DATA KUBWA

작성자: 고우주 / 데이터 쿱와(주)

파이썬 - pandas 활용

1. 데이터 병합

pandas는 두 개 이상의 데이터프레임을 하나로 합치는 데이터 병합(merge)와 연결(concatenate)을 지원

- merge( )
- join( )
- concat( )

In [2]:

```
import pandas as pd
import numpy as np
```

1.1 merge( )

- merge 명령은 두 데이터 프레임의 공통 열 혹은 인덱스를 기준으로 두 개의 테이블을 합친다.
- 기준이 되는 열, 행의 데이터가 키(key) 이다.

In [3]:

```
# 데이터프레임 d1과 d2를 생성
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})

df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [4]:

```
df1
```

Out[4]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [5]:

```
df2
```

Out[5]:

	key	data2
0	a	0
1	b	1
2	d	2

In [6]:

```
#pd.merge(df1, df2)로 합치기
pd.merge(df1, df2, on='key')
```

Out[6]:

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

In [7]:

```
df3 = pd.DataFrame({'left_key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})

df4 = pd.DataFrame({'right_key': ['a', 'b', 'd'],
                    'data2': range(3)})
```

In [8]:

```
df3
```

Out[8]:

	left_key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

In [9]:

```
df4
```

Out[9]:

	right_key	data2
0	a	0
1	b	1
2	d	2

In [10]:

```
# 각각의 key로 합치기
pd.merge(df3, df4, left_on='left_key', right_on='right_key')
```

Out[10]:

	left_key	data1	right_key	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

- inner join

- outer join

- left outer join
- right outer join
- full outer join

In [11]:

```
# Left Outer Join
pd.merge(df1, df2, how='left')
```

Out[11]:

	key	data1	data2
0	b	0	1.0
1	b	1	1.0
2	a	2	0.0
3	c	3	NaN
4	a	4	0.0
5	a	5	0.0
6	b	6	1.0

In [12]:

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                    'data1': range(6)})

df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                    'data2': range(5)})
```

In [13]:

```
df1
```

Out[13]:

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

In [14]:

```
df2
```

Out[14]:

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

In [15]:

```
# Left Join
pd.merge(df1, df2, on='key', how='left')
```

Out[15]:

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

In [16]:

```
# Inner Join
pd.merge(df1, df2, how='inner')
```

Out[16]:

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2

In [17]:

```
left  = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                      'key2': ['one', 'two', 'one'],
                      'lval': [1, 2, 3]})

right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
```

In [18]:

```
left
```

Out[18]:

	key1	key2	lval
0	foo	one	1
1	foo	two	2
2	bar	one	3

In [19]:

```
right
```

Out[19]:

	key1	key2	rval
0	foo	one	4
1	foo	one	5
2	bar	one	6
3	bar	two	7

In [20]:

```
# 여러 key 지정
pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

Out[20]:

	key1	key2	lval	rval
0	foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0
4	bar	two	NaN	7.0

In [21]:

```
# 중복되는 컬럼 이름 자동 변경
pd.merge(left, right, on='key1')
```

Out[21]:

	key1	key2_x	lval	key2_y	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

In [22]:

```
# 중복되는 컬럼 이름 뒤에 붙일 문자열 지정
pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

Out[22]:

	key1	key2_left	lval	key2_right	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

merge 함수 인자 목록

인자	설명
left	머지하려는 DataFrame 중 왼쪽에 위치한 DataFrame
right	머지하려는 DataFrame 중 오른쪽에 위치한 DataFrame
how	조인 방법. 'inner', 'outer', 'left', 'right' 기본 값은 inner
on	조인하려는 로우 이름. 반드시 두 DataFrame 객체 모두에 있는 이름이어야 한다.
left_on	조인 키로 사용할 left DataFrame 컬럼
right_on	조인 키로 사용할 right DataFrame 컬럼
left_index	조인 키로 사용할 left DataFrame의 색인 로우 (다중 색인일 경우의 키)
right_index	조인 키로 사용할 right DataFrame의 색인 로우 (다중 색인일 경우의 키)
suffixes	중복되는 컬럼의 이름 뒤에 붙인 문자열 지정

1.2 색인으로 머지하기

In [23]:

```
left = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                     'value': range(6)})

right = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
```

In [25]:

```
left
```

Out[25]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

In [26]:

```
right
```

Out[26]:

	group_val
a	3.5
b	7.0

In [45]:

```
pd.merge(left1, right1, left_on='key', right_index=True)
```

Out[45]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

In [27]:

```
pd.merge(left, right, left_on='key', right_index=True, how='outer')
```

Out[27]:

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

2.2 join( )

DataFrame 2개 이상 조인 가능

- inner: use intersection of keys from both frames (sql: inner join)
- outer: use union of keys from both frames (sql: full outer join)
- left: use only keys from left frame (sql: left outer join)
- right: use only keys from right frame (sql: right outer join)

In [28]:

```
left.join(right, how='outer')
```

Out[28]:

	key	value	group_val
0	a	0.0	NaN
1	b	1.0	NaN
2	a	2.0	NaN
3	a	3.0	NaN
4	b	4.0	NaN
5	c	5.0	NaN
a	NaN	NaN	3.5
b	NaN	NaN	7.0

In [29]:

```
# Default: how = 'left'
left.join(right, on='key')
```

Out[29]:

	key	value	group_val
0	a	0	3.5
1	b	1	7.0
2	a	2	3.5
3	a	3	3.5
4	b	4	7.0
5	c	5	NaN

In [30]:

```
another = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
                       index=['a', 'c', 'e', 'f'],
                       columns=['New York', 'Oregon'])
another
```

Out[30]:

	New York	Oregon
a	7.0	8.0
c	9.0	10.0
e	11.0	12.0
f	16.0	17.0

In [32]:

```
left.join([right, another])
```

Out[32]:

	key	value	group_val	New York	Oregon
0	a	0	NaN	NaN	NaN
1	b	1	NaN	NaN	NaN
2	a	2	NaN	NaN	NaN
3	a	3	NaN	NaN	NaN
4	b	4	NaN	NaN	NaN
5	c	5	NaN	NaN	NaN

In [33]:

```
left.join([right, another], how='outer', sort=True)
```

```
/anaconda3/lib/python3.7/site-packages/pandas/core/indexes/api.py:69: RuntimeWarning: '<' not suppor
ted between instances of 'int' and 'str', sort order is undefined for incomparable objects
  index = _union_indexes(indexes, sort=sort)
```

Out[33]:

	key	value	group_val	New York	Oregon
0	a	0.0	NaN	NaN	NaN
1	b	1.0	NaN	NaN	NaN
2	a	2.0	NaN	NaN	NaN
3	a	3.0	NaN	NaN	NaN
4	b	4.0	NaN	NaN	NaN
5	c	5.0	NaN	NaN	NaN
a	NaN	NaN	3.5	7.0	8.0
b	NaN	NaN	7.0	NaN	NaN
c	NaN	NaN	NaN	9.0	10.0
e	NaN	NaN	NaN	11.0	12.0
f	NaN	NaN	NaN	16.0	17.0

1.3 concat( )

- concat 명령은 단순히 데이터를 연결(concatenate)
- 기본적으로는 위/아래로 데이터 행을 연결
- 단순히 두 시리즈나 데이터프레임을 연결하기 때문에 인덱스 값이 중복

In [34]:

```
arr = np.arange(12).reshape((3, 4))
arr
```

Out[34]:

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

In [35]:

```
s1 = pd.Series([0, 1],    index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
s3 = pd.Series([5, 6],    index=['f', 'g'])
```

In [36]:

```
pd.concat([s1, s2, s3])
```

Out[36]:

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

In [37]:

```
pd.concat([s1, s2, s3], axis=1, sort=False)
```

Out[37]:

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

In [38]:

```
s4 = pd.concat([s1, s3])
s4
```

Out[38]:

```
a    0
b    1
f    5
g    6
dtype: int64
```

In [39]:

```
pd.concat([s1, s4], axis=1, sort=False)
```

Out[39]:

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

In [40]:

```
# Default: 'outer'
pd.concat([s1, s4], axis=1, join='inner')
```

Out[40]:

	0	1
a	0	0
b	1	1

In [41]:

```
# 필요한 컬럼 선택
pd.concat([s1, s4], axis=1, join_axes=[['a', 'c', 'b', 'e']])
```

Out[41]:

	0	1
a	0.0	0.0
c	NaN	NaN
b	1.0	1.0
e	NaN	NaN

In [42]:

```
# 인덱스, 컬럼 명 설정 (계층적 색인 생성)
result = pd.concat([s1, s2, s3], keys=['one', 'two', 'three'])
result
```

Out[42]:

```
one    a    0
       b    1
two    c    2
       d    3
       e    4
three  f    5
       g    6
dtype: int64
```

2. GroupBy

In [44]:

```
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1' : np.random.randn(5),
                   'data2' : np.random.randn(5)})
df
```

Out[44]:

	key1	key2	data1	data2
0	a	one	-0.219209	0.345139
1	a	two	0.352496	1.309426
2	b	one	1.010584	0.107121
3	b	two	-0.122351	-0.373229
4	a	one	0.072883	0.016763

2-1 Series의 Groupby

In [45]:

```
grouped = df['data1'].groupby(df['key1'])
grouped
```

Out[45]:

```
<pandas.core.groupby.groupby.SeriesGroupBy object at 0x1137cf9b0>
```

In [46]:

```
grouped.mean()
```

Out[46]:

```
key1
a    0.068723
b    0.444116
Name: data1, dtype: float64
```

In [47]:

```
# 2개의 키로 그룹핑
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

Out[47]:

```
key1  key2
a     one    -0.073163
      two     0.352496
b     one     1.010584
      two    -0.122351
Name: data1, dtype: float64
```

In [48]:

```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2005, 2005, 2006, 2005, 2006])

df['data1'].groupby([states, years]).mean()
```

Out[48]:

```
California  2005    0.352496
            2006    1.010584
Ohio        2005   -0.170780
            2006    0.072883
Name: data1, dtype: float64
```

2-2 DataFrame의 Groupby

In [49]:

```
# key2: 숫자 컬럼이 아니어서 pandas가 제외 시킴

df.groupby('key1').mean()
```

Out[49]:

key1	data1	data2
a	0.068723	0.557110
b	0.444116	-0.133054

In [50]:

```
df.groupby(['key1', 'key2']).mean()
```

Out[50]:

key1	key2	data1	data2
a	one	-0.073163	0.180951
	two	0.352496	1.309426
b	one	1.010584	0.107121
	two	-0.122351	-0.373229

In [52]:

```
df.groupby(['key1', 'key2'])
```

Out[52]:

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x1137cfcf8>
```

2-3 groupby 메서드

함수 이름	설명
count	그룹 내에 NA 값이 아닌 값의 수
sum	NA 값이 아닌 값들의 합
mean	NA 값이 아닌 값들의 평균
median	NA 값이 아닌 값들의 산술 중간 값
std, var	표준편차와 분산 (n-1을 분모로 계산)
min, max	NA 값이 아닌 값 중 최소 값, 최대 값
prod	NA 값이 아닌 값의 곱
first, last	NA 값이 아닌 값들 중 첫 번째 값, 마지막 값

In [ ]: