

---

## ▣ Ch06 데이터 전처리

---

> 데이터 전처리(**preprocessing**)는 데이터 분석에 적절하게 데이터를 가공하는 작업

### [06-1] 데이터 전처리

> **pandas** 모듈을 가장 많이 사용

함수	기능
query()	행 추출
df[]	열(변수) 추출
sort_values()	정렬
groupby()	집단별로 나누기
assign()	변수 추가
agg()	통계치 구하기
merge()	데이터 합치기(열)
concat()	데이터 합치기(행)

### [06-2] 조건에 맞는 데이터만 추출하기

> **query()**로 행 추출

```
In [35]: ## [06-2] 조건에 맞는 데이터만 추출하기
# 실습용 데이터 프레임 생성
import pandas as pd
exam = pd.read_csv('exam.csv')
exam.head()
```

```
Out[35]:   id  nclass  math  english  science
          0      1      50       98       50
          1      2      60       97       60
          2      3      45       86       78
          3      4      30       98       58
          4      5      25       80       65
```

#### ▢ 조건에 맞는 데이터만 추출하기

> **query('필터링\_조건식')**을 사용

```
In [3]: exam['nclass'] == 1 #비교 연산: 열 추출 후 비교
```

```
Out[3]: 0      True  
1      True  
2      True  
3      True  
4     False  
5     False  
6     False  
7     False  
8     False  
9     False  
10    False  
11    False  
12    False  
13    False  
14    False  
15    False  
16    False  
17    False  
18    False  
19    False  
Name: nclass, dtype: bool
```

```
In [4]: exam[exam['nclass'] == 1] #exam['nclass'] == 1 비교 결과 True인 모든 행
```

```
Out[4]:   id  nclass  math  english  science  
0   1       1     50       98      50  
1   2       1     60       97      60  
2   3       1     45       86      78  
3   4       1     30       98      58
```

```
In [5]: ## [06-2] 조건에 맞는 데이터만 추출하기  
# exam에서 nclass가 1인 경우만 추출  
exam.query('nclass == 1') #query('필터링_조건식')
```

```
Out[5]:   id  nclass  math  english  science  
0   1       1     50       98      50  
1   2       1     60       97      60  
2   3       1     45       86      78  
3   4       1     30       98      58
```

```
In [6]: # 1반이 아닌 경우  
exam.query('nclass != 1')
```

Out[6]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
4	5	2	25	80	65
5	6	2	50	89	98
6	7	2	80	90	45
7	8	2	90	78	25
8	9	3	20	98	15
9	10	3	50	98	45
10	11	3	65	65	65
11	12	3	45	85	32
12	13	4	46	98	65
13	14	4	48	87	12
14	15	4	75	56	78
15	16	4	58	98	65
16	17	5	65	68	98
17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

## [] 초과, 미만, 이상, 이하 조건 걸기

In [4]:

```
# 수학 점수가 50점을 초과한 경우  
exam.query('math > 50')
```

Out[4]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
1	2	1	60	97	60
6	7	2	80	90	45
7	8	2	90	78	25
10	11	3	65	65	65
14	15	4	75	56	78
15	16	4	58	98	65
16	17	5	65	68	98
17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

## [] 여러 조건을 충족하는 행 추출하기

In [5]:

```
# 1반이면서 수학 점수가 50 점 이상인 경우  
exam.query('nclass == 1 & math >= 50')
```

Out[5]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	50	98	50
<b>1</b>	2	1	60	97	60

In [6]: # 수학 점수가 90점 이상이거나 영어 점수가 90점 이상인 경우  
exam.query('math >= 90 | english >= 90')

Out[6]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	50	98	50
<b>1</b>	2	1	60	97	60
<b>3</b>	4	1	30	98	58
<b>6</b>	7	2	80	90	45
<b>7</b>	8	2	90	78	25
<b>8</b>	9	3	20	98	15
<b>9</b>	10	3	50	98	45
<b>12</b>	13	4	46	98	65
<b>15</b>	16	4	58	98	65

## [] 리스트에 해당하는 행 추출하기

In [7]: # 1, 3, 5반에 해당하면 추출  
exam.query('nclass == 1 | nclass == 3 | nclass == 5')

Out[7]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	50	98	50
<b>1</b>	2	1	60	97	60
<b>2</b>	3	1	45	86	78
<b>3</b>	4	1	30	98	58
<b>8</b>	9	3	20	98	15
<b>9</b>	10	3	50	98	45
<b>10</b>	11	3	65	65	65
<b>11</b>	12	3	45	85	32
<b>16</b>	17	5	65	68	98
<b>17</b>	18	5	80	78	90
<b>18</b>	19	5	89	68	87
<b>19</b>	20	5	78	83	58

In [8]: # 1, 3, 5반에 해당하면 추출  
exam.query('nclass in [1, 3, 5]')

Out[8]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	50	98	50
<b>1</b>	2	1	60	97	60
<b>2</b>	3	1	45	86	78
<b>3</b>	4	1	30	98	58
<b>8</b>	9	3	20	98	15
<b>9</b>	10	3	50	98	45
<b>10</b>	11	3	65	65	65
<b>11</b>	12	3	45	85	32
<b>16</b>	17	5	65	68	98
<b>17</b>	18	5	80	78	90
<b>18</b>	19	5	89	68	87
<b>19</b>	20	5	78	83	58

## ▣ 추출한 행으로 데이터 만들기

In [36]:

```
# nclass가 1인 행 추출해 nclass1에 할당  
nclass1 = exam.query('nclass == 1') #nclass1은 행과, 열이 추출되었으므로 Data Frame  
nclass1['math'].mean()
```

Out[36]:

46.25

In [10]:

```
# nclass가 2인 행 추출해 nclass2에 할당  
nclass2 = exam.query('nclass == 2')  
nclass2['math'].mean()
```

Out[10]:

61.25

---

## (알아 두면 좋아요) 외부 변수를 이용해 추출하기

In [7]:

```
var = 3  
exam.query('nclass == @var')
```

Out[7]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>8</b>	9	3	20	98	15
<b>9</b>	10	3	50	98	45
<b>10</b>	11	3	65	65	65
<b>11</b>	12	3	45	85	32

In [ ]:

## [06-3] 필요한 변수만 추출하기

### [] 변수로 추출하기

```
In [12]: exam['math'] # math 변수로 추출
```

```
Out[12]: 0      50
1      60
2      45
3      30
4      25
5      50
6      80
7      90
8      20
9      50
10     65
11     45
12     46
13     48
14     75
15     58
16     65
17     80
18     89
19     78
Name: math, dtype: int64
```

> 단일 변수로 추출하면 결과는 Series형

```
In [13]: x = exam['math'] # 단일 변수로 추출하면 결과는 Series형
type(x)
```

```
Out[13]: pandas.core.series.Series
```

### [] 변수 나열(list)로 추출하기

```
In [14]: exam[['math', 'english', 'science']] # 변수 list로 추출
```

Out[14]:

	math	english	science
0	50	98	50
1	60	97	60
2	45	86	78
3	30	98	58
4	25	80	65
5	50	89	98
6	80	90	45
7	90	78	25
8	20	98	15
9	50	98	45
10	65	65	65
11	45	85	32
12	46	98	65
13	48	87	12
14	75	56	78
15	58	98	65
16	65	68	98
17	80	78	90
18	89	68	87
19	78	83	58

> 복수 개 변수로 추출하면 결과는 Data Frame형

In [37]:

```
x = exam[['math']] # 단일 변수라도 []로 추출하면 결과는 Data Frame형
type(x)
```

Out[37]:

```
pandas.core.frame.DataFrame
```

In [16]:

```
clist = ['math', 'english', 'science']
x = exam[clist] # 변수 list로 추출
type(x)
```

Out[16]:

```
pandas.core.frame.DataFrame
```

## [] 데이터 프레임의 컬럼 순서를 바꾸기

In [17]:

```
## 데이터 프레임의 컬럼 순서를 바꾸기
clist = ['id', 'nclass', 'science', 'english', 'math'] #컬럼 변수의 순서
exam = exam[clist]
exam
```

Out[17]:

	<b>id</b>	<b>nclass</b>	<b>science</b>	<b>english</b>	<b>math</b>
<b>0</b>	1	1	50	98	50
<b>1</b>	2	1	60	97	60
<b>2</b>	3	1	78	86	45
<b>3</b>	4	1	58	98	30
<b>4</b>	5	2	65	80	25
<b>5</b>	6	2	98	89	50
<b>6</b>	7	2	45	90	80
<b>7</b>	8	2	25	78	90
<b>8</b>	9	3	15	98	20
<b>9</b>	10	3	45	98	50
<b>10</b>	11	3	65	65	65
<b>11</b>	12	3	32	85	45
<b>12</b>	13	4	65	98	46
<b>13</b>	14	4	12	87	48
<b>14</b>	15	4	78	56	75
<b>15</b>	16	4	65	98	58
<b>16</b>	17	5	98	68	65
<b>17</b>	18	5	90	78	80
<b>18</b>	19	5	87	68	89
<b>19</b>	20	5	58	83	78

## ▣ 컬럼 변수 제거하고 추출하기

> 원 데이터 프레임에는 영향을 주지 않음

In [32]:

```
exam.drop(columns = 'math') #자신 구조에는 영향을 주지 않고, 자료만 반환
```

Out[32]:

	<b>id</b>	<b>nclass</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	98	50
<b>1</b>	2	1	97	60
<b>2</b>	3	1	86	78
<b>3</b>	4	1	98	58
<b>4</b>	5	2	80	65
<b>5</b>	6	2	89	98
<b>6</b>	7	2	90	45
<b>7</b>	8	2	78	25
<b>8</b>	9	3	98	15
<b>9</b>	10	3	98	45
<b>10</b>	11	3	65	65
<b>11</b>	12	3	85	32
<b>12</b>	13	4	98	65
<b>13</b>	14	4	87	12
<b>14</b>	15	4	56	78
<b>15</b>	16	4	98	65
<b>16</b>	17	5	68	98
<b>17</b>	18	5	78	90
<b>18</b>	19	5	68	87
<b>19</b>	20	5	83	58

In [30]:

```
df.drop(columns = ['english', 'science']) # math, english 제거
```

Out[30]:

	<b>id</b>	<b>nclass</b>
<b>0</b>	1	1
<b>1</b>	2	1
<b>2</b>	3	1
<b>3</b>	4	1
<b>4</b>	5	2
<b>5</b>	6	2
<b>6</b>	7	2
<b>7</b>	8	2
<b>8</b>	9	3
<b>9</b>	10	3
<b>10</b>	11	3
<b>11</b>	12	3
<b>12</b>	13	4
<b>13</b>	14	4
<b>14</b>	15	4
<b>15</b>	16	4
<b>16</b>	17	5
<b>17</b>	18	5
<b>18</b>	19	5
<b>19</b>	20	5

## [] pandas 함수 조합하기

### <> query() 와 [] 조합하기

In [33]: # nclass가 1인 행만 추출한 다음 english 추출  
exam.query('nclass == 1')['english']

Out[33]: 0 ... 98  
1 ... 97  
2 ... 86  
3 ... 98  
Name: english, dtype: int64

In [40]: exam.query('nclass == 1')['english'].mean()

Out[40]: 94.75

In [35]: # math가 50 이상인 행만 추출한 다음 id, math 추출  
exam.query('math >= 50')[['id', 'math']]

Out[35]:

	<b>id</b>	<b>math</b>
<b>0</b>	1	50
<b>1</b>	2	60
<b>5</b>	6	50
<b>6</b>	7	80
<b>7</b>	8	90
<b>9</b>	10	50
<b>10</b>	11	65
<b>14</b>	15	75
<b>15</b>	16	58
<b>16</b>	17	65
<b>17</b>	18	80
<b>18</b>	19	89
<b>19</b>	20	78

In [36]: # math가 50 이상인 행만 추출한 다음 id, math 앞부분 5행까지 추출  
exam.query('math >= 50')[['id', 'math']].head()

Out[36]:

	<b>id</b>	<b>math</b>
<b>0</b>	1	50
<b>1</b>	2	60
<b>5</b>	6	50
<b>6</b>	7	80
<b>7</b>	8	90

## <> 가독성 있게 코드 줄 바꾸기

In [39]: # math가 50 이상인 행만 추출  
# id, math 추출, # 앞부분 10행 추출  
exam.query('math >= 50')  
[[['id', 'math']]  
.head()

Out[39]:

	<b>id</b>	<b>math</b>
<b>0</b>	1	50
<b>1</b>	2	60
<b>5</b>	6	50
<b>6</b>	7	80
<b>7</b>	8	90

In [ ]:

## [06-4] 순서대로 정렬하기

### [] 오름차/내림차 순으로 정렬하여 추출하기

> 추출을 하므로 대상 데이터 프레임에 영향을 주지 않는다.

```
In [42]: exam.sort_values('math') # math 오름차순 정렬
```

```
Out[42]:   id  nclass  math  english  science
```

8	9	3	20	98	15
4	5	2	25	80	65
3	4	1	30	98	58
2	3	1	45	86	78
11	12	3	45	85	32
12	13	4	46	98	65
13	14	4	48	87	12
0	1	1	50	98	50
9	10	3	50	98	45
5	6	2	50	89	98
15	16	4	58	98	65
1	2	1	60	97	60
10	11	3	65	65	65
16	17	5	65	68	98
14	15	4	75	56	78
19	20	5	78	83	58
6	7	2	80	90	45
17	18	5	80	78	90
18	19	5	89	68	87
7	8	2	90	78	25

```
In [43]: exam.sort_values('math', ascending = False) # math 내림차순 정렬
```

Out[43]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
	<b>7</b>	8	2	90	78
	<b>18</b>	19	5	89	68
	<b>17</b>	18	5	80	78
	<b>6</b>	7	2	80	90
	<b>19</b>	20	5	78	83
	<b>14</b>	15	4	75	56
	<b>16</b>	17	5	65	68
	<b>10</b>	11	3	65	65
	<b>1</b>	2	1	60	97
	<b>15</b>	16	4	58	98
	<b>9</b>	10	3	50	98
	<b>5</b>	6	2	50	89
	<b>0</b>	1	1	50	98
	<b>13</b>	14	4	48	87
	<b>12</b>	13	4	46	98
	<b>11</b>	12	3	45	85
	<b>2</b>	3	1	45	86
	<b>3</b>	4	1	30	98
	<b>4</b>	5	2	25	80
	<b>8</b>	9	3	20	98
					15

In [44]: # nclass, math 오름차순 정렬  
exam.sort\_values(['nclass', 'math'])

Out[44]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
	<b>3</b>	4	1	30	98
	<b>2</b>	3	1	45	86
	<b>0</b>	1	1	50	98
	<b>1</b>	2	1	60	97
	<b>4</b>	5	2	25	80
	<b>5</b>	6	2	50	89
	<b>6</b>	7	2	80	90
	<b>7</b>	8	2	90	78
	<b>8</b>	9	3	20	98
	<b>11</b>	12	3	45	85
	<b>9</b>	10	3	50	98
	<b>10</b>	11	3	65	65
	<b>12</b>	13	4	46	98
	<b>13</b>	14	4	48	87
	<b>15</b>	16	4	58	98
	<b>14</b>	15	4	75	56
	<b>16</b>	17	5	65	68
	<b>19</b>	20	5	78	83
	<b>17</b>	18	5	80	78
	<b>18</b>	19	5	89	68
					87

In [45]: # nclass 오름차순, math 내림차순 정렬  
exam.sort\_values(['nclass', 'math'], ascending = [True, False])

Out[45]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>1</b>	2	1	60	97	60
<b>0</b>	1	1	50	98	50
<b>2</b>	3	1	45	86	78
<b>3</b>	4	1	30	98	58
<b>7</b>	8	2	90	78	25
<b>6</b>	7	2	80	90	45
<b>5</b>	6	2	50	89	98
<b>4</b>	5	2	25	80	65
<b>10</b>	11	3	65	65	65
<b>9</b>	10	3	50	98	45
<b>11</b>	12	3	45	85	32
<b>8</b>	9	3	20	98	15
<b>14</b>	15	4	75	56	78
<b>15</b>	16	4	58	98	65
<b>13</b>	14	4	48	87	12
<b>12</b>	13	4	46	98	65
<b>18</b>	19	5	89	68	87
<b>17</b>	18	5	80	78	90
<b>19</b>	20	5	78	83	58
<b>16</b>	17	5	65	68	98

## [06-5] 파생변수 추가하기

### [] 파생변수 추가하기

#### .assign()

- > 파생 변수(**derived variable**)는 기존 속성들로부터 새롭게 만들어낸 변수
- > .assign()은 메서드이기 때문에 assign()된 데이터 프레임만 반환

In [11]:

```
# total 변수 추가
exam.assign(total = exam['math'] + exam['english'] + exam['science'])
```

Out[11]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>total</b>
<b>0</b>	1	1	50	98	50	198
<b>1</b>	2	1	60	97	60	217
<b>2</b>	3	1	45	86	78	209
<b>3</b>	4	1	30	98	58	186
<b>4</b>	5	2	25	80	65	170
<b>5</b>	6	2	50	89	98	237
<b>6</b>	7	2	80	90	45	215
<b>7</b>	8	2	90	78	25	193
<b>8</b>	9	3	20	98	15	133
<b>9</b>	10	3	50	98	45	193
<b>10</b>	11	3	65	65	65	195
<b>11</b>	12	3	45	85	32	162
<b>12</b>	13	4	46	98	65	209
<b>13</b>	14	4	48	87	12	147
<b>14</b>	15	4	75	56	78	209
<b>15</b>	16	4	58	98	65	221
<b>16</b>	17	5	65	68	98	231
<b>17</b>	18	5	80	78	90	248
<b>18</b>	19	5	89	68	87	244
<b>19</b>	20	5	78	83	58	219

## [] 여러 파생변수 한 번에 추가하기

In [14]:

```
df = exam.assign(total = exam['math'] + exam['english'] + exam['science'],           # 
                 mean = (exam['math'] + exam['english'] + exam['science']) / 3) # mean
df.head()
```

Out[14]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>total</b>	<b>mean</b>
<b>0</b>	1	1	50	98	50	198	66.000000
<b>1</b>	2	1	60	97	60	217	72.333333
<b>2</b>	3	1	45	86	78	209	69.666667
<b>3</b>	4	1	30	98	58	186	62.000000
<b>4</b>	5	2	25	80	65	170	56.666667

## [] df.assign()에 np.where() 적용하기

> import numpy as np 후 사용

> np.where(조건식, True 시 실행문, False 시 실행문)

```
In [49]: import numpy as np  
exam.assign(test = np.where(exam['science'] >= 60, 'pass', 'fail'))
```

```
Out[49]:    id  nclass  math  english  science  test
```

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>test</b>
<b>0</b>	1	1	50	98	50	fall
<b>1</b>	2	1	60	97	60	pass
<b>2</b>	3	1	45	86	78	pass
<b>3</b>	4	1	30	98	58	fall
<b>4</b>	5	2	25	80	65	pass
<b>5</b>	6	2	50	89	98	pass
<b>6</b>	7	2	80	90	45	fall
<b>7</b>	8	2	90	78	25	fall
<b>8</b>	9	3	20	98	15	fall
<b>9</b>	10	3	50	98	45	fall
<b>10</b>	11	3	65	65	65	pass
<b>11</b>	12	3	45	85	32	fall
<b>12</b>	13	4	46	98	65	pass
<b>13</b>	14	4	48	87	12	fall
<b>14</b>	15	4	75	56	78	pass
<b>15</b>	16	4	58	98	65	pass
<b>16</b>	17	5	65	68	98	pass
<b>17</b>	18	5	80	78	90	pass
<b>18</b>	19	5	89	68	87	pass
<b>19</b>	20	5	78	83	58	fall

[실습 6-1] exam 데이터 프레임에 'total'과 'mean', 'result' 파생변수가 추가된 new\_exam 데이터 프레임 생성

- > 'total'은 과목의 합계
- > 'mean'은 전체 과목의 평균
- > 'result'는 'mean'  $\geq 60$  이면 'pass', 아니면 'fail'

```
In [4]: ### [실습 6-1] exam 데이터 프레임에 'total'과 'mean', 'result' 파생변수가 추가
```

```
new_exam
```

Out[4]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>total</b>	<b>mean</b>	<b>result</b>
<b>0</b>	1	1	50	98	50	198	66.000000	pass
<b>1</b>	2	1	60	97	60	217	72.333333	pass
<b>2</b>	3	1	45	86	78	209	69.666667	pass
<b>3</b>	4	1	30	98	58	186	62.000000	pass
<b>4</b>	5	2	25	80	65	170	56.666667	fall
<b>5</b>	6	2	50	89	98	237	79.000000	pass
<b>6</b>	7	2	80	90	45	215	71.666667	pass
<b>7</b>	8	2	90	78	25	193	64.333333	pass
<b>8</b>	9	3	20	98	15	133	44.333333	fall
<b>9</b>	10	3	50	98	45	193	64.333333	pass
<b>10</b>	11	3	65	65	65	195	65.000000	pass
<b>11</b>	12	3	45	85	32	162	54.000000	fall
<b>12</b>	13	4	46	98	65	209	69.666667	pass
<b>13</b>	14	4	48	87	12	147	49.000000	fall
<b>14</b>	15	4	75	56	78	209	69.666667	pass
<b>15</b>	16	4	58	98	65	221	73.666667	pass
<b>16</b>	17	5	65	68	98	231	77.000000	pass
<b>17</b>	18	5	80	78	90	248	82.666667	pass
<b>18</b>	19	5	89	68	87	244	81.333333	pass
<b>19</b>	20	5	78	83	58	219	73.000000	pass

---

## [] lamda 함수를 활용한 이름 줄여쓰기

> lamda 함수?

>> Python 내장함수로서, 함수객체를 반환하는 함수

>> return문이 필요없는 함수

>> 간단한 함수를 선언하면서, 바로 실행되도록 하는 용도로도 사용

In [41]:

```
# 긴 데이터 프레임명 지정
long_name = pd.read_csv('exam.csv')

# long_name 직접 입력
long_name.assign(new = long_name['math'] + long_name['english'] + long_name['science'])
```

Out[41]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>new</b>
<b>0</b>	1	1	50	98	50	198
<b>1</b>	2	1	60	97	60	217
<b>2</b>	3	1	45	86	78	209
<b>3</b>	4	1	30	98	58	186
<b>4</b>	5	2	25	80	65	170
<b>5</b>	6	2	50	89	98	237
<b>6</b>	7	2	80	90	45	215
<b>7</b>	8	2	90	78	25	193
<b>8</b>	9	3	20	98	15	133
<b>9</b>	10	3	50	98	45	193
<b>10</b>	11	3	65	65	65	195
<b>11</b>	12	3	45	85	32	162
<b>12</b>	13	4	46	98	65	209
<b>13</b>	14	4	48	87	12	147
<b>14</b>	15	4	75	56	78	209
<b>15</b>	16	4	58	98	65	221
<b>16</b>	17	5	65	68	98	231
<b>17</b>	18	5	80	78	90	248
<b>18</b>	19	5	89	68	87	244
<b>19</b>	20	5	78	83	58	219

In [55]:

```
## <> lamda 함수를 활용한 이름 줄여쓰기
long_name.assign(new = lambda x: x['math'] + x['english'] + x['science']) # long_n
```

Out[55]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>new</b>
<b>0</b>	1	1	50	98	50	198
<b>1</b>	2	1	60	97	60	217
<b>2</b>	3	1	45	86	78	209
<b>3</b>	4	1	30	98	58	186
<b>4</b>	5	2	25	80	65	170
<b>5</b>	6	2	50	89	98	237
<b>6</b>	7	2	80	90	45	215
<b>7</b>	8	2	90	78	25	193
<b>8</b>	9	3	20	98	15	133
<b>9</b>	10	3	50	98	45	193
<b>10</b>	11	3	65	65	65	195
<b>11</b>	12	3	45	85	32	162
<b>12</b>	13	4	46	98	65	209
<b>13</b>	14	4	48	87	12	147
<b>14</b>	15	4	75	56	78	209
<b>15</b>	16	4	58	98	65	221
<b>16</b>	17	5	65	68	98	231
<b>17</b>	18	5	80	78	90	248
<b>18</b>	19	5	89	68	87	244
<b>19</b>	20	5	78	83	58	219

## <> lamda 함수?

>> 함수 객체를 반환하는 함수

>> return문이 필요없는 함수

>> 간단한 함수를 선언하면서, 바로 실행되도록 하는 용도로도 사용

In [64]: # 매개변수 x, y를 사용하여 x+y를 반환하는 lamda 함수 선언  
lambda x, y:x+y #lamda 함수는 실행 후 함수 객체를 반환

# 선언된 lamda 함수 바로 사용  
(lambda x, y:x+y)(3, 4) #반환된 함수 객체를 사용하여 (3, 4)를 실행

Out[64]: 7

In [63]: #long\_name 객체에서 lamda 함수가 생성되었으므로 매개변수 x는 객체명 long\_name에 해당  
long\_name.assign(new = lambda x: x['math'] + x['english'] + x['science']) # long\_

Out[63]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>new</b>
<b>0</b>	1	1	50	98	50	198
<b>1</b>	2	1	60	97	60	217
<b>2</b>	3	1	45	86	78	209
<b>3</b>	4	1	30	98	58	186
<b>4</b>	5	2	25	80	65	170
<b>5</b>	6	2	50	89	98	237
<b>6</b>	7	2	80	90	45	215
<b>7</b>	8	2	90	78	25	193
<b>8</b>	9	3	20	98	15	133
<b>9</b>	10	3	50	98	45	193
<b>10</b>	11	3	65	65	65	195
<b>11</b>	12	3	45	85	32	162
<b>12</b>	13	4	46	98	65	209
<b>13</b>	14	4	48	87	12	147
<b>14</b>	15	4	75	56	78	209
<b>15</b>	16	4	58	98	65	221
<b>16</b>	17	5	65	68	98	231
<b>17</b>	18	5	80	78	90	248
<b>18</b>	19	5	89	68	87	244
<b>19</b>	20	5	78	83	58	219

In [17]:

```
## lambda 함수의 병행 사용
exam.assign(total = lambda x: x['math'] + x['english'] + x['science'],
            mean = lambda x: x['total'] / 3)
```

Out[17]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>total</b>	<b>mean</b>
<b>0</b>	1	1	50	98	50	198	66.000000
<b>1</b>	2	1	60	97	60	217	72.333333
<b>2</b>	3	1	45	86	78	209	69.666667
<b>3</b>	4	1	30	98	58	186	62.000000
<b>4</b>	5	2	25	80	65	170	56.666667
<b>5</b>	6	2	50	89	98	237	79.000000
<b>6</b>	7	2	80	90	45	215	71.666667
<b>7</b>	8	2	90	78	25	193	64.333333
<b>8</b>	9	3	20	98	15	133	44.333333
<b>9</b>	10	3	50	98	45	193	64.333333
<b>10</b>	11	3	65	65	65	195	65.000000
<b>11</b>	12	3	45	85	32	162	54.000000
<b>12</b>	13	4	46	98	65	209	69.666667
<b>13</b>	14	4	48	87	12	147	49.000000
<b>14</b>	15	4	75	56	78	209	69.666667
<b>15</b>	16	4	58	98	65	221	73.666667
<b>16</b>	17	5	65	68	98	231	77.000000
<b>17</b>	18	5	80	78	90	248	82.666667
<b>18</b>	19	5	89	68	87	244	81.333333
<b>19</b>	20	5	78	83	58	219	73.000000

In [21]:

```
exam.assign(total = exam['math'] + exam['english'] + exam['science'],
            mean = lambda x: x['total'] / 3)
# total 변수를 생성하고 lambda 함수를 호출하는 방식으로 진행
```

Out[21]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>total</b>	<b>mean</b>
<b>0</b>	1	1	50	98	50	198	66.000000
<b>1</b>	2	1	60	97	60	217	72.333333
<b>2</b>	3	1	45	86	78	209	69.666667
<b>3</b>	4	1	30	98	58	186	62.000000
<b>4</b>	5	2	25	80	65	170	56.666667
<b>5</b>	6	2	50	89	98	237	79.000000
<b>6</b>	7	2	80	90	45	215	71.666667
<b>7</b>	8	2	90	78	25	193	64.333333
<b>8</b>	9	3	20	98	15	133	44.333333
<b>9</b>	10	3	50	98	45	193	64.333333
<b>10</b>	11	3	65	65	65	195	65.000000
<b>11</b>	12	3	45	85	32	162	54.000000
<b>12</b>	13	4	46	98	65	209	69.666667
<b>13</b>	14	4	48	87	12	147	49.000000
<b>14</b>	15	4	75	56	78	209	69.666667
<b>15</b>	16	4	58	98	65	221	73.666667
<b>16</b>	17	5	65	68	98	231	77.000000
<b>17</b>	18	5	80	78	90	248	82.666667
<b>18</b>	19	5	89	68	87	244	81.333333
<b>19</b>	20	5	78	83	58	219	73.000000

In [22]:

```
exam.assign(total = exam['math'] + exam['english'] + exam['science'],
            mean   = exam['total'] / 3)
```

```

-----
--> 3789         try:
-> 3790             return self._engine.get_loc(casted_key)
3791     except KeyError as err:
3792
3793     index.pyx in pandas._libs.index.IndexEngine.get_loc()
3794
3795     index.pyx in pandas._libs.index.IndexEngine.get_loc()
3796
3797     pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
3798
3799     pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()
3800
3801     KeyError: 'total'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_22856\2557496153.py in <module>
      1 exam.assign(total = exam['math'] + exam['english'] + exam['science'],
----> 2                 mean = exam['total'] / 3)
      3
~\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
      3894         if self.columns.nlevels > 1:
      3895             return self._getitem_multilevel(key)
-> 3896         indexer = self.columns.get_loc(key)
      3897         if is_integer(indexer):
      3898             indexer = [indexer]
      3899
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key)
      3795         ):
      3796             raise InvalidIndexError(key)
-> 3797         raise KeyError(key) from err
      3798     except TypeError:
      3799         # If we have a listlike key, _check_indexing_error will
raise

KeyError: 'total'

```

---

## [06-6] 집단별로 요약하기

### [] 집단 요약하기

- > 단일 통계값을 구하기 **aggregate** 총 종합
- > **agg()** : 대상 변수에 대해 단일 통계값을 구해줌
- >> **agg(변수명 = (대상\_변수명, 적용할\_함수명))**

>> 적용 가능 내장함수 : 'sum', 'mean', 'median', 'min', 'max', 'count', 'std', 'var'

```
In [75]: # math 평균 구하기  
exam.agg(mean_math = ('math', 'mean')) #결과는 DataFrame 형  
## 받아내는 변수는 결과 데이터 프레임의 index가 된다.
```

```
Out[75]:  
math  
mean_math 57.45
```

```
In [76]: # nclass 개수 구하기  
exam.agg(cnt_nclass = ('nclass', 'count')) #결과는 DataFrame 형
```

```
Out[76]:  
nclass  
cnt_nclass 20
```

## [] 집단별로 요약하기

> 집단별로 통계값을 구하기

> groupby() : 집단별로 통계값을 구해줌

>> groupby(집단구분\_변수명).agg(변수명 = (대상\_변수명, 적용할\_함수명))

```
In [47]: ## exam.groupby('nclass')는 집단별 분리이므로 .count()가 가능  
exam.groupby('nclass').count()
```

```
Out[47]:  
id math english science  
nclass  
1 4 4 4 4  
2 4 4 4 4  
3 4 4 4 4  
4 4 4 4 4  
5 4 4 4 4
```

```
In [42]: ## 'nclass' 별로 'math'의 mean 값 구하기  
exam.groupby('nclass').agg(mean_math = ('math', 'mean'))  
## groupby()의 집단구분_변수명은 결과 데이터 프레임의 index가 된다.
```

```
Out[42]:  
mean_math  
nclass  
1 46.25  
2 61.25  
3 45.00  
4 56.75  
5 78.00
```

## <> 변수를 인덱스로 바꾸지 않기

```
In [82]: exam.groupby('nclass', as_index = False) \
    .agg(mean_math = ('math', 'mean'))
```

Out[82]:

	nclass	mean_math
0	1	1.0
1	2	2.0
2	3	3.0
3	4	4.0
4	5	5.0

## [실습 6-2] 'nclass'별 인원 수 구하기

**new\_exam** 데이터 프레임을 사용하여

> 결과 데이터 프레임에 'nclass'와 'count'가 나타나도록 한다.

```
In [10]: ## 'nclass'별 인원 수 구하기
```

Out[10]:

	nclass	count
0	1	4
1	2	4
2	3	4
3	4	4
4	5	4

In [ ]:

## [] 여러 요약 통계량 한 번에 구하기

```
In [84]: # 'math' 과목에 대해
# 'nclass' 별로
# 평균, 합계, 중앙값, 빈도(학생 수) 구하기
exam.groupby('nclass') \
    .agg(mean_math = ('math', 'mean'),
        sum_math = ('math', 'sum'),
        median_math = ('math', 'median'),
        n = ('nclass', 'count'))
```

```
Out[84]:      mean_math  sum_math  median_math  n
```

nclass					
<b>1</b>	46.25	185	47.5	4	
<b>2</b>	61.25	245	65.0	4	
<b>3</b>	45.00	180	47.5	4	
<b>4</b>	56.75	227	53.0	4	
<b>5</b>	78.00	312	79.0	4	

## ▣ 모든 변수의 요약 통계량 한 번에 구하기

```
In [48]: ## exam.groupby('nclass')는 집단별 분리이므로 .count()가 가능  
exam.groupby('nclass').count()
```

```
Out[48]:      id  math  english  science
```

nclass				
<b>1</b>	4	4	4	4
<b>2</b>	4	4	4	4
<b>3</b>	4	4	4	4
<b>4</b>	4	4	4	4
<b>5</b>	4	4	4	4

```
In [87]: ## 'nclass'별 인원 수 구하기  
exam.groupby('nclass').mean()
```

```
Out[87]:      id  math  english  science
```

nclass				
<b>1</b>	2.5	46.25	94.75	61.50
<b>2</b>	6.5	61.25	84.25	58.25
<b>3</b>	10.5	45.00	86.50	39.25
<b>4</b>	14.5	56.75	84.75	55.00
<b>5</b>	18.5	78.00	74.25	83.25

## ▣ 집단별로 다시 집단 나누기

```
In [25]: ## mpg 데이터 불러오기  
import pandas as pd  
mpg = pd.read_csv('mpg.csv')  
mpg.head()
```

Out[25]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact

In [95]: ## 제조회사(manufacturer)별 구동방식(drv)별 cty 평균 구하기  
mpg.groupby(['manufacturer', 'drv'])  
.agg(mean\_cty = ('cty', 'mean'))

Out[95]:

mean_cty		
manufacturer	drv	mean_cty
audi	4	16.818182
	f	18.857143
chevrolet	4	12.500000
	f	18.800000
dodge	4	14.100000
	f	12.000000
ford	4	15.818182
	r	13.307692
honda	f	14.750000
	r	24.444444
hyundai	f	11.500000
	r	18.642857
jeep	4	13.500000
	r	13.500000
land rover	4	11.333333
	r	11.500000
lincoln	4	20.000000
	r	13.250000
mercury	4	17.000000
	f	13.750000
nissan	4	20.000000
	f	19.285714
pontiac	4	14.933333
	f	14.933333
subaru	4	21.368421
	r	14.933333
toyota	4	20.925926
	f	19.285714
volkswagen	4	11.333333
	f	11.333333

## [] 원하는 집단만 통계 전에 필터링 하기

In [103...]: ## 제조회사(manufacturer)가 'audi'인 것에 대해서만 구동방식(drv)별 cty 평균 구하기  
mpg.query('manufacturer == "audi"')  
.agg(mean\_cty = ('cty', 'mean'))

```
.groupby(['manufacturer', 'drv'])  
.agg(mean_cty = ('cty', 'mean'))
```

Out[103]: mean\_cty

manufacturer	drv	mean_cty
audi	4	16.818182
	f	18.857143

## [] 원하는 집단만 통계 후에 필터링 하기

In [104...]: ## 제조회사(manufacturer)별 구동방식(drv)별 cty 평균이 20 이상인 경우만 구하기  
mpg.groupby(['manufacturer', 'drv'])  
.agg(mean\_cty = ('cty', 'mean'))  
.query('mean\_cty >= 20') # 원하는 결과만 필터링 하기 #query('필터링\_조건식')

Out[104]: mean\_cty

manufacturer	drv	mean_cty
honda	f	24.444444
nissan	f	20.000000
toyota	f	21.368421
volkswagen	f	20.925926

In [ ]:

[실습] 'drv'별로 빈도 수를 구한 다음에 > 빈도 수가 100 초과인 것만 구하기

>> mpg.groupby(['drv'])로 해결하는 방법

In [115...]: ## 'drv'별로 빈도 수를 구한 다음에  
## > 빈도 수가 100 초과인 것만 구하기  
mpg.groupby(['drv'])  
.agg(n = ('drv', 'count'))  
.query('n > 100') # 원하는 결과만 필터링 하기 #query('필터링\_조건식')

Out[115]: n

drv	n
4	103
f	106

>> mpg['drv']로 해결하는 방법

In [106...]: mpg['drv'].value\_counts()

Out[106]: f 106  
4 103  
r 25  
Name: drv, dtype: int64

<문제 발생> mpg['drv'].value\_counts().query('n > 100')

> .query()는 데이터 프레임 메서드이기 때문

```
In [ ]: mpg['drv'].value_counts().query('n > 100')
```

<문제 해결> .query('n > 100')에서 'n'을 프레임에 포함시킴

```
In [116]: mpg['drv'].value_counts() #  
          .to_frame('n')
```

```
Out[116]:  
      n  
-----  
f    106  
4    103  
r     25
```

<> 'n'을 사용하여 필터링 수행

```
In [109]: mpg['drv'].value_counts() #  
          .to_frame('n') #  
          .query('n > 100')
```

```
Out[109]:  
      n  
-----  
f    106  
4    103
```

### [실습 6-3] 원하는 통계 결과만 필터링 하여 구하기

**new\_exam** 데이터 프레임을 사용하여

> 'nclass'별 'mean'의 평균을구하여, 평균 값이 70 이상인 행만 구하기

```
In [15]: ##### > 'nclass'별 'mean'의 평균을구하여, 평균 값이 70 이상인 행만 구하기
```

```
Out[15]:  
      nclass  avg  
-----  
4        5   78.5
```

---

## [06-7] 데이터 합치기

### [] 가로로 합치기(열 합치기) : merge()

: 프레임을 가로(열) 방향으로 합쳐지면서, 프레임에 열 변수를 합하는 기능

> **left**: 병합할 첫 번째 데이터프레임

> **right**: 병합할 두 번째 데이터프레임

> **how**: 병합 방식, 다음 중 하나를 선택

>> 'inner' (기본값): 교집합 병합, 'outer': 합집합

>> 'left': 왼쪽 프레임 기준, 'right': 오른쪽 프레임 기준

> **on**: 병합 기준 열(컬럼)/리스트

> **left\_on**과 **right\_on**: 왼쪽과 오른쪽 프레임의 병합 기준 열을 별도로 지정

> **left\_index**와 **right\_index**: 왼쪽과 오른쪽 프레임의 인덱스를 병합 기준으로 사용할지 여부

> **suffixes**: 중복되는 열 이름이 있는 경우, 추가할 접미사 지정

> **sort**: 병합 후 결과를 정렬

> **indicator**: 결과 데이터프레임에 **\_merge** 열을 추가하여 **source** 프레임 표시

```
In [27]: # 중간고사 데이터 만들기
test1 = pd.DataFrame({'id'      : [1, 2, 3, 4, 5],
                      'midterm' : [60, 80, 70, 90, 85]})

# 기말고사 데이터 만들기
test2 = pd.DataFrame({'id'      : [1, 2, 3, 4, 5],
                      'final'   : [70, 83, 65, 95, 80]})
```

```
In [9]: print(test1)
print(test2)

... id midterm
0 1 ..... 60
1 2 ..... 80
2 3 ..... 70
3 4 ..... 90
4 5 ..... 85
... id final
0 1 .... 70
1 2 .... 83
2 3 .... 65
3 4 .... 95
4 5 .... 80
```

```
In [5]: ## 두 프레임을 id 기준으로 합쳐서 total에 할당
total = pd.merge(test1, test2, how = 'left', on = 'id') # 'id' 기준, 왼쪽으로 합병
total
```

```
Out[5]:   id midterm final
          0     1       60    70
          1     2       80   83
          2     3       70   65
          3     4       90   95
          4     5       85   80
```

## <> nclass 담당 교사 성명(name) 데이터 프레임을 만들어 합병하기

```
In [28]: ## name 데이터 프레임을 만들기  
name = pd.DataFrame({'nclass' : [1, 2, 3, 4, 5],  
                     'teacher' : ['kim', 'lee', 'park', 'choi', 'jung']})  
name
```

Out[28]:

	nclass	teacher
0	1	kim
1	2	lee
2	3	park
3	4	choi
4	5	jung

```
In [29]: ## exam 프레임에 name 프레임 합병하기  
exam_new = pd.merge(exam, name, how = 'left', on = 'nclass')  
exam_new
```

Out[29]:

	id	nclass	math	english	science	teacher
0	1	1	50	98	50	kim
1	2	1	60	97	60	kim
2	3	1	45	86	78	kim
3	4	1	30	98	58	kim
4	5	2	25	80	65	lee
5	6	2	50	89	98	lee
6	7	2	80	90	45	lee
7	8	2	90	78	25	lee
8	9	3	20	98	15	park
9	10	3	50	98	45	park
10	11	3	65	65	65	park
11	12	3	45	85	32	park
12	13	4	46	98	65	choi
13	14	4	48	87	12	choi
14	15	4	75	56	78	choi
15	16	4	58	98	65	choi
16	17	5	65	68	98	jung
17	18	5	80	78	90	jung
18	19	5	89	68	87	jung
19	20	5	78	83	58	jung

## [실습] Outer Join

name 프레임을 nclass = 1을 빼고 생성한 후에,

exam과 name 프레임을 Outer Join 수행

```
In [30]: ## name 데이터 프레임을 만들기 : nclass = 1을 빼고 생성
name = pd.DataFrame({'nclass' : [2, 3, 4, 5],
                     'teacher' : ['lee', 'park', 'choi', 'jung']})
name
```

```
Out[30]:   nclass  teacher
0         2      lee
1         3     park
2         4     choi
3         5    jung
```

```
In [31]: ## exam, name 프레임을 'nclass' 기준으로 Outer Join
exam_all = pd.merge(exam, name, on='nclass', how='outer')
exam_all
```

Out[31]:

	<b>id</b>	<b>nclass</b>	<b>math</b>	<b>english</b>	<b>science</b>	<b>teacher</b>
<b>0</b>	1	1	50	98	50	NaN
<b>1</b>	2	1	60	97	60	NaN
<b>2</b>	3	1	45	86	78	NaN
<b>3</b>	4	1	30	98	58	NaN
<b>4</b>	5	2	25	80	65	lee
<b>5</b>	6	2	50	89	98	lee
<b>6</b>	7	2	80	90	45	lee
<b>7</b>	8	2	90	78	25	lee
<b>8</b>	9	3	20	98	15	park
<b>9</b>	10	3	50	98	45	park
<b>10</b>	11	3	65	65	65	park
<b>11</b>	12	3	45	85	32	park
<b>12</b>	13	4	46	98	65	choi
<b>13</b>	14	4	48	87	12	choi
<b>14</b>	15	4	75	56	78	choi
<b>15</b>	16	4	58	98	65	choi
<b>16</b>	17	5	65	68	98	jung
<b>17</b>	18	5	80	78	90	jung
<b>18</b>	19	5	89	68	87	jung
<b>19</b>	20	5	78	83	58	jung

---

## [] 세로로 합치기(행/열 합치기) : concat()

: 프레임을 세로(행) 방향으로 합치지면서, 프레임에 행 변수를 합하는 기능

> **objs**: 연결할 데이터프레임(리스트, 튜플)

> **axis**: 연결 방향

>> 0: 행(기본값), 1: 열

> **join**: 인덱스 처리

>> 'outer': 외부 조인(Outer Join), 연결 안되도 모두 포함

>> 'inner': 내부 조인(Inner Join), 공통된 인덱스만을 포함

> **ignore\_index**: 새로운 정수 인덱스 생성 (기본값은 False)

> **keys**: 다중 인덱스 생성, 계층적 인덱스를 지정하는 리스트나 배열

> **sort**: 연결 후 결과를 정렬 (기본값은 False)

## > copy: 복사 여부를 지정 (기본값은 True)

```
In [11]: ## 학생 1~5번 시험 데이터 만들기
group_a = pd.DataFrame({'id' : [1, 2, 3, 4, 5],
                       'test' : [60, 80, 70, 90, 85]})

## 학생 6~10번 시험 데이터 만들기
group_b = pd.DataFrame({'id' : [6, 7, 8, 9, 10],
                       'test' : [70, 83, 65, 95, 80]})
```

```
In [13]: print(group_a)
print(group_b)

... id test
0 1 60
1 2 80
2 3 70
3 4 90
4 5 85
... id test
0 6 70
1 7 83
2 8 65
3 9 95
4 10 80
```

```
In [14]: ## group_a와 group_b를 group_all로 행 합치기
group_all = pd.concat([group_a, group_b])
group_all
```

```
Out[14]:   id  test
          0    1    60
          1    2    80
          2    3    70
          3    4    90
          4    5    85
          0    6    70
          1    7    83
          2    8    65
          3    9    95
          4   10    80
```

```
In [47]: ## group_a와 group_b를 group_all로 행 합치기
exam_all = pd.concat([group_a, group_b], ignore_index=True) #행으로 합치기, index
exam_all
```

Out[47]:

	<b>id</b>	<b>test</b>
<b>0</b>	1	60
<b>1</b>	2	80
<b>2</b>	3	70
<b>3</b>	4	90
<b>4</b>	5	85
<b>5</b>	6	70
<b>6</b>	7	83
<b>7</b>	8	65
<b>8</b>	9	95
<b>9</b>	10	80

In [46]:  
## group\_a와 group\_b를 group\_all로 열 합치기  
group\_all = pd.concat([group\_a, group\_b], axis=1) #열로 합치기  
group\_all

Out[46]:

	<b>id</b>	<b>test</b>	<b>id</b>	<b>test</b>
<b>0</b>	1	60	6	70
<b>1</b>	2	80	7	83
<b>2</b>	3	70	8	65
<b>3</b>	4	90	9	95
<b>4</b>	5	85	10	80

---

## [실습 6-4] 데이터 행, 열 분리 및 병합

**new\_exam** 데이터 프레임을 사용하여

<1> **exam\_mean** 데이터 프레임 만들기

<1-1> **new\_exam** 데이터 프레임에서 'id', 'nclass', 'total', 'mean' 변수를 추출하여 **exam\_mean** 만들기

<1-2> **new\_exam** 데이터 프레임에서 'id', 'math', 'english', 'science' 변수를 추출하여 **exam\_subj** 만들기

<2> **exam\_mean**과 **exam\_subj**를 **id**를 기준으로 열 합병하여 **exam\_result** 데이터 프레임 생성

<3> **nclass**로 행 분리된 **exam\_1class**, **exam\_2class**, **exam\_3class**, **exam\_4class**, **exam\_5class** 프레임 생성

<4> **exam\_1class**, **exam\_2class**, **exam\_3class**, **exam\_4class**, **exam\_5class** 프레임들의 모든 행들을 병합하여 **exam\_nclass** 생성

In [22]: ## <1-1> new\_exam 데이터 프레임에서 'id', 'nclass', 'total', 'mean' 변수를 추출하여

```
exam_mean.head()
```

```
Out[22]:
```

	<b>id</b>	<b>nclass</b>	<b>total</b>	<b>mean</b>
<b>0</b>	1	1	198	66.000000
<b>1</b>	2	1	217	72.333333
<b>2</b>	3	1	209	69.666667
<b>3</b>	4	1	186	62.000000
<b>4</b>	5	2	170	56.666667

```
In [23]: ## <1-2> exam_new 데이터 프레임에서 'id', 'math', 'english', 'science' 변수를 추출
```

```
exam_subj.head()
```

```
Out[23]:
```

	<b>id</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	50	98	50
<b>1</b>	2	60	97	60
<b>2</b>	3	45	86	78
<b>3</b>	4	30	98	58
<b>4</b>	5	25	80	65

```
In [25]: ## <2> exam_mean과 exam_subj를 id를 기준으로 열 합병하여 exam_result 데이터 프레임
```

```
exam_result.head()
```

```
Out[25]:
```

	<b>id</b>	<b>nclass</b>	<b>total</b>	<b>mean</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	198	66.000000	50	98	50
<b>1</b>	2	1	217	72.333333	60	97	60
<b>2</b>	3	1	209	69.666667	45	86	78
<b>3</b>	4	1	186	62.000000	30	98	58
<b>4</b>	5	2	170	56.666667	25	80	65

```
In [26]: ## <3> exam_result를 nclass를 기준으로 행 분리하여 별도의 데이터 프레임 exam_1class,
```

```
print(exam_1class)
print(exam_2class)
print(exam_3class)
print(exam_4class)
print(exam_5class)
```

	<u>id</u>	<u>nclass</u>	<u>total</u>	<u>mean</u>	<u>math</u>	<u>english</u>	<u>science</u>
0	1	1	198	66.000000	50	98	50
1	2	1	217	72.333333	60	97	60
2	3	1	209	69.666667	45	86	78
3	4	1	186	62.000000	30	98	58
	<u>id</u>	<u>nclass</u>	<u>total</u>	<u>mean</u>	<u>math</u>	<u>english</u>	<u>science</u>
4	5	2	170	56.666667	25	80	65
5	6	2	237	79.000000	50	89	98
6	7	2	215	71.666667	80	90	45
7	8	2	193	64.333333	90	78	25
	<u>id</u>	<u>nclass</u>	<u>total</u>	<u>mean</u>	<u>math</u>	<u>english</u>	<u>science</u>
8	9	3	133	44.333333	20	98	15
9	10	3	193	64.333333	50	98	45
10	11	3	195	65.000000	65	65	65
11	12	3	162	54.000000	45	85	32
	<u>id</u>	<u>nclass</u>	<u>total</u>	<u>mean</u>	<u>math</u>	<u>english</u>	<u>science</u>
12	13	4	209	69.666667	46	98	65
13	14	4	147	49.000000	48	87	12
14	15	4	209	69.666667	75	56	78
15	16	4	221	73.666667	58	98	65
	<u>id</u>	<u>nclass</u>	<u>total</u>	<u>mean</u>	<u>math</u>	<u>english</u>	<u>science</u>
16	17	5	231	77.000000	65	68	98
17	18	5	248	82.666667	80	78	90
18	19	5	244	81.333333	89	68	87
19	20	5	219	73.000000	78	83	58

In [27]: ## <4> exam\_1class, exam\_2class, exam\_3class, exam\_4class, exam\_5class 프레임들의 모

```
exam_nclass
```

Out[27]:

	<b>id</b>	<b>nclass</b>	<b>total</b>	<b>mean</b>	<b>math</b>	<b>english</b>	<b>science</b>
<b>0</b>	1	1	198	66.000000	50	98	50
<b>1</b>	2	1	217	72.333333	60	97	60
<b>2</b>	3	1	209	69.666667	45	86	78
<b>3</b>	4	1	186	62.000000	30	98	58
<b>4</b>	5	2	170	56.666667	25	80	65
<b>5</b>	6	2	237	79.000000	50	89	98
<b>6</b>	7	2	215	71.666667	80	90	45
<b>7</b>	8	2	193	64.333333	90	78	25
<b>8</b>	9	3	133	44.333333	20	98	15
<b>9</b>	10	3	193	64.333333	50	98	45
<b>10</b>	11	3	195	65.000000	65	65	65
<b>11</b>	12	3	162	54.000000	45	85	32
<b>12</b>	13	4	209	69.666667	46	98	65
<b>13</b>	14	4	147	49.000000	48	87	12
<b>14</b>	15	4	209	69.666667	75	56	78
<b>15</b>	16	4	221	73.666667	58	98	65
<b>16</b>	17	5	231	77.000000	65	68	98
<b>17</b>	18	5	248	82.666667	80	78	90
<b>18</b>	19	5	244	81.333333	89	68	87
<b>19</b>	20	5	219	73.000000	78	83	58

---

## 정리하기

---

In [12]:

```
##사전 데이터 준비
import pandas as pd
import numpy as np
exam = pd.read_csv('exam.csv')
mpg = pd.read_csv('mpg.csv')
test1 = pd.DataFrame({'id' : [1, 2, 3, 4, 5], 'midterm' : [60, 80, 70, 90, 85]})
test2 = pd.DataFrame({'id' : [1, 2, 3, 4, 5], 'final' : [70, 83, 65, 95, 80]})
```

```
group_a = pd.DataFrame({'id' : [1, 2, 3, 4, 5], 'test' : [60, 80, 70, 90, 85]})
```

```
group_b = pd.DataFrame({'id' : [6, 7, 8, 9, 10], 'test' : [70, 83, 65, 95, 80]})
```

In [11]:

```
## 1. 조건에 맞는 데이터만 추출하기
exam.query('english <= 80')
```

```
# 여러 조건 동시 총족
exam.query('nclass == 1 & math >= 50')
```

```
# 여러 조건 중 하나 이상 총족
exam.query('math >= 90 | english >= 90')
exam.query('nclass in [1, 3, 5]')
```

```

## 2. 필요한 변수만 추출하기
exam['math']                                     # 한 변수 추출
exam[['nclass', 'math', 'english']]             # 여러 변수 추출
exam.drop(columns = 'math')                      # 변수 제거
exam.drop(columns = ['math', 'english'])          # 여러 변수 제거

## 3. pandas 명령어 조합하기
exam.query('math >= 50')[['id', 'math']].head()

## 4. 순서대로 정렬하기
exam.sort_values('math')                         # 오름차순 정렬
exam.sort_values('math', ascending = False)       # 내림차순 정렬

# 여러 변수 기준 정렬
exam.sort_values(['nclass', 'math'], ascending = [True, False])

## 5. 파생변수 추가하기
exam.assign(total = exam['math'] + exam['english'] + exam['science'])

# 여러 파생변수 한 번에 추가하기
exam.assign(total = exam['math'] + exam['english'] + exam['science'],
            mean = (exam['math'] + exam['english'] + exam['science']) / 3)

# assign()에 np.where() 적용하기
exam.assign(test = np.where(exam['science'] >= 60, 'pass', 'fail'))

# 추가한 변수를 pandas 코드에 바로 활용하기
exam.assign(total = exam['math'] + exam['english'] + exam['science']) \
    .sort_values('total') \
    .head()

## 6. 집단별로 요약하기
exam.groupby('nclass') \
    .agg(mean_math = ('math', 'mean'))

# 각 집단별로 다시 집단 나누기
mpg.groupby(['manufacturer', 'drv']) \
    .agg(mean_cty = ('cty', 'mean'))

## 7. 데이터 합치기
pd.merge(test1, test2, how = 'left', on = 'id') # 가로로 합치기
pd.concat([group_a, group_b])                  # 세로로 합치기

```

Out[11]:

	<b>id</b>	<b>test</b>
<b>0</b>	1	60
<b>1</b>	2	80
<b>2</b>	3	70
<b>3</b>	4	90
<b>4</b>	5	85
<b>0</b>	6	70
<b>1</b>	7	83
<b>2</b>	8	65
<b>3</b>	9	95
<b>4</b>	10	80

In [ ]: