Groupby

Contents

Groupby 는 데이터를 요약할 때 많이 활용하는 기법입니다. 아래 예제에서 만들어진 DataFrame - df 의 'grp' 컬럼을 이용하여 'a', 'b', 'c' 등의 3 개의 그룹으로 나눌 수 있습니다. 먼저, 그룹을 무시하고 v1, v2 의 평균값을 알아봅니다. 그 다음, 그룹 별로 v1 과 v2 의 평균값을 알아봅니다.

```
import pandas as pd

g_list = ['a','a','a','b','b','b','c','c','c','c']
v1_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
v2_list = [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

df =  pd.DataFrame({'grp': g_list, 'v1': v1_list, 'v2': v2_list}) # 그룹핑을 할 수 있는 컬럼을 가진 DataFrame 생성
```

```
df[['v1', 'v2']].mean() # 전체 평균
```

```
v1     5.5
v2    15.5
dtype: float64
```

```
df.groupby('grp')['v1'].mean() # 그룹별 평균
```

```
grp
a    2.0
b    5.0
c    8.5
Name: v1, dtype: float64
```

그룹별로 여러개의 통계값도 구할 수 있습니다. v1 의 평균, 최대값, 총합을 알아봅니다.

```
df.groupby('grp')['v1'].agg(['mean','max','sum'])
```

grp	mean	max	sum
a	2.0	3	6
b	5.0	6	15
c	8.5	10	34

그룹별로 v1 과 v2 의 평균, 최대값, 총합을 알아봅니다.

```
df.groupby('grp')[['v1','v2']].agg(['mean','max','sum'])
```

	v1			v2		
	mean	max	sum	mean	max	sum
grp						
a	2.0	3	6	12.0	13	36
b	5.0	6	15	15.0	16	45
c	8.5	10	34	18.5	20	74

이번에는 그룹별로 v1 은 평균, v2 는 총합을 알아봅니다.

```
s = {'v1':'mean', 'v2':'sum'}
df.groupby('grp').agg(s)
```

	v1	v2
grp		
a	2.0	36
b	5.0	45
c	8.5	74

그룹별 최대값에서 최소값을 뺀 값을 알아봅니다. lambda 함수를 이용했습니다. lambda 함수의 자세한 활용법은 다루지 않도록 하겠습니다. Apply 함수를 이용한 경우와 Transform 함수를 이용한 경우의 차이점을 알아야 합니다. Apply 를 이용하면 생성된 그룹의 갯 수 만큼의 행을 리턴합니다. Transform 은 그룹핑하기 전의 데이터 행의 갯 수 만큼을 반환합니다. 그룹별 요약된 정보를 원래 데이터에 추가하고 싶을 때는 Transform 이 사용됩니다.

```
df.groupby('grp')['v1'].apply(lambda x: x.max() - x.mean())
```

```
grp
a    1.0
b    1.0
c    1.5
Name: v1, dtype: float64
```

```
df.groupby('grp')['v1'].transform(lambda x: x.max() - x.mean())
```

```
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
5    1.0
6    1.5
7    1.5
8    1.5
9    1.5
Name: v1, dtype: float64
```

이번에는 그룹핑을 하기 위해 활용되는 pd.qcut() 혹은 pd.cut() 메소드에 대하여 알아보겠습니다. 어떤 변수를 그룹 별로 분석하고 싶습니다. 예를 들어, 섹터가 그룹이라면 groupby('섹터')['수익률'].mean() 명령으로 섹터별로 각 섹터에 속하는 종목들의 수익율 평균울 구할 수 있습니다. 하지만 그룹 변수가 없고 연속형 변수를 구간으로 나누어 그룹화하고 싶은 경우 pd.cut() 나 pd.qcut() 을 이용합니다. pd.cut 은 직접 구간을 지정해 그룹을 만들고, pd.qcut 은 분위 수를 이용하여 구간을 만듭니다.

```
pd.qcut(Series, q=10) # 십 분위수로 구간 생성
pd.cut(Series, bins=[a1, a2, a3]) # bins 인수를 이용하면  (a1, a2], (a2, a3]로 구간 생성
```

np.arange(100) 을 이용하여 0 부터 99까지 값을 생성한 후 pd.qcut 와 pd.cut 을 사용 해 보겠습니다.

```
import numpy as np
import pandas as pd

a_list = np.arange(100)
df = pd.DataFrame({'a': a_list})
df.head()
```

	a
0	0
1	1
2	2
3	3
4	4

10 이하의 숫자와 90 초과의 숫자는 해당 구간이 없어서 그룹핑이 되지 않았습니다.

```
rank = pd.cut(df['a'], bins=[10, 25, 75, 90])
df.groupby(rank)['a'].agg(['min','max','count'])
```

	min	max	count
a			
(10, 25]	11	25	15
(25, 75]	26	75	50
(75, 90]	76	90	15

bins 구간이 모든 값을 포함하도록 하기 위해서는 아래와 같이 bins 를 설정합니다.

```
rank = pd.cut(df['a'], bins=[-np.inf, 10, 25, 75, 90, np.inf])
df.groupby(rank).agg(['min','max','count'])
```

	a		
	min	max	count
a			
(-inf, 10.0]	0	10	11
(10.0, 25.0]	11	25	15
(25.0, 75.0]	26	75	50
(75.0, 90.0]	76	90	15
(90.0, inf]	91	99	9

이번에는 제가 주로 활용하는 pd.qcut 입니다. pd.cut 은 bins 로 구간을 설정해야 하나, qcut 는 q 인수로 분위수를 이용하여 구간을 만듭니다. 아래 결과를 보시면 q 의 역할을 아실 수 있을 것이라고 생각합니다.

```
rank = pd.qcut(df['a'], q=10)
df.groupby(rank)['a'].agg(['min','max','count'])
```

a	min	max	count
(-0.001, 9.9]	0	9	10
(9.9, 19.8]	10	19	10
(19.8, 29.7]	20	29	10
(29.7, 39.6]	30	39	10
(39.6, 49.5]	40	49	10
(49.5, 59.4]	50	59	10
(59.4, 69.3]	60	69	10
(69.3, 79.2]	70	79	10
(79.2, 89.1]	80	89	10
(89.1, 99.0]	90	99	10

```
rank = pd.qcut(df['a'], q=5)
df.groupby(rank)['a'].agg(['min','max','count'])
```

a	min	max	count
(-0.001, 19.8]	0	19	20
(19.8, 39.6]	20	39	20
(39.6, 59.4]	40	59	20
(59.4, 79.2]	60	79	20
(79.2, 99.0]	80	99	20