

Ch06 데이터 전처리

```
#.query() #.drop() #.sort_values() #.groupby() #.assign() #.agg() #.merge() #.concat()
```

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

In []:

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)

In []:

06-2 조건에 맞는 데이터만 추출하기

실습용 데이터프레임 구축

```
In [1]: ## [06-2] 조건에 맞는 데이터만 추출하기  
# 실습용 데이터 프레임 생성  
import pandas as pd  
exam = pd.read_csv('exam_csv.csv')  
exam.head()
```

```
Out[1]:   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  
4    5       2     25       80      65
```

.query()

조건에 맞는 데이터만 추출하기

> 데이터프레임 행선택을 사용

```
In [4]: exam['nclass'] == 1 #비교 연산: 열 추출 후 비교
```

```
Out[4]: 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 [5]: exam[exam['nclass'] == 1] #exam['nclass'] == 1 비교 결과 True인 모든 행
```

```
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
```

query('필터링_조건식')을 사용

> 컬럼명에 대한 조건식을 사용

```
In [8]: ## [06-2] 조건에 맞는 데이터만 추출하기
# exam에서 nclass가 1인 경우만 추출
exam.query('nclass == 1') #query('필터링_조건식')
```

Out[8]:

	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 [9]:

```
# 1반이 아닌 경우  
exam.query('nclass != 1')
```

Out[9]:

	id	nclass	math	english	science
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 [10]: # 수학 점수가 50점을 초과한 경우  
exam.query('math > 50')
```

```
Out[10]:   id  nclass  math  english  science  
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 [11]: # 1반이면서 수학 점수가 50 점 이상인 경우  
exam.query('nclass == 1 & math >= 50')
```

```
Out[11]:   id  nclass  math  english  science  
0    1        1     50       98       50  
1    2        1     60       97       60
```

```
In [12]: # 수학 점수가 90점 이상이거나 영어 점수가 90점 이상인 경우  
exam.query('math >= 90 | english >= 90')
```

Out[12]:

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

원하는 행을 리스트로 지정하여 추출하기

In [13]:

```
# 1, 3, 5반에 해당하면 추출  
exam.query('nclass == 1 | nclass == 3 | nclass == 5')
```

Out[13]:

	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
8	9	3	20	98	15
9	10	3	50	98	45
10	11	3	65	65	65
11	12	3	45	85	32
16	17	5	65	68	98
17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

In [14]:

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

Out[14]:

	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
8	9	3	20	98	15
9	10	3	50	98	45
10	11	3	65	65	65
11	12	3	45	85	32
16	17	5	65	68	98
17	18	5	80	78	90
18	19	5	89	68	87
19	20	5	78	83	58

추출한 행으로 데이터 만들기

In [15]:

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

Out[15]:

46.25

In [16]:

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

Out[16]:

61.25

(알아 두면 좋아요) 외부 변수를 이용해 추출하기

```
In [17]: var = 3  
exam.query('nklass == @var')
```

```
Out[17]:
```

	id	nklass	math	english	science	
	8	9	3	20	98	15
	9	10	3	50	98	45
	10	11	3	65	65	65
	11	12	3	45	85	32

```
In [ ]:
```

06-3 필요한 변수만 추출하기

변수로 추출하기

> 단일 변수로 추출하면 결과는 Series형

```
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
```

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

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

변수 나열(list)로 추출하기

> 복수 개 변수로 추출하면 결과는 Data Frame형

```
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

In [37]:

```
x = exam[['math']] # 단일 변수라도 []로 추출하면 결과는 DataFrame형  
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 [18]: ## 데이터 프레임의 컬럼 순서를 바꾸기
clist = ['id', 'nclass', 'science', 'english', 'math'] #컬럼 변수의 순서
exam = exam[clist]
exam
```

```
Out[18]:
```

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

컬럼 변수 제거하고 추출하기

> 원 데이터 프레임에는 영향을 주지 않음

```
In [19]: exam.drop(columns = 'math') #자신 구조에는 영향을 주지 않고, 자료만 반환
```

Out[19]:

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

In [21]: `exam.drop(columns = ['english', 'science']) # math, english 제거`

Out[21]:

	id	nclass	math
0	1	1	50
1	2	1	60
2	3	1	45
3	4	1	30
4	5	2	25
5	6	2	50
6	7	2	80
7	8	2	90
8	9	3	20
9	10	3	50
10	11	3	65
11	12	3	45
12	13	4	46
13	14	4	48
14	15	4	75
15	16	4	58
16	17	5	65
17	18	5	80
18	19	5	89
19	20	5	78

pandas 함수 조합하기

`query()` 와 `[]` 조합하기

```
In [22]: # nclass가 1인 행만 추출한 다음 english 추출  
exam.query('nclass == 1')['english']
```

```
Out[22]: 0    98  
1    97  
2    86  
3    98  
Name: english, dtype: int64
```

```
In [23]: exam.query('nclass == 1')['english'].mean()
```

```
Out[23]: 94.75
```

```
In [24]: # math가 50 이상인 행만 추출한 다음 id, math 추출  
exam.query('math >= 50')[['id', 'math']]
```

```
Out[24]:   id  math
```

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

```
In [25]: # math가 50 이상인 행만 추출한 다음 id, math 앞부분 5행까지 추출  
exam.query('math >= 50')[['id', 'math']].head()
```

Out[25]:

	id	math
0	1	50
1	2	60
5	6	50
6	7	80
7	8	90

가독성 있게 코드 줄 바꾸기

```
In [26]: # math가 50 이상인 행만 추출  
# id, math 추출, # 앞부분 10행 추출  
exam.query('math >= 50')  
    [['id', 'math']]  
    .head()
```

Out[26]:

	id	math
0	1	50
1	2	60
5	6	50
6	7	80
7	8	90

In []:

06-4 순서대로 정렬하기

오름차/내림차 순으로 정렬하여 추출하기

> 추출을 하므로 대상 데이터 프레임에 영향을 주지 않는다.

```
In [27]: exam.sort_values('math') # math 오름차순 정렬
```

```
Out[27]:
```

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

```
In [28]: exam.sort_values('math', ascending = False) # math 내림차순 정렬
```

Out[28]:

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

In [29]:

```
# nclass, math 오름차순 정렬  
exam.sort_values(['nclass', 'math'])
```

Out[29]:

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

In [30]:

```
# nclass 오름차순, math 내림차순 정렬  
exam.sort_values(['nclass', 'math'], ascending = [True, False])
```

Out[30]:

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

In []:

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

Out[31]:

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

여러 파생변수 한 번에 추가하기

In [32]:

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

```
Out[32]:
```

	id	nclass	science	english	math	total	mean
0	1	1	50	98	50	198	66.000000
1	2	1	60	97	60	217	72.333333
2	3	1	78	86	45	209	69.666667
3	4	1	58	98	30	186	62.000000
4	5	2	65	80	25	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 [33]:
```

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

Out[33]:

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

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

`new_exam`

Out[4]:

	id	nclass	math	english	science	total	mean	result
0	1	1	50	98	50	198	66.000000	pass
1	2	1	60	97	60	217	72.333333	pass
2	3	1	45	86	78	209	69.666667	pass
3	4	1	30	98	58	186	62.000000	pass
4	5	2	25	80	65	170	56.666667	fall
5	6	2	50	89	98	237	79.000000	pass
6	7	2	80	90	45	215	71.666667	pass
7	8	2	90	78	25	193	64.333333	pass
8	9	3	20	98	15	133	44.333333	fall
9	10	3	50	98	45	193	64.333333	pass
10	11	3	65	65	65	195	65.000000	pass
11	12	3	45	85	32	162	54.000000	fall
12	13	4	46	98	65	209	69.666667	pass
13	14	4	48	87	12	147	49.000000	fall
14	15	4	75	56	78	209	69.666667	pass
15	16	4	58	98	65	221	73.666667	pass
16	17	5	65	68	98	231	77.000000	pass
17	18	5	80	78	90	248	82.666667	pass
18	19	5	89	68	87	244	81.333333	pass
19	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 [2]: # 선언된 lamda 함수 바로 사용  
(lambda x: 2**x)(10)
```

```
Out[2]: 1024
```

```
In [3]: ## [lamda 함수 활용] 2의 지수승  
result = (lambda x: 2**x)(10)  
print(result)
```

```
1024
```

```
In [4]: # 긴 데이터 프레임명 지정  
long_name = pd.read_csv('exam_csv.csv')  
  
# long_name 직접 입력  
long_name.assign(new = long_name['math'] + long_name['english'] + long_name['science'])
```

Out[4]:

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

In [6]: ## <> lambda 함수를 활용한 이름 줄여쓰기:

```
# long_name으로 lambda가 실행되므로 매개변수 x는 각체명 long_name에 해당
long_name.assign(new = lambda x: x['math'] + x['english'] + x['science'])) # long_name 대신 x 입력
```

Out[6]:

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

In [7]: ## lambda 함수의 병행 사용

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

Out[7]:

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

In [22]:

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

```
-----  
KeyError Traceback (most recent call last)  
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key)  
    3789         try:  
-> 3790             return self._engine.get_loc(casted_key)  
    3791     except KeyError as err:  
  
index.pyx in pandas._libs.index.IndexEngine.get_loc()  
  
index.pyx in pandas._libs.index.IndexEngine.get_loc()  
  
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()  
  
pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()  
  
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)  
  
~\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]  
  
~\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'
```

```
In [48]: exam.assign(total = exam['math'] + exam['english'] + exam['science'],  
                  mean = lambda x: x['total'] / 3)  
# total 변수를 생성하고 lambda 함수를 호출하는 방식으로 진행
```

Out[48]:

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

06-6 집단별로 요약하기

집단 요약하기

> 단일 통계값을 구하기 (aggregate: 총 종합)

```
> agg(): 대상 변수에 대해 단일 통계값을 구해줌  
>> agg(func=적용할_함수명, axis=0) #axis는 열(0) 또는 행(1) 기준  
>> agg(변수명 = (대상_변수명, 적용할_함수명))  
>> 적용 가능 내장함수 : 'sum', 'mean', 'median', 'min', 'max', 'count', 'std', 'var'
```

```
In [20]: import pandas as pd
```

```
# Sample DataFrame  
df = pd.DataFrame({ 'A': [1, 2, 3, 4],  
                    'B': [5, 6, 7, 8],  
                    'C': [9, 10, 11, 12]})  
df
```

```
Out[20]:   A  B  C
```

	A	B	C
0	1	5	9
1	2	6	10
2	3	7	11
3	4	8	12

```
In [21]: ## pandas.agg(): 집계 함수의 이름을 문자열로 전달  
df['A'].agg(func='sum')  
df['A'].agg('sum')
```

```
Out[21]: 10
```

```
In [28]: ## pandas.agg(): 열 기준(default, axis=0) 집계  
df.agg('sum')
```

```
Out[28]: A    10  
         B    26  
         C    42  
        dtype: int64
```

```
In [27]: ## pandas.agg(): 행 기준(axis=1) 집계  
df.agg('sum', axis=1)
```

```
Out[27]: 0    15  
1    18  
2    21  
3    24  
dtype: int64
```

```
In [23]: ## pandas.agg(): 사용자 정의 함수를 사용하여 집계  
def custom_func(x):  
    return x.max() - x.min()  
  
result = df.agg(custom_func)  
result
```

```
Out[23]: A    3  
B    3  
C    3  
dtype: int64
```

```
In [26]: ## pandas.agg(): 행 기준(axis=1) 집계  
result = df.agg(custom_func, axis=1)  
result
```

```
Out[26]: 0    8  
1    8  
2    8  
3    8  
dtype: int64
```

```
In [32]: ## pandas.agg(): 컬럼에 대해 집계, .agg(변수 = (컬럼명, 집계함수))  
# 받아내는 변수는 결과 데이터 프레임의 index가 된다.  
df.agg(avg = ('A', 'mean'))
```

```
Out[32]: A  
avg 2.5
```

```
In [ ]:
```

```
In [8]: ## math 평균 구하기  
# 받아내는 변수는 결과 데이터 프레임의 index가 된다.  
exam.agg(mean_math = ('math', 'mean')) # 결과는 DataFrame 형
```

```
Out[8]:
```

math	
mean_math	57.45

```
In [9]:
```

```
## nclass 개수 구하기  
exam.agg(cnt_nclass = ('nclass', 'count')) #결과는 DataFrame 형
```

```
Out[9]:
```

nclass	
cnt_nclass	20

```
In [ ]:
```

집단별로 요약하기

- > 집단별로 통계값을 구하기
- > **groupby()** : 집단별로 통계값을 구해줌
 - >> **groupby(집단구분_변수명).agg(변수명 = (대상_변수명, 적용할_함수명))**

```
In [51]:
```

```
## exam.groupby('nclass')는 집단별 분리이므로 .count()가 가능  
exam.groupby('nclass').count()
```

```
Out[51]:
```

	id	science	english	math
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 [52]:
```

```
## '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[52]: **mean_math**

nclass	mean_math
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 [33]:

```
# '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[33]:

nclass	mean_math	sum_math	median_math	n
1	46.25	185	47.5	4
2	61.25	245	65.0	4
3	45.00	180	47.5	4
4	56.75	227	53.0	4
5	78.00	312	79.0	4

모든 변수의 요약 통계량 한 번에 구하기

```
In [34]: ## exam.groupby('nclass')는 집단별 분리이므로 .count()가 가능  
exam.groupby('nclass').count()
```

```
Out[34]:      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 [35]: ## 'nclass'별 인원 수 구하기  
exam.groupby('nclass').mean()
```

```
Out[35]:      id  math  english  science
```

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

집단별로 다시 집단 나누기

```
In [36]: ## mpg 데이터 불러오기  
import pandas as pd  
mpg = pd.read_csv('mpg.csv')  
mpg.head()
```

Out[36]:

	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 [37]: ## 제조회사(manufacturer)별 구동방식(drv)별 cty 평균 구하기
mpg.groupby(['manufacturer', 'drv'])
.agg(mean_cty = ('cty', 'mean'))

Out[37]:

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

원하는 집단만 통계 전에 필터링 하기

```
In [38]: ## 제조회사(manufacturer)가 'audi'인 것에 대해서만 구동방식(drv)별 cty 평균 구하기  
mpg.query('manufacturer == "audi"') %>  
  .groupby(['manufacturer', 'drv']) %>  
  .agg(mean_cty = ('cty', 'mean'))
```

Out[38]: mean_cty

manufacturer	drv	mean_cty
audi	4	16.818182
	f	18.857143

원하는 집단만 통계 후에 필터링 하기

```
In [39]: ## 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[39]: mean_cty

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

In []:

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

>> mpg.groupby(['drv'])로 해결하는 방법

```
In [40]: ## '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[40]: n
```

drv	n
4	103
f	106

>> mpg['drv']로 해결하는 방법

```
In [62]: mpg['drv'].value_counts()
```

```
Out[62]: drv  
f    106  
4    103  
r    25  
Name: count, dtype: int64
```

```
In [63]: mpg['drv'].value_counts().query('n > 100')
```

```
-----  
AttributeError: 'Series' object has no attribute 'query'  
Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_6756\1172614733.py in <module>  
----> 1 mpg['drv'].value_counts().query('n > 100')  
  
~\anaconda3\lib\site-packages\pandas\core\generic.py in __getattr__(self, name)  
    6200     ):  
    6201         return self[name]  
-> 6202         return object.__getattribute__(self, name)  
    6203  
    6204     @final  
  
AttributeError: 'Series' object has no attribute 'query'
```

<문제 발생> mpg['drv'].value_counts().query('n > 100')

> .query()는 데이터 프레임 메서드이기 때문

<문제 해결> .query('n > 100')에서 'n'을 프레임에 포함시키고, 'n'을 사용하여 필터링 수행

```
In [65]: mpg['drv'].value_counts() #  
        .to_frame('n')
```

Out[65]: n

drv	n
f	106
4	103
r	25

```
In [66]: mpg['drv'].value_counts() #  
        .to_frame('n') #  
        .query('n > 100')
```

Out[66]: n

drv	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

In []:

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 [41]: # 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 [42]: 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 [43]: ## 두 프레임을 id 기준으로 합쳐서 total에 할당
total = pd.merge(test1, test2, how = 'left', on = 'id') #'id' 기준, 왼쪽으로 합병
total
```

```
Out[43]:   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 [44]: ## name 데이터 프레임을 만들기
name = pd.DataFrame({'nclass' : [1, 2, 3, 4, 5],
                     'teacher' : ['kim', 'lee', 'park', 'choi', 'jung']})
name
```

Out[44]:

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

In [45]:

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

Out[45]:

	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

[연습] Outer Join

name 프레임을 nclass = 1을 빼고 생성한 후에,

exam과 name 프레임을 Outer Join 수행

```
In [46]: ## name 데이터 프레임을 만들기 : nclass = 1을 빼고 생성  
name = pd.DataFrame({'nclass' : [2, 3, 4, 5],  
                     'teacher' : ['lee', 'park', 'choi', 'jung']})  
name
```

```
Out[46]:   nclass  teacher  
0         2      lee  
1         3     park  
2         4     choi  
3         5    jung
```

```
In [49]: ## exam, name 프레임을 'nclass' 기준으로 Inner Join  
exam_join = pd.merge(exam, name, on='nclass')  
exam_join
```

Out[49]:

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

In [47]:

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

Out[47]:

	id	nclass	math	english	science	teacher
0	1	1	50	98	50	NaN
1	2	1	60	97	60	NaN
2	3	1	45	86	78	NaN
3	4	1	30	98	58	NaN
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

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 [50]: ## 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 [51]: 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 [52]: ## group_a와 group_b를 group_all로 행 합치기
group_all = pd.concat([group_a, group_b])
```

```
group_all
```

```
Out[52]:   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 [53]: ## group_a와 group_b를 group_all로 행 합치기
exam_all = pd.concat([group_a, group_b], ignore_index=True) #행으로 합치기, index 무시 후 새로 부여
exam_all
```

Out[53]:

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

In [54]: ## group_a와 group_b를 group_all로 열 합치기

```
group_all = pd.concat([group_a, group_b], axis=1) #열로 합치기  
group_all
```

Out[54]:

	id	test	id	test
0	1	60	6	70
1	2	80	7	83
2	3	70	8	65
3	4	90	9	95
4	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_mean과 exam_subj를 id를 기준으로 열 합병하여 exam_result 데이터 프레임 생성
- <3> nclass로 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 데이터 프레임 만들기`

```
exam_mean.head()
```

Out[22]:

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

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

```
exam_subj.head()
```

Out[23]:

	id	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

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

```
exam_result.head()
```

Out[25]:

	id	nclass	total	mean	math	english	science
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
4	5	2	170	56.666667	25	80	65

In [26]: ## <3> exam_result를 nclass를 기준으로 행 분리하여 별도의 데이터 프레임 exam_1class, exam_2class, exam_3class, exam_4class, exam_5c

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

	id	nclass	total	mean	math	english	science
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
	id	nclass	total	mean	math	english	science
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
	id	nclass	total	mean	math	english	science
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
	id	nclass	total	mean	math	english	science
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
	id	nclass	total	mean	math	english	science
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 모든 행들을 병합하여 exam_nclass 생성

```
exam_nclass
```

Out[27]:

	id	nclass	total	mean	math	english	science
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
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
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
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
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 []:

[과제] 위의 [실습-1], [실습-2], [실습-3], [실습-4] 문제를 해결하기

> 해결한 *.ipynb 파일을 LMS 과제란에 업로드하여 제출

In []:

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]:

	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 []: