파이썬 라이브러리를 활용한 데이터 분석

3장: 판다스 시작하기

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', method='max'
    - **먼저 나타난 순서 대로 순위**
    - **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
```

6과 7의 평균 값

```
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
```

파이썬 라이브러리를 활용한 데이터 분석

5장 3절 기술통계 계산과 요약

2020.06.25(목) 1h

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**
- **누라된 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() cumsun()

- **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

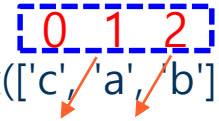
```
In [361]: obj
Out[361]: 0    c
          1    a
          2    d
          3    a
          4    a
          5    b
          6    b
          7    c
          8    c
          dtype: object

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