2022.11~2022.12

Pandas

차 례

<제목 차례>

2.2 Numpy와 Pandas

1 Numpy

Numpy(Numerical Python)는 C언어로 구현된 파이썬 라이브러리로써, 고성능의 수치계산에 적합하여 벡터와 행렬 연산에 사용된다. 넘피(Numpy)는 기본적으로 array라는 단위로 이루어진 행렬(matrix)이다. 넘피 모듈을 불러 'np'라고 정의한다❶.

```
import numpy as np                                                    ❶
```

'0'부터 '9'까지 정수 리스트와❶ 배열을❷ 만든다. Numpy의 arange() 함수로 만든 배열과❸ 짝수 배열을❹ 만들어 리스트로❺ 변경한다.

```
list(range(10))                                                       ❶
np.array(range(0,10))                                                 ❷
np.arange(10)                                                         ❸
np.arange(0,10,2)                                                     ❹
list(np.arange(0,10,2))                                               ❺
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
array([0, 2, 4, 6, 8])
[0, 2, 4, 6, 8]
```

'0'부터 '10'까지 배열과❶ '0'부터 '20'까지 짝수 배열을❷ 만들어 수평과 수직으로 두 개의 배열을 합쳐 확인한다❸❹.

```
np.arange(10)                                                         ❶
np.arange(0, 20, 2)                                                   ❷
np.hstack([np.arange(10), np.arange(0, 20, 2)])                       ❸
np.vstack([np.arange(10), np.arange(0, 20, 2)])                       ❹
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9,  0,  2,  4,  6,  8, 10, 12,
       14, 16, 18])

array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
       [ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18]])
```

리스트는 여러 가지 자료 형태의 원소를 넣을 수 있지만, 메모리 용량이 크고 속도가 느리다❶. 반면에 Numpy array는 같은 자료형태의 원소로 메모리를 최적화하여 연산 속도가 빠르며 전체 연산이 가능하다❷.

```
%time s = sum([i for i in range(1,99999)])                    ❶
%time s = np.arange(99999).sum()                              ❷
```

```
Wall time: 4.99 ms
Wall time: 0 ns
```

무작위 시드(random seed)는 사실 무작위처럼 보이지만 정해진 알고리즘에 의해 생성되는 규칙적인 순열로 시작점이 특정되기 때문에 정해진 숫자가 나오게 된다. 필자와 독자의 코드 결과를 동일하게 재현하기 위해 무작위 시드(seed)를 '1357'로❶ 설정한 후 '1'부터 '100'까지 10개의 무작위 복원추출 배열을 만들어 확인한다❷❸. 최대와 최소값을 확인하고❹❺ 10개 자료의 합계❻, 평균❼, 표준편차를❽ 확인한다.

```
❶ np.random.seed(1357)
❷ m = np.random.randint(1,100,12)
❸ m

❹ m.max()
❺ m.min()
❻ m.sum()
❼ m.mean()
❽ m.std()
```

```
array([52, 71, 72, 80, 72, 53, 71, 46, 68, 59, 55, 19])
80
19
718
59.833333333333336
15.836841716565697
```

최소값 '19'와 최대값 '80'를 가진 원소의 배열 인덱스는 각각 11번째와❶ 3번째이며❷ 최소값부터 최대값까지 순서대로❸ 혹은 그 역순으로 데이터의 위치(index)를❹ 확인한다.

```
m.argmin()                                                    ❶
m.argmax()                                                    ❷
m.argsort()                                                   ❸
np.argsort(-m)                                                ❹
```

```
11
3
array([11,  7,  0,  5, 10,  9,  8,  1,  6,  2,  4,  3], dtype=int64)
array([ 3,  2,  4,  1,  6,  8,  9, 10,  5,  0,  7, 11], dtype=int64)
```

벡터를❶ 재배열하면(reshape(-1,1)) 원소의 개수만큼의 행렬로❷ 변환할 수 있다. 머신러닝 예측 모형은 행렬(matrix)의 입력변수를 요구하기 때문에 벡터는 행렬로 변환시켜야 한다.

```
m.shape                                  ❶
m.reshape(-1,1).shape                    ❷
```

```
(12,)
(12, 1)
```

배열 'm'과❶ 크기를❷ 확인하고 3개 행과 4개 열로 재구성된 행렬을❸ 확인한다❹.

```
m                                        ❶
m.shape                                  ❷
mr = m.reshape(3,4)                      ❸
mr                                       ❹
```

```
array([52, 71, 72, 80, 72, 53, 71, 46, 68, 59, 55, 19])

(12,)

array([[52, 71, 72, 80],
       [72, 53, 71, 46],
       [68, 59, 55, 19]])
```

배열의 두 번째와 세 번째 원소값과❶ 첫 번째와 두 번째 행❷, 전체 행과 두 번째 세 번째 열을❸ 각각 확인한다. 전체 행렬의 세 번째 열까지❹, 두 번째 행과 세 번째 열의 원소값❺, 두 번째까지 행과 두 번째부터 마지막 열까지의 행렬을❻ 확인한다.

```
❶ mr[2][3]
❷ mr[0:2]
❸ mr[:, [1,2]]
❹ mr[:, 0:3]
❺ mr[:2, 2:]
```

```
19

array([[52, 71, 72, 80],
       [72, 53, 71, 46]])

array([[71, 72],
       [53, 71],
       [59, 55]])

array([[52, 71, 72],
       [72, 53, 71],
       [68, 59, 55]])

array([[72, 80],
       [71, 46]])
```

2 Pandas

판다스(Pandas)는 데이터 정제, 분석 등에 가장 많이 사용되는 데이터 분석 라이브러리로, 행과 열로 이루어진 객체를 만들어 데이터를 엑셀처럼 처리할 수 있다. 판다스는 대용량의 데이터를 안정적이고 빠르게 처리하는 장점을 갖고 있다. 판다스 모듈을 'pd'라는 이름으로 읽고❶ 행렬을 객체화하여❷ 데이터프레임으로❸ 만든다.

```
import pandas as pd                                    ❶
data = m.reshape((3,4))                                ❷
pd.DataFrame(data)                                     ❸
```

	0	1	2	3
0	52	71	72	80
1	72	53	71	46
2	68	59	55	19

행과 열 이름을❶❷ 지정하여❸ 데이터프레임을❹ 만들면 엑셀 형태와 같다. 판다의 장점은 대용량 파일을 파이썬 코드로 효과적으로 다룰 수 있다.

```
index = range(1,4)                                     ❶
columns = ['A','B','C','D']                            ❷
df = pd.DataFrame(data, index = index, columns = columns)   ❸
df                                                     ❹
```

	A	B	C	D
1	52	71	72	80
2	72	53	71	46
3	68	59	55	19

특정 행의 데이터만 추출하려면 loc() 명령을 사용하여 '2번' 행을 지정하고❶ 위치기반 인덱싱을 사용하는 iloc() 명령을 사용하여 '1'을 지정하여❷ 두 번째 행의 값을 확인한다. 특정한 자료를 확인하기 위해서 loc()와 iloc()를❸❹ 모두 사용할 수 있다.

```
df.loc[2]                                                    ❶
df.iloc[1]                                                   ❷
df.loc[2, 'B']                                               ❸
df.iloc[1, 1]                                                ❹
```

```
A    72
B    53
C    71
D    46
Name: 2, dtype: int32

A    72
B    53
C    71
D    46
Name: 2, dtype: int32

53

53
```

특정 열만 지정하면❶ 자료형태가 pandas 시리즈(Series)이다❷. 하나 이상의 열을 지정하여 ❸ 자료형태를 확인하고❹. 'A'와 'B' 두 개 열의 '2'번째 행의 데이터를❺ 확인한다.

```
df['A']                                                      ❶
type(df['A'])                                                ❷
df[['A','B']]                                                ❸
type(df[['A','B']])                                          ❹
df[['A','B']].loc[2]                                         ❺
```

```
1    52
2    72
3    68
Name: A, dtype: int32

pandas.core.series.Series
```

	A	B
1	52	71
2	72	53
3	68	59

```
pandas.core.frame.DataFrame

A    72
B    53
Name: 2, dtype: int32
```

데이터프레임을 두 번째 행부터 마지막까지❶ slicing하여 마지막 행만❷ 확인한다.

```
df[1:]                                                    ❶
df[-1:]                                                   ❷
```

	A	B	C	D
2	72	53	71	46
3	68	59	55	19

	A	B	C	D
3	68	59	55	19

자료의 특정 열을 지정하려면 'df['컬럼']'❶ 혹은 'df.컬럼'으로❷ 입력한다. 다양한 함수를 차례로 적용하는 경우는 'df['컬럼']' 방식이 더 편리하다.

```
df['A']                                                   ❶
df.A                                                      ❷
```

```
1    52
2    72
3    68
Name: A, dtype: int32

1    52
2    72
3    68
Name: A, dtype: int32
```

데이터프레임은 성별, 나이, 직업 등 각 열의 데이터 형태인 시리즈(Series) 여러 개가 세로로 쌓아져 만들어진 것이다. 데이터프레임의 각 열은 서로 다른 값(숫자, 문자열, Boolean 등)을 가질 수 있지만 시리즈는 같은 자료형태이다. 시리즈는 1차원 배열의 값에 대응되는 인덱스를 부여할 수 있는 구조이고 데이터프레임은 2차원의 행 인덱스와 열 컬럼을 갖는 레이블이 있는 엑셀 형태의 자료 구조이다.

판다스의 주요 기능은 엑셀과 같이 조건문(filitering), 피봇팅(pivoting)이다.

'60' 이상인 자료를 확인하여❶ 데이터프레임으로 구성하면 자료의 값이 True인 경우의 자료만 가져오고 False인 경우는 'NaN'값을❷ 갖게 된다. 'NaN'은 자료 입력이 없는 결측치이다. '60' 이상인 Bool 형태의 행렬에 1을 곱하여 '1'과 '0'의 '행렬을 만든 후❸ 인덱스와 열을 레이블하여 데이터프레임을❹ 확인한다.

```
df>=60                                                                  ❶
df[df>=60]                                                              ❷
np.array(df>=60)*1 # (df>=60).values*1                                  ❸
pd.DataFrame(np.array(df>=60)*1, index = index, columns = columns)      ❹
```

	A	B	C	D
1	False	True	True	True
2	True	False	True	False
3	True	False	False	False

	A	B	C	D
1	NaN	71.0	72.0	80.0
2	72.0	NaN	71.0	NaN
3	68.0	NaN	NaN	NaN

```
array([[0, 1, 1, 1],
       [1, 0, 1, 0],
       [1, 0, 0, 0]])
```

	A	B	C	D
1	0	1	1	1
2	1	0	1	0
3	1	0	0	0

C열의 자료가 '60'이 넘고❶ B와 C의 열의 값이 각각 '50'과 '70'을 넘는 자료를❷ 확인한다.

```
df[df['C']>60]                                                          ❶
df[(df['B']>50)&(df['C']<70)]                                           ❷
```

	A	B	C	D
1	52	71	72	80
2	72	53	71	46

	A	B	C	D
3	68	59	55	19

코드의 '='기호는 실제 데이터의 내용이 복사되는 것이 아니라 메모리 위치가 복사되는 것이다. 데이터의 내용까지 복사하여 저장한 후❶ 새로운 열 'D'을 만들고❷❸ 컬럼별로 NaN의 개수❹, 고유한 값의 개수❺, 자료형태를❻ 살펴보았다. A 열 결측치는 1개, 고유값은 '72.0'과 '68.0' 2개의 실수형 자료이다. 열 'E'는 결측치가 없고 고유값은 'owl'과 'fox'인 object

형 자료이다. 자료의 형태가 'object'인 의미는 숫자나 문자로 정의되지 않은 자료형태이다.

```
df = df[df>60].copy()                                                    ❶
df['E'] = ['owl', 'fox', 'owl']                                          ❷
df                                                                       ❸
df.isnull().sum()                                                        ❹
df.nunique()                                                             ❺
df.dtypes                                                                ❻
```

	A	B	C	D	E
1	NaN	71.0	72.0	80.0	owl
2	72.0	NaN	71.0	NaN	fox
3	68.0	NaN	NaN	NaN	owl

```
A    1
B    2
C    1
D    2
E    0
dtype: int64

A    2
B    1
C    2
D    1
E    2
dtype: int64

A    float64
B    float64
C    float64
D    float64
E     object
dtype: object
```

결측치 수❶, 고유값의 개수❷, 자료형태는 ❸각각 5개의 원소를 갖는 시리즈다

```
type(df.isnull().sum()); df.isnull().sum().shape                         ❶
type(df.nunique()); df.nunique().shape                                   ❷
type(df.dtypes); df.dtypes.shape                                         ❸
```

```
pandas.core.series.Series
(5,)

pandas.core.series.Series
(5,)

pandas.core.series.Series
(5,)
```

자료 결측치 값에 '0'을 채워 확인한다❶. 명령어 'inplace=True'를❷ 입력하면 데이터프레임에 직접 적용하여 자료를 변경한다❸.

```
df.fillna(0)                                                    ❶
df.fillna(0, inplace = True)  # df = df.fillna(0)               ❷
df                                                              ❸
```

	A	B	C	D	E
1	0.0	71.0	72.0	80.0	owl
2	72.0	0.0	71.0	0.0	fox
3	68.0	0.0	0.0	0.0	owl

	A	B	C	D	E
1	0.0	71.0	72.0	80.0	owl
2	72.0	0.0	71.0	0.0	fox
3	68.0	0.0	0.0	0.0	owl

데이터프레임의 특정 조건에 해당되는 행과 열을 Bool 값으로 찾아내어 삭제할 수 있다. 데이터프레임을 만든 후에❶❷ 행 'c'를 삭제❹, 행 'a'와 'c'를 삭제하여❺ 확인한다.

```
df = pd.DataFrame(mr, index = [i for i in 'abc'], columns = ['A','B','C','D'])   ❶
df                                                                                ❷
df.drop('c')                                                                      ❸
df.drop(['a', 'c'])                                                               ❹
```

	A	B	C	D
a	52	71	72	80
b	72	53	71	46
c	68	59	55	19

	A	B	C	D
a	52	71	72	80
b	72	53	71	46

	A	B	C	D
b	72	53	71	46

열 ‘C’와 열 ‘A’와 ‘C’를 데이터프레임에서 삭제하려면❶❷ 데이터프레임의 열을 특정해주는 ‘axis=1’을 입력하면 된다.

```
df.drop('C', axis=1)                    ❶
df.drop(['A', 'C'], axis=1)             ❷
```

	A	B	D
a	52	71	80
b	72	53	46
c	68	59	19

	B	D
a	71	80
b	53	46
c	59	19

두 번째 행과 같은 ‘d’ 인덱스를❶ 만들어 자료를 확인하고❷ 중복 체크하여❸ 중복이 없는 자료만 추출한다❹.

```
df.loc['d'] = df.loc['b']               ❶
df                                      ❷
df.duplicated()                         ❸
df[~df.duplicated()]                    ❹
```

	A	B	C	D
a	52	71	72	80
b	72	53	71	46
c	68	59	55	19
d	72	53	71	46

```
a    False
b    False
c    False
d     True
dtype: bool
```

	A	B	C	D
a	52	71	72	80
b	72	53	71	46
c	68	59	55	19

3장 데이터 현황분석

3.1 데이터 불러오기

3.2 데이터 분석

3장 데이터 현황분석

3.1 데이터 불러오기

1 데이터 불러오고 확인

IPython 한 셀에 여러 명령어를 입력하여도 모든 결과가 출력되는 모듈과❶ 넘피와 판다스를 ❷❸ 불러와 각 각의 버전을❹ 확인한다.

```
In [1] : from IPython.core.interactiveshell import InteractiveShell                    ❶
         InteractiveShell.ast_node_interactivity = 'all'
         import numpy as np                                                            ❷
         import pandas as pd                                                           ❸
         np.__version__, pd.__version__                                                ❹
Out [1] : ('1.19.1', '1.0.5')
```

'data'폴더에 내려받은 'MCR2019.xlsx'의 2번째 행을 컬럼 명으로 하여(header=1) 데이터프레임을 저장하여❶ 확인하면 관측 수는 4,000개, 컬럼 변수는 3,619개이다❷. 데이터프레임의 처음 5줄의 내용을 확인한다❸. '%tim'로 자료를 읽는데 36.6초가 걸린 걸 확인한다.

```
In [2] : %time df = pd.read_excel('../data/MCR2019.xlsx', header=1)                    ❶
         df.shape                                                                      ❷
         df.head()                                                                     ❸
Wall time: 36.6 s
(4000, 3619)
```

Out [2] :

	조사년도	성별	연령(10대)	연령(1세)	지역(권역)	직업	가구월평균소득(1)	본인월평균소득(1)	한달평균지출액(신용카드+체크카드)	최종학력	...	평소즐겨보는뉴스및콘텐츠	평소즐겨보는뉴스및콘텐츠.1	평소즐겨보는뉴스및콘텐츠.2	평소즐겨보는뉴스및콘텐츠.3	평소즐겨보는뉴스및콘텐츠.4	평소즐겨보는뉴스및콘텐츠.5	평소즐겨보는뉴스및콘텐츠.6	평소즐겨보는뉴스및콘텐츠.7	평소
0	2019	1	1	14	1	6	6	NaN	1	8	...	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	2019	1	1	14	1	6	6	NaN	99	8	...	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	2019	1	1	14	1	6	6	NaN	1	8	...	3	6.0	7	NaN	NaN	NaN	NaN	NaN	
3	2019	1	1	14	1	6	8	NaN	99	8	...	3	4.0	NaN	NaN	NaN	NaN	NaN	NaN	
4	2019	1	1	14	1	6	9	NaN	1	8	...	3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 3619 columns

소비자 구매 행태에 영향을 주는 변수는 응답자의 인구통계학적 특성(성별, 소득, 연령, 학력 등)과 인터넷 및 모바일 이용시간, 즐겨 보는 TV 프로그램 장르, TV 시청 시 즐겨 보는 국내 드라마 장르 등 38개 변수로 압축할 수 있다. 소비자가 좋아하는 TV와 드라마의 장르의 유형은 '뉴스/보도', '교양/생활정보', '시사/토론', '국내 드라마,' '해외 드라마', '영화', '예능/오락', '다큐멘터리', '음악/공연/오디션', '스포츠/골프', '어린이/만화', '교육', '게임', '종교', '경제정보', '취미/레저' 등이다. 소비자가 선호하는 장르 유형은 같은 설문 항목을 반복적으로 구성하여 소비자에게 여러 차례에 걸쳐 응답을 받았다.

데이터 분석에 필요한 38개 변수만 지정하여 2번째 행을 열의 이름으로(header=1) 하여 자료를 읽는다❶. 파이썬의 pickle 모듈을 사용하여 binary 형태로 압축하고 파일 확장자명을 'pkl'로 'MCR2019.pkl' 파일명으로 저장한다❷. 엑셀을 읽는데 34.3초가 걸렸다.

```
%time df = pd.read_excel('../data/MCR2019.xlsx', header= 1, usecols= selected_col)  ❶
%time df.to_pickle('../data/MCR2019.pkl')                                           ❷
```

```
Wall time: 34.3 s
Wall time: 3.99 ms
```

'pkl' 확장자로 중간마다 작업한 내용을 저장하면 다음에 빠르게 해당 파일을 읽을 수 있다❶. 피클로 4.96초 만에 읽는다. 자료의 컬럼 명과❷ 상세 정보를❸ 확인한다. 4,000개의 관측치, 38개의 변수, '성별' 변수의 경우 64bit 정수형으로 결측치는 없다. 전체 자료는 15개의 64bit 실수형, 11개의 64bit 정수형 변수, 12개의 object형 변수를 갖고 있다.

```
%time df = pd.read_pickle('../data/MCR2019.pkl')  ❶
df.columns                                        ❷
df.info()                                         ❸
```

```
Wall time: 4.96 ms

Index(['성별', '연령(1세)', '지역(권역)', '직업', '가구 월평균 소득(1)', '최종학력', '주거 형태',
~~중략 ~~ "TV 시청 시 즐겨 보는 국내 드라마 장르-내용.9',
       'PPL(간접광고) 평가 - 제품/브랜드 구매의향'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 38 columns):
성별                                    4000 non-null int64
연령(1세)                               4000 non-null int64
지역(권역)                               4000 non-null int64
~~ 중략 ~~
TV 시청 시 즐겨 보는 국내 드라마 장르-내용.9     22 non-null object
PPL(간접광고) 평가 - 제품/브랜드 구매의향        4000 non-null int64
dtypes: float64(15), int64(11), object(12)
memory usage: 1.2+ MB
```

각 열의 관측개수, 평균, 표준편차, 최소값, 최대값, 백분율 등 요약 통계를❶ 확인한다.

```
df.describe()                                                   ❶
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부	자녀 총 수	평일 인터넷 이용시간(분)	...	즐겨 보는 TV 프로그램 장르.1	즐겨 보 프로그
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.00000	4000.000000	4000.000000	4000.000000	2603.000000	...	3857.000000	3415.00
mean	1.494500	41.394500	2.945000	3.979000	5.645500	5.72725	1.268750	1.412000	38.854750	65.366884	...	4.228934	5.84
std	0.500032	15.151741	1.758621	1.710933	2.870608	2.20365	0.567982	0.556177	47.194316	61.173436	...	2.191840	2.26
min	1.000000	13.000000	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	0.000000	...	1.000000	1.00
25%	1.000000	28.000000	2.000000	3.000000	4.000000	4.00000	1.000000	1.000000	2.000000	30.000000	...	2.000000	4.00
50%	1.000000	43.000000	2.000000	4.000000	6.000000	5.00000	1.000000	1.000000	2.000000	60.000000	...	4.000000	6.00
75%	2.000000	54.000000	5.000000	5.000000	7.000000	6.00000	1.000000	2.000000	99.000000	60.000000	...	6.000000	7.00
max	2.000000	69.000000	7.000000	7.000000	99.000000	12.00000	5.000000	3.000000	99.000000	720.000000	...	17.000000	16.00

8 rows × 26 columns

판다스의 명령어 'df.dtypes', 'df.index', 'df.columns', 'df.values' 등과 aggregate 함수 명령어 'df.sum()', 'df.count()', 'df.mean()', 'df.std()', 'df.max()', 'df.min()', 'df.median()', 'df.quantile([0.25, 0.5, 0.75])' 등은 독자가 직접 확인하기 바란다. 첫 번째부터 10번째 열까지의 자료만 저장한다❶.

```
df = df[df.columns[0:11]]                                       ❶
```

남성 자료만 추출하여 확인한다❶.

```
df[df['성별'] == 1].head()  ❶
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부	자녀 총 수	평일 인터넷 이용시간(분)	토요일 인터넷 이용시간(분)
0	1	14	1	6	6	8	3	2	99	0.0	120.0
1	1	14	1	6	6	8	1	2	99	30.0	40.0
2	1	14	1	6	6	8	2	2	99	90.0	60.0
3	1	14	1	6	8	8	1	2	99	NaN	NaN
4	1	14	1	6	9	8	1	2	99	0.0	60.0

평일 100분 이상 인터넷을 이용하는 남성 자료를❶ 추출하였다. 특정 조건에 해당하는 데이터를 추출하기 위해서는 Boolean 조건이 상당히 유용하다.

```
df[(df['성별'] == 1) & (df['평일 인터넷 이용시간(분)'] >= 100)]  ❶
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부	자녀 총 수	평일 인터넷 이용시간(분)	토요일 인터넷 이용시간(
6	1	13	1	6	5	8	1	2	99	120.0	12
8	1	15	1	6	6	8	1	2	99	120.0	18
9	1	15	1	6	5	8	1	2	99	180.0	18
16	1	16	1	6	6	9	1	2	99	120.0	12
18	1	17	1	6	6	3	1	2	99	180.0	18
...	...	...	...	...	...	...	...	...	...	...	
3966	1	31	4	3	4	6	1	1	1	180.0	18
3968	1	36	4	3	4	4	2	1	1	120.0	
3971	1	43	4	2	6	6	1	1	1	120.0	12
3973	1	48	4	2	5	7	1	1	2	120.0	12
3975	1	53	4	1	6	7	1	1	2	120.0	12

330 rows × 11 columns

평일 100분 이상 인터넷을 이용하는 남성을 연령 기준 내림차순으로❶ 정리하였다.

```
df[(df['성별']==1) & (df['평일 인터넷 이용시간(분)']>=100)].sort_values('연령(1세)', ascending=False)  ❶
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부	자녀 총 수	평일 인터넷 이용시간(분)	토요일 인터넷 이용시간(
453	1	66	1	2	6	4	1	1	2	210.0	3
2643	1	65	2	1	11	6	1	1	3	180.0	3
1094	1	65	5	4	5	5	1	1	2	120.0	3
1583	1	64	2	3	3	4	1	1	2	120.0	12
437	1	64	1	4	3	4	1	3	2	120.0	6
...	...	...	...	...	...	...	...	...	...	...	
1732	1	13	4	6	5	8	1	2	99	120.0	6
2128	1	13	2	6	6	8	1	2	99	120.0	12
3752	1	13	5	6	6	8	1	2	99	240.0	24
1463	1	13	2	6	5	8	1	2	99	120.0	12
6	1	13	1	6	5	8	1	2	99	120.0	12

330 rows × 11 columns

성별과 직업을 기준으로 평일 인터넷 이용시간(분)을 보여주는 피봇테이블과❶ 그룹핑을❷ 확인한다. 판다스의 피봇테이블과 그룹핑은 같은 기능이다.

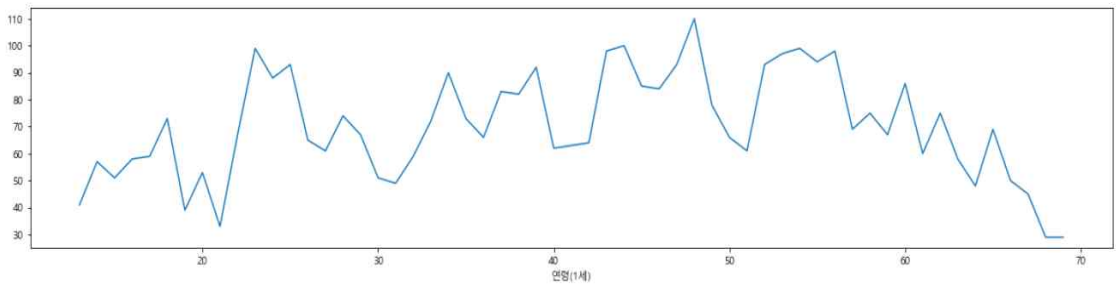
```
pd.pivot_table(df, values='평일 인터넷 이용시간(분)', index= '성별', columns= '직업')  ❶
df.groupby(['성별', '직업'])['평일 인터넷 이용시간(분)'].mean().unstack()  ❷
```

직업	1	2	3	4	5	6	7
성별							
1	73.904494	80.313390	65.957447	51.94	NaN	77.427746	66.153846
2	71.200000	64.487069	52.406417	47.50	49.212544	65.736301	68.636364

직업	1	2	3	4	5	6	7
성별							
1	73.904494	80.313390	65.957447	51.94	NaN	77.427746	66.153846
2	71.200000	64.487069	52.406417	47.50	49.212544	65.736301	68.636364

시각화 모듈인 matplotlib.pyplot을 'plt'으로 불러오고❶ 한글이 깨지지 않도록 글자폰트를 설정하여❷ 연령별로 평일 인터넷 이용시간(분)을 그림의 크기(가로, 세로)를 인치 단위((figure) =(20,4)) 설정하여 선 그래프로❸❹ 시각화하였다.

```
import matplotlib.pyplot as plt                                                    ❶
plt.rcParams['font.family']='Malgun Gothic'                                        ❷
df.groupby('연령(1세)')['평일 인터넷 이용시간(분)'].size().plot(figsize=(20,4))          ❸
plt.show();                                                                        ❹
```



구글에서 ‘pandas.DataFrame.plot’를 검색하면 다양한 시각화 예제가 있다. 시각화 모듈 matplotlib는 고품질의 그래프를 만들기 위한 목적의 패키지이다. 막대그래프, 상자 그림, 선 그래프 등 기본적인 통계 그래프는 물론 데이터를 지도에 대응하거나 3차원 그래프를 제공한다. 자세한 설명은 제5장 데이터 시각화에서 다루도록 하겠다.

3.2 데이터 분석

절차 등과 더불어 데이터의 형태와 구조를 이해하고 데이터 전처리 과정을 수행하여야 한다. 제3장 2절은 제4장의 데이터 전처리를 위해 자료의 현황을 파악하고 분석하는 탐색적 자료 분석(Exploratory data analysis) 과정이다. 파이썬의 명령어와 문법에 친근한 독자는 본 내용을 건너뛰어 제4장 데이터전처리로 넘어가도 된다.

1 데이터 컬럼별 분석

관련 모듈을 불러온다❶. Jupyter Notebook을 사용하다 보면 향후 버전이 올라갈 때 변경될 사항 등을 알려주는 경고 메시지가 거슬리는데 warnings 라이브러리를 사용하여 제거하면 해당 메시지가 나타나지 않는다❷.

```
from IPython.core.interactiveshell import InteractiveShell                    ❶
InteractiveShell.ast_node_interactivity = 'all'
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.family']='Malgun Gothic'
import warnings                                                               ❷
warnings.filterwarnings('ignore')
```

저장한 자료를❶ 불러와 결측치를❷ 확인한다.

```
df = pd.read_pickle('../data/MCR2019.pkl')                                    ❶
df.isnull()                                                                   ❷
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종 학력	주거 형태	결혼 여부	자녀 총 수	평일 인터넷 이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.1	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.4
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	True
1	False	False	False	False	False	False	False	False	False	False	...	False	False	True	True
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	True
3	False	False	False	False	False	False	False	False	False	True	...	True	True	True	True
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3995	False	False	False	False	False	False	False	False	False	True	...	False	False	True	True
3996	False	False	False	False	False	False	False	False	False	True	...	False	True	True	True
3997	False	False	False	False	False	False	False	False	False	True	...	False	False	False	True
3998	False	False	False	False	False	False	False	False	False	True	...	False	True	True	True
3999	False	False	False	False	False	False	False	False	False	True	...	False	True	True	True

4000 rows × 38 columns

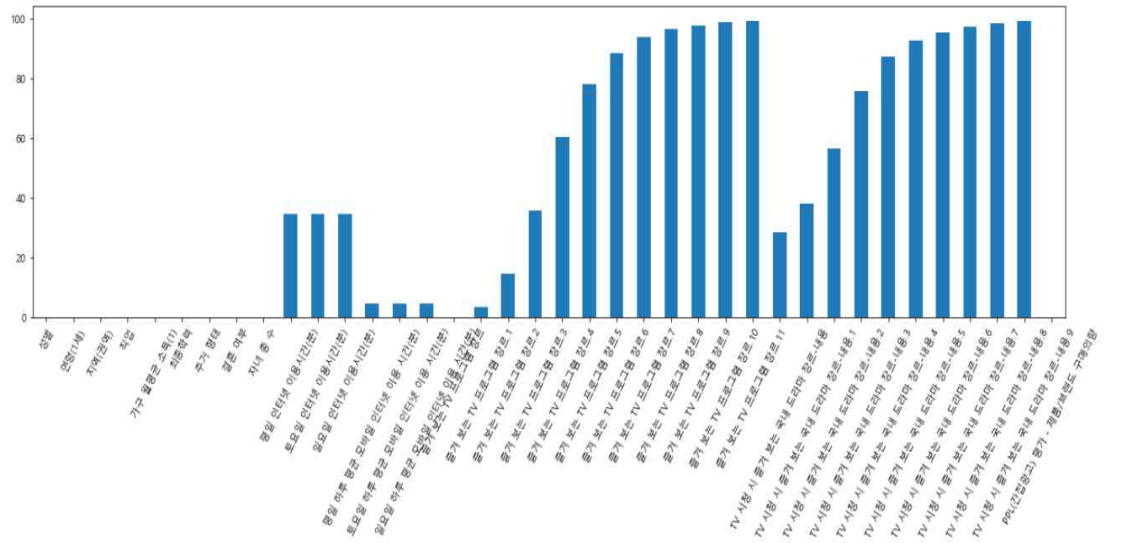
결측치가 있는 필드의 값의 총합을❶ 확인한다. 성별, 연령(1세), 지역(권역) 열은 결측치가 없어 모두 '0', 'TV 시청 시 즐겨 보는 국내 드라마 장르-내용.8'의 결측치는 3,945개이다.

```
df.isnull().sum()                                                    ❶
```

```
성별                                                    0
연령(1세)                                               0
지역(권역)                                              0
직업                                                    0
가구 월평균 소득(1)                                     0
최종학력                                                0
주거 형태                                               0
결혼 여부                                               0
자녀 총 수                                              0
평일 인터넷 이용시간(분)                             1397
토요일 인터넷 이용시간(분)                           1397
일요일 인터넷 이용시간(분)                           1397
~ 중략 ~~
TV 시청 시 즐겨 보는 국내 드라마 장르-내용.7         3892
TV 시청 시 즐겨 보는 국내 드라마 장르-내용.8         3945
TV 시청 시 즐겨 보는 국내 드라마 장르-내용.9         3978
PPL(간접광고) 평가 - 제품/브랜드 구매의향               0
```

결측치 개수를 전체 개수로 나누어 백분율을 막대그래프로 확인한다❶.

```
(df.isnull().sum()/df.shape[0]).plot(kind='bar', rot='60', figsize=(20,5))    ❶
```



'즐겨 보는 TV 프로그램 장르.11' 열의 결측치 개수는 총 3,937개이며❶ 결측치(NaN)를 제외한 4개의 장르 종류와❷ 결측치를 포함한 장르의 종류를❸ 확인한다. 자료의 컬럼 명이 길면 df.columns 명령어를 수행한 후에 cell에서 해당 컬럼 명을 복사하여 사용하면 편리하다.

```
df['즐겨 보는 TV 프로그램 장르.11'].isnull().sum()                    ❶
df['즐겨 보는 TV 프로그램 장르.11'].value_counts()                    ❷
df['즐겨 보는 TV 프로그램 장르.11'].value_counts(dropna=False)        ❸
```

```
3977
16          13
15           4
12           3
13           2
#NULL!       1
Name: 즐겨 보는 TV 프로그램 장르.11, dtype: int64
NaN        3977
16           13
15            4
12            3
13            2
#NULL!        1
Name: 즐겨 보는 TV 프로그램 장르.11, dtype: int64
```

'직업', '성별' 등의 범주형 자료는 value_counts() 명령어를 사용하여 각 유형의 빈도를 확인한다. 총 7가지 직업의 종류의 유형별 빈도와❶ 비중을❷ 확인한다.

```
df['직업'].value_counts()                    ❶
df['직업'].value_counts(normalize=True)      ❷
```

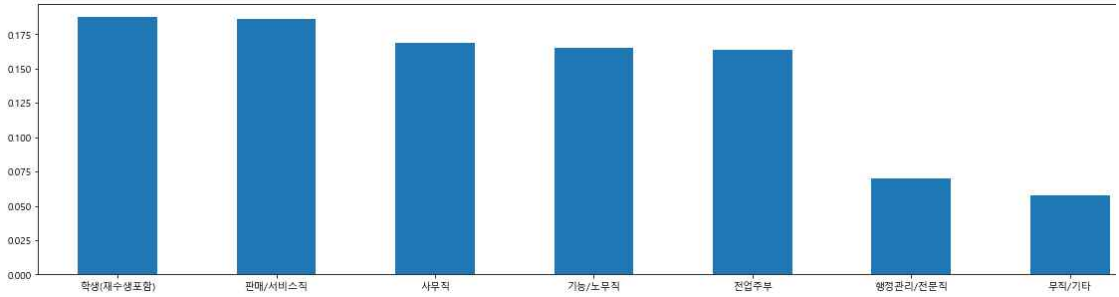
```
6    752                          6    0.18800
3    746                          3    0.18650
2    675                          2    0.16875
4    661                          4    0.16525
5    655                          5    0.16375
1    280                          1    0.07000
7    231                          7    0.05775
Name: 직업, dtype: int64          Name: 직업, dtype: float64
```

'2019년 소비자행태조사 원시데이터_문항보기' 파일을 내려받아 두 번째 A_006의 문항을 확인하면 1:'행정관리/전문직', 2:'사무직', 3:'판매/서비스직', 4:'기능/노무직', 5:'전업주부', 6:'학생(재수생포함)', 7:'무직/기타'의 대응 관계를 알 수 있다. 대응 관계를 사전자료 형태로 저장하여❶ 변경 된 값으로 확인하고❷ X축을 직업명으로 하는 막대그래프를 나타냈다❸. '직업'의 유형별 빈도를 백분율(normalize=True) 막대그래프로❶ 나타낸다. 파이썬 명령어가 긴 경우에는 쉼표 ','에서 줄 바꾸기를 하거나❷ 전체 코드에 괄호를 하여 줄을 바꾸면 된다❸.

```
map_job ={1:'행정관리/전문직', 2:'사무직', 3:'판매/서비스직', 4:'기능/노무직',
          5:'전업주부', 6:'학생(재수생포함)', 7:'무직/기타'}
df['직업'].apply(lambda x: map_job[x] if x in map_job.keys() else x)
(df['직업'].value_counts(normalize=True).rename(index=map_job).
  plot(kind='bar', rot='360', figsize=(20,5)))
```

❶
❷
❸

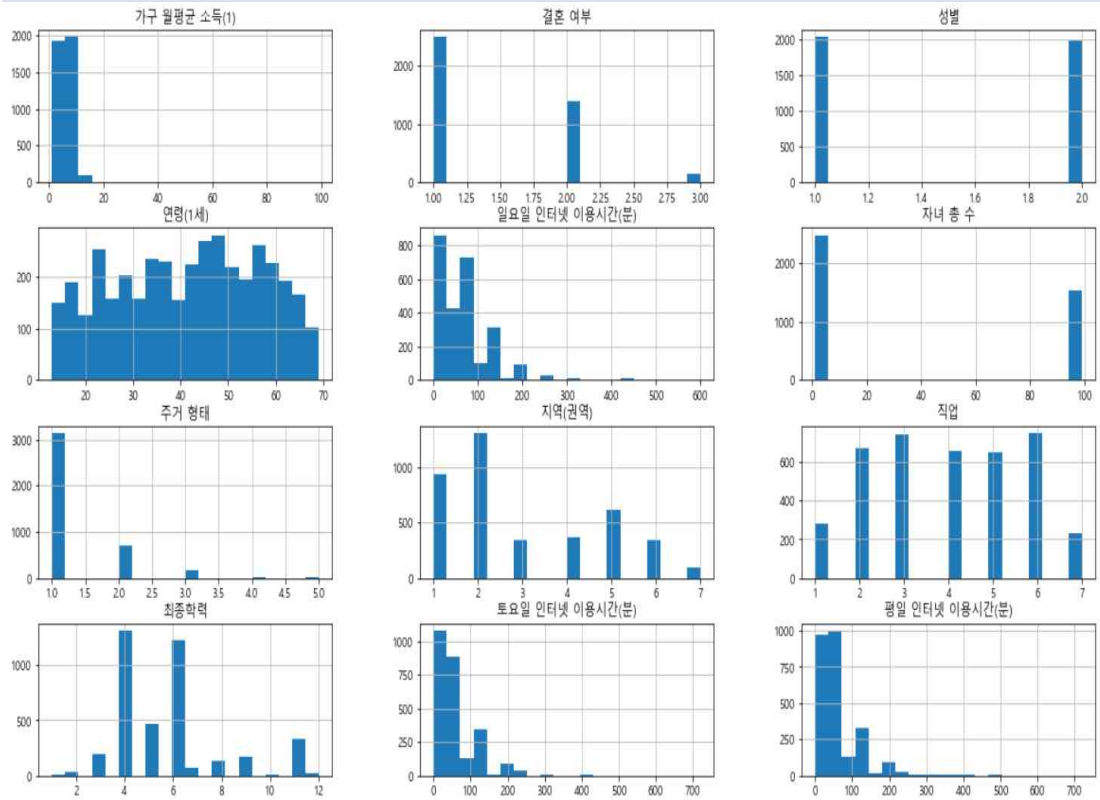
```
0          학생(재수생포함)
1          학생(재수생포함)
           ...
3998         기능/노무직
3999          전업주부
Name: 직업, Length: 4000, dtype: object
```



11번째 컬럼 변수까지 히스토그램을❶ 나타냈다.

```
df[df.columns[0:12]].hist(bins=20, figsize=(20,10))
```

❶



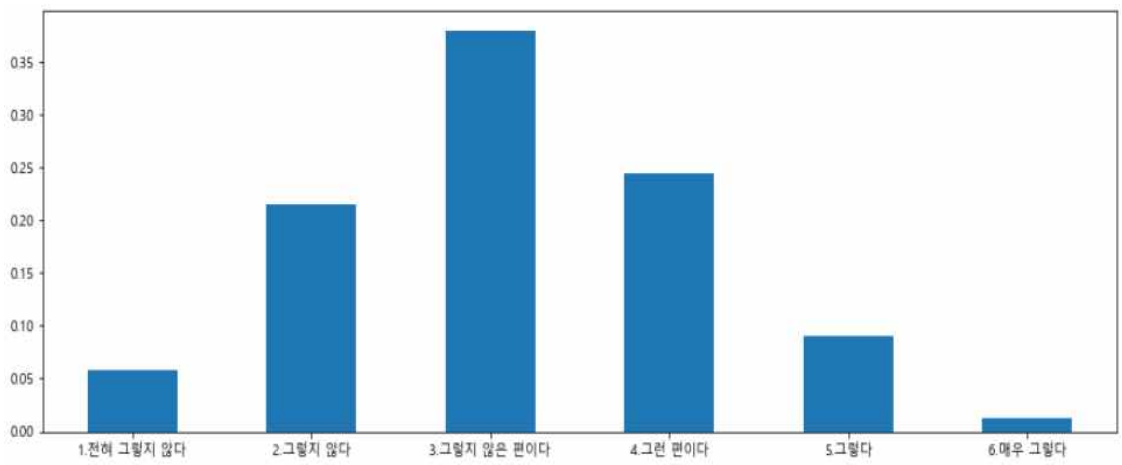
단일 변수를 분석하기 위한 판다스의 명령어는 아래와 같이 다양하다. 직접 명령어를 입력하여 어떤 기능인지 확인하기 바란다.

Tip. 데이터의 컬럼별 기본 속성을 확인하는 pandas 함수 명령어

- df['컬럼명'].index, df['컬럼명'].columns, df['컬럼명'].values
- df['컬럼명'].info(), df['컬럼명'].describe(), df['컬럼명'].isnull(), df['컬럼명'].notnull()
- aggregate 함수 명령어
  df['컬럼명'].sum(), df['컬럼명'].count(), df['컬럼명'].mean(), df['컬럼명'].std(),
  df['컬럼명'].max(), df['컬럼명'].min(), df['컬럼명'].median(),
  df['컬럼명'].quantile([0.25, 0.5, 0.75])

예측하고자 하는 소비자의 구매 경향, 즉 종속변수는 마지막 열 'PPL(간접광고) 평가 - 제품/브랜드 구매의향'이다. 구매의 유형 '1'~'6' 까지 의미는 'MCR2019' 엑셀파일의 두 번째 시트(문항정보)에서 확인할 수 있다. '1'은 '전혀 그렇지 않다', '2'는 '그렇지 않다', '3'은 '그렇지 않은 편이다', 마지막 열을 '구매'라는 열로 새로 만들고❶ '4'는 '그런 편이다', '5'는 '그렇다', '6'은 '매우 그렇다.' 각각 일대일 대응 관계이다❷. '구매'의 유형을 텍스트로 변경하여❸ 구매의 경향 순서로❹ 확인한다❺.

```
df['구매'] = df.iloc[:,-1]                                                    ❶
map_dict={1:'1.전혀 그렇지 않다', 2:'2.그렇지 않다', 3:'3.그렇지 않은 편이다',    ❷
          4:'4.그런 편이다', 5:'5.그렇다', 6:'6.매우 그렇다'}
tf = df['구매'].value_counts(normalize=True).rename(index=map_dict)           ❸
tf.sort_index(inplace=True)                                                   ❹
tf.plot(kind='bar', rot='360', figsize=(20,5))                                ❺
```



2 데이터 인덱싱(Indexing), 필터링(Filtering)

자료에서 특정한 데이터만 골라내는 것을 인덱싱(indexing)이라고 한다. 자료 행 이름(레이블)을 기반으로 하는 loc()와 순서를 나타내는 위치 정수를 기반으로 하는 iloc()로 행과 열을 특정할 수 있다. 관측치 전체 행과 성별부터 결혼 여부 열까지의 레이블을 기준으로 인덱싱하거나❶ 첫 번째 컬럼부터 8번째 컬럼의 위치 정수를 기반으로 인덱싱할 수 있다❷.

```
df.loc[:, '성별' : '결혼 여부']   ❶
df.iloc[:, 0:8]                  ❷
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부
0	1	14	1	6	6	8	3	2
1	1	14	1	6	6	8	1	2
2	1	14	1	6	6	8	2	2
3	1	14	1	6	8	8	1	2
4	1	14	1	6	9	8	1	2
...	...	...	...	...	...	...	...	...
3995	2	58	4	4	1	3	1	3
3996	2	63	4	5	7	4	1	1
3997	2	61	4	4	4	3	2	1
3998	2	67	4	4	2	3	1	3
3999	2	65	4	5	4	4	1	1

4000 rows × 8 columns

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거 형태	결혼 여부
0	1	14	1	6	6	8	3	2
1	1	14	1	6	6	8	1	2
2	1	14	1	6	6	8	2	2
3	1	14	1	6	8	8	1	2
4	1	14	1	6	9	8	1	2
...	...	...	...	...	...	...	...	...
3995	2	58	4	4	1	3	1	3
3996	2	63	4	5	7	4	1	1
3997	2	61	4	4	4	3	2	1
3998	2	67	4	4	2	3	1	3
3999	2	65	4	5	4	4	1	1

4000 rows × 8 columns

자료의 값을 Boolean(참, 거짓)으로 특정 조건을 부여하여 인덱싱한다. 성별 컬럼이 남자인 데이터를 추출하거나❶ 남자 중에서 '성별', '직업', '최종학력', '주거 형태'의 열만❷ 인덱싱하여 필터링할 수 있다.

```
df.loc[df.성별 == 1] # df.loc[df.성별 == 1, :]                    ❶
df.loc[df.성별 == 1, ['성별', '직업', '최종학력', '주거 형태']]    ❷❸
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일 인터넷 이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.4	TV 시청 시 즐겨 보는 국내 드라
0	1	14	1	6	6	8	3	2	99	0.0	...	10.0	13	NaN	
1	1	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	NaN	
2	1	14	1	6	6	8	2	2	99	90.0	...	12.0	14	NaN	
3	1	14	1	6	8	8	1	2	99	NaN	...	NaN	NaN	NaN	
4	1	14	1	6	9	8	1	2	99	0.0	...	10.0	11	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3975	1	53	4	1	6	7	1	1	2	120.0	...	NaN	NaN	NaN	
3976	1	58	4	4	5	4	1	1	1	60.0	...	NaN	NaN	NaN	
3977	1	56	4	7	4	6	1	1	99	NaN	...	NaN	NaN	NaN	
3978	1	61	4	4	4	4	1	1	2	NaN	...	NaN	NaN	NaN	
3979	1	68	4	7	2	6	1	3	99	NaN	...	NaN	NaN	NaN	

2022 rows × 39 columns

	성별	직업	최종학력	주거 형태
0	1	6	8	3
1	1	6	8	1
2	1	6	8	2
3	1	6	8	1
4	1	6	8	1
...	...	...	...	...
3975	1	1	7	1
3976	1	4	4	1
3977	1	7	6	1
3978	1	4	4	1
3979	1	7	6	1

2022 rows × 4 columns

판다스의 loc()는 데이터의 맨 위에서 아래로 행 단위(레코드)로 읽어 내려가기 때문에 열을 특정하여 선택한다. 위치 정수 기반의 iloc()는 loc()와는 다르게 전체 행과 해당하는 열의 인덱스를 동시에 특정하여❸ 데이터를 추출한다.

```
df.iloc[:, [0,3,5,6]]                                                    ❶
```

	성별	직업	최종학력	주거 형태
0	1	6	8	3
1	1	6	8	1
2	1	6	8	2
3	1	6	8	1
4	1	6	8	1
...	...	...	...	...
3995	2	4	3	1
3996	2	5	4	1
3997	2	4	3	2
3998	2	4	3	1
3999	2	5	4	1

4000 rows × 4 columns

성별의 유형 빈도를 살펴보고❶ '1'인(남성) 경우와❷ '1'인(남성) 경우의❸ 열을 확인한다.

```
df.성별.value_counts(dropna=False)                                       ❶
df.loc[df.성별 == 1]['성별']                                             ❷
df.loc[df.성별 ==2]['성별']                                              ❸
```

```
1      2022
2      1978
Name: 성별, dtype: int64
0          1
1          1
2          1
3          1
          ..
3977       1
3978       1
3979       1
Name: 성별, Length: 2022, dtype: int64
473        2
474        2
475        2
476        2
          ..
3997       2
3998       2
3999       2
Name: 성별, Length: 1978, dtype: int64
```

필터링(filtering)은 조건문을 사용하여 자료를 정제하는 작업이다. 연령이 20세 이하인 데이터에서❶ 직업과 최종학력을 인덱싱하고❷ 남자인 경우만❸ 추출한다.

```
df['연령(1세)'] <= 20                                                          ❶
df[df['연령(1세)'] <= 20][['직업', '최종학력']].head()                          ❷
df[(df['연령(1세)'] <= 20) & (df['성별'] == 1)][['직업', '최종학력']].head()    ❸
```

```
0        True
1        True
2        True
3        True
4        True
        ...
3995    False
3996    False
3997    False
3998    False
3999    False
Name: 연령(1세),
```

	직업	최종학력
0	6	8
1	6	8
2	6	8
3	6	8
4	6	8

	직업	최종학력
0	6	8
1	6	8
2	6	8
3	6	8
4	6	8

연령이 35세 이하이거나(’|‘) 남자인 데이터는 2,737개❶ 여자인 데이터는 1,263개이다❷.

```
df[(df['연령(1세)'] <= 35) | (df['성별'] == 1)].shape                           ❶
df[~((df['연령(1세)'] <= 35) | (df['성별'] == 1))].shape                        ❷
```

```
(2737, 39)
(1263, 39)
```

3 데이터 컬럼 간 관계 분석(groupby, pivot)

성별을 '남자'와 '여자'로 변경하는 가장 간단한 명령어는 map() 함수이다❶. 대응할 경우의 수가 많으면 사전형으로 정의한 후에 사용한다❷.

```
df['성별'].map({1:'남자', 2:'여자'})
df['성별'].map({1:'남자', 2:'여자'}).value_counts()
```
❶

```
mapping = {1:' 남자 ', 2:' 여자 '}
df['성별'].map(mapping).value_counts()
```
❷

```
0       남자
1       남자
       ..
3998    여자
3999    여자

남자    2022
여자    1978
```

또 다른 방법은 apply()를 사용하는 것이다. 데이터의 성별 변수를 입력받아 한 줄씩 읽어 '1'이면 '남자' '2'이면 '여자'로❶ 변경한다. 반복문과 리스트를 사용할 수도 있으나 계산속도를 줄이기 위해 apply() 함수를 사용하여 각 열(시리즈)에 모두 적용한다. 조건문이 여러개일 경우에는 'if 조건1 else ~ if 조건2 else ~ if 조건3 else ~'를 입력하면 된다.

```
df.head(2)
df['성별'] = df['성별'].apply(lambda x: '남자' if x == 1 else '여자')
# df['성별'].map(lambda x: '남자' if x == 1 else '여자')
df.head(2)
```
❶

	성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일인터넷이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.4	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.5
0	0	14	1	6	6	8	3	2	99	0.0	...	10.0	13	NaN	NaN
1	0	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	NaN	NaN

2 rows × 39 columns

	성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일인터넷이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.4	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.5
0	여자	14	1	6	6	8	3	2	99	0.0	...	10.0	13	NaN	NaN
1	여자	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	NaN	NaN

2 rows × 39 columns

범주형 변수를 유형별로 그룹화하여 분석하는 함수가 groupby()이다. 남녀 유형별 자료의 개수는 2,022와 1,978이며❶ 성별을 기준으로 그룹화하여 데이터의 수를❷ 확인한다.

```
df.성별.value_counts()                                    ❶
df.groupby('성별').count()                                ❷
```

```
남자    2022
여자    1978
Name: 성별, dtype: int64
```

성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.6	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.7	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.8	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.9	PPL(간접광고) 평가 - 제품/브랜드 구매의향	구매
남자	2022	2022	2022	2022	2022	2022	2022	2022	56	32	13	6	2022	2022
여자	1978	1978	1978	1978	1978	1978	1978	1978	119	76	42	16	1978	1978

2 rows × 38 columns

남녀별로 '평일 인터넷 이용시간(분)'의 평균을 각각 구하면 남자가 높다❶❷. 일반적인 groupby() 함수의 형태는 ❶번 형태이다.

```
df.groupby('성별')['평일 인터넷 이용시간(분)'].mean()                ❶
df['평일 인터넷 이용시간(분)'].groupby(df['성별']).mean()            ❷
```

```
성별
남자    70.737422
여자    58.602431
Name: 평일 인터넷 이용시간(분), dtype: float64
성별
남자    70.737422
여자    58.602431
Name: 평일 인터넷 이용시간(분), dtype: float64
```

분석대상 시리즈나 데이터프레임에 groupby 함수(메서드)를 호출하여 자료를 그룹화한 후 그룹 객체에 대해 그룹 연산을 각각 수행한다.

groupby() 함수의 결과를 데이터프레임의 시리즈 형태로 변경하거나❶❷ 성별을 인덱스로 (reset_index) 변경하여 데이터프레임을❸ 만들 수 있다.

```
df.groupby('성별')['평일 인터넷 이용시간(분)'].mean().to_frame()                    ❶
pd.DataFrame(df.groupby('성별')['평일 인터넷 이용시간(분)'].mean())                  ❷
pd.DataFrame(df.groupby('성별')['평일 인터넷 이용시간(분)'].mean()).reset_index()    ❸
```

성별	평일 인터넷 이용시간(분)
남자	70.737422
여자	58.602431

성별	평일 인터넷 이용시간(분)
남자	70.737422
여자	58.602431

	성별	평일 인터넷 이용시간(분)
0	남자	70.737422
1	여자	58.602431

성별, 직업별 평일과 토요일 인터넷 이용시간의 평균에 차이가 있는지❶ 확인한다.

```
df.groupby(['성별', '직업'])['평일 인터넷 이용시간(분)',                              ❶
                          '토요일 인터넷 이용시간(분)'].mean()
```

성별	직업	평일 인터넷 이용시간(분)	토요일 인터넷 이용시간(분)
남자	1	73.904494	45.758427
	2	80.313390	59.059829
	3	65.957447	56.978723
	4	51.940000	57.340000
	6	77.427746	93.106936
	7	66.153846	62.527473
여자	1	71.200000	52.800000
	2	64.487069	48.711207
	3	52.406417	51.470588
	4	47.500000	42.500000
	5	49.212544	36.689895
	6	65.736301	81.866438
	7	68.636364	69.318182

코딜을 하다가 적당한 시점에 ‘df = pd.read_pickle('../data/파일명.pkl')’로 저장하여 두면 나중에 같은 코드를 실행하는 시간을 절약할 수 있다.

반복문을 통해 df.groupby('성별')['직업', '구매']의 데이터프레임을 읽어와❶ 남자와 여자로 구분된 데이터프레임을❷❸ 각각 확인할 수 있다. 전체 데이터를 남녀 성별로 구분하기만 하고 '직업', '구매' 기준에 대해 관측개수, 평균, 합계 등 어떠한 조작도 하지 않았기 때문에 성별로만 구분된 데이터프레임을 보여주고 있다.

```
for group_id, sub_group in df.groupby('성별')['직업', '구매'] :                ❶
    group_id                                                                  ❷
    sub_group.shape                                                           ❸
    sub_group.head(2)                                                         ❹
```

'남자'

(2022, 39)

	성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일인터넷이용시간(분)	...	TV시청시즐겨보는국내드라마장르-내용.2	TV시청시즐겨보는국내드라마장르-내용.3	TV시청시즐겨보는국내드라마장르-내용.4	TV시청시즐겨보는국내드라마장르-내용.5	TV시청시즐겨보는국내드라마장르-내용.6	TV시청시즐
0	남자	14	1	6	6	8	3	2	99	0.0	...	10.0	13	NaN	NaN	NaN	Na
1	남자	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	NaN	NaN	NaN	Na

2 rows × 39 columns

'여자'

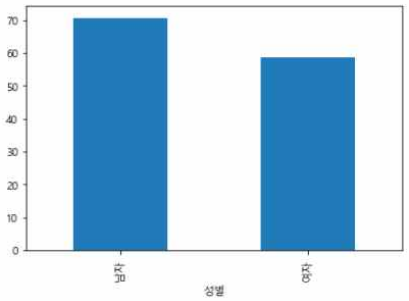
(1978, 39)

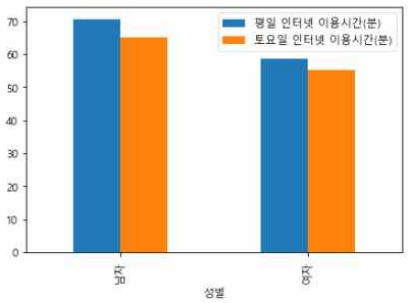
	성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일인터넷이용시간(분)	...	TV시청시즐겨보는국내드라마장르-내용.2	TV시청시즐겨보는국내드라마장르-내용.3	TV시청시즐겨보는국내드라마장르-내용.4	TV시청시즐겨보는국내드라마장르-내용.5	TV시청시즐겨보는국내드라마장르-내용.6	TV시청시즐
473	여자	14	1	6	5	8	1	2	99	60.0	...	NaN	NaN	NaN	NaN	NaN	N
474	여자	14	1	6	6	8	1	2	99	30.0	...	NaN	NaN	NaN	NaN	NaN	N

2 rows × 39 columns

'성별'을 기준으로 '평일 인터넷 이용시간(분)'의 평균과❶ '평일 인터넷 이용시간(분)', '토요일 인터넷 이용시간(분)'의 평균을❷ 확인하면 남자가 여자보다 인터넷이용시간이 더 많다.

```
df.groupby('성별')['평일 인터넷 이용시간(분)'].mean().plot(kind ='bar')                    ❶
df.groupby('성별')['평일 인터넷 이용시간(분)',
                   '토요일 인터넷 이용시간(분)'].mean().plot(kind ='bar')                  ❷
```





성별과 결혼 여부를 기준으로 '평일인터넷 이용평균시간'과❶ 평균연령을❷ 확인한다.

```
df.groupby(['성별', '결혼 여부'])['평일 인터넷 이용시간(분)'].mean()                        ❶
df.groupby(['성별', '결혼 여부'])['연령(1세)', '평일 인터넷 이용시간(분)'].mean()           ❷
```

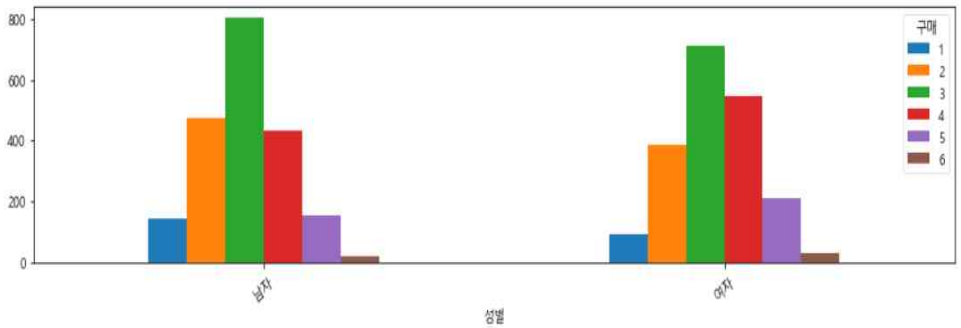
```
성별  결혼 여부
남자  1        67.060766
      2        74.806835
      3        72.857143
여자  1        54.132219
      2        64.836100
      3        53.333333
Name: 평일 인터넷 이용시간(분), dtype: float64
```

		연령(1세)	평일 인터넷 이용시간(분)
성별	결혼 여부		
남자	1	50.338160	67.060766
	2	26.323155	74.806835
	3	56.887097	72.857143
여자	1	48.864329	54.132219
	2	23.400673	64.836100
	3	59.083333	53.333333

'구매' 설문 응답 항목이 5번과 6번, 즉 구매를 적극적으로 하려는 남성은 총 173명(151명+21명)이다. 여성은 총 239명(208명 + 31명)으로 여성이 남성보다 구매의향이 상대적으로 크다❶. '구매' 컬럼의 행 인덱스를 열인덱스로 변경하여❷ 히스토그램으로❸ 확인한다.

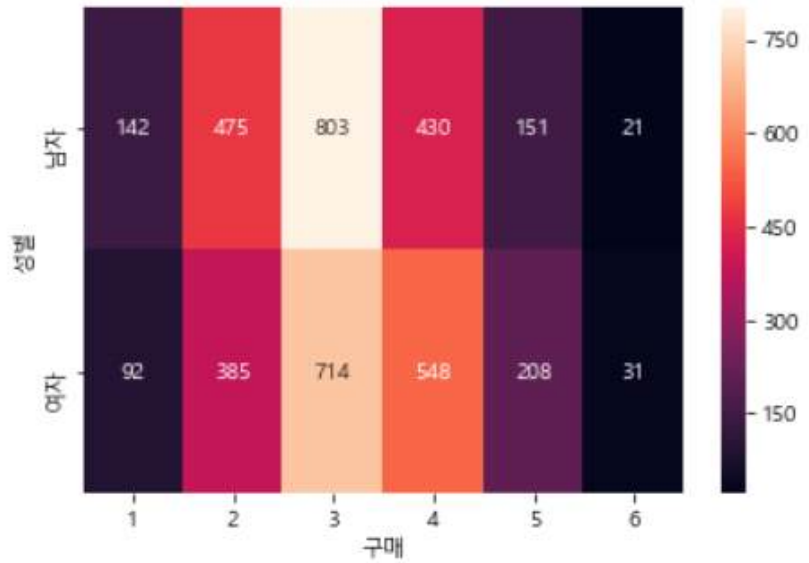
```
df.groupby('성별')['구매'].value_counts()                                    ❶
df.groupby('성별')['구매'].value_counts().unstack() # 행 인덱스를 열 인덱스로 변경   ❷
df.groupby('성별')['구매'].value_counts().unstack().plot.bar(rot='35', figsize=(15,3))  ❸
```

구매	1	2	3	4	5	6
성별						
남자	142	475	803	430	151	21
여자	92	385	714	548	208	31



seaborn모듈을 불러와❶ sns.heatmap() 함수로 성별 구매 경향 빈도수(annot=True)를❷ 구해 정수 형태(fmt='d')로 확인한다. Heatmap은 다양한 값을 갖는 숫자 데이터를 분포 형태와 같이 색상을 이용한 시각화에 적합하다.

```
import seaborn as sns                                                        ❶
ax = sns.heatmap(df.groupby('성별')['구매'].value_counts().unstack(),          ❷
                 fmt='d', annot=True)
```



피봇팅을 사용하여 성별과 구매 경향에 따른 최종학력의 경향을❶ 나타낸다. 성별과 구매 경향의 두 변수를 교차테이블 함수를 사용하여도 같은 결과이다❷. aggfunc의 count대신 mean, std, median, max 등을 사용할 수 있다.

```
df.pivot_table(values=['최종학력'], index=['성별'], columns=['구매'], aggfunc='count')  ❶
pd.crosstab(df['성별'], df['구매'])                                                    ❷
```

	최종학력					
구매	1	2	3	4	5	6
성별						
남자	142	475	803	430	151	21
여자	92	385	714	548	208	31

구매	1	2	3	4	5	6
성별						
남자	142	475	803	430	151	21
여자	92	385	714	548	208	31

성별과 구매 경향 피봇테이블의❶ 행과 열 합계를 구한다❷❸❹.

```
tf = df.pivot_table(values=['최종학력'], index=['성별'], columns=['구매'], aggfunc='count')  ❶
tf.loc['합계_열'] = tf.sum()                                                                 ❷
tf['합계_행'] = tf.sum(axis=1)                                                               ❸
tf                                                                                           ❹
```

	최종학력						합계_행
구매	1	2	3	4	5	6	
성별							
남자	142	475	803	430	151	21	2022
여자	92	385	714	548	208	31	1978
합계_열	234	860	1517	978	359	52	4000

엑셀을 사용하는 이유는 단연 피봇팅 기능일 것이다. 판다스의 groupby 기능과 reset_index을 사용하면 다양한 기능을 구현할 수 있다. 성별을 기준으로 구매 유형을 그룹화하여❶ 인덱스(reset_index)한 후❷ 다시 인덱스(set_index)하였다❸.

```
df.groupby('성별')['구매'].describe()                                    ❶
df.groupby('성별')['구매'].describe().reset_index()                      ❷
df.groupby('성별')['구매'].describe().reset_index().set_index('성별')    ❸
```

성별	count	mean	std	min	25%	50%	75%	max
남자	2022.0	3.017804	1.05873	1.0	2.0	3.0	4.0	6.0
여자	1978.0	3.246714	1.07663	1.0	3.0	3.0	4.0	6.0

	성별	count	mean	std	min	25%	50%	75%	max
0	남자	2022.0	3.017804	1.05873	1.0	2.0	3.0	4.0	6.0
1	여자	1978.0	3.246714	1.07663	1.0	3.0	3.0	4.0	6.0

성별	count	mean	std	min	25%	50%	75%	max
남자	2022.0	3.017804	1.05873	1.0	2.0	3.0	4.0	6.0
여자	1978.0	3.246714	1.07663	1.0	3.0	3.0	4.0	6.0

groupby은 내부적으로 데이터의 분할과 결합을 차례대로 하고 있다. 성별을 기준으로 자료를 나누어❶ 각 자료의 크기와 형태를❷❸ 확인하고 평균, 분산을❹ 결합하는 과정을 확인한다. 여성의 구매 경향 평균이 3.26으로 남성의 3.01보다 높다.

```
for (method, group) in df.groupby('성별'):                              ❶
    print("{0:30s} shape={1}".format(method, group.shape))              ❷
    group.head(2)                                                       ❸
    group['구매'].mean()                                                ❹
```

```
남자                             shape=(2022, 39)
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일 인터넷 이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라
0	남자	14	1	6	6	8	3	2	99	0.0	...	10.0	13	
1	남자	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	

```
2 rows × 39 columns
3.0178041543026706
여자                             shape=(1978, 39)
```

	성별	연령(1세)	지역(권역)	직업	가구 월평균 소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일 인터넷 이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드
473	여자	14	1	6	5	8	1	2	99	60.0	...	NaN	NaN	
474	여자	14	1	6	6	8	1	2	99	30.0	...	NaN	NaN	

```
2 rows × 39 columns
3.2467138523761374
```

4 데이터 결합 (pandas concate, merge)

두 개 이상의 자료를 하나로 합치는 데이터 병합(merge)이나 연결(concatenate)에 대해 살펴보자. 우선 HTML 형태의 화면 출력을 위한 모듈을 읽어오고❶ 결과를 행으로 출력하게 하는 명령문을❷❸ 실행한다.

```
from IPython.display import display, HTML                    ❶
CSS = """.output {flex-direction: row;}"""                   ❷
HTML('<style>{}</style>'.format(CSS))                        ❸
```

관측치(obs)와 컬럼(var)을 인자로 입력받아 자료를 생성하는 함수 make_df()를 만들어❶❷❸ 3개의 데이터프레임을❹ 만들어 확인한다.

```
def make_df(var, obs):                                       ❶
    data = {c: [str(c) + str(i) for i in obs]                ❷
            for c in var}                                    ❸
    return pd.DataFrame(data, obs)                           ❹

make_df('ABCD', [0,1,2,3]); make_df('012', ['A','B']); make_df('ACEGIJ', [1,3,5,7,9])   ❺
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	0	1	2
A	0A	1A	2A
B	0B	1B	2B

	A	C	E	G	I	J
1	A1	C1	E1	G1	I1	J1
3	A3	C3	E3	G3	I3	J3
5	A5	C5	E5	G5	I5	J5
7	A7	C7	E7	G7	I7	J7
9	A9	C9	E9	G9	I9	J9

df1과❶ df2의❷ 자료를 생성한 후에 df1과 df2를 상하 수직으로 연결하는 자료를 만들어 보면 각각의 자료에 없는 컬럼 값에는 'NaN'이 기록된다❸.

```
df1 = make_df('ABCD', [0,1,2,3])                    ❶
df2 = make_df('DEFG', [1,2,3,4])                    ❷
df1; df2; pd.concat([df1, df2])                     ❸
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

	A	B	C	D	E	F	G
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
1	NaN	NaN	NaN	D1	E1	F1	G1
2	NaN	NaN	NaN	D2	E2	F2	G2
3	NaN	NaN	NaN	D3	E3	F3	G3
4	NaN	NaN	NaN	D4	E4	F4	G4

두 개의 결합된(concat) 자료를 append() 함수를❶ 사용하여 똑같은 자료를 만든다.

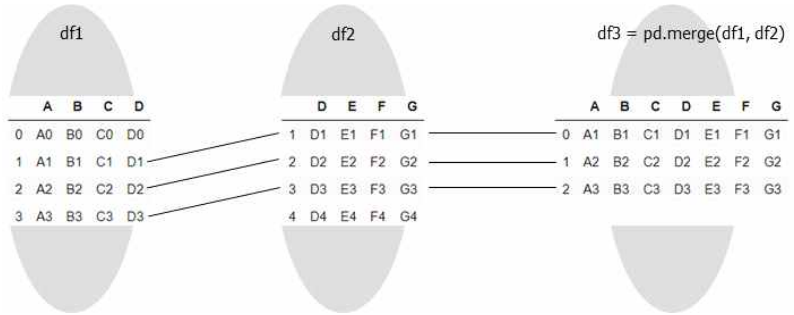
```
df1; df2; df1.append(df2)                           ❶
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

	A	B	C	D	E	F	G
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
1	NaN	NaN	NaN	D1	E1	F1	G1
2	NaN	NaN	NaN	D2	E2	F2	G2
3	NaN	NaN	NaN	D3	E3	F3	G3
4	NaN	NaN	NaN	D4	E4	F4	G4

자료 병합(merge)의 종류는 두 자료의 대응 관계에 따라 일대일(1:1), 일대다(1:M), M:1), 다대다(M:M)로 구분한다. df1과 df2는 'D' 컬럼을 공유하고 있고 'D' 컬럼 기준으로 보면 일대일 대응(1:1) 관계이다. df1과 df2를 pd.merge(df1, df2)로 병합하면❶ 일대일(1:1) 관계로 각각의 컬럼을 공유한다❷.



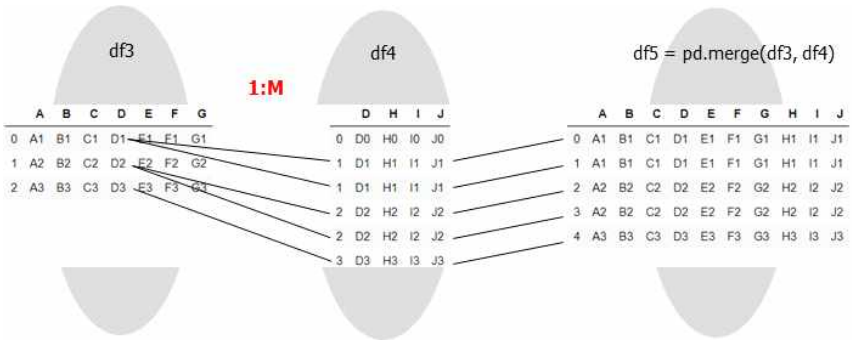

```
df3 = pd.merge(df1, df2)                                    ❶
df1; df2; df3                                               ❷
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

df3과 df4는 'D' 컬럼을 공유하고 있고 'D' 컬럼 기준으로 보면 일대다(1:M) 관계이다. df3과 df4를 pd.merge(df3, df4)로 병합하면❶ 일대일(1:M) 관계로 각각의 컬럼을 공유한다❷.



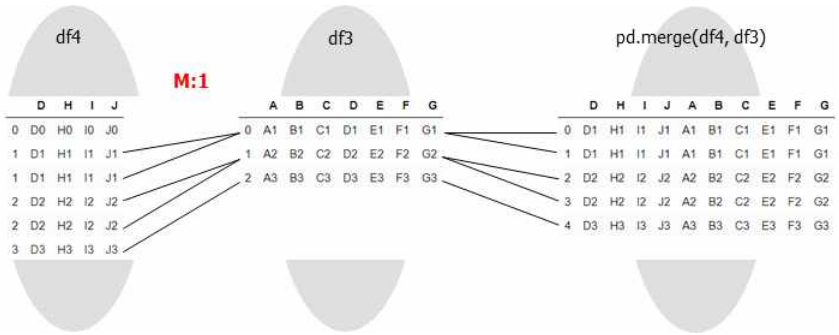
```
df4 = make_df('DHIJ', [0,1,1,2,2,3])                    ❶
df5 = pd.merge(df3, df4)                                ❷
df3; df4; df5                                           ❸
```

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

이젠 df4와 df3의 순서를 바꾸면 'D' 컬럼 기준으로 다대일(M:1) 대응 관계이다. df4과 df3를 pd.merge(df4, df3)로 병합하면❶ 다대일(M:1) 관계로 각각의 컬럼을 공유한다.



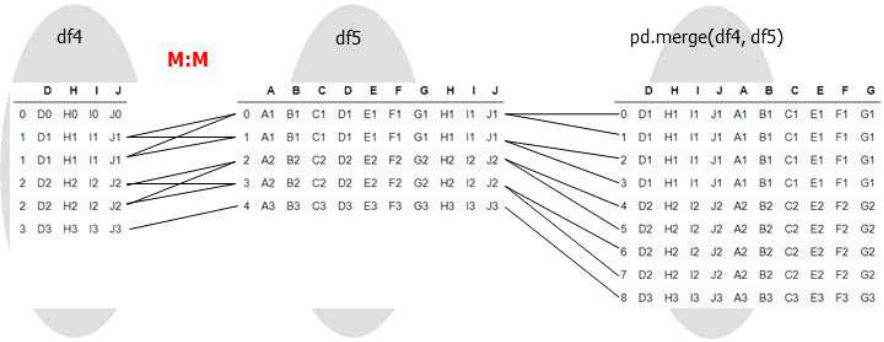

```
df4; df3; pd.merge(df4, df3)                            ❶
```

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

	D	H	I	J	A	B	C	E	F	G
0	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
3	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
4	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4와 df5는 'D' 컬럼을 공유하고 있고 'D' 컬럼 기준으로 보면 다대다(M:M) 관계이다. df4과 df5를 pd.merge(df4, df5)로 병합하면❶ 일대일(M:M) 관계로 각각의 컬럼을 공유한다.



df4; df5; pd.merge(df4, df5) ❶

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

	D	H	I	J	A	B	C	E	F	G
0	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
3	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
4	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
5	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
6	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
7	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
8	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4를 기준으로(how='left') df5와 결합하게 되면 df4의 'D' 열의 모든 값을 보여주게 된다. df4의 'D 열에는 D0에 해당하는 행이 새로 생성된다.

df4; df5; pd.merge(df4, df5, how = 'left') ❶

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

	D	H	I	J	A	B	C	E	F	G
0	D0	H0	I0	J0	NaN	NaN	NaN	NaN	NaN	NaN
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
3	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
4	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
5	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
6	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
7	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
8	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
9	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4와 df6는❶ ‘D’와 ‘H’ 컬럼을 공유하고 있는 다대다(M:M) 관계이다. df4과 df6를 ‘D’와 ‘H’ 컬럼을 기준으로(on =['D', 'H']), 병합하면 공통된 ‘I’ 컬럼을 갖게 되는데 접미사를 (suffixes=('_left', '_right')) 붙여 ‘I’ 컬럼을 각각 공유한다❷.

```
df6= df5[['A', 'B', 'D', 'H', 'I']]                                          ❶
df4; df6; pd.merge(df4, df6, on =['D', 'H'], suffixes=('_left', '_right'))   ❷
```

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	D	H	I
0	A1	B1	D1	H1	I1
1	A1	B1	D1	H1	I1
2	A2	B2	D2	H2	I2
3	A2	B2	D2	H2	I2
4	A3	B3	D3	H3	I3

	D	H	I_left	J	A	B	I_right
0	D1	H1	I1	J1	A1	B1	I1
1	D1	H1	I1	J1	A1	B1	I1
2	D1	H1	I1	J1	A1	B1	I1
3	D1	H1	I1	J1	A1	B1	I1
4	D2	H2	I2	J2	A2	B2	I2
5	D2	H2	I2	J2	A2	B2	I2
6	D2	H2	I2	J2	A2	B2	I2
7	D2	H2	I2	J2	A2	B2	I2
8	D3	H3	I3	J3	A3	B3	I3

Pandas와 Numpy에 대한 자세한 설명은 이 책의 마지막에 있는 Refernce의 참고문헌이나 관련 사이트를 참고하기를 바란다.

4장 데이터 전처리

4.1 결측치(missing value) 대체

4.2 이상치(outliers) 처리

4.3 왜도(Skewness) 처리

4.4 파생변수

4장 데이터 전처리

데이터 전처리 과정은 데이터의 기본 구성 요소인 행과 열(레코드와 컬럼)을 기반으로 이들의 값(필드)을 조작하는 것이다. 자료를 확보하여도 결측치와 이상치가 많은 경우가 있다. 분석과 예측을 하려면 데이터 전처리(Data preprocessing)가 필요하다. 예측 결과에 직접적인 영향을 미치는 중요한 과정이기 때문에, 반드시 예측 단계 전에 이루어져야 한다.

예측 결과가 좋지 않으면 데이터를 재수집하고 다시 전처리해야 하는 반복 작업 과정이다. 업무 프로세스의 특성을 반영하여 자료의 구조와 내용을 충분히 이해하고 수행하도록 한다.

결측치 대체, 이상치 처리 등의 데이터전처리 과정을 수행한다. 관련 모듈을 불러온다❶.

```
from IPython.core.interactiveshell import InteractiveShell                    ❶
InteractiveShell.ast_node_interactivity = 'all'
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.family']='Malgun Gothic'
import warnings
warnings.filterwarnings('ignore')
```

저장한 자료를 불러와❶ '구매의향' 컬럼을 '구매' 컬럼으로 추가하고❷ 확인한다❸❹.

```
df = pd.read_pickle('../data/MCR2019.pkl') # 3장1절의 자료                    ❶
df['구매'] = df.iloc[:,-1]                                                    ❷
df.shape                                                                      ❸
df.head()                                                                     ❹
```

```
(4000, 39)
```

	성별	연령(1세)	지역(권역)	직업	가구월평균소득(1)	최종학력	주거형태	결혼여부	자녀총수	평일인터넷이용시간(분)	...	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.2	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.3	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.4	TV 시청 시 즐겨 보는 국내 드라마 장르-내용.5	
0	1	14	1	6	6	8	3	2	99	0.0	...	10.0	13	NaN	NaN	
1	1	14	1	6	6	8	1	2	99	30.0	...	11.0	NaN	NaN	NaN	
2	1	14	1	6	6	8	2	2	99	90.0	...	12.0	14	NaN	NaN	
3	1	14	1	6	8	8	1	2	99	NaN	...	NaN	NaN	NaN	NaN	
4	1	14	1	6	9	8	1	2	99	0.0	...	10.0	11	NaN	NaN	

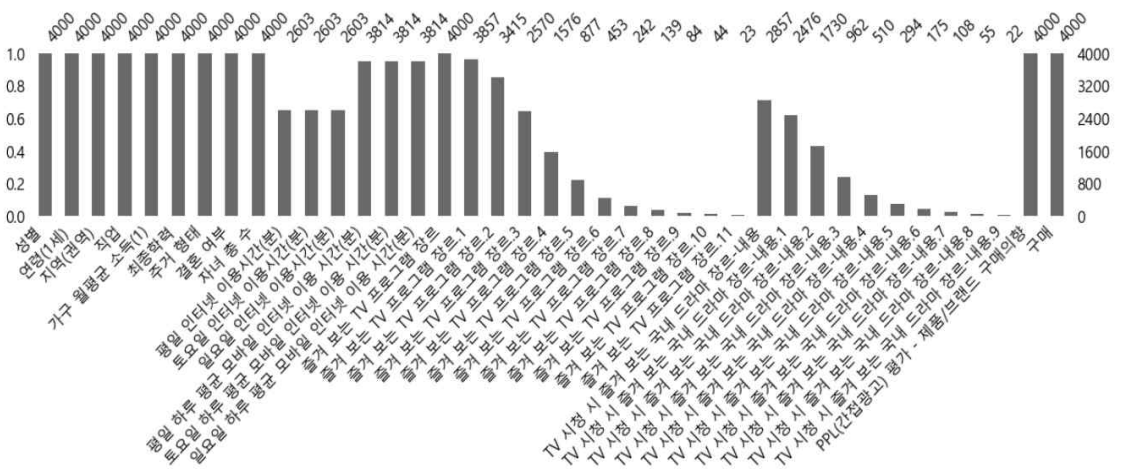
5 rows × 39 columns

4.1 결측치(missing value) 대체

1 결측치 비율이 40% 이상인 변수 제거

데이터 전처리 과정에서 가장 중요한 결측치(누락 데이터) 처리방안을 살펴보자. 시각화 라이브러리를 불러와❶ 막대그래프의 형태로 결측치 비율을❷ 확인한다. 세로축은 변수별로 결측치 비율과 관측치의 개수를 나타낸다. '성별'은 결측치가 없이 자료가 4,000개, 따라서 관측치 비율 막대가 '1.0'을 표시한다. 아니콘다 프롬프트에서 'conda install missingno'를 입력하여 missingno 모듈을 설치한다. 윈도 명령어 프롬프트에서 'pip install missingno'를 실행하거나 주피터 내에서 '!pip install missingno'를 직접 입력해도 된다.

```
import missingno as msno # !pip install missingno                    ❶
msno.bar(df, figsize=(20,3)) # msno.matrix(df, figsize=(20,3))       ❷
```



데이터프레임의 자료형태❶, 비결측치 수❷, 결측치 수❸ 및 고유값❹ 등을 나타내는 사용자 정의함수 summary()를 만들어 관련 정보를 한눈에 확인한다. '평일인터넷이용시간'의 결측치는 1,397개, 고유 값은 35개이다. 고유값이 '1'인 열은 정보가 없어 예측 단계에 제거한다.

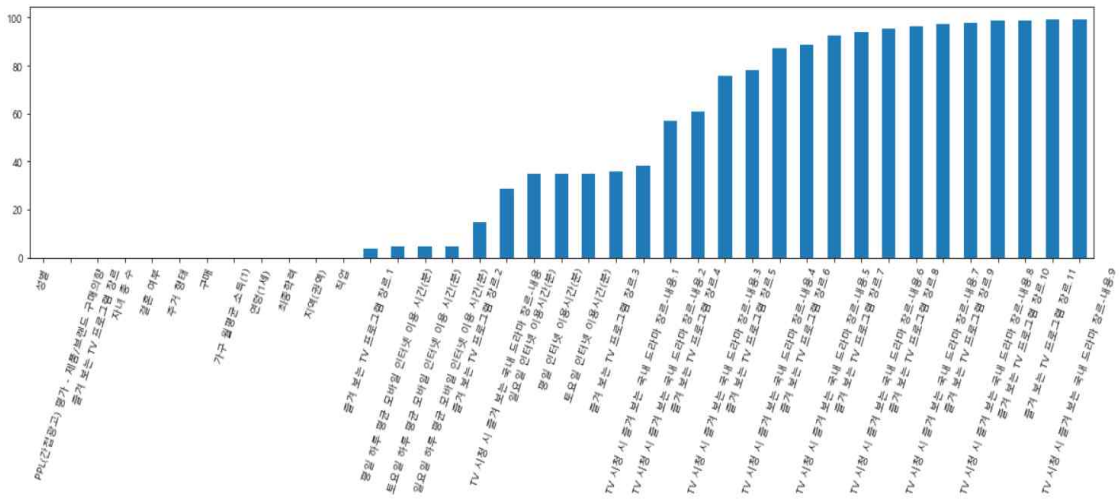
```
def summary(df) :
    tf = pd.DataFrame({'데이터형태(dtypes)' : df.dtypes,                    ❶
                       '비 결측치 수(notnull': df.notnull().sum(),          ❷
                       '결측치 수(null)' : df.isnull().sum(),               ❸
                       '고유값 수(nunique)' : df.nunique()})                ❹
    return tf
summary(df)
```

	데이터형태(dtypes)	비 결측치 수(notnull	결측치 수(null)	고유값 수(nunique)
성별	object	4000	0	2
연령(1세)	int64	4000	0	57
지역(권역)	int64	4000	0	7
직업	int64	4000	0	7
가구 월평균 소득(1)	int64	4000	0	12
최종학력	int64	4000	0	12
주거 형태	int64	4000	0	5
결혼 여부	int64	4000	0	3
자녀 총 수	int64	4000	0	4
평일 인터넷 이용시간(분)	float64	2603	1397	37
토요일 인터넷 이용시간(분)	float64	2603	1397	35

.....(생략)

결측치 비율을❶ 구해 막대 그래프로❷ 확인한다.

```
null_percent = df.isnull().mean()*100                                       ❶
null_percent.sort_values().plot.bar(figsize=(20,4), rot='70')               ❷
```



결측치 비율이 60%가 넘는 변수를❷ 제거하면❸ 39개에서❶ 24개로❹ 감소한다.

```
print('결측치 처리 전 df shape : ', df.shape)                                    ❶
null_filtered = null_percent[null_percent > 60].index                            ❷
df = df.drop(null_filtered, axis =1) # df.drop(missing_col, axis =1, inplace=True)  ❸
print('결측치 처리 후 df shape : ', df.shape)                                    ❹
```

```
결측치 처리 전 df shape :  (4000, 39)
결측치 처리 후 df shape :  (4000, 24)
```

열의 이름을 확인한다❶.

```
df.columns                                                                       ❶
```

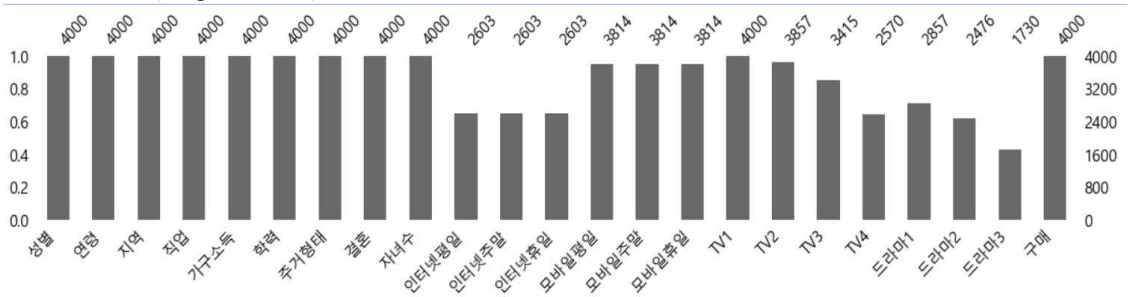
```
Index(['성별', '연령(1세)', '지역(권역)', '직업', '가구 월평균 소득(1)', '최종학력', '주거 형태',
       '결혼 여부', '자녀 총 수', '평일 인터넷 이용시간(분)', '토요일 인터넷 이용시간(분)',
       '일요일 인터넷 이용시간(분)', '평일 하루 평균 모바일 인터넷 이용 시간(분)',
       '토요일 하루 평균 모바일 인터넷 이용 시간(분)', '일요일 하루  인터넷 이용 시간(분)',
       '즐겨 보는 TV 프로그램 장르', '즐겨 보는 TV 프로그램 장르.1', '즐겨 보는  장르.2',
       '즐겨 보는 TV 프로그램 장르.3', 'TV 시청 시 즐겨 보는 국내 드라마 장르-내용',
       'TV 시청 시 즐겨 보는 국내 드라마 장르-내용.1', 'TV 시청 시 즐겨 장르-내용.2',
       'PPL(간접광고) 평가 - 제품/브랜드 구매의향', '구매'],
      dtype='object')
```

새로운 열의 이름을❶ 만들어 데이터프레임에 적용한다❷. '구매의향' 열은 '구매' 열과 내용이 중복되어 제거한다❸.

```
col_name = ['성별', '연령', '지역', '직업', '가구소득', '학력', '주거형태', '결혼', '자녀수',          ❶
            '인터넷평일', '인터넷주말', '인터넷휴일', '모바일평일', '모바일주말', '모바일휴일',
            'TV1', 'TV2', 'TV3', 'TV4', '드라마1', '드라마2', '드라마3', '구매의향', '구매']
df.columns = col_name                                                            ❷
del df['구매의향']                                                               ❸
```

결측치를 확인한다❶.

```
msno.bar(df, figsize=(20,3))                                                     ❶
```



2 자료 형태

데이터의 종류는 범주형(Categorical), 수치형(Numerical), 문자형(String type)으로 구분하며 단순하게 자료의 숫자에 의미가 있으면 '수치형', 없으면 범주형이다.

자료형태			예시	
범주형	명목형 (Nominal)	어떤 대상에 숫자나 기호를 부여하여 구분하며 크기나 순서가 의미 없음	성별, 지역, 직업, 결혼,	범주형
	서열형 (Ordinal)	기준에 따라 대상을 서열화하여 숫자나 기호를 부여하여 순서는 의미가 있으나 크기는 의미 없음	가구소득, 학력, 주거형태	수치형
수치형	연속형 (Continuous)	연속형 수치 자체가 실제적인 의미를 표현	*인터넷/모바일 사용시간(분) 등	
	이산형 (Discrete)	이산형 수치 자체가 실제적인 의미를 표현	연령, 자녀 수	
문자형	텍스트 (String)	문자로 표현되어 있으며 의미는 범주형과 수치형으로 변환이 가능	TV1, TV2,...드라마2, 드라마3	문자형

* 인터넷/모바일 사용시간(분)은 범주형 변수이나 실습을 위해 연속형 숫자로 간주한다.

'성별'은 object 자료형태이고 나머지는 각각 실수형 12개와 정수형 10개로 총 23개의 변수가 있다. 컴퓨터 내 메모리 사용은 718.9+ KB이다❶.

df.info() ❶

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   성별      4000 non-null   int64
 1   연령      4000 non-null   int64
 2   지역      4000 non-null   int64
~~~~~~~
 21  드라마3    1730 non-null   float64
 22  구매      4000 non-null   int64
dtypes: float64(12), int64(11)
memory usage: 718.9 KB
```

'연령', '가구소득', '학력', '자녀수', '구매' 등의 변수는 수치형으로❶❸ 나머지 변수는 범주형으로 변경한 후❷❹ 자료형태를❺ 확인한다. '성별' 변수가 Object 자료형태에서 범주형으로 바뀌면서 컴퓨터 내 메모리 사용이 396.8 KB로 줄어들었다. 수치형 자료로 변환할 경우 df[col] = df[col].apply(pd.to_numeric)을 사용할 수도 있다. 수치형 이외의 모든 자료 형태는 범주형으로 구분하였다. 제6장 특성변수에서 실제 예측을 위해 명목형, 서열형을 구분하여 데이터 변환을 하기로 한다.

```
Nums = ['연령', '가구소득', '학력', '자녀수', '인터넷평일', '인터넷주말', '인터넷휴일',          ❶
        '모바일평일', '모바일주말', '모바일휴일', '구매']
Cats = [item for item in df.columns if item not in Nums]                              ❷
for col in Nums :                                                                     ❸
        df[col] = df[col].astype('float64')
for col in Cats :                                                                     ❹
        df[col] = df[col].astype('category')
df.info()                                                                             ❺
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   성별       4000 non-null   category
 1   연령       4000 non-null   float64
 2   지역       4000 non-null   category
~~~~~~~
 21  드라마3     1730 non-null   category
 22  구매       4000 non-null   float64
dtypes: category(12), float64(11)
memory usage: 396.8 KB
```

컬럼명을 집합형태로 변경하여 ','를 구분자로 원소를 문자로 나열하여❶ 개수를 확인하면❷ 총 23개이다. 수치형과 범주형 자료의 원소와❸❺ 개수는❹❻ 각각 11개와 12개이다.

```
','.join(set(df.columns))                                                             ❶
len(set(df.columns))                                                                  ❷
','.join(set(Nums))                                                                   ❸
len(set(Nums))                                                                        ❹
','.join(set(Cats))                                                                   ❺
len(set(Cats))                                                                        ❻
```

'학력,모바일주말,인터넷휴일,인터넷평일,드라마1,모바일평일,지역,TV2,가구소득,모바일휴일,연령,드라마2,자녀수,성별,TV1,구매,TV3,드라마3,인터넷주말,결혼,직업,TV4,주거형태'

23

'가구소득,학력,모바일주말,모바일휴일,인터넷휴일,인터넷주말,구매,인터넷평일,연령,모바일평일,자녀수'

11

'TV1,TV3,결혼,드라마1,드라마2,직업,지역,성별,TV2,드라마3,TV4,주거형태'

12

파이썬의 집합자료(Set)는 합집합, 교집합, 차집합의 연산이 가능하다. 리스트(List)의 경우는 중복된 원소를 가질 수 있지만, 집합은 중복할 수 없는 유일한 원소만 갖는다. 전체 변수와 수치형 변수의 합집합의 원소가 같은지❶ 확인한다. 전체 변수(df.columns)와 수치형 변수(Num)의 교집합이 수치형 변수와 같은지❷ 범주형 변수(Cats)의 경우도 같은 방식으로❸ 확인한다. 차집합 연산도❹ 확인한다.

```
set(df.columns) | set(Nums) == set(df.columns)                    ❶
set(df.columns) & set(Nums) == set(Nums)                          ❷
set(df.columns) & set(Cats) == set(Cats)                          ❸
set(df.columns) - set(Nums) - set(Cats) # 공집합이면 OK            ❹
```

```
True
True
True
set()
```

범주형 자료의 값 변경

사용자정의함수 summary()로 자료요약 정보를 확인한다❶.

```
summary(df)                                                       ❶
```

	데이터형태(dtypes)	비 결측치 수(notnull	결측치 수(null)	고유값 수(nunique)
성별	category	4000	0	2
연령	float64	4000	0	57
지역	category	4000	0	7
직업	category	4000	0	7
가구소득	float64	4000	0	12
학력	float64	4000	0	12
주거형태	category	4000	0	5
결혼	category	4000	0	3
자녀수	float64	4000	0	4
인터넷평일	float64	2603	1397	37
인터넷주말	float64	2603	1397	35
인터넷휴일	float64	2603	1397	31

.....(생략)

소비자의 설문응답 내용을 쉽게 확인하기 위해 명목형 컬럼을 응답 결과로 변경한다. 명목형 컬럼 '성별', '지역', '직업', '결혼'의 값을 'MCR 2019' 자료의 세 번째 시트 '보기정보'에서 의미를 확인하여 변경한다❶❷❸.

```
df['지역'] = df['지역'].map({1:'서울', 2:'경기인천', 3:'대전충청세종', 4:'광주전라제주',
                             5:'부산울산경남', 6:'대구경북', 7:'강원'})                    ❶
df['직업'] = df['직업'].map({1:'행정관리전문직', 2:'사무직', 3:'판매서비스직', 4:'기능노무직',
                             5:'전업주부', 6:'학생재수생', 7:'무직기타'})                  ❷
df['결혼'] = df['결혼'].map({1:'기혼', 2:'미혼', 3:'기타_이혼별거사별'})                   ❸
```

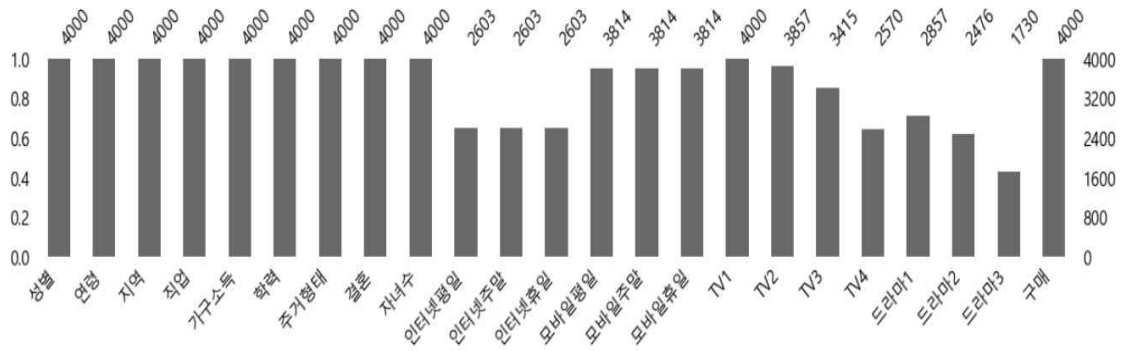
서열형 컬럼 '학력', '주거형태'의 값을 변경한다❶❷❸.

```
df['학력'] = df['학력'].map({1:'정규교육없음', 2:'초등학교졸업', 3:'중학교졸업',
                             4:'고등학교졸업', 5:'전문대학졸업', 6:'대학교졸업', 7:'대학원졸업',
                             8:'중학교재학', 9:'고등학교재학', 10:'전문대학재학',
                             11:'대학교재학', 12:'대학원재학'})                           ❶
df['주거형태'] = df['주거형태'].map({1:'자가', 2:'전세', 3:'월세', 4:'영구임대', 5:'사택기타'})  ❷
```

3 결측치 채우기 (missing value imputation)

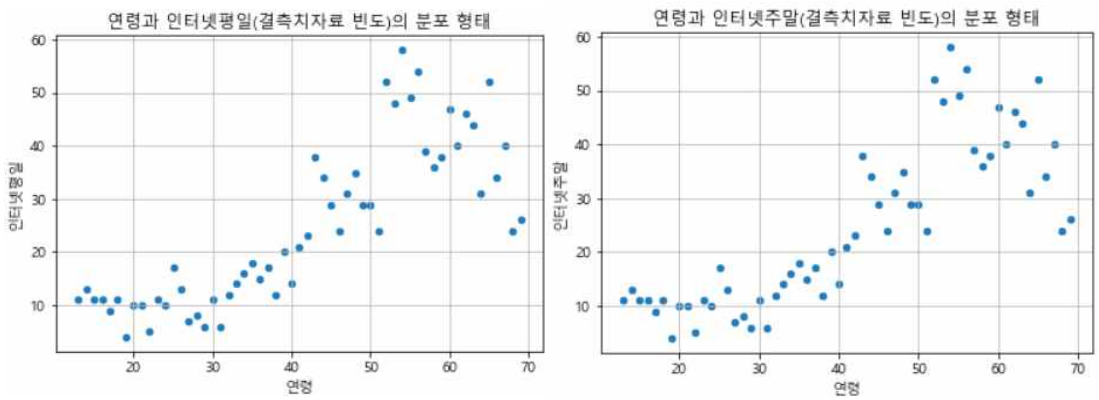
결측치 현황을 확인한다❶. '인터넷평일', '인터넷주말', '인터넷휴일', '모바일평일', '모바일주말', '모바일휴일', 'TV2', 'TV3', 'TV4', '드라마1', '드라마2', '드라마3' 등의 결측치를 확인한다. 결측치가 있는 경우 자료의 형태, 막대그래프, 평균과 최빈값을 확인하여 가장 적합한 값으로 대체한다.

```
msno.bar(df, figsize=(20,3))
```
❶



'인터넷평일', '인터넷주말' 컬럼에서 결측치가 있는 자료를(무응답자) 대상으로 연령과 결측치의 빈도를 확인하면❶ 대부분 '40대 이상' 이다.

```
for col in ['인터넷평일', '인터넷주말']:
    df[df[col].isnull()]['연령'].value_counts().reset_index().plot.scatter(x='index',y='연령')
    plt.xlabel('연령')
    plt.ylabel(col)
    plt.title('연령과 '+ col+'(결측치자료 빈도)의 분포 형태')
    plt.grid(True)
```
❶



‘40세’를 기준으로❶ 반복문을 통해 결측치가 있는 '인터넷평일', '인터넷주말', '인터넷휴일', '모바일평일', '모바일주말', '모바일휴일'의 열을 읽어와❷ 평균으로 결측치를 대체한다❹❺.

```
❶ over_40 = (df['연령'] >= 40); under_40 = (df['연령'] < 40)
❷ cols = ['인터넷평일','인터넷주말','인터넷휴일','모바일평일','모바일주말','모바일휴일']
❸ for col in cols :
❹       df.loc[over_40, col] = df.loc[over_40, col].fillna(df.loc[over_40,col].mean())
❺       df.loc[under_40, col] = df.loc[under_40, col].fillna(df.loc[under_40,col].mean())
```

문자형 자료인 'TV2', 'TV3', 'TV4', '드라마1', '드라마2', '드라마3’ 컬럼의❶ 결측치는 반복문을 통해 ❷ 최빈값(mode)으로❸ 대체한다.

```
❶ col_02 = ['TV2', 'TV3', 'TV4', '드라마1', '드라마2', '드라마3']
❷ for col in col_02 :
❸       df[col].fillna(df[col].mode()[0], inplace=True)
```

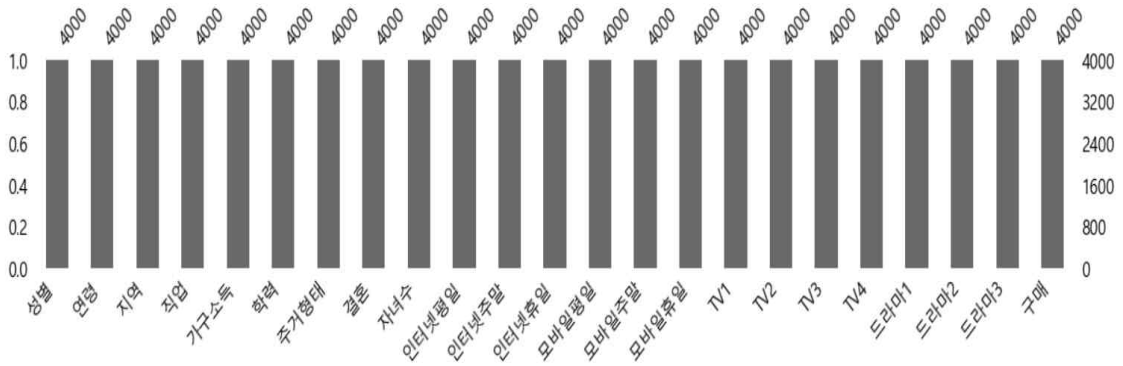
결측치와 무한대 값이 입력변수에 존재하면 예측의 입력자료로 사용할 수 없게 된다. 전처리에서 발생할 수 있는 결측치, 양수와 음수의 무한 값(np.inf, -np.inf)이 있는지 확인한다❶.

```
print('nan, inf & -inf 의 개수는 ?  ', df.isin([np.nan, np.inf, -np.inf]).any(1).sum())      ❶
```

```
nan, inf & -inf 의 개수는 ?   0
```

결측치 처리결과를 그래프로 확인하면 결측치 비율이 60%가 넘는 열은 모두 제거되고 모든 결측치는 대체되었다❶.

```
msno.bar(df, figsize=(20,3))                                                                ❶
```



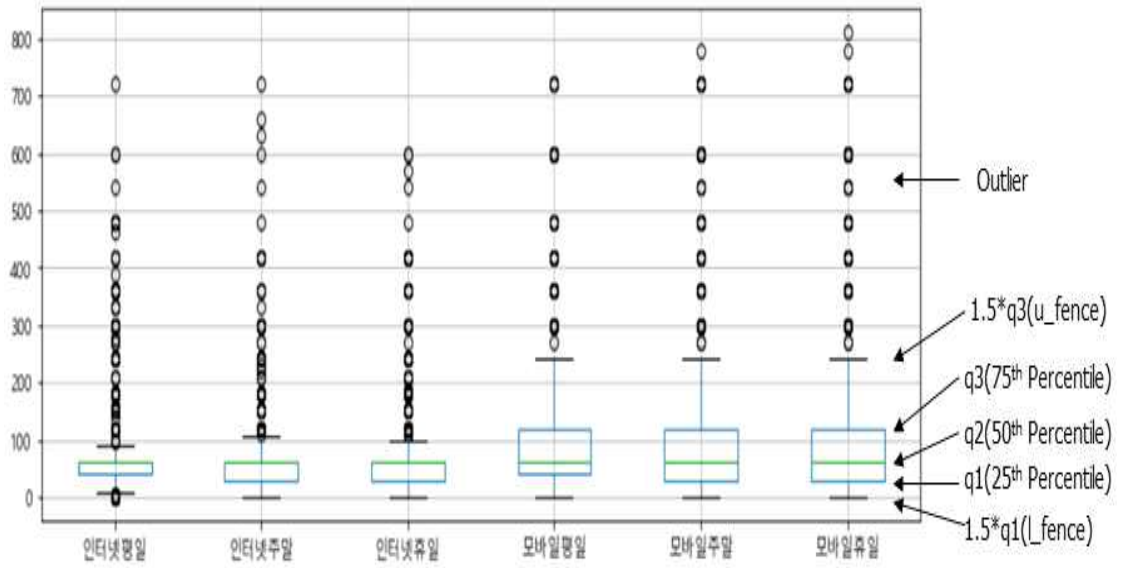
4.2 이상치(outliers) 처리

1 이상치(Outliers) 처리

변수의 값이 다른 값에 비해 극단적으로 작거나 큰 이상치(Outlier)는 예측결과를 왜곡하거나 예측 성능을 떨어뜨리게 된다.

이상치가 발생할 수 있는 실수형 변수의❶ 상자 그래프를❷ 사분위(Inter-quartile range, IQR)로 파악한다. 제1사 분위수(25% 백분위 수, q1), 중앙값인 제2사 분위수(50% 백분위 수, q2), 제3 사분위수(75% 백분위 수, q3)를 확인한다.

```
suspects = ['인터넷평일', '인터넷주말', '인터넷휴일', '모바일평일',                    ❶
            '모바일주말', '모바일휴일']
df.boxplot(column=suspects , figsize=(12,4))                                    ❷
```



상자의 윗면과 아랫면은 제3사 분위 수(q3)와 제1사 분위 수(q1)를 나타낸다. 중앙값(median, q2)은 상자 안의 초록색 선이다. 상자의 위와 아래에 있는 실선은 각각 q3의 1.5배, q1의 1.5배를 가감한 값이다. 검은색 실선 밖에 자료가 있으면 이상치이다.

이 책에서는 자료의 손실을 방지하기 위해 '인터넷평일' 변수의 15%, 95%의 백분위 수를 각각 q1, q3로 설정하고❶❷ 사분위 수 범위 구간에서 1.5배만큼 떨어져 있는 자료를❸ 이상치로 정의하여❹ 개수, 형태, 인덱스, 값을❺❻❼❽ 확인한다.

```
❶ q1, q3 = df['인터넷평일'].quantile(0.15), df['인터넷평일'].quantile(0.95)
❷ iqr = q3 - q1
❸ l_fence, u_fence = q1 - 1.5*iqr , q3 + 1.5*iqr
❹ outliers = df['인터넷평일'][(df['인터넷평일'] < l_fence) | (df['인터넷평일'] > u_fence)]
❺ outliers.shape
❻ outliers.head(10)
❼ outliers.index
❽ outliers.values
```

```
(40,)

23      300.0
37      300.0
115     330.0
151     480.0
163     540.0
186     360.0
225     270.0
235     480.0
242     300.0
252     360.0
Name: 인터넷평일, dtype: float64

Int64Index([  23,   37,  115,  151,  163,  186,  225,  235,  242,  252,  277,
              290,  307,  337,  359,  363,  481,  545,  592,  662, 1374, 1375,
             1474, 1480, 1486, 1911, 1913, 1952, 2102, 2174, 2259, 2287, 2356,
             2416, 2425, 2571, 2789, 2915, 3041, 3442],
            dtype='int64')

array([300., 300., 330., 480., 540., 360., 270., 480., 300., 360., 390.,
       600., 300., 300., 360., 330., 330., 600., 280., 480., 420., 300.,
       360., 360., 420., 480., 360., 270., 480., 465., 300., 360., 270.,
       360., 300., 300., 300., 300., 420., 720.])
```

사분위 범위를 넓게 설정하면 데이터의 누락을 방지할 수 있다. 이상치를 df['인터넷_평일'].drop(outliers.index, axis = 0)으로 제거하는 방법이 있으나 관측치가 줄어든다.

사용자정의함수 Outliers()를 정의하여❶ 사분위 구간의 상하 1.5배인 상한선(u_fence)과 하한선(l_fence)을 설정한다❷❸. 각 변수의 이상치 개수❹, 상하한선❺을 확인한다. 반복문을 통해 변수를 하나씩 호출하여❼ 이상치를 대체한다❽.

```
❶ def Outliers(col):
        q1, q3 =col.quantile(0.15), col.quantile(0.95)
        iqr = q3 - q1
        l_fence, u_fence = q1 - 1.5*iqr , q3 + 1.5*iqr
        outliers = col[(col < l_fence) | (col > u_fence)]
❷       col[(col < l_fence)] = l_fence
❸       col[(col > u_fence)] = u_fence
❹       print(col.name,'의 총 이상치 관측 수:', outliers.count())
❺       print('Lower fence: ', l_fence, 'Upper fence: ', u_fence)
❻       return col
❼ for col in suspects :
❽       df[col] = Outliers(df[col])
```

```
인터넷평일 의 총 이상치 관측 수: 40
Lower fence:  -105.0 Upper fence:  255.0
인터넷주말 의 총 이상치 관측 수: 24
Lower fence:  -130.0 Upper fence:  270.0
인터넷휴일 의 총 이상치 관측 수: 20
Lower fence:  -180.0 Upper fence:  300.0
모바일평일 의 총 이상치 관측 수: 28
Lower fence:  -195.0 Upper fence:  405.0
모바일주말 의 총 이상치 관측 수: 23
Lower fence:  -285.0 Upper fence:  555.0
모바일휴일 의 총 이상치 관측 수: 25
Lower fence:  -285.0 Upper fence:  555.0
The numbers of nan, inf & -inf :  0
```

Nan값, 양수와 음수의 무한 값(np.inf, -np.inf) 등을 확인한다❶.

```
print('nan, inf & -inf 의 개수는 ?  ', df.isin([np.nan, np.inf, -np.inf]).any(1).sum())      ❶
```

```
nan, inf & -inf 의 개수는 ?   0
```

2 이상한 응답 값 처리

설문 응답 항목에 '99', '999', '9999' 등의 값을 입력하는 경우가 있다. 각 열의 최대값을❶ 살펴보면 가구소득, 자녀수, TV1의 열에 '99' 값이 있는 것을 확인할 수 있다.

```
df.max()                                                    ❶
```

```
성별                  2
연령                 69
지역                 서울
직업          행정관리전문직
가구소득             99
학력          초등학교졸업
주거형태             전세
결혼                 미혼
자녀수               99
인터넷평일           255
인터넷주말           270
인터넷휴일           300
모바일평일           405
모바일주말           555
모바일휴일           555
TV1                99
TV2                17
TV3                16
TV4                16
드라마1              16
드라마2              15
드라마3              16
구매                 6
dtype: object
```

반복문으로 데이터의 모든 변수를 하나씩 읽어❶ '99' 값이 있는 데이터가 한 줄이라도 있는 경우에❷ 해당 변수명과 크기를❸ 확인한다. 해당 열의 value_count()의 인덱스와 범주형 유형 빈도를❹❺ 보면 가구소득, TV1 열에는 99 값의 개수가 각각 2개, 1개이다.

```
for col in df.columns :                                                    ❶
    if df[(df[col] == 99)].shape[0] > 0 :                                  ❷
        col, df[df[col] == 99].shape[0]                                    ❸
        df[col].value_counts().index                                       ❹
        df[col].value_counts().values                                      ❺
```

```
('가구소득', 2)

Float64Index([6.0, 5.0, 4.0, 7.0, 3.0, 8.0, 9.0, 2.0, 11.0, 10.0, 1.0, 99.0],
dtype='float64')

array([969, 772, 600, 487, 398, 278, 189, 118, 90, 51, 46, 2], dtype=int64)

('자녀수', 1,524)

Float64Index([2.0, 99.0, 1.0, 3.0], dtype='float64')

array([1715, 1524, 585, 176], dtype=int64)

('TV1', 1)

CategoricalIndex([1, 4, 7, 2, 6, 3, 5, 9, 10, 8, 99, 12], categories=[1, 2, 3, 4, 5, 6,
7, 8, ...], ordered=False, dtype='category')

array([2899, 511, 262, 117, 104, 56, 26, 15, 6, 2, 1, 1], dtype=int64)
```

자녀 수를 '99'로 응답한 빈도가 1,524개이다. 무응답자를 '99'로 자료 처리한 것으로 추정된다. 따라서 99의 의미는 자녀가 없다는 의미로 간주하여 '0'으로 대체한다❶. 가구소득, TV1 열의 '99' 값을 갖는 인덱스와 범주 유형 빈도를 비교한 결과, 오타로 추정되기에 해당 행은 제거한다❷❸. 각 열의 최대값을 확인한 결과 '99'의 값을 갖는 필드는 없다❹.

```
df['자녀수'] = df['자녀수'].apply(lambda x: 0 if x == 99 else x)           ❶
df.drop(df.loc[df['가구소득'] == 99].index, inplace = True)                ❷
df.drop(df.loc[df['TV1'] == 99].index, inplace = True)                     ❸
(df.max() == 99).sum()                                                     ❹
```

```
0
```

4.3 왜도(Skewness) 처리

왜도(Skewness)는 데이터 분포의 비대칭 정도로 자료가 한쪽으로 쏠리는 유형을 의미한다. 왜도 종류는 양의 왜도(positive skewness)와 음의 왜도(negative skewness)가 있다. 왜도 값이 양수이면 오른쪽에 꼬리를 가진(왼쪽에 데이터가 많은) 형태이고, 음수 값이면 왼쪽에 꼬리를 가진(오른쪽에 데이터가 많은) 형태이다. 정규분포의 왜도는 '0'이다.

인터넷과 모바일의 평일과 주중 이용시간 변수의❶ 왜도를 살펴보고❷ '+2' 초과이거나 '-2' 미만이면 왜도가 있다고 가정하여 각 열의 왜도 값을❸ 확인한다.

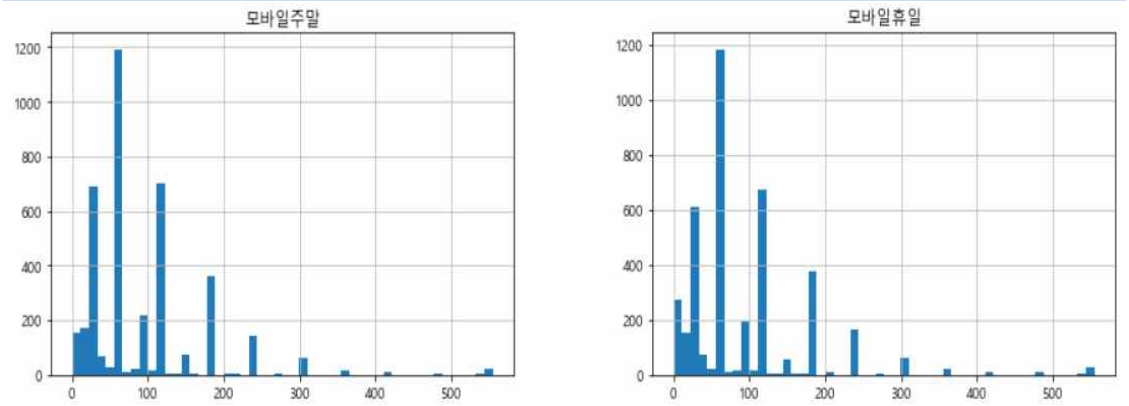
```
check = ['인터넷평일','인터넷주말','인터넷휴일','모바일평일','모바일주말','모바일휴일']                ❶
df[check].skew()                                                                                ❷
df[check].skew()[(df[check].skew() > 2) | (df[check].skew() < -2)]                              ❸
```

```
인터넷평일     1.992342
인터넷주말     1.877968
인터넷휴일     2.118610
모바일평일     1.947162
모바일주말     2.328548
모바일휴일     2.216040
dtype: float64

인터넷휴일     2.118610
모바일주말     2.328548
모바일휴일     2.216040
dtype: float64
```

왜도 값이 큰 '모바일주말'과 '모바일휴일' 변수의❶ 막대그래프는 오른쪽에 꼬리를 가진(왼쪽에 데이터가 많은) 양의 왜도를 갖는다❷.

```
sk = ['모바일주말', '모바일휴일']                                                                ❶
df[sk].hist(bins=50, figsize=(15,4))                                                            ❷
```



양의 왜도가 있는 자료를 처리하는 방법은 로그변환(Log transformation), 제곱근 루트변환(Square root), 박스콕스 변환(Box-Cox Transform) 등이 있다. 이 책에서는 로그변환과 제곱근 루트의 왜도 개선 정도를 비교하였다. 반복문으로 왜도가 높은 두 개 변수를 하나씩 읽어❶ 변수명❷ 왜도 값❸, 로그변환 값❹, 제곱근 루트변환 값의❺❻ 왜도 값을 계산하였다.

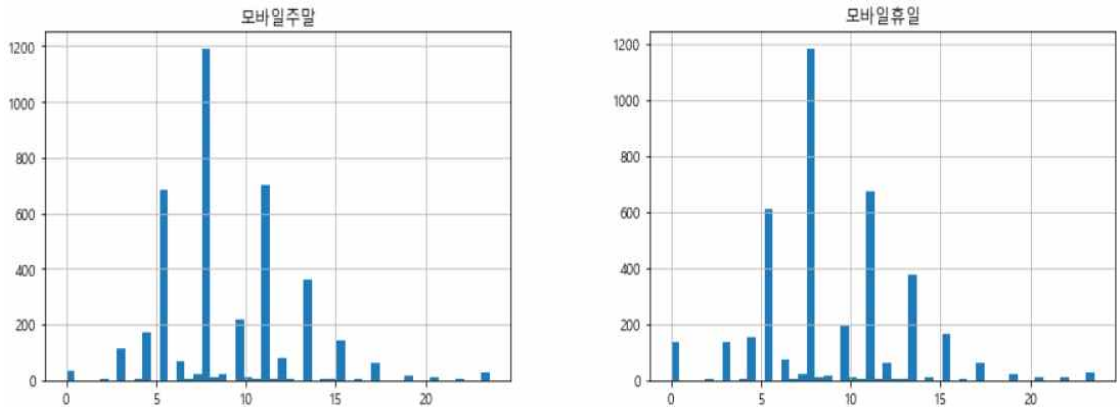
```
❶ for i in sk :
❷       print('변수명 :', i)
❸       print('원자료_skewness :', df[i].skew())
❹       print('log변환_skewness :', np.log(1 + df[i]).skew()) # -inf방지
❺       print('squareroot변환_skewness :', np.sqrt(df[i]).skew())
❻       print('*'*50)
```

```
변수명 : 모바일주말
원자료_skewness : 2.3285478340362737
log변환_skewness : -0.9822699243989281
squareroot변환_skewness : 0.8042985452979947
**************************************************
변수명 : 모바일휴일
원자료_skewness : 2.2160403737777896
log변환_skewness : -1.7017025705614766
squareroot변환_skewness : 0.49218534855428253
**************************************************
```

제곱근루트 변환이 로그변환방식보다 왜도 개선 효과가 크므로 왜도가 있는 각 열에 제곱근 루트 변환하여❶❷ 왜도를 제거한다. 왜도 제거 후에 분포의 치우침이 개선되었다.

```
for i in sk :                                                    ❶
    df[i] = np.sqrt(df[i])
df[sk].skew()                                                    ❷
df[sk].hist(bins=50, figsize=(15,4))                             ❸
```

```
모바일주말      0.804299
모바일휴일      0.492185
```



결측치, 양과 음의 무한대 값 등이 데이터에 없는 것을 확인한다❶.

```
print('The numbers of nan, inf & -inf ',df.isin([np.nan, np.inf, -np.inf]).any(1).sum())  ❶
The numbers of nan, inf & -inf :  0
```

현재 데이터프레임의 인덱스의 순서를 확인해보면❶❷❸ 데이터 전처리 과정에서 특정 행을 삭제하였기 때문에 인덱스 871번, 2230번, 3203번이 결번이다.

```
set([i for i in range(len(df))]) - set(df.index)                          ❶
for i in set([i for i in range(len(df))]) - set(df.index) :                ❷
    df[df.index == i]                                                      ❸
{871, 2230, 3203}
```

```
성별 연령 지역 직업 가구소득 학력 주거형태 결혼 자녀수 인터넷평일 ... 모바일주말 모바일휴일 TV1

0 rows × 23 columns
```

```
성별 연령 지역 직업 가구소득 학력 주거형태 결혼 자녀수 인터넷평일 ... 모바일주말 모바일휴일 TV1

0 rows × 23 columns
```

```
성별 연령 지역 직업 가구소득 학력 주거형태 결혼 자녀수 인터넷평일 ... 모바일주말 모바일휴일 TV1

0 rows × 23 columns
```

데이터프레임의 인덱스 순번을 '0'번부터 차례대로 부여하기 위해서 'df.index = [i for i in range(len(df))]' 명령어를 사용하거나 reset_index(drop = True)❶ 명령어를 사용한다. 데이터프레임의 순번을 결번 없이 차례대로 만들어야 자료의 결합(merge), 처리에 오류가 없다.

```
df.reset_index(drop = True, inplace= True) # df.index = [i for i in range(len(df))]  ❶
```

데이터 전처리로 자료 중에 'object' 형태를 확인하고❶ 범주형으로 변경한다❷.

```
df.columns[df.dtypes == 'object']                                          ❶
df['학력']= df['학력'].astype('category')                                   ❷
Index(['학력'], dtype='object')
```

TV와 드라마의 선호 장르 유형을 수치화 변수로 만들기 전에 현재까지 작업한 데이터프레임을 '4장3절자료.pkl' 파일로 'data' 폴더에 저장한다❶.

```
df.to_pickle('../data/4장3절자료.pkl')                                    ❶
```

발생빈도가 낮은 범주형 변수 값(rare values)은 발생빈도가 높은 유형에 포함하거나, 범주 유형이 많은 경우(high cardinality)는 다른 대표 값(평균, 중앙값)으로 대체하여 예측 성능을 높일 수 있다.

연속형 자료를 이산형 자료로 바꾸는 것을 'Feature discretization'이라 한다. 유사한 범주의 값을 갖는 자료로 그룹핑, 여러 변수를 선형적으로 분리할 수 없는 경우 등에 'Feature discretization' 데이터 전처리를 하면 예측 성능이 높아지게 된다. 제8장 머신러닝 응용에서 소개한다. 자세한 내용은 사이킷런에서 'Feature discretization'를 검색하여 확인할 수 있다. 이산형 자료를 연속형으로 변경하는 경우에는 scipy.signal.cont2discrete()을 참고하기 바란다. 참고문헌의 'A Short Guide for Feature Engineering and Feature Selection'의 문서와 관련 깃허브를 참고하기 바란다.

3.4 파생변수

파생변수란 기존의 변수를 조합하고 특정의 조건을 가정하여 새로운 변수를 만들어 내는 것을 의미한다. 가장 간단한 파생변수는 '연령대별 평균인터넷 이용시간', '학력과 소득을 동시에 반영한 상호작용' 변수 등이 그 예이다. 이 책에서는 '드라마1', '드라마2', '드라마3' 등의 열에서 선택한 유형 16개의 장르 유형을 새로운 파생변수로 만들었다.

1 종속변수와 자료 저장

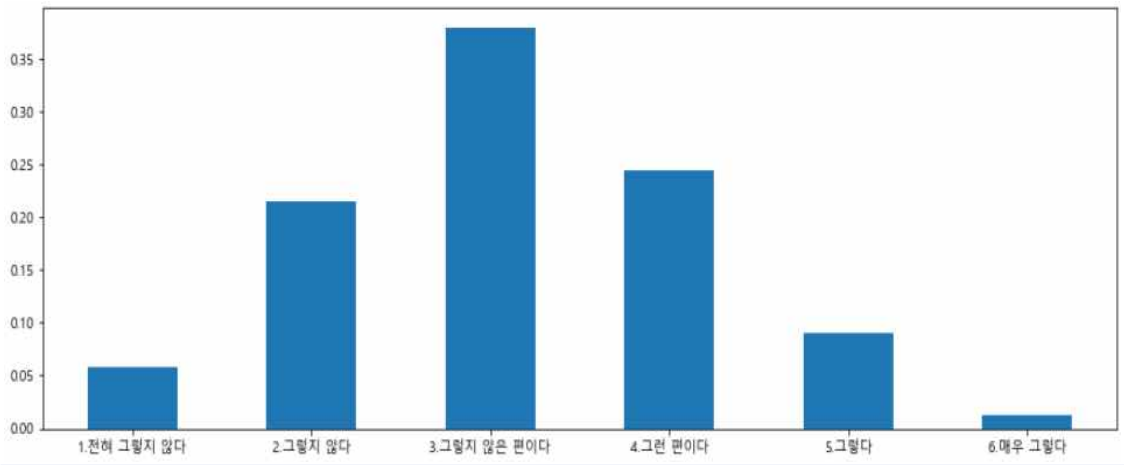
자 이제 현재까지의 데이터전 처리가 완료된 자료를 다시 저장하고❶ 소비자의 구매 경향의 비중을 살펴보았다❷.

```
df.to_pickle('../data/df_03.pkl')                    ❶
df['구매'].value_counts(normalize=True)              ❷
```

```
3.0    0.379284
4.0    0.244183
2.0    0.215161
5.0    0.089817
1.0    0.058544
6.0    0.013010
Name: 구매, dtype: float64
```

구매의 유형 '1'~'6' 까지 의미는 'MCR2019' 엑셀파일의 두 번째 시트(문항정보)에서 확인할 수 있습니다. '1'은 '전혀 그렇지 않다', '2'는 '그렇지 않다', '3'은 '그렇지 않은 편이다', '4'는 '그런 편이다', '5'는 '그렇다', '6'은 '매우 그렇다.' 각각 일대일 대응 관계입니다❶. '구매'의 유형을 텍스트로 변경하여❷ 구매의 경향 순서로❸ 그림으로 확인한다❹.

```
map_dict={1:'1.전혀 그렇지 않다', 2:'2.그렇지 않다', 3:'3.그렇지 않은 편이다',          ❶
          4:'4.그런 편이다', 5:'5.그렇다', 6:'6.매우 그렇다'}
tf = df['구매'].value_counts(normalize=True).rename(index=map_dict)                  ❷
tf.sort_index(inplace=True)                                                          ❸
tf.plot(kind='bar', rot='360', figsize=(20,5))                                       ❹
```



y 값의 유형을 '그런편이다' 이상과 '그렇다'의 두 개 그룹의 자료로 각각 만들어 저장하겠다.

데이터프레임을 tf로 복사한다❶.

```
❶ tf = df.copy()
```

응답 항목이 '4'번, '5'번, '6'번의 경우에만 구매의향이 있다고 간주하여 해당 열의 값을 모두 '1'로 변경하고❶❷❸ '구매' 변수는 삭제한 후❹ '구매의향' 변수를 '구매' 변수 이름으로❺ 변경하였다. 구매 분류유형은 '0'과 '1'로 이진 분류하고 구매의향이 있는 비율은 34.70%로❻ 나타났다. 'df_04'로 저장하여❼ 제4장~ 제6장의 기계학습 분야 예측모델 적용에 사용하도록 하겠다.

```
❶ tf['구매의향'] = 0
❷ buy = tf[((tf['구매'] == 4) | (tf['구매'] == 5) | (tf['구매'] == 6))].index
❸ tf.loc[buy, '구매의향'] = 1
❹ del tf['구매']
❺ tf = tf.rename(columns={'구매의향': '구매'})
❻ print('구매비율 {:0.2f}%'.format(tf['구매'].value_counts()[1]/len(tf)*100))
❼ tf.to_pickle('../data/df_04.pkl')
구매비율 34.70%
```

다시 한번 데이터프레임을 tf로 복사한다❶.

```
❶ tf = df.copy()
```

구매의향이 '5'인 '그렇다'와 '6'번 '매우 그렇다'의 경우만 구매의향이 있다고 간주하고 'df_07'로 저장하여 제7장의 불균형자료의 기계학습 분야 예측모델 적용에 사용한다.

```
❶ tf['구매의향'] = 0
   buy = tf[((tf['구매'] == 5) | (tf['구매'] == 6))].index
   tf.loc[buy, '구매의향'] = 1
   del tf['구매']
   tf = tf.rename(columns={'구매의향': '구매'})
   print('구매비율 {:0.2f}%'.format(tf['구매'].value_counts()[1]/len(tf)*100))
   tf.to_pickle('../data/df_07.pkl')
구매비율 10.28%
```

3.5 데이터 전처리 방법

데이터 전처리에 아주 유용한 몇 가지 기능을 소개한다.

```
df = pd.read_pickle('../data/dataset.pkl')                                    ❶
```

1 매핑과 그룹핑 분할(get_group())

'2019년 소비자행태조사 원시데이터_문항보기' 파일을 다운받아 두 번째 A_006의 문항을 확인하면 1:'행정관리/전문직', 2:'사무직', 3:'판매/서비스직', 4:'기능/노무직', 5:'전업주부', 6:'학생(재수생포함)', 7:'무직/기타'의 대응 관계를 알 수 있다. 대응 관계를 사전자료 형태로❷ 저장하여 '직업' 열의 값에 대응되는 직업명으로 변환하여❸ 전후 자료를❶❹ 확인한다.

```
df.head(3)                                                                    ❶
map_job ={1:'행정관리/전문직', 2:'사무직', 3:'판매/서비스직', 4:'기능/노무직',    ❷
          5:'전업주부', 6:'학생(재수생포함)', 7:'무직/기타'}
df['직업'] = df['직업'].apply(lambda x: map_job[x] if x in map_job.keys() else x)  ❸
df.head(3)                                                                    ❹
```

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
0	1	14.0	1	6	6.0	8.0	3	2	0.0	0.0	...	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1
1	1	14.0	1	6	6.0	8.0	1	2	0.0	30.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0
2	1	14.0	1	6	6.0	8.0	2	2	0.0	90.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1

3 rows × 49 columns

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
0	1	14.0	1	학생(재수생포함)	6.0	8.0	3	2	0.0	0.0	...	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1
1	1	14.0	1	학생(재수생포함)	6.0	8.0	1	2	0.0	30.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0
2	1	14.0	1	학생(재수생포함)	6.0	8.0	2	2	0.0	90.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1

3 rows × 49 columns

직업유형별로 그룹화된 6개의 자료를❶ 반복문을 통해 하나씩 읽어❷ 직업 유형의 이름과 자료의 크기와❸ 처음 3줄을❹ 확인하였다.

```
df_job = df.groupby('직업')                                    ❶
for name, group in df_job:                                     ❷
    name, group.shape                                          ❸
    group.head(2)                                              ❹
```

('행정관리/전문직', (280, 49))

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
71	1	26.0	1	행정관리/전문직	10.0	6.0	1	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
96	1	26.0	1	행정관리/전문직	5.0	6.0	3	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

2 rows × 49 columns

('사무직', (674, 49))

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
73	1	29.0	1	사무직	8.0	6.0	1	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
81	1	29.0	1	사무직	6.0	5.0	3	2	0.0	60.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

2 rows × 49 columns

('판매/서비스직', (746, 49))

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
83	1	28.0	1	판매/서비스직	7.0	6.0	1	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
90	1	25.0	1	판매/서비스직	6.0	5.0	1	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

2 rows × 49 columns

직업유형별로 그룹화된 6개의 자료중에서 '사무직'의 직업 유형을 가진 특정 자료를 저장하여❶ 처음 2줄과❷ 크기를❸ 확인한다.

```
df_job_사무직 = df_job.get_group('사무직')                      ❶
df_job_사무직.head(2)                                          ❷
df_job_사무직.shape                                            ❸
```

	성별	연령	지역	직업	가구소득	학력	주거형태	결혼	자녀수	인터넷평일	...	dra8	dra9	dra10	dra11	dra12	dra13	dra14	dra15	dra16	구매
73	1	29.0	1	사무직	8.0	6.0	1	2	0.0	60.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
81	1	29.0	1	사무직	6.0	5.0	3	2	0.0	60.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0

2 rows × 49 columns

(674, 49)

df_job.get_group('사무직')의 데이터 크기는 (674, 49)이며 직업 유형으로 빈도를 확인한 결과인 사무직 674개와❶ 같은 것을 알 수 있다.

```
df['직업'].value_counts()                                      ❶
```

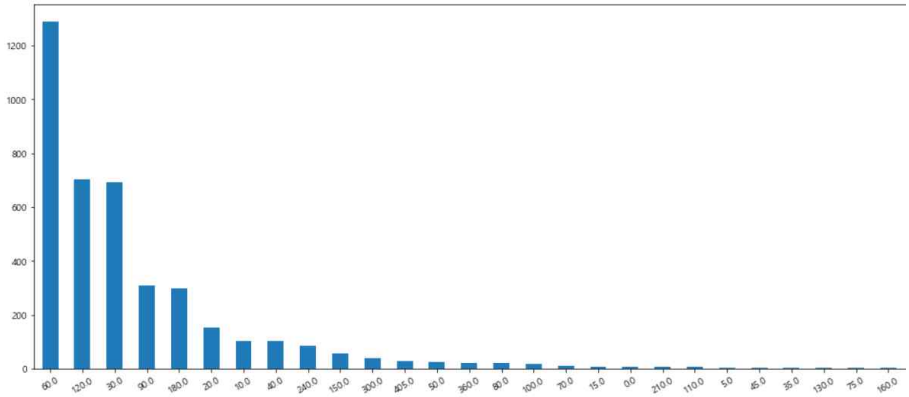
```
학생(재수생포함)      752
판매/서비스직         746
사무직               674
기능/노무직           661
전업주부              654
행정관리/전문직       280
무직/기타             230
Name: 직업, dtype: int64
```

2 수치형을 범주형 데이터로 변환

'모바일 평일 이용시간'을❶ 5개 구간으로 구분된 범주형 데이터로 만들려고 한다. '모바일 평일 이용시간'의 이용시간이 대부분 60분, 120분, 30분, 90분 등에 집중되어 있고 나머지 이용시간별로는 발생빈도가 아주 적은 롱테일(Longtail)❷ 형태이다.

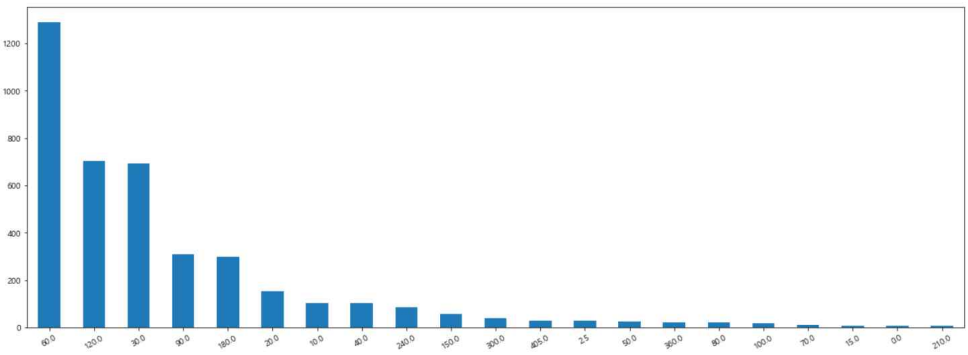
```
df.모바일평일.dtype                                                    ❶
df.모바일평일.value_counts().plot(kind = 'bar', rot='30', figsize=(20,6))   ❷
```

```
dtype('float64')
```



발생빈도가 상위 15위에 있는 모바일이용시간과❶ 나머지 빈도는 모두 합쳐 '모바일평일_new'이라는 새로운 컬럼을❷ 생성하여 분포형태를❸ 살펴보았다.

```
top_20 = df['모바일평일'].value_counts().nlargest(20).index                    ❶
df['모바일평일_new'] = df['모바일평일'].where(df['모바일평일'].isin(top_20),     ❷
                                                other = 2.5)
df.모바일평일_new.value_counts().plot(kind = 'bar', figsize=(20,4))           ❸
```



‘모바일평일_new’컬럼을 5등급으로 분류할 수도 있으나 데이터의 왜도 때문에 특정 백분위 등급에 해당되지 않는 자료는 ‘Nan’값이❶ 생길 수 있다. 임의로 3개로 분류하거나❷ 4개로 분류하여 '평균미만', '평균', '평균이상', '상위' 라벨을 부여하여❸ 범주형 데이터로 만든다.

```
pd.cut(df['모바일평일_new'], bins = [0, 25, 50, 75, 100]).head()                          ❶
pd.qcut(df['모바일평일_new'], q = 3).head()                                               ❷
pd.qcut(df['모바일평일_new'], q = 4, labels = ['평균미만', '평균', '평균이상', '상위']).head()  ❸
```

```
0          NaN
1    (25.0, 50.0]
2          NaN
3    (50.0, 75.0]
4          NaN
Name: 모바일평일_new, dtype: category
Categories (4, interval[int64]): [(0, 25] < (25, 50] < (50, 75] < (75, 100]]
0     (90.0, 405.0]
1    (-0.001, 60.0]
2     (90.0, 405.0]
3    (-0.001, 60.0]
4     (90.0, 405.0]
Name: 모바일평일_new, dtype: category
Categories (3, interval[float64]): [(-0.001, 60.0] < (60.0, 90.0] < (90.0, 405.0]]
0      상위
1    평균미만
2    평균이상
3      평균
4      상위
Name: 모바일평일_new, dtype: category
Categories (4, object): [평균미만 < 평균 < 평균이상 < 상위]
```

3 무작위 데이터 생성

윈도우 명령프롬프트에서 'pip install Faker'를 실행하여 Fake()모듈[1]을 설치하거나 가상의 데이터를 무작위로 만들 수 있는 Faker 모듈을 설치한다❶. 모듈을 불러오고❷ 한국어버젼으로❸ 무작위추출로❹ 이름, 주소, 소속회사, 전화번호를❺❻❼❽ 생성하였다. 영어, 중국어, 프랑스어 등의 언어로 성명, 카드번호, 인터넷 주소 등을 무작위로 생성할 수 있다.

```
!pip install Faker                                  ❶
from faker import Faker                             ❷
fake = Faker('ko_KR')                               ❸
Faker.seed(1357)                                    ❹
fake.name()                                         ❺
fake.address()                                      ❻
fake.company()                                      ❼
fake.phone_number()                                 ❽
```

```
'이민서'
'경상북도 포천시 삼성거리 (승민박리)'
'유한회사 오윤백'
'011-537-3309'
```

가상의 이름, 주민번호, 소속회사, 이메일을 리스트로❶ 만든 후에 저장하여 확인하였다❷.

```
data = [[fake.name(), fake.ssn(), fake.company(), fake.email()] for i in range(10)]   ❶
pd.DataFrame(data, columns = ['이름', '주민번호', '회사', '이메일'])                    ❷
```

	이름	주민번호	회사	이메일
0	박서현	710017-2344474	주식회사 이	jiweongim@nate.com
1	박종수	660820-1764450	백박	co@ju.kr
2	박시우	840802-2746999	김조	minjunbag@baggim.com
3	윤시우	750625-2384433	이이	cgim@yu.kr
4	박영환	580317-2325031	홍최김	minseogyun@hanhong.kr
5	최정순	700516-2068394	주식회사 김	dhan@dreamwiz.com
6	박순옥	550224-2003877	김이	gimjiyeong@daum.net
7	허승현	010326-2112463	(주) 박박민	misuggim@naver.com
8	강선영	910716-1184903	(주) 김구강	seomisug@yu.kr
9	조승민	200214-1404449	최박	isunog@naver.com

1) https://pypi.org/project/Faker/

5개의 서로 다른 데이터를 10개씩 생성하여 리스트에 저장한 후 하나의 데이터로 합쳐서 확인할 수 있다.

```
dataset =[]                                                                    ❶
for i in range(5) :                                                            ❷
    data = [[fake.name(), fake.ssn(), fake.company(), fake.email()] for i in   ❸
range(10)]                                                                     ❹
    dataset.append(pd.DataFrame(data, columns = ['이름', '주민번호', '회사', '이메일']))  ❺
pd.concat((data for data in dataset), ignore_index = True).head()              ❻
pd.concat((data for data in dataset), ignore_index = True).shape               ❼
```

	이름	주민번호	회사	이메일
0	박유진	790820-1879534	유한회사 진	ngim@yu.kr
1	이서연	990911-2671628	허김	singeonu@simgim.com
2	윤은지	230216-1086232	강박	hyeonsug85@jusighoesa.com
3	강정순	600526-1233647	장김이	wgim@dreamwiz.com
4	이은지	690819-1267892	김심	seongho40@gimijo.net

```
(50, 4)
```

위도와 경도도 가상의 데이터를 생성할 수 있다. fake.local_latlng(country_code='KR', coords_only=False)는 우리나라 지도위에서 위도와 경도를 ('37.65639', '126.835', 'Goyang-si', 'KR', 'Asia/Seoul') 튜플(tuple) 형태로 생성한다. 첫 번째와 두 번째의 원소만 이용하여 가상의 위도와 경도를 이용하여 지도에 표시하여 볼 수 있다.

4 데이터 결합 (pandas concate, merge)

두 개 이상의 자료를 하나로 합치는 데이터 병합(merge)이나 연결(concatenate)에 대해 살펴보도록 한다. 우선 HTML 형태의 화면 출력을 위한 모듈을 읽어오고❶ 쥬피터노크북의 명령어 결과는 행으로 출력하게 하는 명령문을 입력하여❷❸ 실행한다.

```
from IPython.display import display, HTML                    ❶
CSS = """.output {flex-direction: row;}"""                   ❷
HTML('<style>{}</style>'.format(CSS))                        ❸
```

관측치(obs)와 컬럼(var)을 인자로 입력받아 자료를 생성하는 함수 make_df()를 만들어❶❷❸ 3개의 자료를❹ 만들었다.

```
def make_df(var, obs):                                       ❶
    data = {c: [str(c) + str(i) for i in obs]                ❷
            for c in var}                                    ❸
    return pd.DataFrame(data, obs)                           ❹

make_df('ABCD', [0,1,2,3]); make_df('012', ['A','B']); make_df('ACEGIJ', [1,3,5,7,9])   ❺
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	0	1	2
A	0A	1A	2A
B	0B	1B	2B

	A	C	E	G	I	J
1	A1	C1	E1	G1	I1	J1
3	A3	C3	E3	G3	I3	J3
5	A5	C5	E5	G5	I5	J5
7	A7	C7	E7	G7	I7	J7
9	A9	C9	E9	G9	I9	J9

df1과❶ df2의❷ 자료를 생성한 후에 df1와 df2를 상하수직으로 연결하는 자료를 만들어 보면 각각의 자료에 없는 컬럼의 값에는 nan값이 기록된된 자료가❸ 생성된다.

```
df1 = make_df('ABCD', [0,1,2,3])                                    ❶
df2 = make_df('DEFG', [1,2,3,4])                                    ❷
df1; df2; pd.concat([df1, df2])                                     ❸
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

	A	B	C	D	E	F	G
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
1	NaN	NaN	NaN	D1	E1	F1	G1
2	NaN	NaN	NaN	D2	E2	F2	G2
3	NaN	NaN	NaN	D3	E3	F3	G3
4	NaN	NaN	NaN	D4	E4	F4	G4

두 개의 결합된(concat) 자료를 append()함수로❶ 똑같은 자료를 만들 수 있다.

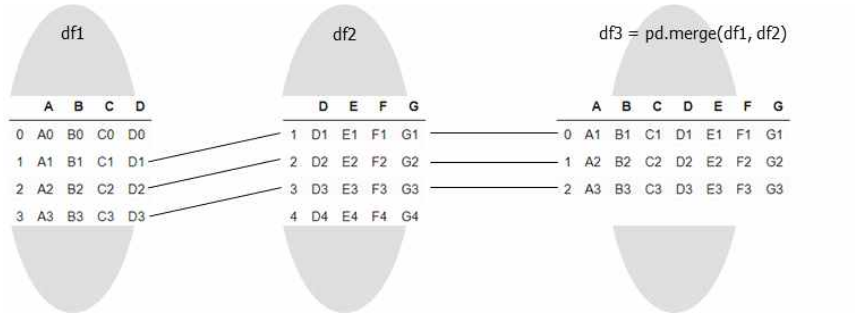
```
df1; df2; df1.append(df2)                                           ❶
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

	A	B	C	D	E	F	G
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN
1	NaN	NaN	NaN	D1	E1	F1	G1
2	NaN	NaN	NaN	D2	E2	F2	G2
3	NaN	NaN	NaN	D3	E3	F3	G3
4	NaN	NaN	NaN	D4	E4	F4	G4

두 개의 자료 병합(merge)의 종류는 자료의 대응 관계가 일대일대응(1:1), 일대다대응(1:M), M:1), 다대다(M:M)이 있다. df1과 df2 두 개의 자료는 'D'컬럼을 공유하고 있고 'D'컬럼 기준으로 보면 일대일대응(1:1) 관계이다. df1과 df2을 pd.merge(df1, df2)로 병합하면❶ 일대일대응(1:1) 관계의 행으로 df1과 df2 각각의 컬럼을 공유한다❷.




```
df3 = pd.merge(df1, df2)                                                    ❶
df1; df2; df3                                                               ❷
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

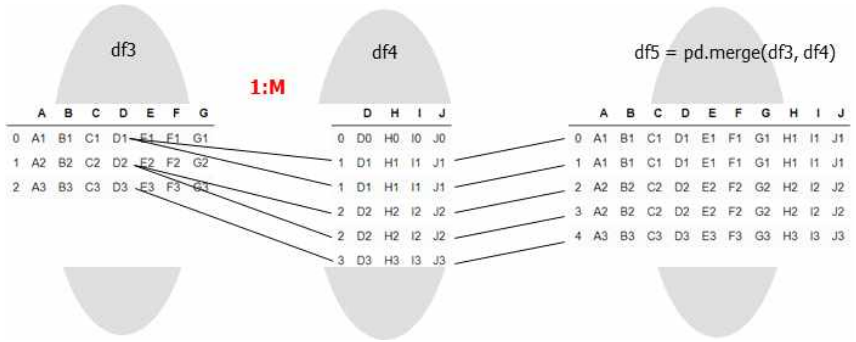
df2

	D	E	F	G
1	D1	E1	F1	G1
2	D2	E2	F2	G2
3	D3	E3	F3	G3
4	D4	E4	F4	G4

df3

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

df3과 df4 두 개의 자룐는 'D'컬럼을 공유하고 있고 'D'컬럼 기준으로 보면 일대다대응(1:M) 관계이다. df3과 df4을 pd.merge(df3, df4)로 병합하면❶ 일대일대응(1:M) 관계의 행으로 df3과 df4 각각의 컬럼을 공유한다❷.



```
df4 = make_df('DHIJ', [0,1,1,2,2,3])
df5 = pd.merge(df3, df4)
df3; df4; df5
```

❶
❷

df3

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

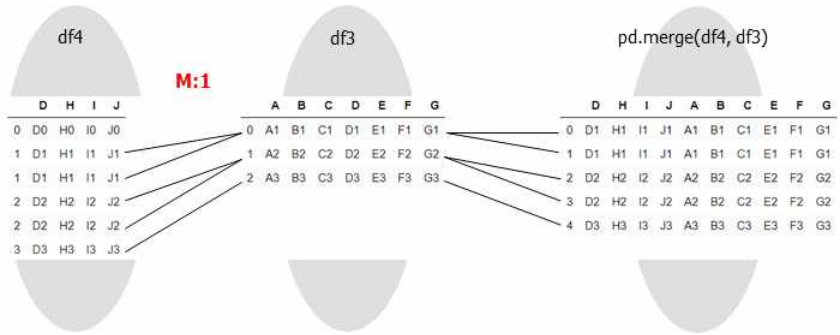
df4

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

df5

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

이젠 df4와 df3의 순서를 바꾸어 'D'컬럼 기준으로 보면 다대일대응(M:1) 관계이다. df4과 df3을 pd.merge(df4, df3)로 병합하면❶ 일대일대응(M:1) 관계의 행으로 df4과 df3 각각의 컬럼을 공유하게 된다.




```
df4; df3; pd.merge(df4, df3)
```

❶

df4

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

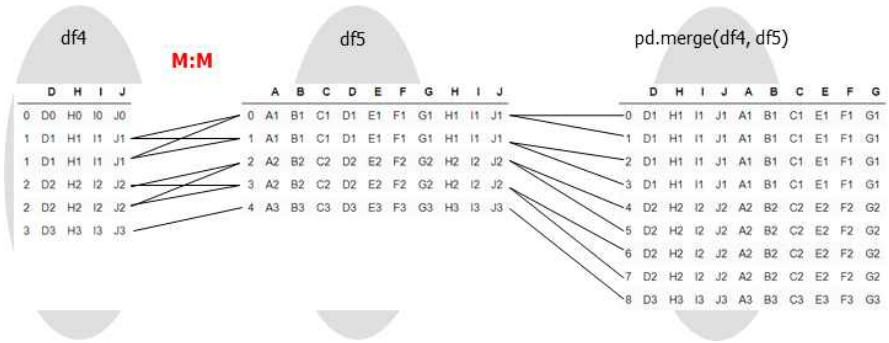
df3

	A	B	C	D	E	F	G
0	A1	B1	C1	D1	E1	F1	G1
1	A2	B2	C2	D2	E2	F2	G2
2	A3	B3	C3	D3	E3	F3	G3

pd.merge(df4, df3)

	D	H	I	J	A	B	C	E	F	G
0	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
3	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
4	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4와 df5 두 개의 자료는 'D'컬럼을 공유하고 있고 'D'컬럼 기준으로 보면 다대다대응(M:M) 관계이다. df4과 df5을 pd.merge(df4, df5)로 병합하면❶ 일대일대응(M:M) 관계의 행으로 df4과 df5 각각의 컬럼을 공유하게 된다.



df4; df5; pd.merge(df4, df5) ❶

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

	D	H	I	J	A	B	C	E	F	G
0	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
3	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
4	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
5	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
6	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
7	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
8	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4를 기준으로(how='left') df5와 결합하게 되면 df4의 'D' 열의 모든 값을 보여주게 된다. df4의 'D열의 D0에 해당되는 행이 생성된다.

df4; df5; pd.merge(df4, df5, how = 'left') ❶

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	C	D	E	F	G	H	I	J
0	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
1	A1	B1	C1	D1	E1	F1	G1	H1	I1	J1
2	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
3	A2	B2	C2	D2	E2	F2	G2	H2	I2	J2
4	A3	B3	C3	D3	E3	F3	G3	H3	I3	J3

	D	H	I	J	A	B	C	E	F	G
0	D0	H0	I0	J0	NaN	NaN	NaN	NaN	NaN	NaN
1	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
2	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
3	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
4	D1	H1	I1	J1	A1	B1	C1	E1	F1	G1
5	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
6	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
7	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
8	D2	H2	I2	J2	A2	B2	C2	E2	F2	G2
9	D3	H3	I3	J3	A3	B3	C3	E3	F3	G3

df4는 df5의 일부 컬럼으로 만든 df6❶ 두 개의 자료는 'D'와 'H'컬럼을 공유하고 있는 다대다대응(M:M) 관계이다. df4과 df6을 'D'와 'H'컬럼을 기준으로(on =['D', 'H']) pd.merge(df4, df6)로 병합하면 공통된 'I'컬럼을 갖게 되는데 접두사를(suffixes=('_left', '_right')) 붙여 'I' 컬럼을 각각 공유한다❷.

```
df6= df5[['A', 'B', 'D', 'H', 'I']]                                                    ❶
df4; df6; pd.merge(df4, df6, on =['D', 'H'], suffixes=('_left', '_right'))             ❷
```

	D	H	I	J
0	D0	H0	I0	J0
1	D1	H1	I1	J1
1	D1	H1	I1	J1
2	D2	H2	I2	J2
2	D2	H2	I2	J2
3	D3	H3	I3	J3

	A	B	D	H	I
0	A1	B1	D1	H1	I1
1	A1	B1	D1	H1	I1
2	A2	B2	D2	H2	I2
3	A2	B2	D2	H2	I2
4	A3	B3	D3	H3	I3

	D	H	I_left	J	A	B	I_right
0	D1	H1	I1	J1	A1	B1	I1
1	D1	H1	I1	J1	A1	B1	I1
2	D1	H1	I1	J1	A1	B1	I1
3	D1	H1	I1	J1	A1	B1	I1
4	D2	H2	I2	J2	A2	B2	I2
5	D2	H2	I2	J2	A2	B2	I2
6	D2	H2	I2	J2	A2	B2	I2
7	D2	H2	I2	J2	A2	B2	I2
8	D3	H3	I3	J3	A3	B3	I3