과 라벨로 자동 정렬 데이터 베이스 에서 join연산과 비슷하게 작동

[14]:
```
print(obj3)
print(obj4)
obj4 + obj3
```

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

[14]:
```
California         NaN
Ohio           70000.0
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

name 속성을 이용하여 시리즈 데이터에 이름 index.name 속성으로 시리즈의 인덱스에도 이름

[15]:
```
obj4.name = 'population'
obj4.index.name = 'state'
obj4
```

[15]:
```
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

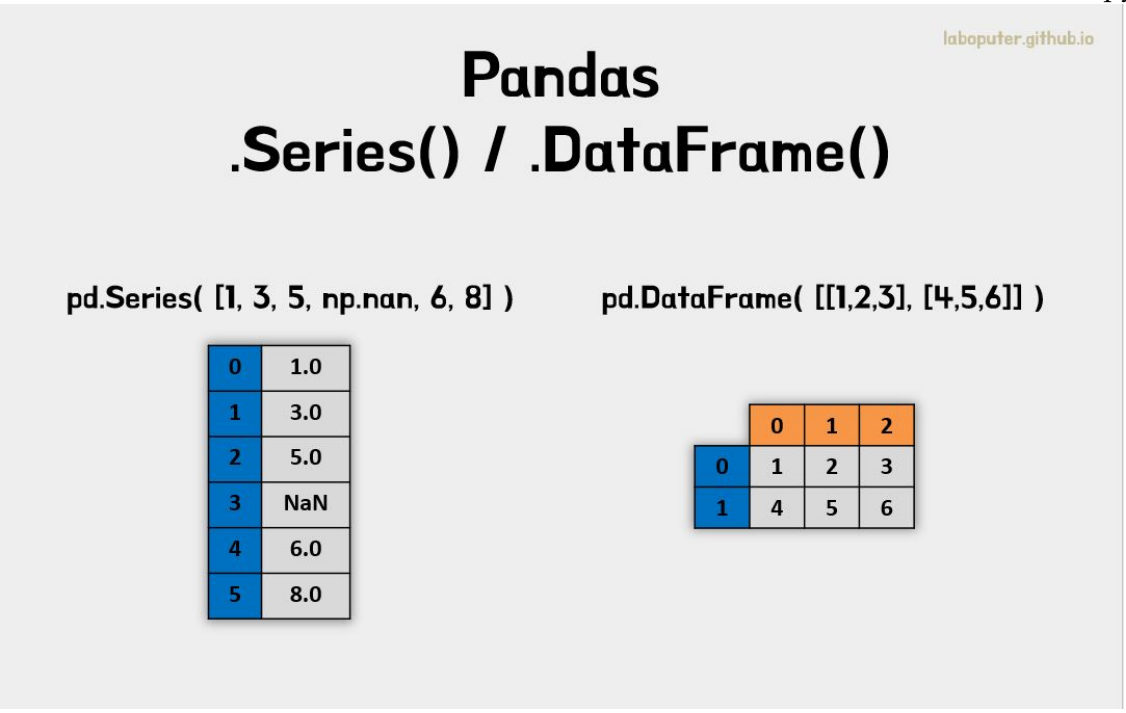
[16]:
```
obj
obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
obj
```

[16]:
```
Bob      4
Steve    7
Jeff    -5
Ryan     3
```

```
dtype: int64
```

1.1.2 DataFrame

특징 * DataFrame은 스프레드 시트 형식의 자료구조이고 여러개의 컬럼이 있는데 각 컬럼은 서로 다른 종류의 값(숫자, 문자열, 불리언)을 담을 수 있음 * DataFrame은 로우와 컬럼에 대한 색인을 갖고 있는데 Series객체를 담고 있는 파이썬의 사전으로 생각 가능 * DataFrame은 물리적으로는 2차원 이지미만 계층적 색인을 이용하여 고차원 데이터 표현 가능 가장 흔하게 사용되는 표현 방법은 아래와 같이 리스트에 담긴 사전이용하거나 numpy배열이용



출처 :https://laboputer.github.io/assets/img/ml/python/pandas/2.JPG

[17]:
```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

[18]:
```
frame
```

[18]:

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

head메서드를 이용하여 처음 5개의 행만 출력 가능

[19]:
```
frame.tail()
```

[19]:
```
    state  year  pop
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
5  Nevada  2003  3.2
```

columns를 지정하면 원하는 순서를 가진 DataFrame객체 생성

[20]:
```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

[20]:
```
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
5  2003  Nevada  3.2
```

Series와 마찬가지로 사전에 없는 값을 넘기면 결측치로 저장

[21]:
```
frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                      index=['one', 'two', 'three', 'four',
                             'five', 'six'])
print(frame2)
frame2.index
```

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

[21]:
```
Index(['one', 'two', 'three', 'four', 'five', 'six'], dtype='object')
```

DataFrame은 Series처럼 사전 형식의 표기법이나 속성 형식으로 표기 가능

[22]:
```
frame2['state'] # 사전 형식
frame2.year     #속성 형식
```

[22]:
```
one      2000
two      2001
three    2002
four     2001
```

```
five     2002
six      2003
Name: year, dtype: int64
```

행의 위치나 loc속성을 이용하여 이름을 통해 접근 가능

[23]:
```
frame2.loc['three']
```

[23]:
```
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

[24]:
```
frame2['debt'] = 16.5
print(frame2)
frame2['debt'] = np.arange(6.)
print("\n\n",frame2)
```

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


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

[25]:
```
val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
frame2['debt'] = val
frame2
```

[25]:
```
       year   state  pop  debt
one    2000    Ohio  1.5   NaN
two    2001    Ohio  1.7  -1.2
three  2002    Ohio  3.6   NaN
four   2001  Nevada  2.4  -1.5
five   2002  Nevada  2.9  -1.7
six    2003  Nevada  3.2   NaN
```

frame2.state이 'Ohio'인지 아닌즈에 대한 불리언 값을 담고 있는 'eastern'이라는 새로운 컬럼 값을 생성

[26]:
```
frame2['eastern'] = frame2.state == 'Ohio'
frame2
```

[26]:

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

del예약어를 이용하여 컬럼 삭제 가능

[27]:
```
del frame2['eastern']
frame2.columns
```

[27]:
```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

중첩된 사전을 DataFrame에 넘기면 바깥에 있는 사전의 키는 컬럼이 되고 안에 있는 키는 로우가 된다.

[28]:
```
pop = {'Nevada': {2001: 2.4, 2002: 2.9},
       'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

[29]:
```
frame3 = pd.DataFrame(pop)
frame3
```

[29]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2000	NaN	1.5

Numpy배열과 유사한 문법으로 데이터 전치 가능

[30]:
```
frame3.T
```

[30]:

	2001	2002	2000
Nevada	2.4	2.9	NaN
Ohio	1.7	3.6	1.5

[31]:
```
pd.DataFrame(pop, index=[2001, 2002, 2003])
```

[31]:

	Nevada	Ohio
2001	2.4	1.7
2002	2.9	3.6
2003	NaN	NaN

[32]:
```
pdata = {'Ohio': frame3['Ohio'][:-1],
         'Nevada': frame3['Nevada'][:2]}
pd.DataFrame(pdata)
```

[32]:
```
      Ohio  Nevada
2001   1.7     2.4
2002   3.6     2.9
```

[33]:
```
frame3.index.name = 'year'; frame3.columns.name = 'state'
frame3
```

[33]:
```
state  Nevada  Ohio
year
2001      2.4   1.7
2002      2.9   3.6
2000      NaN   1.5
```

Series와 유사하게 values속성은 DataFrame에 저장된 데이터를 2차원 배열로 반환

[34]:
```
frame3.values
```

[34]:
```
array([[2.4, 1.7],
       [2.9, 3.6],
       [nan, 1.5]])
```

[35]:
```
frame2.values
```

[35]:
```
array([[2000, 'Ohio', 1.5, nan],
       [2001, 'Ohio', 1.7, -1.2],
       [2002, 'Ohio', 3.6, nan],
       [2001, 'Nevada', 2.4, -1.5],
       [2002, 'Nevada', 2.9, -1.7],
       [2003, 'Nevada', 3.2, nan]], dtype=object)
```

1.1.3 색인 객체 (Index Objects)

색인 객체는 표형식의 데이터에서 각 행과 열에 대한 이름과 다른 메타데이터를 저장하는 객체 Series나 DataFrame객체를 생성할 때 사용되는 배열이나 다른 순차적인 이름은 내부적으로 색인으로 변환

[36]:
```
obj = pd.Series(range(3), index=['a', 'b', 'c'])
index = obj.index
index
index[1:]
```

[36]:
```
Index(['b', 'c'], dtype='object')
```

색인 객체는 변경 불가능

index[1] = 'd' # TypeError

[37]:
```
labels = pd.Index(np.arange(3))
print(labels)
obj2 = pd.Series([1.5, -2.5, 0], index=labels)
print("\n\n",obj2)
print("\n\n",obj2.index is labels)
```

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


 0    1.5
1   -2.5
2    0.0
dtype: float64


 True
```

[38]:
```
print(frame3)
print("\n\n",frame3.columns)
print("\n\n",'Ohio' in frame3.columns)
print("\n\n", 2003 in frame3.index)
```

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


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


 True


 False
```

파이썬의 집합과는 달리 pandas의 인덱스는 중복 값을 허용.

[39]:
```
dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])
print(dup_labels)
dl = pd.Series([1,2,3,4],index= dup_labels)
print("\n\n",dl)
print("\n\n",dl['foo'])
```

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

```
 foo    1
foo    2
bar    3
bar    4
dtype: int64
```

```
 foo    1
foo    2
dtype: int64
```

1.2 핵심 기능

1.2.1 재색인 기능

새로운 색인데 맞도록 객체를 새로 생성
Series객체에 대해서 reindex를 호출하면 데이터를 새로운 색인데 맞게 재배열하고 존재하지 않는 색이 있다면 NaN을 추가

[40]:
```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
obj
```

[40]:
```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

[41]:
```
obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
obj2
```

[41]:
```
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

시계열 같은 순자적인 데이터 재색인할 때 값을 보간 하거나 채워 넣어야 할 경우, method 옵션을 이용해 이를 해결 가능 하며 ffill같은 메서드를 이용하여 누락된 값을 직전의 값으로 채워 넣을 수 있다.

[42]:
```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
obj3
obj3.reindex(range(6), method='ffill')
```

[42]:
```
0      blue
1      blue
2    purple
3    purple
```

```
4    yellow
5    yellow
dtype: object
```

[43]:
```
obj3.reindex(range(6))
```

[43]:
```
0      blue
1       NaN
2    purple
3       NaN
4    yellow
5       NaN
dtype: object
```

reindex는 색인과 컬럼 또는 둘다 변경 가능

[44]:
```
import pandas as pd
import numpy as np
frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['Ohio', 'Texas', 'California'])

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

[44]:

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

컬럼은 reindex 예약어를 사용해서 재색인

[45]:
```
states = ['Texas', 'Utah', 'California']
frame.reindex(columns=states)
```

[45]:

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

[46]:
```
frame
```

[46]:

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

[47]:
```
frame.loc[['a','c']]
```

[47]:
```
   Ohio  Texas  California
a     0      1           2
c     3      4           5
```

1.2.2 하나의 로우나 컬럼 삭제 하기

drop 메서들를 사용하여 선택한 값들이 삭제된 새로운 객체 얻기 가능

주의: drop함수는 특정 행 또는 열을 drop하고난 DataFrame을 반환한다. 즉, 반환을 받지 않으면 기존의 DataFrame은 그대로이다. 아니면, inplace=True라는 인자를 추가하여, 반환을 받지 않고서도 기존의 DataFrame이 변경되도록 한다.

[48]:
```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
obj
new_obj = obj.drop('c')
new_obj
obj.drop(['d', 'c'],axis=0)
```

[48]:
```
a    0.0
b    1.0
e    4.0
dtype: float64
```

[49]:
```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
```

[49]:
```
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

[50]:
```
data.drop(['Colorado', 'Ohio'])
```

[50]:
```
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
```

[51]:
```
data.drop('two', axis=1)
data.drop(['two', 'four'], axis='columns')
```

[51]:
```
          one  three
Ohio        0      2
Colorado    4      6
```

```
Utah        8     10
New York   12     14
```

[52]:
```
obj.drop('c', inplace=True)
obj
```

[52]:
```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

1.2.3 색인하기, 선택하기, 거르기

Numpy 의 색인과 pandas색인은 유사하게 동작하지만 정수가 아니어도 된다

[53]:
```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
display(obj)
obj['b']
obj[1]
obj[2:4]
obj[['b', 'a', 'd']]
obj[[1, 3]]
obj[obj < 2]
```

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

[53]:
```
a    0.0
b    1.0
dtype: float64
```

라벨 이름으로 슬라이싱 하면 시작점과 끝점을 포함한다는 것이 일반 파이썬의 슬라이싱과 다른 점

[54]:
```
obj['b':'c']
```

[54]:
```
b    1.0
c    2.0
dtype: float64
```

슬라이싱 문법으로 선택된 영역에 값을 대입하는 것은 생각하는 대로 동작 한다.

[55]:
```
obj['b':'c'] = 5
obj
```

[55]:
```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

색인으로 DataFrame 에서 하나 이상의 컬럼 값을 가져 올 수 있다.

[56]:
```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
data['two']
data[['three', 'one']]
```

[56]:

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

슬라이싱으로 로우를 선택하거나, 불리언 배열로 로우 선택 가능
[ ]연산자에 단일 값을 넘기거나 리스트를 넘겨서 여러 컬럼 값을 선택 할 수 있음

[57]:
```
data[:2]
data[data['three'] > 5]
```

[57]:

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

스칼라 값으로 비교시에는 불리언 DataFrame을 사용해서 값을 선택

[58]:
```
data < 5
data[data < 5] = 0
data
```

[58]:

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

loc and iloc으로 선택하기

[59]:
```
data
```

[59]:
```
          one  two  three  four
Ohio        0    0      0     0
Colorado    0    5      6     7
Utah        8    9     10    11
New York   12   13     14    15
```

[60]:
```
data.loc['Colorado', ['two', 'three']]
```

[60]:
```
two      5
three    6
Name: Colorado, dtype: int64
```

[61]:
```
data.iloc[2, [3, 0, 1]]
```

[61]:
```
four    11
one      8
two      9
Name: Utah, dtype: int64
```

[62]:
```
data.iloc[2]
```

[62]:
```
one       8
two       9
three    10
four     11
Name: Utah, dtype: int64
```

[63]:
```
data.iloc[[1, 2], [3, 0, 1]]
```

[63]:
```
          four  one  two
Colorado     7    0    5
Utah        11    8    9
```

[64]:
```
data.loc[:'Utah', 'two']
data.iloc[:, :3][data.three > 5]
```

[64]:
```
          one  two  three
Colorado    0    5      6
Utah        8    9     10
New York   12   13     14
```

1.2.4 정수 색인

아래의 예에서 라벨 색인이 0,1,2를 포함하는 경우 사용자가 라벨 색인으로 선택하려는 것인지 정수 색인으로 선택하려는 것인지 추측하기 쉽지 않음

[65]:
```
ser = pd.Series(np.arange(3.))
ser
#ser[-1]
```

[65]:
```
0    0.0
1    1.0
2    2.0
dtype: float64
```

[66]:
```
ser = pd.Series(np.arange(3.))
```

[67]:
```
ser
```

[67]:
```
0    0.0
1    1.0
2    2.0
dtype: float64
```

반면 아래와 같이 정수 색인이 아닌 경우 이러한 모호함은 사라짐

[68]:
```
ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
ser2[-1]
```

[68]:
```
2.0
```

일관성 유지를 위하여 정수값을 담고 있는 축 색인이 있다면 우선적으로 라벨을 먼저 찾음 저금더 세밀하게 사용하고 싶다면 라벨에 대해서는 `loc`를 사용하고 정수 색인에 대해서는 `iloc`을 사용

[69]:
```
ser[:1]
ser.loc[:1]
ser.iloc[:1]
```

[69]:
```
0    0.0
dtype: float64
```

1.2.5 산술 연산과 데이터 정렬

pandas에서 가장 중요한 기능 중 하나는 다른 색인을 가지고 있는 객체 간의 산술 연산 객체를 더할때 짝이 맞지 않는 색인이 있다면 결과에 두색인이 통합
서로 겹치는 색인이 없는 경우 데이터는 NA 값

[70]:
```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
               index=['a', 'c', 'e', 'f', 'g'])
s1
```

[70]:
```
a    7.3
c   -2.5
```

```
d    3.4
e    1.5
dtype: float64
```

[71]:
```
s2
```

[71]:
```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

[72]:
```
s1 + s2
```

[72]:
```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

Dataframe의 경우 정렬은 로우와 컬럼 모두 적용

[73]:
```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),
                   index=['Ohio', 'Texas', 'Colorado'])
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),
                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])
df1
```

[73]:

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

[74]:
```
df2
```

[74]:

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

c 와 e 컬럼이 양쪽 DataFrame객체에 존재하지 않으므로 결과에서 모두 없는 값으로 나타남
로우 역시 마찬가지로 양쪽에 다 존재하지 않는 라벨에 대해서는 없는 값으로 나타남

[75]:
```
df1 + df2
```

[75]:
```
            b   c     d   e
Colorado  NaN NaN   NaN NaN
Ohio      3.0 NaN   6.0 NaN
Oregon    NaN NaN   NaN NaN
Texas     9.0 NaN  12.0 NaN
Utah      NaN NaN   NaN NaN
```

[76]:
```
df1 = pd.DataFrame({'A': [1, 2]})
df2 = pd.DataFrame({'B': [3, 4]})
df1
df2
df1 - df2
```

[76]:
```
    A   B
0 NaN NaN
1 NaN NaN
```

산술 연산 메서드를 채워 넣을 값 지정하기

[ ]:
```

```

[77]:
```
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
                   columns=list('abcd'))
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                   columns=list('abcde'))
df1
```

[77]:
```
     a    b     c     d
0  0.0  1.0   2.0   3.0
1  4.0  5.0   6.0   7.0
2  8.0  9.0  10.0  11.0
```

[78]:
```
df2.loc[1, 'b'] = np.nan
df2
```

[78]:
```
      a     b     c     d     e
0   0.0   1.0   2.0   3.0   4.0
1   5.0   NaN   7.0   8.0   9.0
2  10.0  11.0  12.0  13.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

겹쳐지지 않는 부분은 NA

[79]:
```
df1 + df2
```

[79]:
```
      a     b     c     d   e
0   0.0   2.0   4.0   6.0 NaN
1   9.0   NaN  13.0  15.0 NaN
```

```
2  18.0  20.0  22.0  24.0 NaN
3   NaN   NaN   NaN   NaN NaN
```

df1에 add 메서드를 사용하고 df2와 fill_value 값을 인자로 전달

[80]:
```
df1.add(df2, fill_value=0)
```

[80]:
```
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

r로 시작하는 메서더는 계산 인자를 뒤집어 계산

[81]:
```
1 / df1
df1.rdiv(1)
```

[81]:
```
       a         b         c         d
0    inf  1.000000  0.500000  0.333333
1  0.250  0.200000  0.166667  0.142857
2  0.125  0.111111  0.100000  0.090909
```

[82]:
```
df1.reindex(columns=df2.columns, fill_value=0)
```

[82]:
```
     a    b     c     d  e
0  0.0  1.0   2.0   3.0  0
1  4.0  5.0   6.0   7.0  0
2  8.0  9.0  10.0  11.0  0
```

DataFrame 과 Series 간의 연산 브로드 캐스팅을 이용하여 빼기

[83]:
```
arr = np.arange(12.).reshape((3, 4))
arr
```

[83]:
```
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])
```

[84]:
```
arr[0]
```

[84]:
```
array([0., 1., 2., 3.])
```

[85]:
```
arr - arr[0]
```

[85]:
```
array([[0., 0., 0., 0.],
       [4., 4., 4., 4.],
       [8., 8., 8., 8.]])
```

[86]:
```
frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                     columns=list('bde'),
                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
```

[86]:

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

[87]:
```
series = frame.iloc[0]
series
```

[87]:
```
b    0.0
d    1.0
e    2.0
Name: Utah, dtype: float64
```

[88]:
```
frame - series
```

[88]:

	b	d	e
Utah	0.0	0.0	0.0
Ohio	3.0	3.0	3.0
Texas	6.0	6.0	6.0
Oregon	9.0	9.0	9.0

기본적으로 DataFrame과 Series간의 산술 연산은 Series의 색인을 DataFrame의 컬럼에 맞추고 아래로우로 전파

[89]:
```
series2 = pd.Series(range(3), index=['b', 'e', 'f'])
frame + series2
```

[89]:

	b	d	e	f
Utah	0.0	NaN	3.0	NaN
Ohio	3.0	NaN	6.0	NaN
Texas	6.0	NaN	9.0	NaN
Oregon	9.0	NaN	12.0	NaN

[90]:
```
frame
```

[90]:

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

[91]:
```
series3 = frame['d']
series3
```

[91]:
```
Utah       1.0
Ohio       4.0
Texas      7.0
Oregon    10.0
Name: d, dtype: float64
```

axis='index' 나axis=0 은 DataFrame의 로우를 따라 연산을 수행

[92]:
```
frame.sub(series3, axis='index')
```

[92]:
```
          b    d    e
Utah   -1.0  0.0  1.0
Ohio   -1.0  0.0  1.0
Texas  -1.0  0.0  1.0
Oregon -1.0  0.0  1.0
```

1.2.6 함수 적용과 매핑

[93]:
```
frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])
frame
```

[93]:
```
                b         d         e
Utah   -0.204708  0.478943 -0.519439
Ohio   -0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189 -1.296221
```

[94]:
```
np.abs(frame)
```

[94]:
```
                b         d         e
Utah    0.204708  0.478943  0.519439
Ohio    0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189  1.296221
```

함수 f는 Sereies의 최대값과 최소값 차이를 계산 하는 함수

[95]:
```
f = lambda x: x.max() - x.min()
frame.apply(f)
```

[95]:
```
b    1.802165
d    1.684034
e    2.689627
```

```
dtype: float64
```

함수의 인자로 axis='columns'를 넘기면 각 행에 대해 한번씩 수행

[96]:
```
frame.apply(f, axis='columns')
```

[96]:
```
Utah      0.998382
Ohio      2.521511
Texas     0.676115
Oregon    2.542656
dtype: float64
```

[97]:
```
def f(x):
    return pd.Series([x.min(), x.max()], index=['min', 'max'])
frame.apply(f)
```

[97]:
```
             b         d         e
min -0.555730  0.281746 -1.296221
max  1.246435  1.965781  1.393406
```

frame객체에서 실수값을 문자열 포맷으로 변환하고 싶다면 applymap을 이용해서 사용

[98]:
```
format = lambda x: '%.2f' % x
frame.applymap(format)
```

[98]:
```
             b     d      e
Utah     -0.20  0.48  -0.52
Ohio     -0.56  1.97   1.39
Texas     0.09  0.28   0.77
Oregon    1.25  1.01  -1.30
```

[99]:
```
frame['e'].map(format)
```

[99]:
```
Utah      -0.52
Ohio       1.39
Texas      0.77
Oregon    -1.30
Name: e, dtype: object
```

1.2.7 정렬과 순위

[100]:
```
obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
obj.sort_index()
```

[100]:
```
a    1
b    2
c    3
```

```
d    0
dtype: int64
```

[101]:
```
frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                     index=['three', 'one'],
                     columns=['d', 'a', 'b', 'c'])
frame.sort_index()
frame.sort_index(axis=1)
```

[101]:
```
       a  b  c  d
three  1  2  3  0
one    5  6  7  4
```

[102]:
```
frame.sort_index(axis=1, ascending=False)
```

[102]:
```
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

[103]:
```
obj = pd.Series([4, 7, -3, 2])
obj.sort_values()
```

[103]:
```
2   -3
3    2
0    4
1    7
dtype: int64
```

[104]:
```
obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
obj.sort_values()
```

[104]:
```
4   -3.0
5    2.0
0    4.0
2    7.0
1    NaN
3    NaN
dtype: float64
```

[105]:
```
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
frame
frame.sort_values(by='b')
```

[105]:
```
   b  a
2 -3  0
3  2  1
0  4  0
```

```
1  7  1
```

[106]:
```
frame.sort_values(by=['a', 'b'])
```

[106]:
```
   b  a
2 -3  0
0  4  0
3  2  1
1  7  1
```

[107]:
```
obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
obj.rank()
```

[107]:
```
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

[108]:
```
obj.rank(method='first')
```

[108]:
```
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

[109]:
```
# Assign tie values the maximum rank in the group
obj.rank(ascending=False, method='max')
```

[109]:
```
0    2.0
1    7.0
2    2.0
3    4.0
4    5.0
5    6.0
6    4.0
dtype: float64
```

[110]:
```
frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                      'c': [-2, 5, 8, -2.5]})
```

```
frame
frame.rank(axis='columns')
```

[110]:
```
     b    a    c
0  3.0  2.0  1.0
1  3.0  1.0  2.0
2  1.0  2.0  3.0
3  3.0  2.0  1.0
```

1.2.8 Axis Indexes with Duplicate Labels

[111]:
```
obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
obj
```

[111]:
```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

[112]:
```
obj.index.is_unique
```

[112]:
```
False
```

[113]:
```
obj['a']
obj['c']
```

[113]:
```
4
```

[114]:
```
df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
df
df.loc['b']
```

[114]:
```
          0         1         2
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228
```

1.3 기술 통계 계산과 요약

pandas는 일반적인 수학 메서드와 통계 메서드를 가지고 있다. 행이나 열에서의 단일 값(합 또는 평균)을 구하는 축소 또는 요약 통계가 가능

[115]:
```
import pandas as pd
import numpy as np
```

[116]:
```
df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
                   [np.nan, np.nan], [0.75, -1.3]],
                  index=['a', 'b', 'c', 'd'],
                  columns=['one', 'two'])
df
```

[116]:
```
     one  two
a  1.40  NaN
b  7.10 -4.5
c   NaN  NaN
d  0.75 -1.3
```

[117]:
```
df.sum()
```

[117]:
```
one    9.25
two   -5.80
dtype: float64
```

[118]:
```
df.sum(axis='columns')
```

[118]:
```
a    1.40
b    2.60
c    0.00
d   -0.55
dtype: float64
```

[119]:
```
df.mean(axis='columns', skipna=False)
```

[119]:
```
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

idxmax() 나idxmin()같은 메서드는 최솟값 혹은 최대 값을 가지고 있는 색인과 같은 간접 통계 반환

[120]:
```
df.idxmax()
```

[120]:
```
one    b
two    d
dtype: object
```

[121]:
```
df.cumsum()
```

[121]:
```
     one  two
a  1.40  NaN
b  8.50 -4.5
```

```
c   NaN  NaN
d  9.25 -5.8
```

[122]:
```
df.describe()
```

[122]:
```
            one       two
count  3.000000  2.000000
mean   3.083333 -2.900000
std    3.493685  2.262742
min    0.750000 -4.500000
25%    1.075000 -3.700000
50%    1.400000 -2.900000
75%    4.250000 -2.100000
max    7.100000 -1.300000
```

[123]:
```
obj = pd.Series(['a', 'a', 'b', 'c'] * 4)
obj.describe()
```

[123]:
```
count     16
unique     3
top        a
freq       8
dtype: object
```

함수	설명
count	전체 성분의 (NaN이 아닌) 값의 갯수를 계산
min, max	전체 성분의 최솟, 최댓값을 계산
argmin, argmax	전체 성분의 최솟값, 최댓값이 위치한 (정수)인덱스를 반환
idxmin, idxmax	전체 인덱스 중 최솟값, 최댓값을 반환
quantile	전체 성분의 특정 사분위수에 해당하는 값을 반환 (0~1 사이)
sum	전체 성분의 합을 계산
mean	전체 성분의 평균을 계산
median	전체 성분의 중간값을 반환
mad	전체 성분의 평균값으로부터의 절대 편차(absolute deviation)의 평균을 계산
std, var	전체 성분의 표준편차, 분산을 계산
cumsum	맨 첫 번째 성분부터 각 성분까지의 누적합을 계산 (0에서부터 계속 더해짐)
cumprod	맨 첫번째 성분부터 각 성분까지의 누적곱을 계산 (1에서부터 계속 곱해짐)

1.3.1 상관관계와 공분산

conda install pandas-datareader

[124]:
```
import pandas as pd
price = pd.read_pickle('examples/yahoo_price.pkl')
volume = pd.read_pickle('examples/yahoo_volume.pkl')
```

[125]:
```
import pandas_datareader.data as web
all_data = {ticker: web.get_data_yahoo(ticker)
            for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']}

price = pd.DataFrame({ticker: data['Adj Close']
                      for ticker, data in all_data.items()})
volume = pd.DataFrame({ticker: data['Volume']
                       for ticker, data in all_data.items()})
```

주식의 퍼센트 변화율 계산후 맨 마지막 5개를 보여준다

[126]:
```
returns = price.pct_change()
returns.tail()
```

[126]:

Date	AAPL	IBM	MSFT	GOOG
2020-09-14	0.030000	0.005187	0.006764	-0.000947
2020-09-15	0.001560	0.002867	0.016406	0.014586
2020-09-16	-0.029514	0.014538	-0.017866	-0.013325
2020-09-17	-0.015964	0.005635	-0.010436	-0.016681
2020-09-18	-0.031720	-0.017291	-0.012419	-0.023764

corr 메서드는 NA가 아니며 정렬된 색인에서 연속하는 두 Series에 대해 상관관계를 계산하고 cov메서드는 공분산을 계산한다

[127]:
```
returns['MSFT'].corr(returns['IBM'])
returns['MSFT'].cov(returns['IBM'])
```

[127]:
```
0.00016138011318456486
```

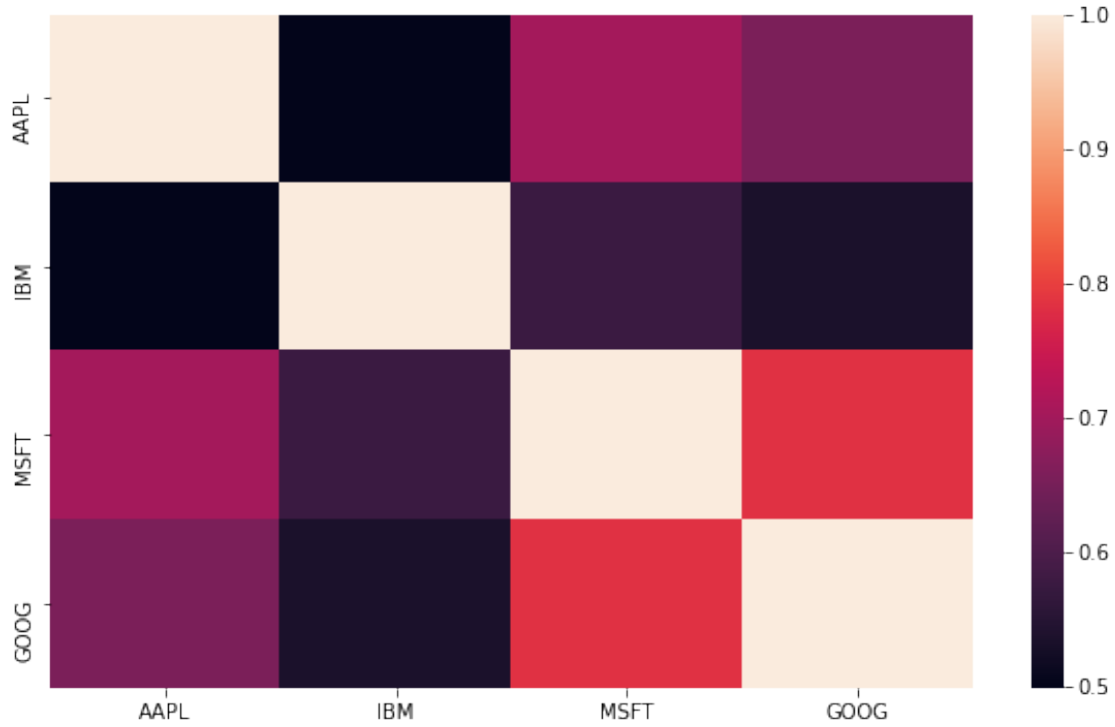
[128]:
```
import seaborn as sns
returns.corr()
```

[128]:

	AAPL	IBM	MSFT	GOOG
AAPL	1.000000	0.498388	0.703872	0.655596
IBM	0.498388	1.000000	0.576192	0.534534
MSFT	0.703872	0.576192	1.000000	0.783305
GOOG	0.655596	0.534534	0.783305	1.000000

[129]:
```
sns.heatmap(returns.corr())
```

[129]:
```
<AxesSubplot:>
```



[130]:
```
returns.corrwith(returns.IBM)
```

[130]:
```
AAPL    0.498388
IBM     1.000000
MSFT    0.576192
GOOG    0.534534
dtype: float64
```

[131]:
```
returns.corrwith(volume)
```

[131]:
```
AAPL   -0.071676
IBM    -0.098103
MSFT   -0.054559
GOOG   -0.150853
dtype: float64
```

1.3.2 유일값, 값 세기, 멤버십

[132]:
```
obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

[133]:
```
uniques = obj.unique()
uniques
```

[133]:
```
array(['c', 'a', 'd', 'b'], dtype=object)
```

[134]:
```
obj.value_counts()
```

[134]:
```
c    3
a    3
b    2
d    1
dtype: int64
```

[135]:
```
pd.value_counts(obj.values, sort=False)
```

[135]:
```
d    1
a    3
c    3
b    2
dtype: int64
```

isin 메서드는 어떤 값이 Series에 존재 하는지 나타내는 불리언 벡터를 반환
Series나 DataFrame의 컬럼에서 값을 골라내고 싶을 때 유용하게 사용

[136]:
```
obj
mask = obj.isin(['b', 'c'])
mask
obj[mask]
```

[136]:
```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

[137]:
```
to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])
unique_vals = pd.Series(['c', 'b', 'a'])
pd.Index(unique_vals).get_indexer(to_match)
```

[137]:
```
array([0, 2, 1, 1, 0, 2])
```

[138]:
```
data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
                     'Qu2': [2, 3, 1, 2, 3],
                     'Qu3': [1, 5, 2, 4, 4]})
data
```

[138]:

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2

```
3    3    2    4
4    4    3    4
```

[139]:
```
result = data.apply(pd.value_counts).fillna(0)
result
```

[139]:

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0