>> DataFrame group by 이해하기

group by : 그룹별로 묶어 집계할 때 많이 사용/ 각 그룹과 그룹에 속한 index를 dict형태로 표현

1. train.csv의 Pclass를 그룹지어 본다.

```
import pandas as pd
df = pd.read_csv('./data/train.csv')
df.head()
```

```
class_group = train_d.groupby('Pclass')
class_group
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x7fcdca(
8>
```

//객체형태로 출력된다.

>> 그룹의 속성을 살펴보기 위해 groups속성을 이용한다. dict 형태인 것을 확인하자.

```
class_group.groups
```

```
{1: Int64Index([  1,   3,   6,  11,  23,  27,  30,  31,  34,  35,
             ...
             853, 856, 857, 862, 867, 871, 872, 879, 887, 889],
            dtype='int64', length=216),
 2: Int64Index([  9,  15,  17,  20,  21,  33,  41,  43,  53,  56,
```

//딕셔너리 형태의 키와 값이 나온다.

>> 성별, 생존여부도 각각 구해보자

sex_g = train_d.groupby('Sex')

sex_g.groups

```
{'female': Int64Index([  1,   2,   3,   8,
             ...
             866, 871, 874, 875, 879, 880,
            dtype='int64', length=314),
 'male': Int64Index([  0,   4,   5,   6,
             ...
             873, 876, 877, 878, 881, 883,
            dtype='int64', length=577)}
```

//키는 female, male, 값은 각각내용

Survived_g = train_d.groupby('Survived')

Survived_g.groups

2. group별 함수적용해 보기

- 그룹 데이터에 적용 가능한 통계함수(NaN은 제외하여 연산)
- count, sum, mean, std, var, min, max 등 구할수 있다.

class_group.count() //NaN은 개수에 포함하지 않아 개수가 다르게 나온다.

class_group.sum() //숫자인것만 계산
class_group.mean()
class_group.max()
class_group.min()
>> sum, mean, max, min 해본다.

>> Age의 mean만 출력해 본다.
class_g.mean()['Age'] // 1등석의 평균 연령이 높다.

>> Survived(생존)의 mean을 구하여 비교해 보자
Survived_g.mean()['Age'] //생존/사망의 평균연령을 볼 수 있다.

3. 열을 index로 변경하기

train_d.head() //기본 index는 0부터 인 것을 확인
train_d.set_index('Pclass') //Pclass를 index로 한다. 중복된 값이 많아 인덱스로 좋지않다.

train_d.set_index(['Pclass','Sex']) //multi index를 지정한다. [ ] 유의하기!

>> index를 없애고 싶으면
  train2 = train_d.set_index(['Pclass','Sex'])
  train2.reset_index() //다시 열로 올라가는 것을 확인할 수 있다.

>> 'Age'를 index로 설정하고, 원하는 index로 그룹핑하기.
train_d.set_index('Age').groupby(level=0).mean()
//index가 하나면 level=0

>> 인덱스 2개지정하기
train_d.set_index(['Age','Sex']).groupby(level=1).mean()
//둘이상이면 level=0 or level=1 지정하여 그룹한다.
//모두 구하고싶으면 lavel=[0,1] 지정한다.
train3.groupby(level=[0,1]).mean()

나이를 좀 정리해서 다시 확인하기 위해 나이대를 정리한다.
>> 나이대별로 생존율 구하기
앞서 만들었던 age_categorize 함수를 이용하여 나이대별 평균을 구하고 생존율만 출력해 본다.

```
def age_categorize(age):
    if pd.isnull(age):
        return -1
    return age//10*10
```

```
train_d.set_index('Age').groupby(age_categorize).mean()
```

노약자의 생존율이 높은 것을 볼수 있다.
생존율만 보기위해 열을 지정한다.

```
train_d.set_index('Age').groupby(age_categorize).mean()['Survived']
```

```
-1.0    0.293785
 0.0    0.612903
 10.0   0.401961
 20.0   0.350000
 30.0   0.437126
```

>> 멀티레벨을 그룹하기
train_d.set_index(['Pclass','Sex']).groupby(level=[0,1]).mean()
//클래스별 남,녀 생존율을 비교해 볼수 있다.

>> aggregate함수를 이용해 멀티함수를 호출하여 구룹별로 한번에 볼 수 있다.
train_d.set_index(['Pclass','Sex']).groupby(level=[0,1]).aggregate([np.mean, np.sum, np.max])

>> DataFrame transform 이해하기

train_d.groupby('Pclass').mean() //원본과 다른 결과 복사 pclass개수만큼 row생김.
train_d.groupby('Pclass').transform(np.mean) //원본 index 그대로 이용 결과를 도출한다.
각 인덱스별로 평균은 1,2,3등석의 내용으로 채워진다.
이값을 기존의 데이터프레임에 추가하기가 좋다. 인덱스가 동일하므로

train_d['Age2']= train_d.groupby('Pclass').transform(np.mean)['Age']
그룹한 평균값중 Age값만 Age2열에 추가한다.
//Age2가 column으로 쉽게 추가할 수 있다.

4. pivot, pivot_table 함수의 이해 및 활용하기
pivot : dataframe의 형태를 원하는 데로 변경할 수 있다.
즉, index, column, 값 등 테이블의 모양을 원하는 데로 변경가능하다.

```
import numpy as np
import pandas as pd


df = pd.DataFrame({
    '지역' : ['서울','서울','서울','경기','경기','부산','서울','서울','부산','경기','경기','경기'],
    '요일' : ['월요일','화요일','수요일','월요일','화요일','월요일','목요일','금요일','화요일','수요일','목요일','금요일'],
    '강수량': [100,80,1000,200,200,100,50,100,200,100,50,100],
    '강수확률':[80,70,90,10,20,30,50,90,20,80,50,10]})
df
```

// 지역과 요일별로 강수량과 강수확률을 가진 데이터를 이용하여 원하는 테이블로 변경해 보자.

df = pd.DataFrame({ '지역' : ['서울','서울','서울','경기','경기','부산','서울','서울','부산','경기','경기','경기'],'요일' : ['월요일','화요일','수요일','월요일','화요일','월요일','목요일','금요일','화요일','수요일','목요일','금요일'],'강수량': [100,80,1000,200,200,100,50,100,200,100,50,100],'강수확률':[80,70,90,10,20,30,50,90,20,80,50,10]})

>> 지역은 인덱스 요일을 컬럼으로 나머지는 값으로 나타낸다.

df.pivot(index = '지역', columns='요일')

또는

```
df.pivot('지역','요일')
```

	강수량					강수확률				
요일	금요일	목요일	수요일	월요일	화요일	금요일	목요일	수요일	월요일	화요일
지역										
경기	100.0	50.0	100.0	200.0	200.0	10.0	50.0	80.0	10.0	20.0
부산	NaN	NaN	NaN	100.0	200.0	NaN	NaN	NaN	30.0	20.0
서울	100.0	50.0	1000.0	100.0	80.0	90.0	50.0	90.0	80.0	70.0

>> 요일을 인덱스, 지역을 컬럼, 강수량을 값으로 만들어 보자.

df.pivot('요일','지역')

//중복값이 있을경우

>> 서울의 두 번째 요일을 '월요일'로 수정하여 요일을 중복시킨다.

df.iloc[1,2]='월요일'

df.pivot('요일','지역') //중복된값 때문에 실행할 수없다. 이런경우는 pivot_table 으로 해야 한다.

pivot_table : 기능적으로 pivot과 동일, 중복데이터 처리가 가능한 것이 다르다.

df.pivot_table(df, '요일','지역') //중복값은 평균으로 채운다.

>> pivot_table의 'aggfunc'을 이용하여 중복의 값의 평균으로 구하여 값을 채운다.

```
df.pivot_table(df, index='요일',columns='지역', aggfunc=np.mean)
```

	강수량			강수확률		
지역	경기	부산	서울	경기	부산	서울
요일						
금요일	100.0	NaN	100.0	10.0	NaN	90.0
목요일	50.0	NaN	50.0	50.0	NaN	50.0
수요일	100.0	NaN	1000.0	80.0	NaN	90.0
월요일	200.0	100.0	90.0	10.0	30.0	75.0
화요일	200.0	200.0	NaN	20.0	20.0	NaN

5. stack, unstack 함수의 이해 및 활용하기
stack : 컬럼 레벨에서 인덱스 레벨로 변경
unstack : 인덱스 레벨에서 컬럼 레벨로 변경
둘은 역관계에 있다.

// 아래의 데이터를 만든다.

```
import numpy as np
import pandas as pd
df = pd.DataFrame({
    '지역' : ['서울','서울','서울','경기','경기','부산','서울','서울','부산','경기','경기','경기'],
    '요일' : ['월요일','화요일','수요일','월요일','월요일','화요일','월요일','목요일','금요일','화요일','수요일','목요일','금요일'],
    '강수량': [100,80,1000,200,200,100,50,100,200,100,50,100],
    '강수확률':[80,70,90,10,20,30,50,90,20,80,50,10]})
df
```

>> '지역'과 '요일'을 index로 변경해 new_df에 지정하기
new_df =df.set_index(['지역','요일'])
new_df

>> new_df.index //multi index 임을 확인한다.
>> '지역' index를 column으로 변경한다. 지역이 첫 번째인덱스 이므로 '0'.
new_df.unstack(0) //index값은 0부터 시작한다.
>> '요일' index를 column으로 변경해 보자
new_df.unstack(1)

>> stack을 이해하기 위해 다음의 예제를 살펴보자
new_df.unstack(1).stack(0) //인덱스(1)은 컬럼으로, 컬럼의(0)은 인덱스로변경

6. Concat 함수로 데이터 프레임 병합하기
>> 열레벨, 행레벨로 연결한다.

>>컬럼명이 같이 두 개의 데이터 프레임을 만든다.

```
df1 = pd.DataFrame({'key1': np.arange(10) ,'value1' : np.random.randn(10)})
df2 = pd.DataFrame({'key1': np.arange(10) ,'value1' : np.random.randn(10)})
```

```
df1
```

	key1	value1
0	0	-1.686507
1	1	1.712181
2	2	-0.748686
3	3	-0.175325
4	4	2.299809
5	5	-1.733592
6	6	-0.350002
7	7	-0.871389
8	8	-1.767577
9	9	-1.164493

>> pd.concat([df1, df2]) //df1과 df2를 연결한다.
행레벨로 연결된다. df1 아래에 df2 추가된다. index가 유지됨을 볼 수 있다.

>> pd.concat([df1, df2], ignore_index=True) //새로운 index로 된다.
>> 열레벨로 연결해 보자 : axis=1(열) axis=0(행, 기본)
pd.concat([df1, df2],axis=1)

>> 컬럼명이 같은 경우니 연결하기 쉬웠지만 다른경우를 보자
df3 = pd.DataFrame({'key2' : np.arange(10), 'value2' : np.random.randn(10)})
pd.concat([df1, df3], ignore_index=True)

7. Merge & join 함수로 데이터 프레임 병합하기 :특정컬럼끼리 병합하기

```
customer = pd.DataFrame({'customer_id' : np.arange(6),
                         'name': ['철수', '영희', '길동', '영수', '수민', '동건'],
                         '나이' : [40,20,21,30,31,18]})
customer
```
//고객정보

```
orders = pd.DataFrame({'customer_id' : [1,1,2,2,2,3,3,1,4,9],
                       'item': ['치약','칫솔','이어폰','헤드셋','수건','샘수','수건','치약','샘수','케이스'],
                       'quantity' : [1,2,1,1,3,2,2,3,2,1]})
orders.head()
```
//주문내역

>> customer_id를 기준으로 두 dataframe을 'merge'함수로 연결해보자.
pd.merge(customer, orders, on='customer_id', how='inner')

//on-> 연결할 컬럼, how='inner' 일치하는 데이터만 연결한다. 나오지 않는 데이터를 확인해 보라.

>> left, right, outer 비교해 보기
pd.merge(customer, orders, on='customer_id', how='right') //orders기준
pd.merge(customer, orders, on='customer_id', how='left') //customer기준
pd.merge(customer, orders, on='customer_id', how='outer') //모두

//index를 기준으로 join하기위해 'customer_id'를 index로 만든다

```
cst1 = customer.set_index('customer_id')
odr1 = orders.set_index('customer_id')
```

//index의 변화를 확인한다.

//index를 기준으로 merge를 한다.

```
pd.merge(cst1, odr1, left_index=True, right_index=True)
```

customer_id	name	나이	item	quantity
1	영희	20	치약	1
1	영희	20	칫솔	2
1	영희	20	치약	3
2	길동	21	이어폰	1
2	길동	21	헤드셋	1
2	길동	21	수건	3
3	영수	30	생수	2
3	영수	30	수건	2
4	수민	31	생수	2