

수DA쟁이

Python for Data Analysis

5.2(p146~)~5.4

Department of Mathematics
Gyeongsang National University
Youngmin Shin

목차

1. 산술 연산과 데이터 정렬 – 5.2.5
2. 함수 적용과 매핑 - 5.2.6
3. 정렬과 순위 – 5.2.7
4. 중복 색인 – 5.2.8
5. 기술 통계 계산과 요약 – 5.3
6. 상관관계와 공분산 – 5.3.1
7. 유일 값, 값 세기, 멤버십 – 5.3.2

1. 산술연산과 데이터 정렬

Pandas에서 가장 중요한 기능 중 하나는 다른 색인을 가지고 있는 객체 간의 산술 연산이다.

```
In [1]: import pandas as pd  
import numpy as np  
from pandas import Series, DataFrame
```

1. 산술연산과 데이터 정렬

5.2.5 산술 연산과 데이터 정렬

```
In [67]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index = ['a', 'c', 'd', 'e'])
```

```
In [68]: s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],  
                        index = ['a', 'c', 'e', 'f', 'g'])
```

```
In [69]: s1
```

```
Out [69]: a    7.3  
         c   -2.5  
         d    3.4  
         e    1.5  
         dtype: float64
```

```
In [70]: s2
```

```
Out [70]: a   -2.1  
         c    3.6  
         e   -1.5  
         f    4.0  
         g    3.1  
         dtype: float64
```

```
In [71]: s1 + s2
```

```
Out [71]: a    5.2  
         c    1.1  
         d   NaN  
         e    0.0  
         f   NaN  
         g   NaN  
         dtype: float64
```

서로 겹치는 색인이 없는 경우 데이터는 NA 값이 된다.

1. 산술연산과 데이터 정렬

```
In [72]: df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns = list('bcd'),
                             index = ['Ohio', 'Texas', 'Colorado'])
```

```
In [73]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns = list('bce'),
                             index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
In [74]: df1
```

```
Out[74]:
```

	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 [75]: df2
```

```
Out[75]:
```

	b	c	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 [76]: df1 + df2
```

```
Out[76]:
```

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	5.0	NaN	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	11.0	NaN	NaN
Utah	NaN	NaN	NaN	NaN

1. 산술연산과 데이터 정렬

```
In [77]: df1 = pd.DataFrame({'A': [1, 2]})
```

```
In [78]: df2 = pd.DataFrame({'B': [3, 4]})
```

```
In [79]: df1
```

Out[79]:

	A
0	1
1	2

```
In [81]: df1 - df2
```

Out[81]:

	A	B
0	NaN	NaN
1	NaN	NaN

```
In [80]: df2
```

Out[80]:

	B
0	3
1	4

공통되는 컬럼 라벨이나 로우 라벨이 없는 DataFrame을 더하면 결과에 아무것도 나타나지 않는다.

1. 산술연산과 데이터 정렬

```
In [88]: df1.add(df2, fill_value = 0)
```

Out [88]:

	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

df1에 add 메서드를 사용하고, df2와 fill_value 값을 인자로 전달한다.

```
In [89]: 1 / df1
```

Out [89]:

	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

```
In [90]: df1.rdiv(1)
```

Out [90]:

	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

1. 산술연산과 데이터 정렬

표 5-5 산술 연산 메서드

메서드	설명
add, radd	덧셈(+)을 위한 메서드
sub, rsub	뺄셈(-)을 위한 메서드

메서드	설명
div, rdiv	나눗셈(/)을 위한 메서드
floordiv, rfloordiv	소수점 내림(//) 연산을 위한 메서드
mul, rmul	곱셈(*)을 위한 메서드
pow, rpow	역승(**)을 위한 메서드

1. 산술연산과 데이터 정렬

```
In [92]: arr = np.arange(12.).reshape((3, 4))
```

```
In [93]: arr
```

```
Out[93]: array([[ 0.,  1.,  2.,  3.],  
               [ 4.,  5.,  6.,  7.],  
               [ 8.,  9., 10., 11.]])
```

```
In [94]: arr[0]
```

```
Out[94]: array([0., 1., 2., 3.])
```

```
In [95]: arr - arr[0]
```

```
Out[95]: array([[0., 0., 0., 0.],  
               [4., 4., 4., 4.],  
               [8., 8., 8., 8.]])
```

각 계산은 각 로우에 대해 한 번씩만 수행된다. 이를 브로드캐스팅이라고 한다.

1. 산술연산과 데이터 정렬

```
In [96]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                             columns = list('bde'),
                             index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
In [97]: series = frame.iloc[0]
```

```
In [98]: frame
```

Out [98]:

	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 [99]: series
```

Out [99]:

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

```
In [100]: frame - series
```

Out [100]:

	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

DataFrame과 Series 간의 산술 연산은 Series의 색인을 DataFrame의 컬럼에 맞추고 아래 로우로 전파한다.

1. 산술연산과 데이터 정렬

```
In [101]: series2 = pd.Series(range(3), index = ['b', 'e', 'f'])
```

```
In [102]: frame + series2
```

Out[102]:

	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

만약 색인 값은 DataFrame의 컬럼이나 Series의 색인에서 찾을 수 없다면 그 객체는 형식을 맞추기 위해 재색인된다.

1. 산술연산과 데이터 정렬

```
In [106]: frame.sub(series3, axis = 'index')
```

Out[106]:

	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

```
In [103]: series3 = frame['d']
```

```
In [104]: frame
```

Out[104]:

	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 [105]: series
```

Out[105]:

b	0.0
d	1.0
e	2.0

Name: Utah, dtype: float64

sub은 뺄셈을 해준다. axis 값은 연산을 적용할 축 번호다.

axis = 'index'나 axis = 0 은 DataFrame의 로우를 따라 연산을 수행하라는 의미다.

2. 함수 적용과 매핑

pandas 객체에도 NumPy의 유니버설 함수를 적용할 수 있다.

```
In [107]: frame = pd.DataFrame(np.random.randn(4, 3), columns = list('bde'),  
                                index = ['Utah', 'Ohio', 'Texas', 'Oregon'])
```

```
In [108]: frame
```

```
Out[108]:
```

	b	d	e
Utah	-0.467428	0.843950	-0.199600
Ohio	0.442832	-1.201518	-0.894999
Texas	0.530576	1.233779	-0.231575
Oregon	0.668035	0.541446	0.720571

```
In [109]: np.abs(frame)
```

```
Out[109]:
```

	b	d	e
Utah	0.467428	0.843950	0.199600
Ohio	0.442832	1.201518	0.894999
Texas	0.530576	1.233779	0.231575
Oregon	0.668035	0.541446	0.720571

2. 함수 적용과 매핑

```
In [110]: f = lambda x: x.max() - x.min()
```

lambda를 사용해서 함수를 만들

```
In [111]: frame.apply(f)
```

```
Out[111]: b    1.135463  
         d    2.435297  
         e    1.615570  
         dtype: float64
```

각 컬럼에 대해 한번씩 수행되어 결괏값을 계산을 적용해 Series를 반환한다.

```
In [112]: frame.apply(f, axis = 'columns')
```

```
Out[112]: Utah    1.311378  
         Ohio    1.644350  
         Texas    1.465354  
         Oregon    0.179125  
         dtype: float64
```

apply 함수에 axis = 'columns'인자를 넘기면 각 로우에 대해 한번씩만 수행된다.

2. 함수 적용과 매핑

```
In [113]: def f(x):  
           return pd.Series([x.min(), x.max()], index = ['min', 'max'])
```

```
In [114]: frame.apply(f)
```

```
Out[114]:
```

	b	d	e
min	-0.467428	-1.201518	-0.894999
max	0.668035	1.233779	0.720571

Series로 반환할 수도 있다.

2. 함수 적용과 매핑

```
In [115]: format = lambda x: '%.2f' % x
```

```
In [116]: frame.applymap(format)
```

Out[116]:

	b	d	e
Utah	-0.47	0.84	-0.20
Ohio	0.44	-1.20	-0.89
Texas	0.53	1.23	-0.23
Oregon	0.67	0.54	0.72

```
In [117]: frame['e'].map(format)
```

Out[117]:

Utah	-0.20
Ohio	-0.89
Texas	-0.23
Oregon	0.72

Name: e, dtype: object

applymap을 사용해 이와 같이 적용할 수 있다. 이 메서드의 이름이 applymap인 이유는 Series는 각 원소에 적용할 함수를 지정하기 위한 map 메서드를 가지고 있기 때문이다. 이 때, map은 리스트의 요소를 지정된 함수로 처리해주는 함수입니다.

3. 정렬과 순위

```
In [118]: obj = pd.Series(range(4), index = ['d', 'a', 'b', 'c'])
```

```
In [119]: obj.sort_index()
```

```
Out[119]: a    1  
         b    2  
         c    3  
         d    0  
         dtype: int64
```

로우나 컬럼의 색인을 알파벳순으로 정렬하려면 정렬된 새로운 객체를 반환하는 `sort_index` 메서드를 사용하면 된다.

3. 정렬과 순위

```
In [120]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                                index = ['three', 'one'],
                                columns = ['d', 'a', 'b', 'c'])
```

```
In [121]: frame
```

Out[121]:

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

```
In [122]: frame.sort_index()
```

Out[122]:

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

```
In [123]: frame.sort_index(axis = 1)
```

Out[123]:

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

DataFrame은 로우나 컬럼 중 하나의 축을 기준으로 정렬할 수 있다.

3. 정렬과 순위

```
In [124]: frame.sort_index(axis = 1, ascending = False)
```

```
Out[124]:
```

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

데이터는 기본적으로 오름차순으로 정렬되고 내림차순으로 정렬할 수도 있다.

```
In [125]: obj = pd.Series([4, 7, -3, 2])
```

```
In [126]: obj.sort_values()
```

```
Out[126]:
```

2	-3
3	2
0	4
1	7

dtype: int64

값에 따라 정렬하려면 `sort_values` 메서드를 사용하면 된다.

3. 정렬과 순위

```
In [127]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
```

```
In [128]: obj.sort_values()
```

```
Out[128]: 4    -3.0  
          5     2.0  
          0     4.0  
          2     7.0  
          1    NaN  
          3    NaN  
          dtype: float64
```

정렬할 때 비어 있는 값은 기본적으로 **Series** 객체에서 가장 마지막에 위치한다.

3. 정렬과 순위

```
In [129]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
```

```
In [130]: frame
```

Out[130]:

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

```
In [131]: frame.sort_values(by = 'b')
```

Out[131]:

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

DataFrame에서 하나 이상의 컬럼에 있는 값으로 정렬을 하는 경우 `sort_values` 함수의 `by` 옵션에 하나 이상의 컬럼 이름을 넘기면 된다.

```
In [132]: frame.sort_values(by = ['a', 'b'])
```

Out[132]:

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

여러개의 컬럼을 정렬하려면 컬럼 이름이 담긴 리스트를 전달하면 된다.

3. 정렬과 순위

```
In [133]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
```

```
In [134]: obj
```

```
Out[134]: 0    7  
          1   -5  
          2    7  
          3    4  
          4    2  
          5    0  
          6    4  
          dtype: int64
```

3. 정렬과 순위

```
In [135]: obj.rank()
```

```
Out[135]: 0    6.5
          1    1.0
          2    6.5
          3    4.5
          4    3.0
          5    2.0
          6    4.5
          dtype: float64
```

순위 매기는 것, 동점인 항목에 대해서는 평균 순위를 매긴다.

```
In [136]: obj.rank(method = 'first')
```

```
Out[136]: 0    6.0
          1    1.0
          2    7.0
          3    4.0
          4    3.0
          5    2.0
          6    5.0
          dtype: float64
```

먼저 나온것이 6등 나중이 7등

```
[137]: obj.rank(ascending = False, method = 'max')
```

```
[137]: 0    2.0
          1    7.0
          2    2.0
          3    4.0
          4    5.0
          5    6.0
          6    4.0
          dtype: float64
```

내림차순

3. 정렬과 순위

```
In [138]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                                'c': [-2, 5, 8, -2.5]})
```

```
In [139]: frame
```

```
Out [139]:
```

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

```
Out [140]:
```

	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

DataFrame에 대해서도 순위를 매길 수 있다.

표 5-6 순위의 동물을 처리하는 메서드

메서드	설명
'average'	기본값. 같은 값을 가지는 항목들의 평균값을 순위로 삼는다.
'min'	같은 값을 가지는 그룹을 낮은 순위로 매긴다.
'max'	같은 값을 가지는 그룹을 높은 순위로 매긴다.
'first'	데이터 내의 위치에 따라 순위를 매긴다.
'dense'	method='min'과 같지만 같은 그룹 내에서 모두 같은 순위를 적용하지 않고 1씩 증가시킨다.

4. 중복 색인

pandas의 reindex 같은 많은 함수 들에서 색인 값이 유일해야 할 의무는 없다.

```
In [141]: obj = pd.Series(range(5), index = ['a', 'a', 'b', 'b', 'c'])
```

```
In [142]: obj
```

```
Out[142]: a    0  
         a    1  
         b    2  
         b    3  
         c    4  
         dtype: int64
```

4. 중복 색인

```
In [143]: obj.index.is_unique
```

```
Out[143]: False
```

`is_unique` 속성은 해당 값이 유일한지 아닌지를 알려준다.

```
In [144]: obj['a']
```

```
Out[144]: a    0
          a    1
          dtype: int64
```

색인 값이 중복되지 않는다면 스칼라 값을 반환하지만, 중복된다면 `series`를 반환한다.

```
In [145]: obj['c']
```

```
Out[145]: 4
```

이는 라벨이 반복되는 지 여부에 따라 색인을 이용해서 선택한 결과가 다를 수 있기 때문에 코드를 복잡하게 만들 수 있다.

4. 중복 색인

```
In [146]: df = pd.DataFrame(np.random.randn(4, 3), index = ['a', 'a', 'b', 'b'])
```

```
In [147]: df
```

Out[147]:

	0	1	2
a	1.285726	-0.478725	2.204005
a	-0.835233	-0.135697	-1.589904
b	0.694174	-0.080903	1.277626
b	0.740127	-0.655580	-1.045586

```
In [148]: df.loc['b']
```

Out[148]:

	0	1	2
b	0.694174	-0.080903	1.277626
b	0.740127	-0.655580	-1.045586

5. 기술 통계 계산과 요약

```
In [149]: 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'])
```

Pandas 객체는 일반적인 수학 메서드와 통계 메서드를 가지고 있다.

```
In [150]: df
```

```
Out[150]:
```

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

```
In [151]: df.sum()
```

```
Out[151]: one    9.25
          two   -5.80
          dtype: float64
```

DataFrame의 sum 메서드를 호출하면 각 컬럼의 합을 담은 Series를 반환한다.

```
In [152]: df.sum(axis = 'columns')
```

```
Out[152]: a    1.40
          b    2.60
          c    0.00
          d   -0.55
          dtype: float64
```

axis = 'columns' 또는 axis = 1 옵션을 넘기면 각 컬럼의 합을 반환한다.

5. 기술 통계 계산과 요약

```
In [153]: df.mean(axis = 'columns', skipna = False)
```

```
Out [153]: a      NaN  
b      1.300  
c      NaN  
d     -0.275  
dtype: float64
```

skipna는 누락된 값을 제외할 것인지 정하는 옵션, 기본값은 True이다

```
In [154]: df.idxmax()
```

```
Out [154]: one      b  
two      d  
dtype: object
```

idxmin이나 idxmax 같은 메서드는 최솟값 혹은 최댓값을 가지고 있는 색인값을 반환한다.

```
In [155]: df.cumsum()
```

```
Out [155]:
```

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

cumsum은 누산이다.(축적)

5. 기술 통계 계산과 요약

```
In [156]: df.describe()
```

```
Out[156]:
```

	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

describe는 한 번에 여러개의 통계 결과를 만들어낸다.

```
In [157]: obj = pd.Series(['a', 'a', 'b', 'c']*4)
```

```
In [158]: obj
```

```
Out[158]: 0    a
          1    a
          2    b
          3    c
          4    a
          5    a
          6    b
          7    c
          8    a
          9    a
         10    b
         11    c
         12    a
         13    a
         14    b
         15    c
dtype: object
```

```
In [159]: obj.describe()
```

```
Out[159]: count      16
          unique      3
          top         a
          freq        8
          dtype: object
```

수치 데이터가 아닐 경우 describe는 다른 요약 통계를 생성한다.

5. 기술 통계 계산과 요약

표 5-8 요약 통계 관련 메서드

메서드	설명
count	NA 값을 제외한 값의 수를 반환한다.
describe	Series나 DataFrame의 각 컬럼에 대한 요약 통계를 계산한다.
min, max	최솟값과 최댓값을 계산한다.
argmin, argmax	각각 최솟값과 최댓값을 담고 있는 색인의 위치(정수)를 반환한다.
idxmin, idxmax	각각 최솟값과 최댓값을 담고 있는 색인의 값을 반환한다.
quantile	0부터 1까지의 분위수를 계산한다.
sum	합을 계산한다.

메서드	설명
mean	평균을 계산한다.
median	중간값(50% 분위)을 반환한다.
mad	평균값에서 평균절대편차를 계산한다.
prod	모든 값의 곱
var	표본분산의 값을 계산한다.
std	표본표준편차의 값을 계산한다.
skew	표본비대칭도(3차 적률)의 값을 계산한다.
kurt	표본첨도(4차 적률)의 값을 계산한다.
cumsum	누적합을 계산한다.
cummin, cummax	각각 누적 최솟값과 누적 최댓값을 계산한다.
cumprod	누적곱을 계산한다.
diff	1차 산술차를 계산한다(시계열 데이터 처리 시 유용하다).
pct_change	퍼센트 변화율을 계산한다.

6. 상관관계와 공분산

상관 관계와 공분산은 두 쌍의 인자가 필요하다. 두 쌍을 야후!에서 받아와서 했다.

```
In [160]: conda install pandas-datareader
```

```
Collecting package metadata (current_repodata.json): ...working... done  
Solving environment: ...working... done
```

```
# All requested packages already installed.
```

```
Note: you may need to restart the kernel to use updated packages.
```

conda를 통해 pandas-datareader 패키지 다운

```
In [161]: 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()})
```

야후! 금융 사이트에서 구한 주식가격과 시가총액을 담고 있는 DataFrame만들기

6. 상관관계와 공분산

In [162]: price

Out [162]:

	AAPL	IBM	MSFT	GOOG
Date				
2016-05-12	21.098566	119.175903	47.233212	713.309998
2016-05-13	21.140608	118.279099	46.838917	710.830017
2016-05-16	21.925322	119.672348	47.526649	716.489990
2016-05-17	21.834238	118.503326	46.640190	706.229980
2016-05-18	22.084131	117.974838	46.917206	706.630005
...
2021-05-05	127.882790	143.615356	246.470001	2356.739990
2021-05-06	129.520004	146.779999	249.729996	2381.350098
2021-05-07	130.210007	145.460007	252.460007	2398.689941
2021-05-10	126.849998	146.169998	247.179993	2341.659912
2021-05-11	125.910004	144.220001	246.229996	2308.760010

1258 rows × 4 columns

In [163]: volume

Out [163]:

	AAPL	IBM	MSFT	GOOG
Date				
2016-05-12	305258800.0	3249200.0	24102800.0	1360700
2016-05-13	177571200.0	2398000.0	22592300.0	1314500
2016-05-16	245039200.0	3069100.0	20032000.0	1317100
2016-05-17	187667600.0	3490600.0	27803500.0	2001200
2016-05-18	168249600.0	2491100.0	24907500.0	1766800
...
2021-05-05	84000900.0	3622800.0	21901300.0	1090300
2021-05-06	78128300.0	7503500.0	26491100.0	1030900
2021-05-07	78892700.0	7002500.0	27010100.0	1163600
2021-05-10	88071200.0	6983400.0	29299900.0	1300300
2021-05-11	126053700.0	7123500.0	33628300.0	1603300

1258 rows × 4 columns

6. 상관관계와 공분산

```
In [164]: returns = price.pct_change()
```

```
In [165]: returns.tail()
```

```
Out [165]:
```

	AAPL	IBM	MSFT	GOOG
Date				
2021-05-05	0.001956	-0.003636	-0.005327	0.001058
2021-05-06	0.012802	0.022036	0.013227	0.010442
2021-05-07	0.005327	-0.008993	0.010932	0.007282
2021-05-10	-0.025805	0.004881	-0.020914	-0.023775
2021-05-11	-0.007410	-0.013341	-0.003843	-0.014050

각 주식의 퍼센트 변화율 계산하고, tail을 통해 마지막 5줄을 확인한다.

6. 상관관계와 공분산

```
In [166]: returns['MSFT'].corr(returns['IBM'])
```

```
Out[166]: 0.5307537417392566
```

```
In [167]: returns['MSFT'].cov(returns['IBM'])
```

```
Out[167]: 0.00015043577234161114
```

`corr`메서드는 NA가 아니며 정렬된 색인에서 연속하는 두 `series`에 대해 상관관계를 계산하고 `cov`메서드는 공분산을 계산한다.

```
In [168]: print(returns.MSFT.corr(returns.IBM))  
print(returns.MSFT.cov(returns.IBM))
```

```
0.5307537417392566  
0.00015043577234161114
```

이런 식으로도 가능하다.

6. 상관관계와 공분산

```
In [169]: returns.corr()
```

```
Out[169]:
```

	AAPL	IBM	MSFT	GOOG
AAPL	1.000000	0.446473	0.723717	0.658067
IBM	0.446473	1.000000	0.530754	0.493688
MSFT	0.723717	0.530754	1.000000	0.771043
GOOG	0.658067	0.493688	0.771043	1.000000

```
In [170]: returns.cov()
```

```
Out[170]:
```

	AAPL	IBM	MSFT	GOOG
AAPL	0.000363	0.000140	0.000238	0.000211
IBM	0.000140	0.000269	0.000150	0.000136
MSFT	0.000238	0.000150	0.000299	0.000224
GOOG	0.000211	0.000136	0.000224	0.000282

DataFrame에서 corr과 cov 메서드는 DataFrame 행렬에서 상관관계와 공분산을 계산한다.

6. 상관관계와 공분산

```
In [171]: returns.corrwith(returns.IBM)
```

```
Out[171]: AAPL    0.446473  
          IBM     1.000000  
          MSFT    0.530754  
          GOOG    0.493688  
          dtype: float64
```

`DataFrame`의 `corrwith` 메서드를 사용하면 다른 `series`나 `DataFrame`과의 상관관계를 계산한다. `series`를 넘기면 각 컬럼에 대해 계산한 상관관계를 담고 있는 `Series`를 반환한다.

```
In [172]: returns.corrwith(volume)
```

```
Out[172]: AAPL    -0.050805  
          IBM     -0.099293  
          MSFT    -0.061588  
          GOOG    -0.114190  
          dtype: float64
```

`DataFrame`을 넘기면 맞아떨어지는 컬럼 이름에 대한 상관 관계를 계산한다. 위는 시가총액의 퍼센트 변화율에 대한 상관관계를 계산해보았다.

6. 상관관계와 공분산

```
In [173]: returns.corrwith(volume,axis = 1)
```

```
Out[173]: Date
2016-05-12      NaN
2016-05-13    0.852449
2016-05-16    0.988225
2016-05-17    0.740397
2016-05-18    0.849632
...
2021-05-05    0.486394
2021-05-06   -0.240617
2021-05-07    0.265714
2021-05-10   -0.481592
2021-05-11    0.527231
Length: 1258, dtype: float64
```

`axis = 1`이라는 옵션을 넘기면 각 컬럼에 대한 상관관계와 공분산을 계산한다.

7. 유일 값, 값 세기, 멤버십

```
In [174]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
In [175]: uniques = obj.unique()
```

```
In [176]: uniques
```

```
Out[176]: array(['c', 'a', 'd', 'b'], dtype=object)
```

`unique` 메서드는 중복되는 값을 제거하고 유일값만 담고 있는 `Series`를 반환한다.

```
In [177]: obj.value_counts()
```

```
Out[177]: a    3  
c    3  
b    2  
d    1  
dtype: int64
```

`value_counts` 메서드는 `Series`에서 도수를 계산하여 반환한다.

7. 유일 값, 값 세기, 멤버십

```
In [181]: mask = obj.isin(['b', 'c'])
```

```
In [182]: mask
```

```
Out[182]: 0    True
          1   False
          2   False
          3   False
          4   False
          5    True
          6    True
          7    True
          8    True
          dtype: bool
```

```
In [183]: obj[mask]
```

```
Out[183]: 0    c
          5    b
          6    b
          7    c
          8    c
          dtype: object
```

isin 메서드는 어떤 값이 Series에 존재하는 지 나타내는 불리언 벡터를 반환하는데, Series나 DataFrame의 컬럼에서 값을 골라내고 싶을 때 유용하게 사용할 수 있다.

7. 유일 값, 값 세기, 멤버십

```
In [184]: to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])
```

```
In [185]: unique_vals = pd.Series(['c', 'b', 'a'])
```

```
In [186]: pd.Index(unique_vals).get_indexer(to_match)
```

```
Out[186]: array([0, 2, 1, 1, 0, 2], dtype=int64)
```

`Index.get_indexer` 메서드는 여러 값이 들어 있는 배열에서 유일한 값의 색인 배열을 구할 수 있다.

7. 유일 값, 값 세기, 멤버십

```
In [189]: data = pd.DataFrame({'Qu1': [1, 3, 4, 3, 4],
                                'Qu2': [2, 3, 1, 2, 3],
                                'Qu3': [1, 5, 2, 4, 4]})
```

```
In [190]: data
```

Out [190]:

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

내림차순으로 정렬해준다.

```
In [191]: result = data.apply(pd.value_counts).fillna(0)
```

```
In [192]: result
```

Out [192]:

	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

Thank you!