파이썬 라이브러리를 활용한 데이터 분석

5장: 판다스 시작하기

파이썬 라이브러리를 활용한 데이터 분석

5장 2절 핵심기능

재색인

- **obj.reindex()**
  - 새로운 색인에 맞도록 새로 생성
  - 존재하지 않는 색인의 값에는 NaN 추가

```
In [70]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
         obj
Out[70]: d    4.5
         b    7.2
         a   -5.3
         c    3.6
         dtype: float64

In [71]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
         obj2
Out[71]: a   -5.3
         b    7.2
         c    3.6
         d    4.5
         e    NaN
         dtype: float64

In [72]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
         obj3
Out[72]: 0      blue
         2    purple
         4    yellow
         dtype: object

In [75]: obj3.reindex(range(6))
Out[75]: 0      blue
         1       NaN
         2    purple
         3       NaN
         4    yellow
         5       NaN
         dtype: object

In [74]: obj3.reindex(range(6), method='ffill')
Out[74]: 0      blue
         1      blue
         2    purple
         3    purple
         4    yellow
         5    yellow
         dtype: object
```

누락된 값을 이전 값으로 저장

loc[행명, 열명]

- 이름을 여러 개
  - 리스트 형태로
- 슬라이싱도 가능
  - frame.loc['a':'c', 'Texas':'California']
- 열을 재색인
  - 인자 columns에 대입

```
In [135]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                               index=['a', 'c', 'd'],
                               columns=['Ohio', 'Texas', 'California'])
          frame
```

Out[135]:

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

```
In [136]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])
          frame2
```

Out[136]:

	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

```
In [137]: states = ['Texas', 'Utah', 'California']
          frame.reindex(columns = states)
```

Out[137]:

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

```
In [126]: frame.loc[['a', 'c'], ['Texas', 'California']]
```

Out[126]:

	Texas	California
a	1	2
c	4	5

하나의 행이나 열 삭제

- **선택한 값들이 삭제된 객체를 반환**
  - 옵션 inplace=True
    - **원본 객체에 반영**
- **행 삭제**
  - drop('행명')
    - **['행명1', '행명2', ...]**
- **열 삭제**
  - drop('열명', axis=1)
    - **['열명1', '열명2', ...]**

```
In [156]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
          obj.drop('c', inplace=True)
          obj

Out[156]: a    0.0
          b    1.0
          d    3.0
          e    4.0
          dtype: float64
```

```
In [144]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                              index=['Ohio', 'Colorado', 'Utah', 'New York'],
                              columns=['one', 'two', 'three', 'four'])
          data
```

Out[144]:

	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

```
In [145]: data.drop(['Colorado', 'Ohio'])
```

Out[145]:

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

```
In [146]: data.drop('two', axis=1)
```

Out[146]:

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

```
n [147]: data.drop(['two', 'four'], axis='columns')
```

ut[147]:

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

시리즈 색인 p203

- **정수와 인덱스 라벨 모두 가능**
  - 라벨 슬라이싱은 끝점도 포함

```
In [157]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
          obj
Out[157]: a    0.0
          b    1.0
          c    2.0
          d    3.0
          dtype: float64

In [158]: obj['b']
Out[158]: 1.0

In [159]: obj[1]
Out[159]: 1.0

In [160]: obj[2:4]
Out[160]: c    2.0
          d    3.0
          dtype: float64

In [161]: obj[['b', 'a', 'd']]
Out[161]: b    1.0
          a    0.0
          d    3.0
          dtype: float64

In [162]: obj[[1, 3]]
Out[162]: b    1.0
          d    3.0
          dtype: float64

In [163]: obj[obj < 2]
Out[163]: a    0.0
          b    1.0
          dtype: float64

In [164]: obj['b':'c']
Out[164]: b    1.0
          c    2.0
          dtype: float64

In [165]: obj['b':'c'] = 5
          obj
Out[165]: a    0.0
          b    5.0
          c    5.0
          d    3.0
          dtype: float64
```

데이터프레임의 열과 행 참조 p206

- df['열명'], df[['열명1', '열명2', ...]]
- df[m:n]
  - 슬라이싱으로 행를 선택
- df[정수]
  - 이름이 정수인 열이 없으면 오류
- df[조건]
  - 조건이 참인 행를 선택

```
In [166]: data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                              index=['Ohio', 'Colorado', 'Utah', 'New York'],
                              columns=['one', 'two', 'three', 'four'])
          data
```

Out[166]:

	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

```
In [167]: data['two']
```

```
Out[167]: Ohio         1
          Colorado     5
          Utah         9
          New York    13
          Name: two, dtype: int32
```

```
In [168]: data[['three', 'one']]
```

Out[168]:

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

```
In [170]: data[data['three'] > 5]
```

Out[170]:

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

```
In [171]: data < 5
```

Out[171]:

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

loc과 iloc으로 선택하기

- **이름 선택: df.loc(행명, 열명)**
  - 슬라이싱도 가능: 끝도 포함
- **정수 색인 선택: df.iloc(행번호, 열번호)**
  - 슬라이싱도 가능

Selection with loc and iloc

```
In [173]: data
```

Out[173]:

	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

```
In [174]: data.loc['Colorado', ['two', 'three']]
Out[174]: two      5
          three    6
          Name: Colorado, dtype: int32

In [175]: data.iloc[2, [3, 0, 1]]
Out[175]: four    11
          one      8
          two      9
          Name: Utah, dtype: int32

In [176]: data.iloc[2]
Out[176]: one       8
          two       9
          three    10
          four     11
          Name: Utah, dtype: int32

In [177]: data.iloc[[1, 2], [3, 0, 1]]
```

Out[177]:

	four	one	two
Colorado	7	0	5
Utah	11	8	9

```
In [178]: data.loc[:'Utah', 'two']
Out[178]: Ohio        0
          Colorado    5
          Utah        9
          Name: two, dtype: int32

In [179]: data.iloc[:, :3]
```

Out[179]:

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

```
In [180]: data.iloc[:, :3][data.three > 5]
```

Out[180]:

	one	two	three
Colorado	0	5	6
Utah	8	9	10
New York	12	13	14

데이터프레임 색인

Table 5-4. Indexing options with DataFrame

Type	Notes
`df[val]`	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
`df.loc[val]`	Selects single row or subset of rows from the DataFrame by label
`df.loc[:, val]`	Selects single column or subset of columns by label
`df.loc[val1, val2]`	Select both rows and columns by label
`df.iloc[where]`	Selects single row or subset of rows from the DataFrame by integer position
`df.iloc[:, where]`	Selects single column or subset of columns by integer position
`df.iloc[where_i, where_j]`	Select both rows and columns by integer position
`df.at[label_i, label_j]`	Select a single scalar value by row and column label
`df.iat[i, j]`	Select a single scalar value by row and column position (integers)
`reindex` method	Select either rows or columns by labels
`get_value, set_value` methods	Select single value by row and column label

Quiz

- **다음 pandas 프로그램에서 실행 결과가 다른 하나는? (4)**
  - 4번은 마지막 행의 데이터프레임이 출력

  - df = pd.DataFrame(np.arange(12).reshape(3, 4),
  - index=list('abc'), columns=list('ABCD'))

    - **print(df['D'][2])**
    - **print(df.loc['c', 'D'])**
    - **print(df.iloc[2, 3])**
    - **print(df[2:])**

5.2.4 정수 색인 p209

- **정수 색인**
  - iloc() 사용 권장
- **라벨 색인**
  - ser.loc[:1]

```
In [182]: ser = pd.Series(np.arange(3.))
          ser
Out[182]: 0    0.0
          1    1.0
          2    2.0
          dtype: float64

In [190]: ser[0]
          #ser[-1] # 색인 자체가 정수, -1을 라벨로 인지하고 검색, 오류 발생
Out[190]: 0.0

In [191]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
          ser2[-1]
Out[191]: 2.0

In [192]: ser[:1]
Out[192]: 0    0.0
          dtype: float64

In [193]: ser.loc[:1]
Out[193]: 0    0.0
          1    1.0
          dtype: float64

In [194]: ser.iloc[:1]
Out[194]: 0    0.0
          dtype: float64
```

라벨이므로
1까지이다.

시리즈 산술 연산과 데이터 정렬

- **+ 연산**
  - 연산에 참여하는 값이 하나라도 na라면 결과는
    - **NaN**

```
In [195]: 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

Out[195]: a    7.3
          c   -2.5
          d    3.4
          e    1.5
          dtype: float64

In [196]: s2

Out[196]: a   -2.1
          c    3.6
          e   -1.5
          f    4.0
          g    3.1
          dtype: float64

In [197]: s1 + s2

Out[197]: a    5.2
          c    1.1
          d    NaN
          e    0.0
          f    NaN
          g    NaN
          dtype: float64
```

데이터프레임 산술 연산과 데이터 정렬

- 양쪽에 겹치지 않은 부분은 NaN

```
In [198]: 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
```

Out[198]:

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

```
In [199]: df2
```

Out[199]:

	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

```
In [200]: df1 + df2
```

Out[200]:

	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

산술 연산 메소드에 채워 넣을 값 지정하기

- 옵션 fill_vaule=0

```
In [205]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
                             columns=list('abcd'))
          df1
```

Out[205]:

	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

```
In [206]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                             columns=list('abcde'))
          df2.loc[1, 'b'] = np.nan
          df2
```

Out[206]:

	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

```
In [207]: df1 + df2
```

Out[207]:

	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

```
In [208]: df1.add(df2, fill_value=0)
```

Out[208]:

	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

데이터프레임과 시리즈 간의 연산(1)

- **브로드캐스팅과 동일**
  - 칼럼에 맞추고 로우로 전파

```
In [215]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                               columns=list('bde'),
                               index=['Utah', 'Ohio', 'Texas', 'Oregon'])
          frame
```

Out[215]:

	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

b d e

0 1 2

0 1 2

0 1 2

0 1 2

```
In [216]: series = frame.iloc[0]
          series
```

```
Out[216]: b    0.0
          d    1.0
          e    2.0
          Name: Utah, dtype: float64
```

```
In [217]: frame - series
```

Out[217]:

	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

데이터프레임과 시리즈 간의 연산(2)

- **색인 값을 DataFrame의 열이나 Series의 색인에서 찾을 수 없다면**
  - 그 개체는 형식을 맞추기 위해 재색인
  - 중복되지 않는 부분은 NaN

```
In [218]: frame
```

Out[218]:

	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

```
In [221]: series2 = pd.Series(range(3), index=['b', 'e', 'f'])
          series2
```

```
Out[221]: b    0
          e    1
          f    2
          dtype: int64
```

```
In [222]: frame + series2
```

Out[222]:

	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

시리즈 값을 열로 전파

- **행 연산: axis='index' 또는 0**
  - 원래는 칼럼에 맞추고 로우로 전파하나 축이 0이므로 그대로 열로 전파

```
In [224]: series3 = frame['d']
          series3
Out[224]: Utah      1.0
          Ohio      4.0
          Texas     7.0
          Oregon    10.0
          Name: d, dtype: float64
```

1 4 7 10 / 1 4 7 10

```
In [225]: frame
Out[225]:
```

	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

```
In [233]: frame.sub(series3)
Out[233]:
```

	Ohio	Oregon	Texas	Utah	b	d	e
Utah	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Ohio	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Texas	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Oregon	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
In [226]: frame.sub(series3, axis='index')
Out[226]:
```

	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

5.2.6 함수 적용과 매핑 p217

- **Numpy 유니버설 함수 적용 가능**
  - 각 원소에 적용되는 메소드
- **df.apply(함수, axis=0)**
  - 축 행에 따른 연산이 기본
  - axis=1
    - **축 열에 따라 계산**

```
In [234]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),
                               index=['Utah', 'Ohio', 'Texas', 'Oregon'])
          frame
```

Out[234]:

	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

```
In [235]: np.abs(frame)
```

Out[235]:

	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

```
In [236]: f = lambda x: x.max() - x.min()
          frame.apply(f)
```

```
Out[236]: b    1.802165
          d    1.684034
          e    2.689627
          dtype: float64
```

```
In [237]: frame.apply(f, axis='columns')
```

```
Out[237]: Utah      0.998382
          Ohio      2.521511
          Texas     0.676715
          Oregon    2.542656
          dtype: float64
```

열에서 최대-최소

행에서 최대-최소

함수 반환 값이 시리즈

- 여러 값을 가진 시리즈 반환도 가능

```
In [238]: def f(x):
              return pd.Series([x.min(), x.max()], index=['min', 'max'])
          frame
```

Out[238]:

	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

```
In [240]: frame.apply(f)
```

Out[240]:

	b	d	e
min	-0.555730	0.281746	-1.296221
max	1.246435	1.965781	1.393406

```
In [241]: frame.apply(f, axis=1)
```

Out[241]:

	min	max
Utah	-0.519439	0.478943
Ohio	-0.555730	1.965781
Texas	0.092908	0.769023
Oregon	-1.296221	1.246435

배열의 원소에 적용되는 함수 사용

- **df.applymap(함수명)**
  - Series.map(함수명)

```
In [243]: format = lambda x: '%.2f' % x
          frame
```

Out[243]:

	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

```
In [244]: frame.applymap(format)
```

Out[244]:

	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

```
In [246]: frame['e'].map(format)
```

```
Out[246]: Utah      -0.52
          Ohio       1.39
          Texas      0.77
          Oregon    -1.30
          Name: e, dtype: object
```

정렬과 순위

- **행, 열의 색인을 정렬: sort_index**
  - 옵션 axis=

```
In [249]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                               index=['three', 'one'],
                               columns=['d', 'a', 'b', 'c'])
          frame
```

Out[249]:

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

```
In [250]: frame.sort_index()
```

Out[250]:

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

```
In [251]: frame.sort_index(axis=1)
```

Out[251]:

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

```
In [252]: frame.sort_index(axis=1, ascending=False)
```

Out[252]:

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

값에 따른 정렬

- **메소드 df.sort_values()**
  - 시리즈에서 NaN는 마지막에 배치
  - 데이터프레임에서 반드시 필요한 인자 by='열명'
    - **정렬할 열명, 없으면 오류**
    - **by=['열명1', '열명2' ...]**

```
In [254]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
          obj.sort_values()

Out[254]: 4   -3.0
          5    2.0
          0    4.0
          2    7.0
          1    NaN
          3    NaN
          dtype: float64
```

```
In [255]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
          frame

Out[255]:
```

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

```
In [260]: frame.sort_values(by='b')

Out[260]:
```

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

```
In [257]: frame.sort_values(by=['a', 'b'])

Out[257]:
```

	b	a
2	-3	0
0	4	0
3	2	1
1	7	1

값에 따른 정렬 axis=1

- 지정된 행의 값에 따라 정렬

```
In [387]: df
```

Out[387]:

	col1	col2	col3
0	A	2	0
1	A	1	1
2	B	9	9
3	NaN	8	4
4	D	7	2
5	C	4	3

```
In [388]: df.sort_values(by=3, axis=1)
```

Out[388]:

	col3	col2	col1
0	0	2	A
1	1	1	A
2	9	9	B
3	4	8	NaN
4	2	7	D
5	3	4	C

시리즈 항목의 순위

- **메소드 series.rank(), df.rank()**
  - 동점인 항목은 평균 순위가 기본
  - 옵션 method='first'
    - **먼저 나타난 순서 대로 순위**
  - method='max'
    - **동등이면 큰 값으로**
      - **1등이 3개이면 모두 3**
  - 옵션 ascending=False
    - **내림차순으로**

```
In [274]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
          obj.rank()

Out[274]: 0    6.5
          1    1.0
          2    6.5
          3    4.5
          4    3.0
          5    2.0
          6    4.5
          dtype: float64

In [275]: obj.rank(method='first')

Out[275]: 0    6.0
          1    1.0
          2    7.0
          3    4.0
          4    3.0
          5    2.0
          6    5.0
          dtype: float64

In [276]: # Assign tie values the maximum rank in the group
          obj.rank(ascending=False, method='max')

Out[276]: 0    2.0
          1    7.0
          2    2.0
          3    4.0
          4    5.0
          5    6.0
          6    4.0
          dtype: float64
```

데이터프레임 항목의 순위

- **메소드 df.rank()**
  - 동점인 항목은 평균 순위가 기본
  - 옵션 method='first', method='max'
    - **먼저 나타나 순서 대로 순위**
    - **동등이면 큰 값으로**
      - **1등이 3개이면 모두 3**
  - 옵션 ascending=False
    - **내리차순으로**
- **데이터프레임에서**
  - 모든 열에 대해 순위를 매김
  - axis=1
    - **모든 행에 대해 각 값의 순위를 매김**
      - **3등, 2등, 1등**

```
In [277]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                                'c': [-2, 5, 8, -2.5]})
          frame
```

Out[277]:

	b	a	c
0	4.3	0	-2.0
1	7.0	1	5.0
2	-3.0	0	8.0
3	2.0	1	-2.5

```
In [278]: frame.rank(axis='columns')
```

Out[278]:

	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

각 열에서 등수 표시

- **옵션 axis=0**
  - 이것이 기본
    - **각 열에서의 값의 등수 표시**

```
In [61]: frame
```

Out[61]:

	b	a	c
0	4.3	0	-2.0
1	7.0	1	5.0
2	-3.0	0	8.0
3	2.0	1	-2.5

```
In [60]: frame.rank(axis=0)
```

Out[60]:

	b	a	c
0	3.0	1.5	2.0
1	4.0	3.5	3.0
2	1.0	1.5	4.0
3	2.0	3.5	1.0

중복 색인

- **색인 값은 중복 가능**
  - 시리즈에서 참조 시 결과가 여러 개면 시리즈 반환

```
In [283]: obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
          obj

Out[283]: a    0
          a    1
          b    2
          b    3
          c    4
          dtype: int64

In [284]: obj.index.is_unique

Out[284]: False

In [285]: obj['a']

Out[285]: a    0
          a    1
          dtype: int64

In [286]: obj['c']

Out[286]: 4
```

```
In [287]: df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
          df
```

Out[287]:

	0	1	2
a	0.274992	0.228913	1.352917
a	0.886429	-2.001637	-0.371843
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228

```
In [289]: df.loc['b']
```

Out[289]:

	0	1	2
b	1.669025	-0.438570	-0.539741
b	0.476985	3.248944	-1.021228

```
In [292]: df[1] #열 자체의 레이블이 1

Out[292]: a    0.228913
          a   -2.001637
          b   -0.438570
          b    3.248944
          Name: 1, dtype: float64
```

Quiz

- **4. 다음 pandas 프로그램에서 실행 결과는? (1)**
  - df.sum() : 행 방향으로 열의 합을 구함

  - a = pd.DataFrame([[-1, 2], [1, -2]])
  - np.abs(a).sum().sum()

  - 6
  - 0
  - 3
  - 4

파이썬 라이브러리를 활용한 데이터 분석

5장 3절 기술통계 계산과 요약

메소드 df.sum()

- **기본이 축 0을 중심**
  - 열의 합을 반환
  - axis=0, 'index'가 기본
  - 옵션 axis=1, 'columns'
    - **축 1을 중심으로 행 합을 반환**
- **누락된 데이터는 제외하고 계산**
  - 옵션 skipna=True가 기본
  - skipna=False로 하면 결과는 NaN

```
In [294]: 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
```

Out[294]:

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
In [299]: df.sum(axis='index')
Out[299]: one    9.25
          two   -5.80
          dtype: float64
```

```
In [300]: df.sum(axis='columns')
Out[300]: a    1.40
          b    2.60
          c    0.00
          d   -0.55
          dtype: float64
```

```
In [301]: df.mean(axis='columns', skipna=False)
Out[301]: a      NaN
          b    1.300
          c      NaN
          d   -0.275
          dtype: float64
```

메소드 idxmax() cumsum()

- **cumsum()**
  - Na는 0으로 취급하며, 그 위치는 그대로 Na로 반환

```
In [306]: df.cumsum()
```

Out[306]:

	one	two
a	1.40	NaN
b	8.50	-4.5
c	NaN	NaN
d	9.25	-5.8

```
In [311]: df.cumsum(axis=1)
```

Out[311]:

	one	two
a	1.40	NaN
b	7.10	2.60
c	NaN	NaN
d	0.75	-0.55

```
In [312]: df
```

Out[312]:

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
In [303]: df.idxmax()
```

```
Out[303]: one    b
          two    d
          dtype: object
```

```
In [304]: df.idxmax(axis=1)
```

```
Out[304]: a    one
          b    one
          c    NaN
          d    one
          dtype: object
```

describe()

- **여러 개의 통계 결과**
  - 수치 값이 아니면 다른 통계량

```
In [315]: obj = pd.Series(['a', 'a', 'b', 'c'] * 4)

In [316]: obj.describe()

Out[316]: count      16
          unique      3
          top         a
          freq        8
          dtype: object
```

가장 많이 출현

```
In [313]: df
```

Out[313]:

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
In [314]: df.describe()
```

Out[314]:

	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

상관관계

- **두 주식 간의 상관관계(corr)가 어느 정도인가?**
  - 마이크로스프트와 IBM
    - **corr <= .3: 약한 상관관계**
    - **.3 < corr <= .7: 강한 상관관계**
    - **.7<= corr: 매우 강한 상관관계**

In [320]:
```
price = pd.read_pickle('examples/yahoo_price.pkl')
price.head()
```

Out[320]:

Date	AAPL	GOOG	IBM	MSFT
2010-01-04	27.990226	313.062468	113.304536	25.884104
2010-01-05	28.038618	311.683844	111.935822	25.892466
2010-01-06	27.592626	303.826685	111.208683	25.733566
2010-01-07	27.541619	296.753749	110.823732	25.465944
2010-01-08	27.724725	300.709808	111.935822	25.641571

In [324]:
```
returns = price.pct_change()
returns.tail()
```

Out[324]:

Date	AAPL	GOOG	IBM	MSFT
2016-10-17	-0.000680	0.001837	0.002072	-0.003483
2016-10-18	-0.000681	0.019616	-0.026168	0.007690
2016-10-19	-0.002979	0.007846	0.003583	-0.002255
2016-10-20	-0.000512	-0.005652	0.001719	-0.004867
2016-10-21	-0.003930	0.003011	-0.012474	0.042096

In [327]:
```
returns['MSFT'].corr(returns['IBM'])
```

Out[327]: 0.4997636114415114

전체 상관관계 분석

- **전체**

```
In [330]: returns.corr()
```

Out[330]:

	AAPL	GOOG	IBM	MSFT
AAPL	1.000000	0.407919	0.386817	0.389695
GOOG	0.407919	1.000000	0.405099	0.465919
IBM	0.386817	0.405099	1.000000	0.499764
MSFT	0.389695	0.465919	0.499764	1.000000

- **IBM과 다른 회사 간의 상관관계**

```
In [337]: returns.corrwith(returns.IBM)
```

```
Out[337]: AAPL    0.386817
          GOOG    0.405099
          IBM     1.000000
          MSFT    0.499764
          dtype: float64
```

유일 값, 값 세기

- unique()
- value_counts()

```
In [354]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])

In [355]: uniques = obj.unique()
          uniques

Out[355]: array(['c', 'a', 'd', 'b'], dtype=object)

In [356]: obj.value_counts() # 값을 내림차 순으로

Out[356]: a    3
          c    3
          b    2
          d    1
          dtype: int64

In [357]: pd.value_counts(obj.values, sort=False)

Out[357]: c    3
          d    1
          b    2
          a    3
          dtype: int64

In [359]: pd.value_counts(obj.values, sort=True)

Out[359]: a    3
          c    3
          b    2
          d    1
          dtype: int64

In [360]: pd.value_counts(obj.values).sort_index() # 인덱스를 오름차 순으로

Out[360]: a    3
          b    2
          c    3
          d    1
          dtype: int64
```

series.isin(['값1', '값2', ...])

- 어떤 값이 시리즈에 있는 지 검사
  - 논리 벡터를 반환
- obj[obj.isin(['b', 'c'])]
  - 값이 b 또는 c인 값만 시리즈 반환

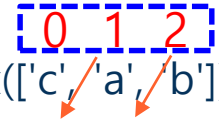
```
In [361]: obj
Out[361]: 0    c
          1    a
          2    d
          3    a
          4    a
          5    b
          6    b
          7    c
          8    c
          dtype: object

In [362]: mask = obj.isin(['b', 'c'])
          mask
Out[362]: 0     True
          1    False
          2    False
          3    False
          4    False
          5     True
          6     True
          7     True
          8     True
          dtype: bool

In [363]: obj[mask]
Out[363]: 0    c
          5    b
          6    b
          7    c
          8    c
          dtype: object
```

Index.get_indexer()

- **pd.Index(unique_vals).get_indexer(to_match)**
  - 인자인 to_match 원소 값이 유일한 값으로 구성된 Index와 매칭되는 첨자로 구성되는 배열을 반환
    - **결과는 인자인 to_macth 수와 일치**
  - get_indexer()를 호출하는 인덱스는 반드시 원소 값이 unique해야 함
- **간단 예제**
  - index = pd.Index(['c', 'a', 'b'])
  - index.get_indexer(['a', 'b', 'x'])
    - **array([1, 2, -1])**




```
In [364]: to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])
          to_match

Out[364]: 0    c
          1    a
          2    b
          3    b
          4    c
          5    a
          dtype: object

In [367]: unique_vals = pd.Series(['c', 'b', 'a'])
          unique_vals

Out[367]: 0    c
          1    b
          2    a
          dtype: object

In [368]: pd.Index(unique_vals)

Out[368]: Index(['c', 'b', 'a'], dtype='object')
                  0    1    2

In [369]: pd.Index(unique_vals).get_indexer(to_match)

Out[369]: array([0, 2, 1, 1, 0, 2], dtype=int64)
```

데이터프레임에 value_count() 적용

- 각 열에서 값이 나온 수 계산
  - 축 0에 따라
    - 각 값의 출현 횟수를 세어
      - 각 값이 인덱스로
      - 출현 수가 값으로 대입

In [375]:
```
data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
                     'Qu2': [2, 3, 1, 2, 3],
                     'Qu3': [1, 5, 2, 4, 4]})
data
```

Out[375]:

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

In [376]:
```
data.apply(pd.value_counts) #축 0에 따라 값의 수를 저장
```

Out[376]:

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

1열에서
1이 1개
2는 없고
3은 2개

In [377]:
```
result = data.apply(pd.value_counts).fillna(0)
result
```

Out[377]:

	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

인덱스(로우 라벨)는 전체 값의 유일한 값을 가짐

각 행에서 값이 나온 수를 계산

- 옵션 axis=1

```
In [379]: data
```

Out[379]:

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

```
In [380]: data.apply(pd.value_counts, axis=1)
```

Out[380]:

	1	2	3	4	5
0	2.0	1.0	NaN	NaN	NaN
1	NaN	NaN	2.0	NaN	1.0
2	1.0	1.0	NaN	1.0	NaN
3	NaN	1.0	1.0	1.0	NaN
4	NaN	NaN	1.0	2.0	NaN

1행에서
1이 2개
2는 1개

```
In [382]: result = data.apply(pd.value_counts, axis=1).fillna(0)
result
```

Out[382]:

	1	2	3	4	5
0	2.0	1.0	0.0	0.0	0.0
1	0.0	0.0	2.0	0.0	1.0
2	1.0	1.0	0.0	1.0	0.0
3	0.0	1.0	1.0	1.0	0.0
4	0.0	0.0	1.0	2.0	0.0

생각해 봅시다.

- **데이터프레임에 행과 열의 합을 추가**
  - 함수 apply
  - 행 추가
    - **df.loc[행마지막번호] = df.열합**
  - 열 추가
    - **df['열명'] = df.행합**

문제: 데이터프레임에 행과 열의 합을 추가

```
In [137]: df = pd.DataFrame(np.arange(12).reshape(4, 3))
          df
```

Out[137]:

	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8
3	9	10	11

```
In [138]: df.apply(sum)
```

```
Out[138]: 0    18
          1    22
          2    26
          dtype: int64
```

```
In [139]: df.apply(sum, 1)
```

```
Out[139]: 0     3
          1    12
          2    21
          3    30
          dtype: int64
```

```
In [142]: df['합'] = df.apply(sum, 1)
          df
```

Out[142]:

	0	1	2	합
0	0	1	2	9
1	3	4	5	36
2	6	7	8	63
3	9	10	11	90

```
In [145]: df.loc[len(df)] = df.apply(sum)
          df
```

Out[145]:

	0	1	2	합
0	0	1	2	9
1	3	4	5	36
2	6	7	8	63
3	9	10	11	90
4	18	22	26	198
5	36	44	52	396

Quiz

- **5. 다음 pandas 프로그램에서 실행 결과가 다른 하나는? (3)**

  - pd.Index(['a', 'b']).get_indexer(['b', 'a', 'c', 'x']).sum()

  - 0
  - 1
  - -1
  - -2