

# 파이썬 라이브러리를 활용한 데이터 분석

## 10장 데이터 집계와 그룹 연산

2020.07.03 3h

# 그룹 통계와 보고서 작성

## • 그룹 연산

- 그룹으로 나눠 여러 총계량을 파악
  - 연산 `groupby()`



## • 피벗테이블

- 데이터를 하나 이상의 키(index)로 수집해서 분류해 열(columns)과 값(value)의 평균(옵션 `aggfunc`로 수정 가능)을 기술
  - 연산 `pivot_table()`

	student	school	class	grade
0	Andy	Z	english	10
1	Bernie	Y	english	100
2	Cindy	Z	english	1000
3	Deb	Y	english	10000
4	Andy	Z	math	20
5	Bernie	Y	math	200
6	Cindy	Z	math	2000
7	Deb	Y	math	20000
8	Andy	Z	physics	30
9	Bernie	Y	physics	300
10	Cindy	Z	physics	3000
11	Deb	Y	physics	30000

### Pivot 1

```
df_long.pivot_table(index=["student", "school"],
                     columns='class',
                     values='grade', reset_index())
```

class	student	school	english	math	physics
0	Andy	Z	10	20	30
1	Bernie	Y	100	200	300
2	Cindy	Z	1000	2000	3000
3	Deb	Y	10000	20000	30000

## • 주요 내용

- 키로 판다스 객체를 여러 조각으로 나누기
- 그룹 요약 통계 계산: 합, 평균, 표준편차, 사용자 정의 함수
- 피벗테이블 교차일람표 작성
- 변위치 분석 등

# 참고 사이트

- 국내

- <https://datascienceschool.net/view-notebook/76dcd63bba2c4959af15bec41b197e7c/>
- <https://rfriend.tistory.com/383>
- <https://ponyozzang.tistory.com/291>

- 국외

- <https://www.geeksforgeeks.org/pandas-groupby/>
- <https://realpython.com/pandas-groupby/>
- [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/groupby.html](https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html)
- <https://towardsdatascience.com/how-to-use-the-split-apply-combine-strategy-in-pandas-groupby-29e0eb44b62e>

# 파일 ch10-study.ipynb

# Python pandas의 groupby() 메카닉

## • 집단, 그룹별로 데이터를 집계, 요약하는 방법

– Split => Apply (function) => Combine

- 전체 데이터를 그룹 별로 나누고(split), 각 그룹별로 집계 함수를 적용(apply) 한 후, 그룹별 집계 결과를 하나로 합치는(combine) 단계를 거치게 됨

– 분리

- 키를 기준으로 분리

– axis=0, 1

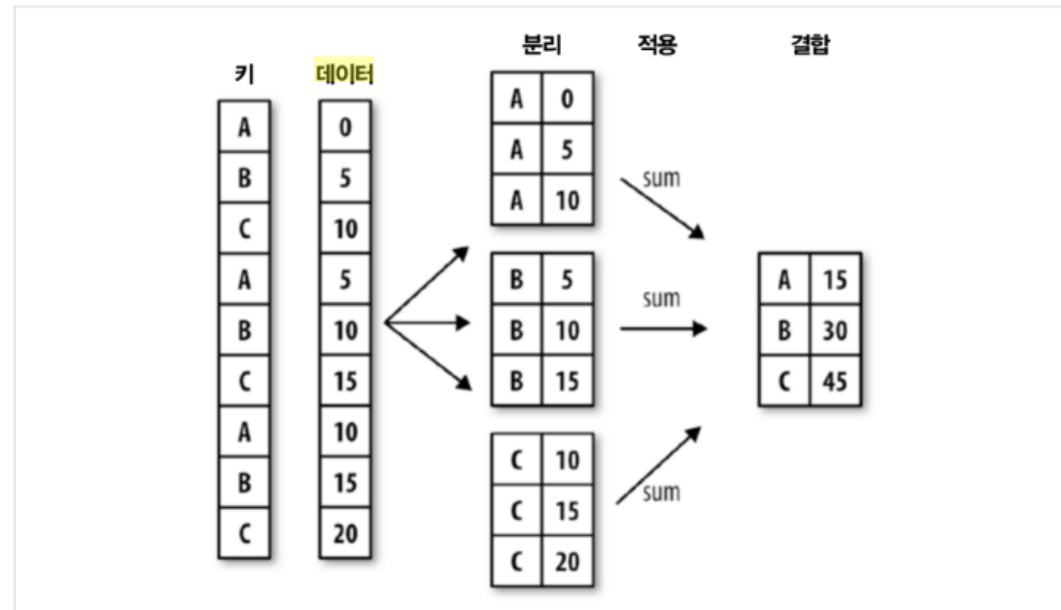
– 적용

- 필요 함수의 적용

– 결합

- 함수를 적용한 결과를 객체에 결합

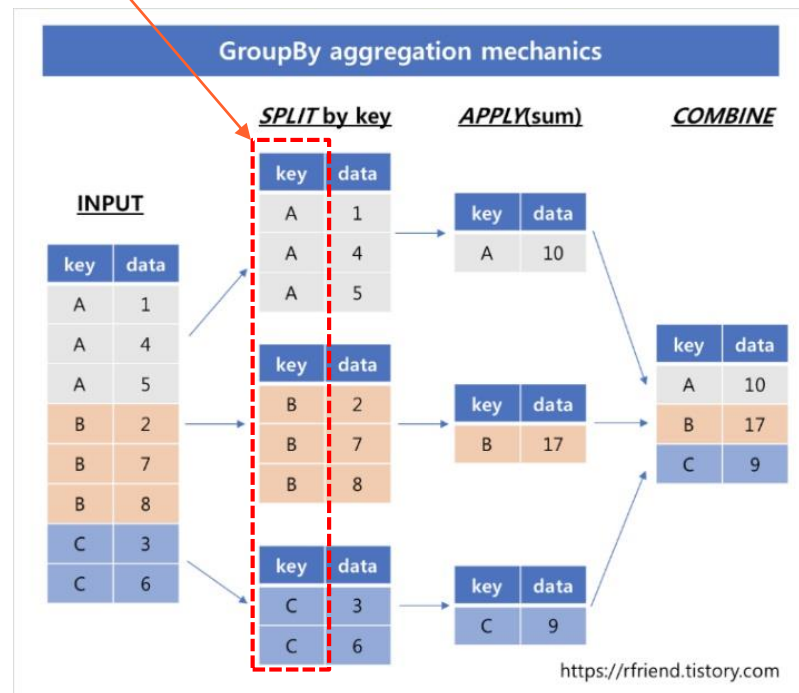
그림 10-1 그룹 연산 예시



# 그룹 색인

## • 색인의 다양한 형태

- 그룹으로 묶을 축과 동일한 길이의 리스트나 배열
- DataFrame의 컬럼 이름을 지칭하는 값
- 그룹으로 묶을 값과 그룹 이름에 대응하는 사전이나 Series 객체
- 축 색인 혹은 색인 내의 개별 이름에 대해 실행되는 함수

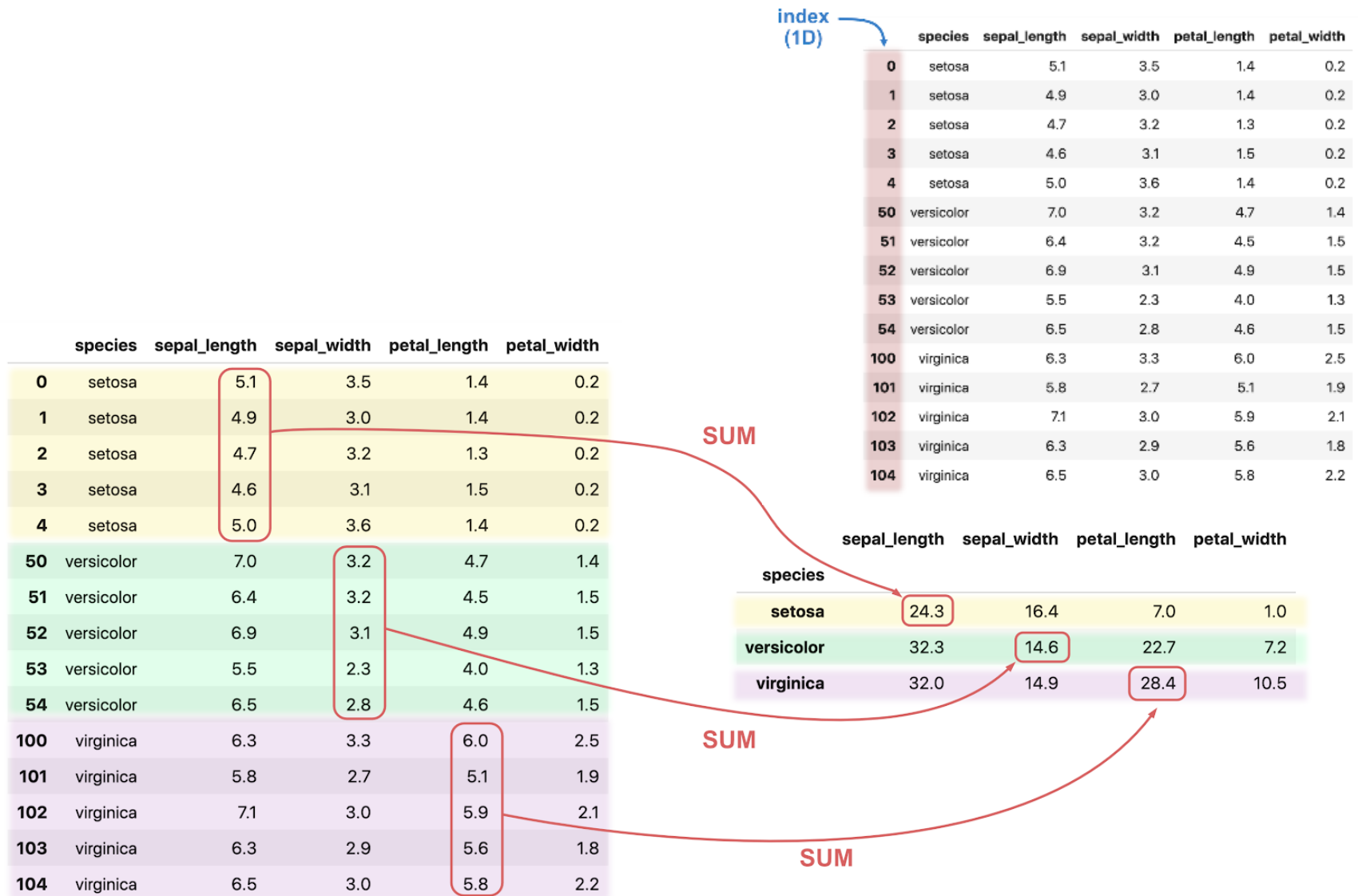


# Groupby apply(분리 적용 결합)의 이해

- <https://towardsdatascience.com/how-to-use-the-split-apply-combine-strategy-in-pandas-groupby-29e0eb44b62e>



# Groupby apply(분리 적용 결합)의 사례





# Pandas GroupBy 이해: splitting

- 1. Group the unique values from the Team column

	Name	Team	Position	Age	Weight
0	Avery Bradley	Boston Celtics	PG	25.0	180.0
1	Jae Crowder	Boston Celtics	SF	25.0	235.0
2	John Holland	Boston Celtics	SG	27.0	205.0
3	R.j. Hunter	Boston Celtics	SG	22.0	185.0
4	Sergey Karasev	Brooklyn Nets	SG	22.0	208.0
5	sean Kilpatrick	Brooklyn Nets	SG	26.0	219.0
6	Shane Larkin	Brooklyn Nets	PG	23.0	175.0
7	Brook Lopez	Brooklyn Nets	C	28.0	275.0
8	Chris Johnson	Utah Jazz	SF	26.0	206.0
9	Trey Lyles	Utah Jazz	PF	20.0	234.0
10	Shelvin Mack	Utah Jazz	PG	26.0	203.0
11	Raul Pleiss	Utah Jazz	PG	24.0	179.0

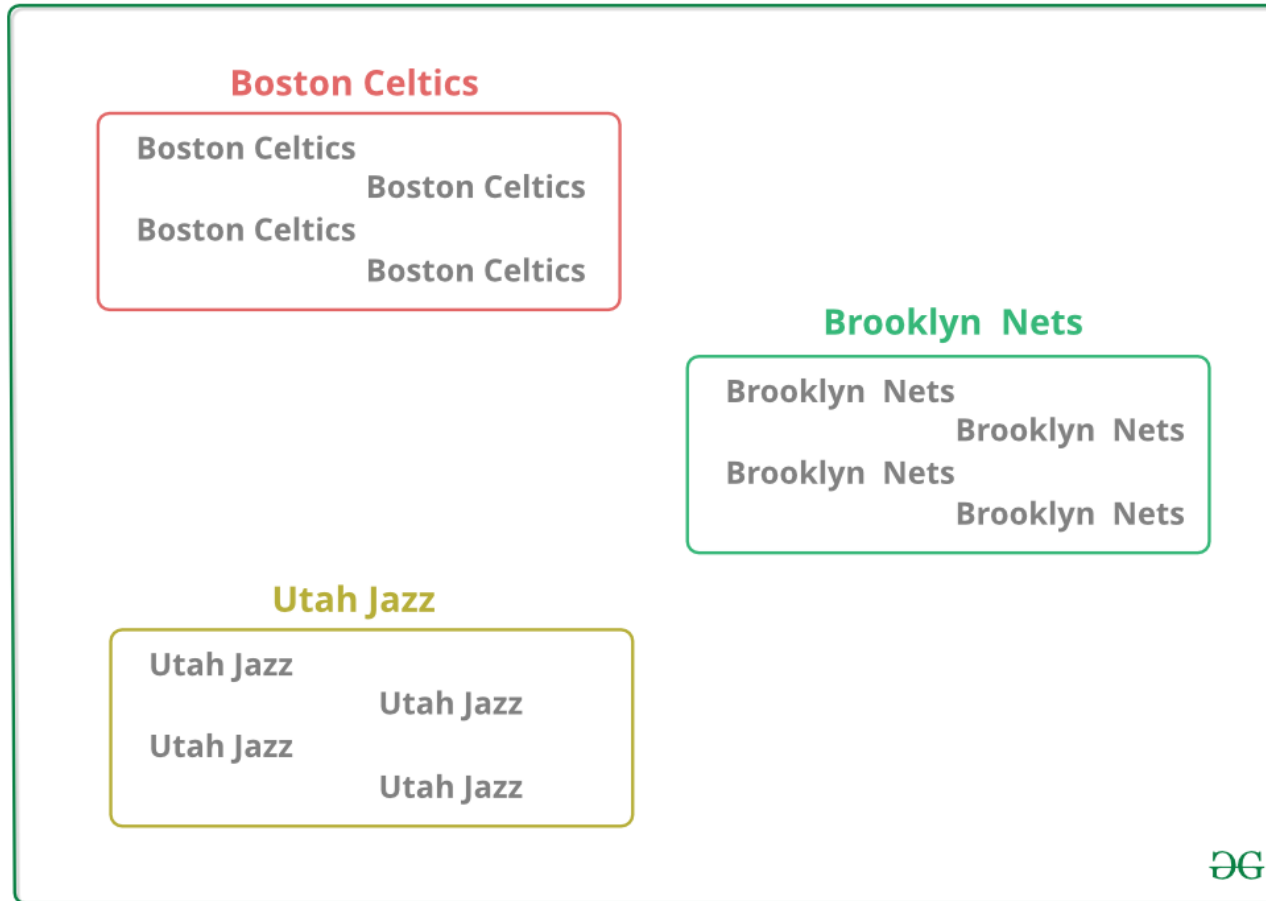
Boston Celtics  
Boston Celtics  
Boston Celtics  
Boston Celtics

Brooklyn Nets  
Brooklyn Nets  
Brooklyn Nets  
Brooklyn Nets

Utah Jazz  
Utah Jazz  
Utah Jazz  
Utah Jazz

# Pandas GroupBy의 이해: splited bucket

- 2. Now there's a bucket for each group



# Pandas GroupBy의 이해: bucket made

- 3. Toss the other data into the buckets

	Name	Team	Position	Age	Weight
0	Avery Bradley	Boston Celtics	PG	25.0	180.0
1	Jae Crowder	Boston Celtics	SF	25.0	235.0
2	John Holland	Boston Celtics	SG	27.0	205.0
3	R.j. Hunter	Boston Celtics	SG	22.0	185.0
4	Sergey Karasev	Brooklyn Nets	SG	22.0	208.0
5	Sean Kilpatrick	Brooklyn Nets	SG	26.0	219.0
6	Shane Larkin	Brooklyn Nets	PG	23.0	175.0
7	Brook Lopez	Brooklyn Nets	C	28.0	275.0
8	Chris Johnson	Utah Jazz	SF	26.0	206.0
9	Trey Lyles	Utah Jazz	PF	20.0	234.0
10	Shelvin Mack	Utah Jazz	PG	26.0	203.0
11	Raul Pleiss	Utah Jazz	PG	24.0	179.0

Name	Position	Age	Weight
Avery Bradley	PG	25.0	180.0
Jae Crowder	SF	25.0	235.0
John Holland	SG	27.0	205.0
R.j. Hunter	SG	22.0	185.0

Name	Position	Age	Weight
Sergey Karasev	SG	22.0	208.0
Sean Kilpatrick	SG	26.0	219.0
Shane Larkin	PG	23.0	175.0
Brook Lopez	C	28.0	275.0

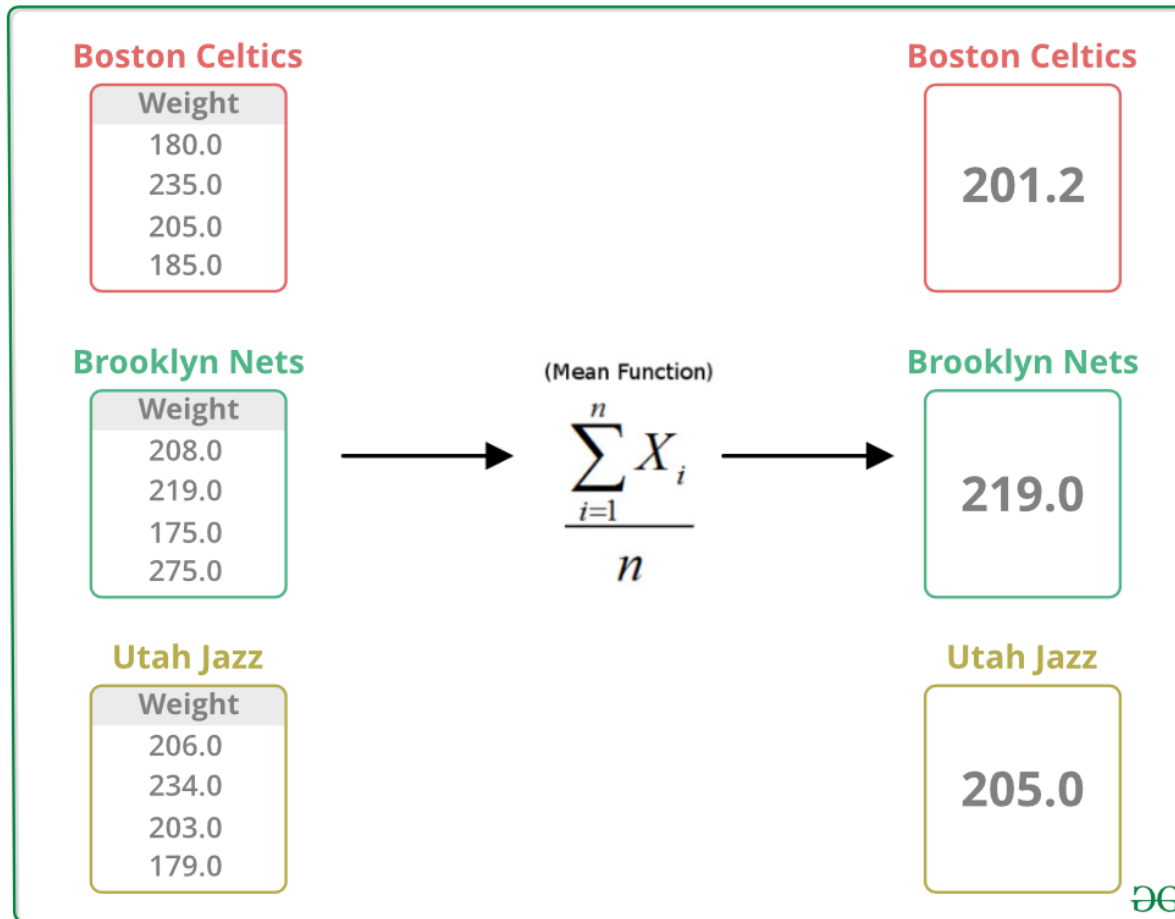
  

Name	Position	Age	Weight
Chris Johnson	SF	26.0	206.0
Trey Lyles	PF	20.0	234.0
Shelvin Mack	PG	26.0	203.0
Raul Pleiss	PG	24.0	179.0



# Pandas GroupBy의 이해: apply function to bucket

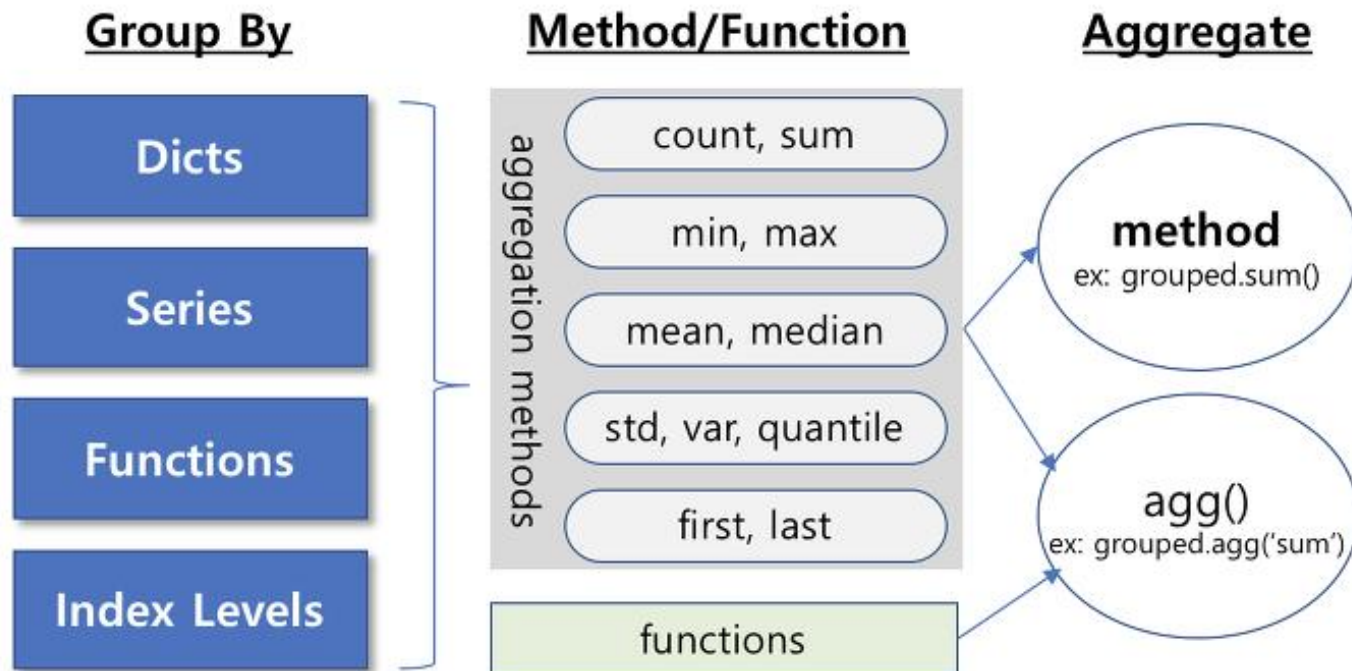
- 4. Apply a function on the weight column of each bucket.



# Python pandas의 다양한 GroupBy 집계 방법



## pandas Group By



<http://rfriend.tistory.com>

# 데이터 집계와 그룹 연산

- 데이터 분석 과정

- 여러 자료를 취합해 하나의 테이블 집합 준비
- 그룹 통계를 구하거나
- 피벗 테이블로 보고서 만들고
- 시각화 하기

- 데이터 집합을 나누고 요약하는 방법 제공

- Groupby
  - SQL 기능과 비슷
  - 복잡한 그룹 연산도 적당한 python, pandas, numpy 함수의 조합으로 해결

- 주요 내용

- 하나 이상의 키를 이용해 pandas 객체를 여러 조각으로 나누는 방법
- 그룹 요약 집계 함수(합, 평균, 표준편차, 사용자 정의 함수) 적용 방법
- 피벗 테이블과 교차일람표 구하는 방법
- 변위치 분석과 통계 집단 분석을 수행하는 방법

# 10장 데이터 집계와 그룹 연산

## GroupBy 메카닉

2h

# 분리 적용 결합

## • 해들리 위کم

- R 프로그래밍 패키지의 저자
- 분리-적용-결합 용어 사용
  - 하나 이상의 키로 분리
  - 함수를 각 그룹에 적용해 새로운 값을 생성
  - 함수 적용 결과를 하나의 객체로 결합

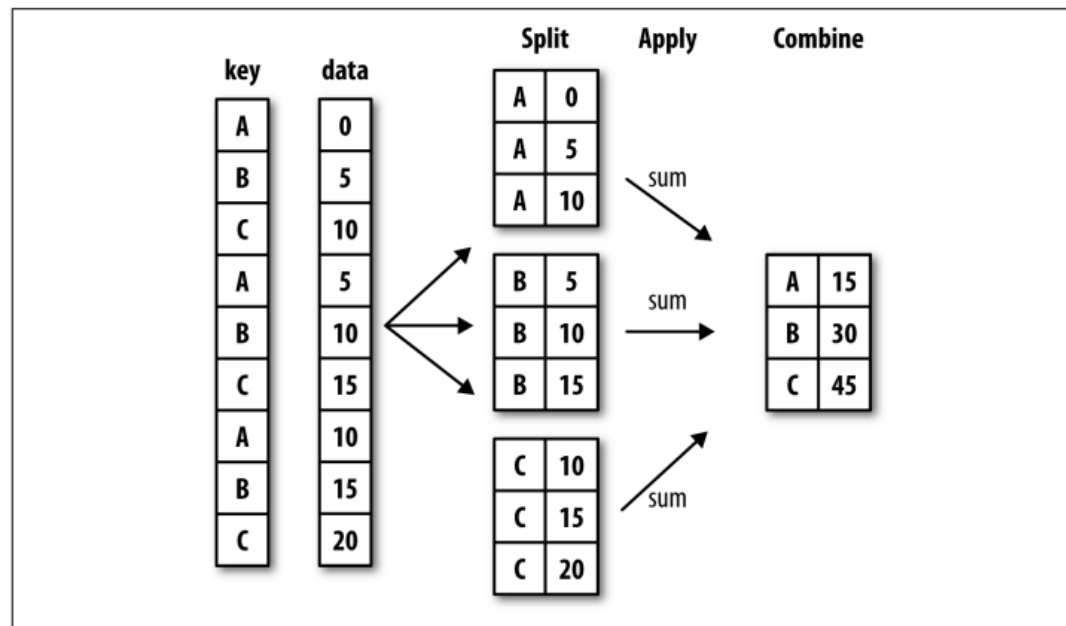


Figure 10-1. Illustration of a group aggregation



# 키로 묶고, 칼럼으로 선택, 함수 적용

- `df['칼럼']`  
`.groupby(df['키'])`  
`.함수()`

- 데이터가 그룹 색인에 따라 수집되고 `key1` 칼럼에 있는 유일한 값으로 색인되는 새로운 시리즈 객체가 생성

```
In [3]: 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 [3]:
```

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
In [7]: grouped = df['data1'].groupby(df['key1'])
grouped
```

```
Out [7]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x00000260B3D7D4C8>
```

```
In [9]: grouped.mean()
```

```
Out [9]: key1
a      0.746672
b     -0.537585
Name: data1, dtype: float64
```

# 키로 묶고, 칼럼으로 선택, 함수 적용

- `df['칼럼']`  
`.groupby(df['키'])`  
`.함수()`
- Key1으로 묶고 각 그룹에서 data1 평균
  - Data1에 대해 groupby 메소드를 호출하고 key1칼럼을 참조
  - 반환값
    - GroupBy 객체
  - 평균값 구하기
    - GroupBy 객체에 메소드 `mean()` 호출

```
In [3]: 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 [3]:
```

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
In [7]: grouped = df['data1'].groupby(df['key1'])
grouped
```

```
Out [7]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x00000260B3D7D4C8>
```

```
In [9]: grouped.mean()
```

```
Out [9]: key1
a    0.746672
b   -0.537585
Name: data1, dtype: float64
```

# 두 개 이상의 키의 조합

- 데이터가 그룹 색인에 따라 수집되고 [key1, key2] 칼럼에 있는 계층적으로 색인되는 새로운 시리즈 객체가 생성

```
In [10]: means = df.groupby(['key1', 'key2'])['data1'].mean()
```

```
Out [10]:
```

key1	key2	data1
a	one	0.880536
	two	0.478943
b	one	-0.519439
	two	-0.555730

Name: data1, dtype: float64

```
In [11]: means.unstack()
```

```
Out [11]:
```

	key2	one	two
key1			
a		0.880536	0.478943
b		-0.519439	-0.555730

```
In [3]: 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 [3]:

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

# Groupby 키: 내부 열이 아니어도 가능

## • 그룹의 색인

– 모두 시리즈 객체

### • 길이만 같다면 어떤 배열도 가능

– 행수와 groupby의 키 배열의 수가 같으면 가능

In [14]:

df

Out [14]:

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

행의 길이와 같으면 가능

```
In [12]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
```

```
Out [12]: California 2005    0.478943
           2006   -0.519439
           Ohio    2005   -0.380219
           2006    1.965781
           Name: data1, dtype: float64
```

# 그룹 이후에 특별한 칼럼을 지정하지 않는 경우

- **df.groupby('키').함수()**
  - df.groupby('key1').mean()
  - 지정한 키를 제외한 전체 데이터(수 자료형 열)에 대한 결과
- **성가신 컬럼(nuisance column)**
  - Key2가 표시되지 않은 이유
    - 함수 인자로 적합하지 않은 자료 값
    - 숫자가 아니라서

In [25]: df

Out [25]:

	key1	key2	data1	data2
0	a	one	1.007189	0.886429
1	a	two	-1.296221	-2.001637
2	b	one	0.274992	-0.371843
3	b	two	0.228913	1.669025
4	a	one	1.352917	-0.438570

In [26]: df.groupby('key1').mean()

Out [26]:

	data1	data2
key1		
a	0.354628	-0.517926
b	0.251952	0.648591

# 함수 size()

- 키를 2개로

```
In [27]: df.groupby(['key1', 'key2']).mean()
```

Out [27]:

		data1	data2
key1	key2		
a	one	1.180053	0.223930
	two	-1.296221	-2.001637
b	one	0.274992	-0.371843
	two	0.228913	1.669025

- 각 그룹의 크기를 조회
  - size()

```
In [28]: df.groupby(['key1', 'key2']).size()
```

```
Out [28]: key1  key2
a      one      2
        two      1
b      one      1
        two      1
dtype: int64
```

# 그룹 간 순회하기

## • GroupBy 객체

### – 이터레이션 지원

#### • 튜플: (그룹 이름, 데이터 묶음)

- 그룹 이름은 키나 키들의 튜플
- 키에 따른 데이터 묶음(데이터 프레임 등), 모든 열 포함

```
In [28]: for (k1, k2), group in df.groupby(['key1', 'key2']):
          print((k1, k2))
          print(group)
          print()
```

```
('a', 'one')
  key1 key2  data1  data2
0    a  one  1.007189  0.886429
4    a  one  1.352917 -0.438570
```

```
('a', 'two')
  key1 key2  data1  data2
1    a  two -1.296221 -2.001637
```

```
('b', 'one')
  key1 key2  data1  data2
2    b  one  0.274992 -0.371843
```

```
('b', 'two')
  key1 key2  data1  data2
3    b  two  0.228913  1.669025
```

```
In [29]: df
```

```
Out [29]:
```

	key1	key2	data1	data2
0	a	one	1.007189	0.886429
1	a	two	-1.296221	-2.001637
2	b	one	0.274992	-0.371843
3	b	two	0.228913	1.669025
4	a	one	1.352917	-0.438570

```
In [30]: for name, group in df.groupby('key1'):
          print(name)
          print(group)
          print()
```

```
a
  key1 key2  data1  data2
0    a  one  1.007189  0.886429
1    a  two -1.296221 -2.001637
4    a  one  1.352917 -0.438570
```

```
b
  key1 key2  data1  data2
2    b  one  0.274992 -0.371843
3    b  two  0.228913  1.669025
```

# 키에 원하는 자료만 추출 가능

- 사전형으로 만들어
  - `dict(list(df.groupby('키')))`
- 하나 참조
  - 객체['키의\_하나']

```
In [31]: dict(list(((1, 10), (2, 20), (3, 30))))
```

```
Out [31]: {1: 10, 2: 20, 3: 30}
```

```
In [32]: pieces = dict(list(df.groupby('key1')))
pieces
```

```
Out [32]: 'a'  key1 key2      data1      data2
0    a  one  1.007189  0.886429
1    a  two -1.296221 -2.001637
4    a  one  1.352917 -0.438570,
'b'  key1 key2      data1      data2
2    b  one  0.274992 -0.371843
3    b  two  0.228913  1.669025
```

```
In [33]: pieces['b']
```

```
Out [33]:
```

	key1	key2	data1	data2
2	b	one	0.274992	-0.371843
3	b	two	0.228913	1.669025



# 다른 축으로 groupby

- 기본이 axis=0
- 다른 축도 가능
- df의 칼럼 타입에 따라 그룹으로

```
In [37]: df.dtypes
```

```
Out [37]: key1      object
          key2      object
          data1   float64
          data2   float64
          dtype: object
```

```
In [38]: grouped = df.groupby(df.dtypes, axis=1)
```

```
In [39]: for dtype, group in grouped:
          print(dtype)
          print(group)
          print()
```

```
float64
      data1      data2
0  1.007189  0.886429
1 -1.296221 -2.001637
2  0.274992 -0.371843
3  0.228913  1.669025
4  1.352917 -0.438570
```

```
object
      key1 key2
0      a  one
1      a  two
2      b  one
3      b  two
4      a  one
```

# 칼럼이나 칼럼의 일부만 선택

- `df.groupby('key1')['data1'] == df['data1'].groupby(df['key1'])`
  - `df.groupby('key1')['data1'].mean()`
    - 결과가 Series
- `df.groupby('key1')[['data2']] == df[['data2']].groupby(df['key1'])`
  - `df.groupby('key1')['data2'].mean()`
    - 결과가 하나의 열로 구성된 DataFrame
- data2 칼럼에 대해서만 평균 구하고
  - 결과를 DataFrame으로 받고 싶으면

```
In [48]: df.groupby(['key1', 'key2'])[['data2']].mean()
```

```
Out[48]:
```

		data2
key1	key2	
a	one	0.223930
	two	-2.001637
b	one	-0.371843
	two	1.669025

# DataFrameGroupBy와 SeriesGroupBy(1)

- Groupby 색인으로 얻은 객체에 열 참조 방법
  - 리스트나 배열로 인자로 넘긴 경우
    - DataFrameGroupBy 객체가 됨
  - 하나의 컬럼 이름만 인자로 넘긴 경우
    - SeriesGroupBy

```
In [11]: df['data1'].groupby(df['key1'])
```

```
Out[11]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x0000010FC1D68608>
```

```
In [12]: df[['data1']].groupby(df['key1'])
```

```
Out[12]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000010FC1D6F5C8>
```

```
In [13]: df.groupby(df['key1'])['data2']
```

```
Out[13]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x0000010FC1CFCE48>
```

```
In [14]: df.groupby(df['key1'])[['data2']]
```

```
Out[14]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000010FC1D3F788>
```

# DataFrameGroupBy와 SeriesGroupBy(2)

- 평균의 결과
  - 위는 열이 하나인 DataFrame, 아래는 Series

```
In [63]: df.groupby(['key1', 'key2'])[['data2']].mean()
```

```
Out[63]:
```

		data2
key1	key2	
a	one	0.223930
	two	-2.001637
b	one	-0.371843
	two	1.669025

```
In [65]: df.groupby(['key1', 'key2'])['data2'].mean()
```

```
Out[65]: key1  key2
a      one    0.223930
       two   -2.001637
b      one   -0.371843
       two    1.669025
Name: data2, dtype: float64
```

# 그룹 별로 칼럼의 값을 모두 더하기, **axis=1**

- 축 **axis=1**로 중심으로 더하기
- 더하는 그룹은 사전으로
  - a, b, e: red로 그룹화
  - c, d: blue로 그룹화
  - f:

- 사용하지 않는 그룹 키는 아무 문제 없음

```
In [71]: people.iloc[2:3, [1, 2]] = np.nan # Add a few NA values
people
```

Out [71]:

	a	b	c	d	e
Joe	0.286350	0.377984	-0.753887	0.331286	1.349742
Steve	0.069877	0.246674	-0.011862	1.004812	1.327195
Wes	-0.919262	NaN	NaN	0.758363	-0.660524
Jim	0.862580	-0.010032	0.050009	0.670216	0.852965
Travis	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302

```
In [74]: mapping = {'a': 'red', 'b': 'red',
                    'c': 'blue', 'd': 'blue',
                    'e': 'red', 'f': 'orange'}
```

```
In [75]: by_column = people.groupby(mapping, axis=1)
by_column.sum()
```

Out [75]:

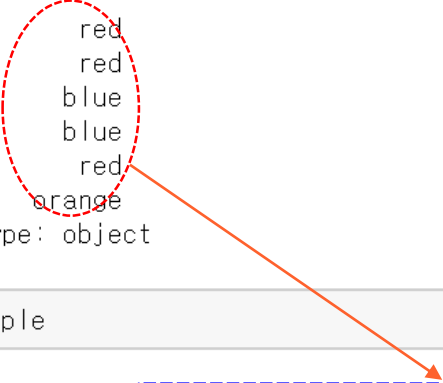
	blue	red
Joe	-0.422601	2.014076
Steve	0.992950	1.643745
Wes	0.758363	-1.579786
Jim	0.720225	1.705513
Travis	-2.956703	-2.197664

# 키 대신 Series로도 가능

- 함수 `count()`
  - 출현 횟수
    - NaN는 제외

```
In [76]: map_series = pd.Series(mapping)
map_series
```

```
Out [76]: a      red
b      red
c     blue
d     blue
e      red
f    orange
dtype: object
```



```
In [77]: people
```

```
Out [77]:
```

	a	b	c	d	e
Joe	0.286350	0.377984	-0.753887	0.331286	1.349742
Steve	0.069877	0.246674	-0.011862	1.004812	1.327195
Wes	-0.919262	NaN	NaN	0.758363	-0.660524
Jim	0.862580	-0.010032	0.050009	0.670216	0.852965
Travis	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302

```
In [78]: people.groupby(map_series, axis=1).count()
```

```
Out [78]:
```

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

# 함수로 그룹핑하기

- 그룹 색인으로 넘기는 함수는 색인 값 하나마다 한 번씩 호출
  - 반환 값은 그 그룹의 이름으로 사용
- 함수 `len`을 그룹핑 인자로
  - 사람의 이름을 색인 값으로
  - 이름의 길이 별로 그룹핑
- `axis=1`
  - 모든 열의 길이로 그룹핑
    - 모두 1

In [82]: `people`

Out [82]:

	a	b	c	d	e
Joe	0.286350	0.377984	-0.753887	0.331286	1.349742
Steve	0.069877	0.246674	-0.011862	1.004812	1.327195
Wes	-0.919262	NaN	NaN	0.758363	-0.660524
Jim	0.862580	-0.010032	0.050009	0.670216	0.852965
Travis	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302

In [83]: `people.groupby(len).sum()`

Out [83]:

	a	b	c	d	e
3	0.229668	0.367952	-0.703877	1.759864	1.542183
5	0.069877	0.246674	-0.011862	1.004812	1.327195
6	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302

In [84]: `people.groupby(len, axis=1).sum()`

Out [84]:

	1
Joe	1.591475
Steve	2.636695
Wes	-0.821423
Jim	2.425738
Travis	-5.154367

# 여러 개의 키를 함수나 사전, 시리즈의 배열 사용

In [86]:

people

Out [86]:

	a	b	c	d	e
Joe	0.286350	0.377984	-0.753887	0.331286	1.349742
Steve	0.069877	0.246674	-0.011862	1.004812	1.327195
Wes	-0.919262	NaN	NaN	0.758363	-0.660524
Jim	0.862580	-0.010032	0.050009	0.670216	0.852965
Travis	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302

In [85]:

```
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()
```

Out [85]:

		a	b	c	d	e
3	one	-0.919262	0.377984	-0.753887	0.331286	-0.660524
	two	0.862580	-0.010032	0.050009	0.670216	0.852965
5	one	0.069877	0.246674	-0.011862	1.004812	1.327195
6	two	-0.955869	-0.023493	-2.304234	-0.652469	-1.218302



# 다중 색인에서 색인 단계로 그룹핑

- 옵션 `level=그룹핑_이름`

```
In [87]: columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'JP', 'JP'],
                                             [1, 3, 5, 1, 3]],
                                             names=['cty', 'tenor'])

columns
```

```
Out [87]: MultiIndex([( 'US', 1),
                     ( 'US', 3),
                     ( 'US', 5),
                     ( 'JP', 1),
                     ( 'JP', 3)],
                     names=['cty', 'tenor'])
```

```
In [88]: hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
hier_df
```

```
Out [88]:
```

	cty	US		JP	
	tenor	1	3	5	1
0		-1.332610	1.074623	0.723642	0.690002
1		-0.503087	-0.622274	-0.921169	-0.726213
2		0.051316	-1.157719	0.816707	0.433610
3		1.824875	-0.997518	0.850591	-0.131578

```
In [91]: hier_df.groupby(level='tenor', axis=1).count()
```

```
Out [91]:
```

tenor	1	3	5
0	2	2	1
1	2	2	1
2	2	2	1
3	2	2	1

```
In [90]: hier_df.groupby(level='cty', axis=1).count()
```

```
Out [90]:
```

cty	JP	US
0	2	3
1	2	3
2	2	3
3	2	3

## 가중치 평균



파이썬 GroupBy를 사용해 그룹별 가중 평균 구하기  
(Group Weighted Average by GroupBy Operation)

## Original Dataset

Group	Value	Weight
A	1	0.0
A	2	0.1
A	3	0.2
A	4	0.3
A	5	0.4
B	6	0.0
B	7	0.1
B	8	0.2
B	9	0.3
B	10	0.4

## 1 Split

Group	Value	Weight
A	1	0.0
A	2	0.1
A	3	0.2
A	4	0.3
A	5	0.4

Group	Value	Weight
B	6	0.0
B	7	0.1
B	8	0.2
B	9	0.3
B	10	0.4

## 2 Apply

Weighted Average of Group A

$$= \frac{1 \cdot 0.0 + 2 \cdot 0.1 + 3 \cdot 0.2 + 4 \cdot 0.3 + 5 \cdot 0.4}{0.0 + 0.1 + 0.2 + 0.3 + 0.4}$$

$$= \frac{4}{1} = 4$$

Weighted Average of Group B

$$= \frac{6 \cdot 0.0 + 7 \cdot 0.1 + 8 \cdot 0.2 + 9 \cdot 0.3 + 10 \cdot 0.4}{0.0 + 0.1 + 0.2 + 0.3 + 0.4}$$

$$= \frac{9}{1} = 9$$

## 3 Combine

Group	Weighted Average
A	4
B	9

R, Python 분석과 프로그래밍의 친구 <http://rfriend.tistory.com>

```
# group weighted average by category
grouped = df.groupby('grp_col')
weighted_avg_func = lambda g: np.average(g['val'], weights=g['weight'])
grouped.apply(weighted_avg_func)
```



## [ Python GroupBy ]

데이터프레임에 그룹 단위 통계량 칼럼 추가하기  
(Add Group-level statistics column at DataFrame)

			①	②	③	
	col	group_1	group_2	count_col	sum_col	max_col
0	1.0	a	c	2.0	3.0	2.0
1	2.0	a	c	2.0	3.0	2.0
2	NaN	a	c	2.0	3.0	2.0
3	4.0	a	d	1.0	4.0	4.0
4	NaN	a	d	1.0	4.0	4.0
5	6.0	b	e	2.0	13.0	7.0
6	7.0	b	e	2.0	13.0	7.0
7	NaN	b	e	2.0	13.0	7.0
8	9.0	b	f	2.0	19.0	10.0
9	10.0	b	f	2.0	19.0	10.0

- ① `df['count_col'] = df.groupby(['group_1', 'group_2']).col.transform('count')`
- ② `df['sum_col'] = df.groupby(['group_1', 'group_2']).col.transform('sum')`
- ③ `df['max_col'] = df.groupby(['group_1', 'group_2']).col.transform('max')`

<http://rfriend.tistory.com>