apply for dataframe

apply for dataframe

- apply를 사용하면 dataframe의 각 row 또는 column에 특정 함수를 적용할 수 있음
- 사용 방법
  - **df1.apply(f1, axis=0 or 1)**
  - df1: dataframe 객체, f1은 특정 함수
  - axis=0 or index→각 column
  - axis=1 or columns→각 row에 적용

(a)

```
d1= pd.read_table('data/pandas_data3.txt', index_col=0)
d2=d1.iloc[:3,1:]
d2

def f2(x):
    return x.max()-x.min()
```

(b)

```
d2.apply(f2, axis=0)
d2.apply(f2, axis='index')
```

(c)

```
d2.apply(f2, axis=1)
d2.apply(f2, axis='columns')
```

Out[17]: (a)

name	year	mid	final	att	proj
kim2	1	12	36	5	75
kim3	2	17	20	5	96
kim6	2	28	20	4	83

Out[17]: (b)

```
year      1
mid      16
final    16
att       1
proj     21
dtype: int64
```

Out[17]: (c)

```
name
kim2    74
kim3    94
kim6    81
dtype: int64
```

apply for dataframe (cont'd)

- 간단한 함수는 lambda로 정의 가능

In [18]:

```
d1=pd.read_table('data/pandas_data3.txt', index_col=0)
d2=d1.iloc[:3,1:]
d2
```

(a)

```
def f2(x):
    return x.max()-x.min()
```

(b)

```
d2.apply(lambda x: x.max()-x.min())
d2.apply(f2)
```

(default axis=0)

Out[18]: (a)

name	year	mid	final	att	proj
kim2	1	12	36	5	75
kim3	2	17	20	5	96
kim6	2	28	20	4	83

Out[18]: (b)

```
year      1
mid      16
final    16
att       1
proj     21
dtype: int64
```

apply for dataframe (cont'd)

- 이미 정의되어있는 함수가 있으면 함수 바로 사용 가능 (1)
- dataframe class의 맴버함수들도 사용 가능 (2)

(a)

```
d1=pd.read_table('data/pandas_data3.txt', index_col=0)
d2=d1.iloc[:3,1:]
d2
```

```
def f2(x):
    return x.mean()
```

(b)

```
d2.apply(f2, axis=1)
d2.apply(lambda x: x.mean(), axis=1)
d2.apply(np.mean, axis=1) (1)
d2.mean(axis=1) (2)
```

(a)

name	year	mid	final	att	proj
kim2	1	12	36	5	75
kim3	2	17	20	5	96
kim6	2	28	20	4	83

(b)

```
name
kim2    25.8
kim3    28.0
kim6    27.4
dtype: float64
```

map vs apply

map

```
df1['기말성적']=df1['기말'].map(lambda x: x*0.5)
df1.head()

def f1(x):
    return x*0.5

df1['기말성적2']=df1['기말'].map(f1)
df1.head()
```

1. Series에 적용
2. 함수의 x: **Series의 각 element**
3. 함수의 결과: scalar (숫자 하나)
4. 최종결과: Series의 각 element에 함수(f1)을 적용한 **series**

apply

```
def f2(x):
    return x.max()-x.min()

ds1=df2.apply(f2)
ds2=df2.apply(f2,axis=1)
```

1. dataframe에 적용
2. 함수의 x: **dataframe의 각 series**
   1) axis=0: 각 column (default)
   2) axis=1: 각 row
3. 함수의 결과: scalar (숫자 하나)
4. 최종결과: dataframe의 각 series에 함수(f2)를 적용한 **series**

apply for dataframe (새로운 column, row 추가)

In [57]:

```
d1=pd.read_table(' data/pandas_data3.txt ', index_col=0)
d2=d1.iloc[:5,1:]
d2
```

(a)

```
def f3(x):
    return np.mean(x)+np.std(x)

d2['num_of_3mul']=d2.apply(lambda x: np.sum(x%3==0), axis=1)
d2.loc['temp']=d2.apply(f3)
d2
```

(b)

Out[57]:

(a)

name	year	mid	final	att	proj
kim2	1	12	36	5	75
kim3	2	17	20	5	96
kim6	2	28	20	4	83
kim7	2	20	44	6	82
kim8	1	20	24	8	88

Out[57]:

(b)

name	year	mid	final	att	proj	num_of_3mul
kim2	1.000000	12.0	36.0	5.000000	75.000000	3.000000
kim3	2.000000	17.0	20.0	5.000000	96.000000	1.000000
kim6	2.000000	28.0	20.0	4.000000	83.000000	0.000000
kim7	2.000000	20.0	44.0	6.000000	82.000000	1.000000
kim8	1.000000	20.0	24.0	8.000000	88.000000	1.000000
temp	2.089898	24.6	38.4	6.956466	91.768501	2.179796

groupby and apply

```
d1=pd.read_table('pandas_data3.txt', index_col=0)
d1=d1.head(10)

for key, col in d1.groupby('gender')['mid']:
    print(key)
    col

d1.groupby('gender')['mid'].apply(np.sum)
```

(a)

(b)

```
d1.groupby('gender')['mid'].sum()
```

새롭게 정의한 함수 사용 가능

(a)

```
F

Out[17]: name
         kim2     12
         kim8     20
         lee10    28
         Name: mid, dtype: int64

         M

Out[17]: name
         kim3     17
         kim6     28
         kim7     20
         kim9     23
         kim10    16
         lee5     21
         lee9     20
         Name: mid, dtype: int64
```

(b)

```
Out[17]: gender
         F      60
         M     145
         Name: mid, dtype: int64
```