단원 07
데이터프레임 정보와 검색 활용

인공지능소프트웨어학과

강환수 교수

7.1 데이터프레임 정보와 조건 검색

-

데이터프레임 준비

- **파일 '2016년졸음운전.csv'을 읽어 데이터프레임을 준비**
  - 파일의 첫 열을 데이터프레임의 인덱스로 사용하기 위해 옵션 index_col=0을 지정해 읽기

```
import pandas as pd

df = pd.read_csv('data/2016년졸음운전.csv',
                 encoding='euc-kr', index_col=0)
df
```

✓ 0.8s

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df.describe()
```

✓ 0.0s Python

	사고(건)	사망(명)	부상(명)
count	12.000000	12.000000	12.000000
mean	202.750000	8.166667	408.250000
std	23.014324	3.186144	60.051682
min	168.000000	4.000000	328.000000
25%	186.000000	5.750000	363.500000
50%	200.000000	7.000000	402.500000
75%	219.500000	10.500000	441.250000
max	239.000000	13.000000	522.000000

```
df.dtypes
```

✓ 0.0s

```
사고(건)    int64
사망(명)    int64
부상(명)    int64
dtype: object
```

```
df.T
```

✓ 0.0s Python

연월	2016년1월	2016년2월	2016년3월	2016년4월	2016년5월	2016년6월	2016년7월	2016년8월	2016년9월	2016년10월	2016년11월	2016년12월
사고(건)	192	174	217	216	239	200	227	230	187	183	200	168
사망(명)	5	6	7	7	13	12	9	7	13	10	5	4
부상(명)	387	328	435	419	522	362	460	490	347	367	418	364

```
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Index: 12 entries, 2016년1월 to 2016년12월
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   사고(건)   12 non-null     int64
 1   사망(명)   12 non-null     int64
 2   부상(명)   12 non-null     int64
dtypes: int64(3)
memory usage: 384.0+ bytes
```

데이터프레임 합 계산

- **df.sum() df.sum(axis=0)**
  - 모든 열의 합
  - 축 axis=0에 따라 이동(위에서 아래로 수직으로)하며 합
- **df.sum(axis=1)**
  - 축 axis=1에 따라 이동(왼쪽에서 오른쪽으로 수평으로)하며 합 구하기
  - 모든 행의 합

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df.sum()
✓ 0.0s
사고(건)     2433
사망(명)       98
부상(명)     4899
dtype: int64
```

```
df.sum(axis=0)
✓ 0.0s
사고(건)     2433
사망(명)       98
부상(명)     4899
dtype: int64
```

```
df.sum(axis=1)
✓ 0.0s
연월
2016년1월      584
2016년2월      508
2016년3월      659
2016년4월      642
2016년5월      774
2016년6월      574
2016년7월      696
2016년8월      727
2016년9월      547
2016년10월     560
2016년11월     623
2016년12월     536
dtype: int64
```

데이터프레임 합 계산

df.cumsum(axis= 0 | 1)

- 함수 cumsum()
  - 누적 합을 구하는 함수
  - df.cumsum(axis=0)
    - 위에서 아래로 수직으로 누적 합을 구하기
  - df.cumsum(axis=1)
    - 왼쪽에서 오른쪽으로 수평으로 누적 합

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df.cumsum(axis=0)
```
✓ 0.0s

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	366	11	715
2016년3월	583	18	1150
2016년4월	799	25	1569
2016년5월	1038	38	2091
2016년6월	1238	50	2453
2016년7월	1465	59	2913
2016년8월	1695	66	3403
2016년9월	1882	79	3750
2016년10월	2065	89	4117
2016년11월	2265	94	4535
2016년12월	2433	98	4899

```
df.cumsum(axis=1)
```
✓ 0.0s

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	197	584
2016년2월	174	180	508
2016년3월	217	224	659
2016년4월	216	223	642
2016년5월	239	252	774
2016년6월	200	212	574
2016년7월	227	236	696
2016년8월	230	237	727
2016년9월	187	200	547
2016년10월	183	193	560
2016년11월	200	205	623
2016년12월	168	172	536

데이터프레임 평균 계산

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

- **df.mean() df.mean(axis=0)**
  - 모든 열의 평균을 구하기
  - 축 axis=0에 따라 이동(위에서 아래로 수직으로)하며 평균
- **df.mean(axis=1)**
  - 축 axis=1에 따라 이동(왼쪽에서 오른쪽으로 수평으로)하며 평균
  - 모든 행의 평균을 구하기
- **df.mean(axis=None)**
  - 전체 평균

```
df.mean()
```

```
사고(건)     202.750000
사망(명)       8.166667
부상(명)     408.250000
dtype: float64
```

```
df.mean(axis=0)
```

```
사고(건)     202.750000
사망(명)       8.166667
부상(명)     408.250000
dtype: float64
```

```
df.mean(axis=1)
```

```
연월
2016년1월     194.666667
2016년2월     169.333333
2016년3월     219.666667
2016년4월     214.000000
2016년5월     258.000000
2016년6월     191.333333
2016년7월     232.000000
2016년8월     242.333333
2016년9월     182.333333
2016년10월    186.666667
2016년11월    207.666667
2016년12월    178.666667
dtype: float64
```

```
df.mean(axis=None)
```

✓ 0.0s

```
206.38888888888889
```

데이터프레임 최대 최소 계산

- **df.max(), df.max(axis=0) 모든 열의 최댓값**
  - 축 axis=0에 따라 이동(위에서 아래로 수직으로)하며 최댓값
- **df.max(axis=1)**
  - 축 axis=1에 따라 이동(왼쪽에서 오른쪽으로 수평으로)하며 최댓값
  - 모든 행의 최댓값
- **df.max(axis=None)**
  - 전체에서 최댓값

- **df.min(), df.min(axis=0) 모든 열의 최솟값**
  - 축 axis=0에 따라 이동(위에서 아래로 수직으로)하며 최솟값
  - 모든 열의 최솟값
- **df.min(axis=1)**
  - 축 axis=1에 따라 이동(왼쪽에서 오른쪽으로 수평으로)하며 최솟값
  - 모든 행의 최솟값
- **df.min(axis=None)**
  - 전체에서 최솟값

```
df.max()
```

```
사고(건)    239
사망(명)     13
부상(명)    522
dtype: int64
```

```
df.max(axis=0)
```

```
사고(건)    239
사망(명)     13
부상(명)    522
dtype: int64
```

```
df.max(axis=1)
```

```
연월
2016년1월    387
2016년2월    328
2016년3월    435
2016년4월    419
2016년5월    522
2016년6월    362
2016년7월    460
2016년8월    490
2016년9월    347
2016년10월   367
2016년11월   418
2016년12월   364
dtype: int64
```

```
df.max(axis=None)
```

✓ 0.0s

```
522
```

```
df.min()
```

```
사고(건)    168
사망(명)      4
부상(명)    328
dtype: int64
```

```
df.min(axis=0)
```

```
사고(건)    168
사망(명)      4
부상(명)    328
dtype: int64
```

```
df.min(axis=1)
```

```
연월
2016년1월      5
2016년2월      6
2016년3월      7
2016년4월      7
2016년5월     13
2016년6월     12
2016년7월      9
2016년8월      7
2016년9월     13
2016년10월    10
2016년11월     5
2016년12월     4
dtype: int64
```

```
df.min(axis=None)
```

✓ 0.0s

```
4
```

데이터프레임 조회

- **df[['사망(명)', '부상(명)']].sum(axis=1)**
  - 모든 행의 '사망(명)'과 '부상(명)'의 합

- **df.iloc[:, 1:].sum(axis=1)**
  - 모든 행의 '사망(명)'과 '부상(명)'의 합

```
df[['사망(명)', '부상(명)']].sum(axis=1)
```

```
연월
2016년1월     392
2016년2월     334
2016년3월     442
2016년4월     426
2016년5월     535
2016년6월     374
2016년7월     469
2016년8월     497
2016년9월     360
2016년10월    377
2016년11월    423
2016년12월    368
dtype: int64
```

```
df.iloc[:, 1:].sum(axis=1)
```

```
연월
2016년1월     392
2016년2월     334
2016년3월     442
2016년4월     426
2016년5월     535
2016년6월     374
2016년7월     469
2016년8월     497
2016년9월     360
2016년10월    377
2016년11월    423
2016년12월    368
dtype: int64
```

새로운 열 추가

- 새로운 df2를 만든 후
- '사망(명)'+'부상(명)'의 합을 새로운 '재해자수' 열에 대입

- df2.loc['총계'] = df2.sum(axis=0)
  - 새로운 행 '총계' 생성
  - 모든 열의 합을 추가
  - axis=0는 생략 가능

```
df2 = df.copy()
df2['재해자수'] = df[['사망(명)', '부상(명)']].sum(axis=1)
df2
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년1월	192	5	387	392
2016년2월	174	6	328	334
2016년3월	217	7	435	442
2016년4월	216	7	419	426
2016년5월	239	13	522	535
2016년6월	200	12	362	374
2016년7월	227	9	460	469
2016년8월	230	7	490	497
2016년9월	187	13	347	360
2016년10월	183	10	367	377
2016년11월	200	5	418	423
2016년12월	168	4	364	368

```
df2.sum()
```

```
사고(건)     2433
사망(명)       98
부상(명)     4899
재해자수       4997
dtype: int64
```

```
df2.loc['총계'] = df2.sum(axis=0)
df2
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년1월	192	5	387	392
2016년2월	174	6	328	334
2016년3월	217	7	435	442
2016년4월	216	7	419	426
2016년5월	239	13	522	535
2016년6월	200	12	362	374
2016년7월	227	9	460	469
2016년8월	230	7	490	497
2016년9월	187	13	347	360
2016년10월	183	10	367	377
2016년11월	200	5	418	423
2016년12월	168	4	364	368
총계	4866	196	9798	9994

블리안 인덱싱

- 데이터프레임 원소의 비교 논리 값은 다음으로 반환
  - 원소 값이 10 초과인 자료가 있는 부분은 True가 표시
- 위의 블리안 인덱싱 조건을 df2[조건] 내부에 기술
  - True인 원소만 표시
  - False인 원소는 NaN이 표시
- df2[[True, False, ..., False]]
  - True인 행만 표시
  - 인위적으로 True와 False를 13개 만들어 실행
    - True인 처음 6개의 행만 표시

```
df2 > 10
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년1월	True	False	True	True
2016년2월	True	False	True	True
2016년3월	True	False	True	True
2016년4월	True	False	True	True
2016년5월	True	True	True	True
2016년6월	True	True	True	True
2016년7월	True	False	True	True
2016년8월	True	False	True	True
2016년9월	True	True	True	True
2016년10월	True	False	True	True
2016년11월	True	False	True	True
2016년12월	True	False	True	True
총계	True	True	True	True

```
df2[df2 > 10]
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년1월	192	NaN	387	392
2016년2월	174	NaN	328	334
2016년3월	217	NaN	435	442
2016년4월	216	NaN	419	426
2016년5월	239	13.0	522	535
2016년6월	200	12.0	362	374
2016년7월	227	NaN	460	469
2016년8월	230	NaN	490	497
2016년9월	187	13.0	347	360
2016년10월	183	NaN	367	377
2016년11월	200	NaN	418	423
2016년12월	168	NaN	364	368
총계	4866	196.0	9798	9994

```
df2[ [True]*6 + [False]*7 ]
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년1월	192	5	387	392
2016년2월	174	6	328	334
2016년3월	217	7	435	442
2016년4월	216	7	419	426
2016년5월	239	13	522	535
2016년6월	200	12	362	374

조건과 isin()

- ‘재해자수’ 열의 비교 논리 값은 다음으로 반환
  - 13개의 논리 값
- ‘재해자수'가 400명을 초과한 행을 표시
- 열 이름 ’사망(명)‘이 7인 논리 값의 시리즈가 반환된
- 함수 Series.isin([value0, value1, value2, ...])
  - 시리즈의 각 요소가 전달된 값 시퀀스의 요소와 정확히 일치하는지 여부를 보여주는 논리 값 시리즈를 반환
  - df['사망(명)'] == 7
    - 열 이름 ’사망(명)‘이 7인 논리 값의 시리즈가 반환

```
df2['재해자수'] > 400
```

```
연월
2016년1월     False
2016년2월     False
2016년3월      True
2016년4월      True
2016년5월      True
2016년6월     False
2016년7월      True
2016년8월      True
2016년9월     False
2016년10월    False
2016년11월     True
2016년12월    False
총계            True
Name: 재해자수, dtype: bool
```

```
df2[df2['재해자수'] > 400]
```

연월	사고(건)	사망(명)	부상(명)	재해자수
2016년3월	217	7	435	442
2016년4월	216	7	419	426
2016년5월	239	13	522	535
2016년7월	227	9	460	469
2016년8월	230	7	490	497
2016년11월	200	5	418	423
총계	2433	98	4899	4997

```
df['사망(명)'] == 7
```

```
연월
2016년1월     False
2016년2월     False
2016년3월      True
2016년4월      True
2016년5월     False
2016년6월     False
2016년7월     False
2016년8월      True
2016년9월     False
2016년10월    False
2016년11월    False
2016년12월    False
Name: 사망(명), dtype: bool
```

```
df['사망(명)'].isin([7])
```

```
연월
2016년1월     False
2016년2월     False
2016년3월      True
2016년4월      True
2016년5월     False
2016년6월     False
2016년7월     False
2016년8월      True
2016년9월     False
2016년10월    False
2016년11월    False
2016년12월    False
Name: 사망(명), dtype: bool
```

isin(), str.contains()

- '사망(명)'이 7명 또는 10명인 모든 행을 표시
- 사망(명)은 7 또는 10이거나 부상(명)은 350을 초과하는 행을 모두 출력
- 행 명이나 열 명의 값이 문자열
  - 문자열에 포함된 내용은 함수 contains('부분문자열')로 검색

```
df['사망(명)'].isin([7, 10])
```

```
연월
2016년1월     False
2016년2월     False
2016년3월      True
2016년4월      True
2016년5월     False
2016년6월     False
2016년7월     False
2016년8월      True
2016년9월     False
2016년10월     True
2016년11월    False
2016년12월    False
Name: 사망(명), dtype: bool
```

```
df[ df['사망(명)'].isin([7, 10]) ]
```

연월	사고(건)	사망(명)	부상(명)
2016년3월	217	7	435
2016년4월	216	7	419
2016년8월	230	7	490
2016년10월	183	10	367

```
df[ (df['사망(명)'].isin([7, 10])) | (df['부상(명)'] > 350) ]
```

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df[ df.index.str.contains('3') ]
```

연월	사고(건)	사망(명)	부상(명)
2016년3월	217	7	435

논리 첨자로 값 대입

- **df3의 원소 값이 10보다 큰 모든 수를**
  - 음수로 수정

```
df3 = df.copy()
df3
```

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df3[df3 > 10] = -df3
df3
```

연월	사고(건)	사망(명)	부상(명)
2016년1월	-192	5	-387
2016년2월	-174	6	-328
2016년3월	-217	7	-435
2016년4월	-216	7	-419
2016년5월	-239	-13	-522
2016년6월	-200	-12	-362
2016년7월	-227	9	-460
2016년8월	-230	7	-490
2016년9월	-187	-13	-347
2016년10월	-183	10	-367
2016년11월	-200	5	-418
2016년12월	-168	4	-364

함수 df.query(조건)

- 조건을 만족하는 행을 표시

```
df = pd.DataFrame({'A': range(1, 6),
                   'B': range(6, -3, -2),
                   'C': range(10, 5, -1),
                   'D clmns': range(3, -2, -1),
                   'E': ['python', 'java', 'python', 'go', 'python']})
df
```

	A	B	C	D clmns	E
0	1	6	10	3	python
1	2	4	9	2	java
2	3	2	8	1	python
3	4	0	7	0	go
4	5	-2	6	-1	python

```
df.query('A > B')
```

	A	B	C	D clmns	E
2	3	2	8	1	python
3	4	0	7	0	go
4	5	-2	6	-1	python

```
df[df.A > df.B]
```

	A	B	C	D clmns	E
2	3	2	8	1	python
3	4	0	7	0	go
4	5	-2	6	-1	python

열 이름을 바로 기술 가능

- **열 E가 "python"인 행을 반환**
  - 외부와 내부의 따옴표를 작은 따옴표와 큰 따옴표로 구분
- **열 E가 "python"이 아닌 행을 반환**
  - 외부와 내부의 따옴표를 구분

```
df.query('C == 10')
```

	A	B	C	D clmns	E
0	1	6	10	3	python

```
df.query('A > 0')
```

	A	B	C	D clmns	E
0	1	6	10	3	python
1	2	4	9	2	java
2	3	2	8	1	python
3	4	0	7	0	go
4	5	-2	6	-1	python

```
df.query('E == "python"')
```

	A	B	C	D clmns	E
0	1	6	10	3	python
2	3	2	8	1	python
4	5	-2	6	-1	python

```
df.query("E != 'python'")
```

	A	B	C	D clmns	E
1	2	4	9	2	java
3	4	0	7	0	go

열명 중간에 빈 문자열이 있는 경우

- 반드시 열 명을 백쿼트(back quote, Tab 키 위의 키) `D clmns`를 사용
- 만일 “D clmns”으로 사용
  - 열 A의 값 문자열로 “D clums”를 찾으므로 오류는 발생하지 않으나 원하는 결과가 표시되지 않음

```
df.query('A == `D clmns`')
```

	A	B	C	D clmns	E
1	2	4	9	2	java

```
df.query('A == "D clmns"')
```

A	B	C	D clmns	E

```
df[df.A == df["D clmns"]]
```

	A	B	C	D clmns	E
1	2	4	9	2	java

```
df.query('A > 3 | B > 8')
```

	A	B	C	D clmns	E
3	4	0	7	0	go
4	5	-2	6	-1	python

```
df.query('A > 2 & C < 8')
```

	A	B	C	D clmns	E
3	4	0	7	0	go
4	5	-2	6	-1	python

```
df.query("E in ['python', 'go']")
```

	A	B	C	D clmns	E
0	1	6	10	3	python
2	3	2	8	1	python
3	4	0	7	0	go
4	5	-2	6	-1	python

df.query() 내부의 질의 문자열 내부에서 변수를 사용

- @변수명을 사용
- f 문자열로 사용이 가능
  - f 문자열 내부에서 변수 num은 {num}으로 사용이 가능

- 열 유형이 문자열
  - 열명.str.contains('부분문자열')을 사용
    - '부분문자열'이 포함된 행의 데이터프레임을 반환
- 열의 내용이 문자열
  - 열명.str.startswith('시작문자열')을 사용
    - '시작문자열'로 시작되는 행을 반환
- 열의 내용이 문자열
  - 열명.str.endswith('끝나는문자열')을 사용
    - '끝나는문자열'로 종료되는 행의 데이터프레임을 반환

```
num = 3
df.query('B > @num')
```

	A	B	C	D clmns	E
0	1	6	10	3	python
1	2	4	9	2	java

```
num = 3
df.query(f'B > {num}')
```

	A	B	C	D clmns	E
0	1	6	10	3	python
1	2	4	9	2	java

```
s = "E.str.contains('py')"
df.query(s)
```

	A	B	C	D clmns	E
0	1	6	10	3	python
2	3	2	8	1	python
4	5	-2	6	-1	python

```
s = 'E.str.startswith("g")'
df.query(s)
```

	A	B	C	D clmns	E
3	4	0	7	0	go

```
s = "E.str.endswith('on')"
df.query(s)
```

	A	B	C	D clmns	E
0	1	6	10	3	python
2	3	2	8	1	python
4	5	-2	6	-1	python

7.2 데이터프레임 정렬과 수정

-

df.head(n)

- **df.tail(n)**

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df.head()
```

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522

```
df.head(7)
```

연월	사고(건)	사망(명)	부상(명)
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460

```
df.tail()
```

연월	사고(건)	사망(명)	부상(명)
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

```
df.tail(6)
```

연월	사고(건)	사망(명)	부상(명)
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364

행 제목(레이블)으로 정렬

- **axis=0**
  - ascending = false
    - 내림차순

```
df.sort_index()
```

연월	사고(건)	사망(명)	부상(명)
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347

```
df.sort_index(axis=0)
```

연월	사고(건)	사망(명)	부상(명)
2016년10월	183	10	367
2016년11월	200	5	418
2016년12월	168	4	364
2016년1월	192	5	387
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년5월	239	13	522
2016년6월	200	12	362
2016년7월	227	9	460
2016년8월	230	7	490
2016년9월	187	13	347

```
df.sort_index(ascending=False)
```

연월	사고(건)	사망(명)	부상(명)
2016년9월	187	13	347
2016년8월	230	7	490
2016년7월	227	9	460
2016년6월	200	12	362
2016년5월	239	13	522
2016년4월	216	7	419
2016년3월	217	7	435
2016년2월	174	6	328
2016년1월	192	5	387
2016년12월	168	4	364
2016년11월	200	5	418
2016년10월	183	10	367

열 제목(레이블)으로 정렬

- **axis=1**
  - ascending = false
    - 내림차순

```
df.sort_index(axis=1)
```

연월	부상(명)	사고(건)	사망(명)
2016년1월	387	192	5
2016년2월	328	174	6
2016년3월	435	217	7
2016년4월	419	216	7
2016년5월	522	239	13
2016년6월	362	200	12
2016년7월	460	227	9
2016년8월	490	230	7
2016년9월	347	187	13
2016년10월	367	183	10
2016년11월	418	200	5
2016년12월	364	168	4

```
df.sort_index(axis=1, ascending=False)
```

연월	사망(명)	사고(건)	부상(명)
2016년1월	5	192	387
2016년2월	6	174	328
2016년3월	7	217	435
2016년4월	7	216	419
2016년5월	13	239	522
2016년6월	12	200	362
2016년7월	9	227	460
2016년8월	7	230	490
2016년9월	13	187	347
2016년10월	10	183	367
2016년11월	5	200	418
2016년12월	4	168	364

특정한 열의 내용으로 정렬

df.sort_values(by = '열명')

- **인자 by**
  - 열 하나 또는 열 목록 가능
- **인자 ascending = False**
  - 내림차순

```
df.sort_values(by='사망(명)')
```

연월	사고(건)	사망(명)	부상(명)
2016년12월	168	4	364
2016년1월	192	5	387
2016년11월	200	5	418
2016년2월	174	6	328
2016년3월	217	7	435
2016년4월	216	7	419
2016년8월	230	7	490
2016년7월	227	9	460
2016년10월	183	10	367
2016년6월	200	12	362
2016년5월	239	13	522
2016년9월	187	13	347

```
df.sort_values(by=['사망(명)', '부상(명)'])
```

연월	사고(건)	사망(명)	부상(명)
2016년12월	168	4	364
2016년1월	192	5	387
2016년11월	200	5	418
2016년2월	174	6	328
2016년4월	216	7	419
2016년3월	217	7	435
2016년8월	230	7	490
2016년7월	227	9	460
2016년10월	183	10	367
2016년6월	200	12	362
2016년9월	187	13	347
2016년5월	239	13	522

데이터프레임 행 값으로 정렬

df.sort_values(by = ' 행명', axis=1)

- 행의 값에 따라 열의 순서가 정해짐
  - 인자 ascending = False
    - 내림차순

```
df.sort_values(by='2016년12월', axis=1)
```

연월	사망(명)	사고(건)	부상(명)
2016년1월	5	192	387
2016년2월	6	174	328
2016년3월	7	217	435
2016년4월	7	216	419
2016년5월	13	239	522
2016년6월	12	200	362
2016년7월	9	227	460
2016년8월	7	230	490
2016년9월	13	187	347
2016년10월	10	183	367
2016년11월	5	200	418
2016년12월	4	168	364

```
df.sort_values(by=['2016년12월'], axis=1, ascending=False)
```

연월	부상(명)	사고(건)	사망(명)
2016년1월	387	192	5
2016년2월	328	174	6
2016년3월	435	217	7
2016년4월	419	216	7
2016년5월	522	239	13
2016년6월	362	200	12
2016년7월	460	227	9
2016년8월	490	230	7
2016년9월	347	187	13
2016년10월	367	183	10
2016년11월	418	200	5
2016년12월	364	168	4

데이터프레임 생성과 열명 지정

```
import numpy as np
import pandas as pd

dates = pd.date_range("20240102", periods=6)
df = pd.DataFrame(np.arange(24).reshape(6, 4), index=dates)
df
```

	0	1	2	3
2024-01-02	0	1	2	3
2024-01-03	4	5	6	7
2024-01-04	8	9	10	11
2024-01-05	12	13	14	15
2024-01-06	16	17	18	19
2024-01-07	20	21	22	23

```
df.columns = list('ABCD')
df
```

	A	B	C	D
2024-01-02	0	1	2	3
2024-01-03	4	5	6	7
2024-01-04	8	9	10	11
2024-01-05	12	13	14	15
2024-01-06	16	17	18	19
2024-01-07	20	21	22	23

열 추가

먼저 새로운 열에 삽입할 시리즈를 하나 생성

- **열 'E'에 시리즈 s1을 대입**
  - s1에는 2023-01-07이 없으므로 결측값에는 NaN 값이 삽입

```
s1 = pd.Series([1, 2, 3, 4, 5], index=pd.date_range("20240102", periods=5))
s1
```

```
2024-01-02    1
2024-01-03    2
2024-01-04    3
2024-01-05    4
2024-01-06    5
Freq: D, dtype: int64
```

```
df["E"] = s1
df
```

	A	B	C	D	E
2024-01-02	0	1	2	3	1.0
2024-01-03	4	5	6	7	2.0
2024-01-04	8	9	10	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	NaN

df.loc['행', '열'] = value

행 열 위치의 원소 값 수정

```
df.loc['2024-01-02', 'A'] = 10
df
```

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	5	6	7	2.0
2024-01-04	8	9	10	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	NaN

```
df.at[dates[2], df.columns[2]] = 100
df
```

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	50	6	7	2.0
2024-01-04	8	9	100	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	NaN

```
df.loc[dates[1], 'B'] = 50
df
```

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	50	6	7	2.0
2024-01-04	8	9	10	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	NaN

```
df.at[df.index[-1], df.columns[-1]] = 230
df
```

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	50	6	7	2.0
2024-01-04	8	9	100	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	230.0

df.iloc[m, n] = value

df.iat[m, n] = value

- **df.iloc[:, 4] = 600**
  - 열 첨자 4인 모든 값을 600으로 수정

```
df.iloc[1, 2] = 60
df
```

✓ 0.0s

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	50	60	7	2.0
2024-01-04	8	9	100	11	3.0
2024-01-05	12	13	14	15	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	230.0

```
df.iloc[:, 4] = [500] * len(df)
df
```

✓ 0.0s

	A	B	C	D	E
2024-01-02	10	1	2	3	500.0
2024-01-03	4	50	60	7	500.0
2024-01-04	8	9	100	11	500.0
2024-01-05	12	13	14	150	500.0
2024-01-06	16	17	18	19	500.0
2024-01-07	20	21	22	23	500.0

```
df.iat[3, 3] = 150
df
```

✓ 0.0s

	A	B	C	D	E
2024-01-02	10	1	2	3	1.0
2024-01-03	4	50	60	7	2.0
2024-01-04	8	9	100	11	3.0
2024-01-05	12	13	14	150	4.0
2024-01-06	16	17	18	19	5.0
2024-01-07	20	21	22	23	230.0

```
df.iloc[:, 4] = 600
df
```

✓ 0.0s

	A	B	C	D	E
2024-01-02	10	1	2	3	600.0
2024-01-03	4	50	60	7	600.0
2024-01-04	8	9	100	11	600.0
2024-01-05	12	13	14	150	600.0
2024-01-06	16	17	18	19	600.0
2024-01-07	20	21	22	23	600.0

데이터프레임과 DatetimeIndex 생성

```
import numpy as np
import pandas as pd

dates = pd.date_range("20250302", periods=7, freq='W-MON')
df = pd.DataFrame(np.arange(35).reshape(7, 5), index=dates, columns=list('ABCDE'))
df
```

✓ 0.0s

	A	B	C	D	E
2025-03-03	0	1	2	3	4
2025-03-10	5	6	7	8	9
2025-03-17	10	11	12	13	14
2025-03-24	15	16	17	18	19
2025-03-31	20	21	22	23	24
2025-04-07	25	26	27	28	29
2025-04-14	30	31	32	33	34

```
subdates = pd.date_range("20250307", periods=5, freq='W-MON')
subdates
```

✓ 0.0s

```
DatetimeIndex(['2025-03-10', '2025-03-17', '2025-03-24', '2025-03-31',
               '2025-04-07'],
              dtype='datetime64[ns]', freq='W-MON')
```

시리즈 생성과 열 추가

결측 값을 대입

```
s = pd.Series(np.linspace(10, 17, 5), index=subdates)
s
```

✓ 0.0s

```
2025-03-10    10.00
2025-03-17    11.75
2025-03-24    13.50
2025-03-31    15.25
2025-04-07    17.00
Freq: W-MON, dtype: float64
```

```
df['F'] = s
df
```

✓ 0.2s

	A	B	C	D	E	F
2025-03-03	0	1	2	3	4	NaN
2025-03-10	5	6	7	8	9	10.00
2025-03-17	10	11	12	13	14	11.75
2025-03-24	15	16	17	18	19	13.50
2025-03-31	20	21	22	23	24	15.25
2025-04-07	25	26	27	28	29	17.00
2025-04-14	30	31	32	33	34	NaN

```
df2 = df.copy()
df2.iloc[3, 4] = np.nan
df2.iloc[1, :] = np.nan
df2
```

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	NaN
2025-03-10	NaN	NaN	NaN	NaN	NaN	NaN
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	NaN	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	NaN

함수 isna()

결측 값 검사 함수

- isnull()로도 가능

```
df2
```

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	NaN
2025-03-10	NaN	NaN	NaN	NaN	NaN	NaN
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	NaN	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	NaN

```
pd.isna(df2)
```

	A	B	C	D	E	F
2025-03-03	False	False	False	False	False	True
2025-03-10	True	True	True	True	True	True
2025-03-17	False	False	False	False	False	False
2025-03-24	False	False	False	False	True	False
2025-03-31	False	False	False	False	False	False
2025-04-07	False	False	False	False	False	False
2025-04-14	False	False	False	False	False	True

```
df2.isna()
```

	A	B	C	D	E	F
2025-03-03	False	False	False	False	False	True
2025-03-10	True	True	True	True	True	True
2025-03-17	False	False	False	False	False	False
2025-03-24	False	False	False	False	True	False
2025-03-31	False	False	False	False	False	False
2025-04-07	False	False	False	False	False	False
2025-04-14	False	False	False	False	False	True

```
df2.isnull()
```

	A	B	C	D	E	F
2025-03-03	False	False	False	False	False	True
2025-03-10	True	True	True	True	True	True
2025-03-17	False	False	False	False	False	False
2025-03-24	False	False	False	False	True	False
2025-03-31	False	False	False	False	False	False
2025-04-07	False	False	False	False	False	False
2025-04-14	False	False	False	False	False	True

df.count()

열 마다 결측 값이 없는 개수

- **axis=1**
  - 행마다 결측 값이 없는 개수

```
df2.count()
✓ 0.0s
A    6
B    6
C    6
D    6
E    5
F    4
dtype: int64
```

```
df2.count(axis=1)
✓ 0.0s
2025-03-03    5
2025-03-10    0
2025-03-17    6
2025-03-24    5
2025-03-31    6
2025-04-07    6
2025-04-14    5
Freq: W-MON, dtype: int64
```

함수 value_counts()

df2.열명.value_counts()

- df2.열명.value_counts()
- df2['열명'].value_counts()
  - 열에서 유일한 값들의 출현 빈도수를 표시, nan은 미포함
  - 유형은 Series, 유일 값이 첨자
  - 키워드 인자 dropna=False
    - nan도 포함해서 유일한 값들의 출현 빈도수를 표시
- to_frame()으로 호출
  - 데이터프레임 반환

```
s = df2.E.value_counts()
s
```
✓ 0.0s

```
E
4.0     1
14.0    1
24.0    1
29.0    1
34.0    1
Name: count, dtype: int64
```

```
type(s)
```
✓ 0.0s

```
pandas.core.series.Series
```

```
s[4]
```
✓ 0.0s

```
1
```

```
s.values
```
✓ 0.0s

```
array([1, 1, 1, 1, 1], dtype=int64)
```

```
df2['D'].value_counts()
```
✓ 0.0s

```
D
3.0     1
13.0    1
18.0    1
23.0    1
28.0    1
33.0    1
Name: count, dtype: int64
```

```
df2['D'].value_counts().to_frame()
```
✓ 0.0s

D	count
3.0	1
13.0	1
18.0	1
23.0	1
28.0	1
33.0	1

```
df2.E.value_counts(dropna=False)
```
✓ 0.0s

```
E
NaN     2
4.0     1
14.0    1
24.0    1
29.0    1
34.0    1
Name: count, dtype: int64
```

df2.value_counts()

- 데이터프레임에서 각 개별 행의 빈도를 포함하는 시리즈를 반환
  - 물론 결측값 nan은 제외
    - dropna = False로 포함
  - 모든 행의 값 구성이 다르므로 모든 행이 첨자이며, 값이 모두 1

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	NaN
2025-03-10	NaN	NaN	NaN	NaN	NaN	NaN
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	NaN	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	NaN

```
df2.value_counts(dropna=False)
```

```
A     B     C     D     E     F
0.0   1.0   2.0   3.0   4.0   NaN      1
10.0  11.0  12.0  13.0  14.0  11.75    1
15.0  16.0  17.0  18.0  NaN   13.50    1
20.0  21.0  22.0  23.0  24.0  15.25    1
25.0  26.0  27.0  28.0  29.0  17.00    1
30.0  31.0  32.0  33.0  34.0  NaN      1
NaN   NaN   NaN   NaN   NaN   NaN      1
dtype: int64
```

```
s1 = df2.value_counts()
s1
```

✓ 0.0s

```
A     B     C     D     E     F
10.0  11.0  12.0  13.0  14.0  11.75    1
20.0  21.0  22.0  23.0  24.0  15.25    1
25.0  26.0  27.0  28.0  29.0  17.00    1
Name: count, dtype: int64
```

```
s1.index
```

```
MultiIndex([(10.0, 11.0, 12.0, 13.0, 14.0, 11.75),
            (20.0, 21.0, 22.0, 23.0, 24.0, 15.25),
            (25.0, 26.0, 27.0, 28.0, 29.0,  17.0)],
           names=['A', 'B', 'C', 'D', 'E', 'F'])
```

```
s1.values
```

```
array([1, 1, 1], dtype=int64)
```

함수 dropna()

- **dropna() dropna(axis=0)**
  - NaN인 결측값이 하나라도 있는 행을 제거한 데이터프레임을 반환
  - 기본적으로 제거된 것이 df2에 자동으로 반영되지 않음
- **dropna(axis=1)**
  - NaN인 결측값이 하나라도 있는 열을 제거한 데이터프레임을 반환
    - 다음에서 모든 열이 제거

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	NaN
2025-03-10	NaN	NaN	NaN	NaN	NaN	NaN
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	NaN	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	NaN

```
df2.dropna()
```
✓ 0.0s

	A	B	C	D	E	F
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00

```
df2.dropna(axis=0)
```
✓ 0.0s

	A	B	C	D	E	F
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00

```
df2.dropna(axis=1)
```

2025-03-03
2025-03-10
2025-03-17
2025-03-24
2025-03-31
2025-04-07
2025-04-14

결측값에 값을 대입

- **df2[df2.isna()] = 500**
  - 모든 결측값에 500을 대입

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	NaN
2025-03-10	NaN	NaN	NaN	NaN	NaN	NaN
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	NaN	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	NaN

```
df2[ df2.isna() ] = 500
df2
```

✓ 0.2s

	A	B	C	D	E	F
2025-03-03	0.0	1.0	2.0	3.0	4.0	500.00
2025-03-10	500.0	500.0	500.0	500.0	500.0	500.00
2025-03-17	10.0	11.0	12.0	13.0	14.0	11.75
2025-03-24	15.0	16.0	17.0	18.0	500.0	13.50
2025-03-31	20.0	21.0	22.0	23.0	24.0	15.25
2025-04-07	25.0	26.0	27.0	28.0	29.0	17.00
2025-04-14	30.0	31.0	32.0	33.0	34.0	500.00

결측값을 채우는 함수 fillna()

- 함수 호출 df.fillna(10)의 결과
  - df 결측값을 모두 10으로 채움
  - 함수 fillna()의 인자 inplace=False가 기본
    - df 자체에 수정이 반영되지는 않음
    - 여전히 df는 많은 결측값이 있는 이전과 동일

```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, np.nan],
                   [np.nan, 3, np.nan, 4]],
                  columns=list("ABCD"))
df
```

✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

```
df.fillna(10)
```

✓ 0.0s

	A	B	C	D
0	10.0	2.0	10.0	0.0
1	3.0	4.0	10.0	1.0
2	10.0	10.0	10.0	10.0
3	10.0	3.0	10.0	4.0

```
df
```

✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

fillna(value, limit=n, axis= 0 | 1)

열(axis= 0)이나 행(axis= 1)에서 결측 값에 value를 채우는 최대 수

- df.fillna('missing', axis=0), df.fillna('missing')
  - 모든 결측값에 문자열 'missing'을 채움
  - 키워드 인자 limit이 없으면 인자 axis=0 | 1은 별 의미가 없음
  - 키워드 인자 limit=1
    - 각 열에서 최대 1개만 채움
    - C 열의 첫 행만 missing으로 채워짐
- 다음은 동일 기능
  - df.fillna('missing', limit=1)
  - df.fillna('missing', limit=1, axis=0)

```
df
✓ 0.0s
```

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

```
df.fillna('missing', axis=0)
✓ 0.0s
```

	A	B	C	D
0	missing	2.0	missing	0.0
1	3.0	4.0	missing	1.0
2	missing	missing	missing	missing
3	missing	3.0	missing	4.0

```
df.fillna('missing', limit=1, axis=1)
```

	A	B	C	D
0	missing	2.0	NaN	0.0
1	3.0	4.0	missing	1.0
2	missing	NaN	NaN	NaN
3	missing	3.0	NaN	4.0

```
df.fillna('missing', limit=1)
✓ 0.0s
```

	A	B	C	D
0	missing	2.0	missing	0.0
1	3.0	4.0	NaN	1.0
2	NaN	missing	NaN	missing
3	NaN	3.0	NaN	4.0

```
df.fillna('value', axis=1)
```

	A	B	C	D
0	value	2.0	value	0.0
1	3.0	4.0	value	1.0
2	value	value	value	value
3	value	3.0	value	4.0

fillna(value)

value에 스칼라, dict, Series 또는 DataFrame 대입 가능, 리스트는 대입 불가능

- values = {"A": 100, "B": 200, "C": 300, "D": 400}
  - 채워 넣는 값으로 dict를 사용한 예
  - 사전의 키는 열 명이며, 값은 결측값을 대체할 값
  - 인자 value=values로 호출
    - 각각 열 별로 채워진 값이 다름
- df.fillna(value=values, limit=2)
  - 키워드 인자 limit=2
    - 각 열 별로 채워지는 수를 2개로 제한

```
df
```

✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

```
values = {"A": 100, "B": 200, "C": 300, "D": 400}
df.fillna(value=values)
```

	A	B	C	D
0	100.0	2.0	300.0	0.0
1	3.0	4.0	300.0	1.0
2	100.0	200.0	300.0	400.0
3	100.0	3.0	300.0	4.0

```
df.fillna(value=values, limit=2)
```

	A	B	C	D
0	100.0	2.0	300.0	0.0
1	3.0	4.0	300.0	1.0
2	100.0	200.0	NaN	400.0
3	NaN	3.0	NaN	4.0

fillna(value)

시리즈와 데이터프레임으로 대입

```
df
```

✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

- **df2에 열 D가 없으므로 열 D인 결측값은 대체되지 못함**

```
s = pd.Series([100, 200, 400], index=list('ACD'))
s
```

✓ 0.0s

```
A    100
C    200
D    400
dtype: int64
```

```
df.fillna(value=s, limit=1)
```

✓ 0.0s

	A	B	C	D
0	100.0	2.0	200.0	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	400.0
3	NaN	3.0	NaN	4.0

```
df2 = pd.DataFrame(-np.ones((4, 4)), columns=list("ABCE"))
df2
```

	A	B	C	E
0	-1.0	-1.0	-1.0	-1.0
1	-1.0	-1.0	-1.0	-1.0
2	-1.0	-1.0	-1.0	-1.0
3	-1.0	-1.0	-1.0	-1.0

```
df.fillna(df2)
```

	A	B	C	D
0	-1.0	2.0	-1.0	0.0
1	3.0	4.0	-1.0	1.0
2	-1.0	-1.0	-1.0	NaN
3	-1.0	3.0	-1.0	4.0

결측값을 위한 함수 fillna(method='ffill')

향후 obj.ffill()와 obj.bfill() 메소드로 대체될 함수

- **df.fillna(method='ffill')**
  - 결측값을 달리 지정하지 않고
  - 바로 열의 바로 이전에 있는 내용으로 채움
    - 마지막 유효한 값을 다음 유효한 값으로 전파하여 채우는 방식
  - 행 0, 열 A인 원소는 이전 값이 없으므로 채워지지 못함
    - 열 C도 마찬가지

```
df
✓ 0.0s
```

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

- **df.fillna(method='ffill', limit=1)**
  - 키워드 인자 limit=1로 이전 값으로 최대 하나만 채움

- **df.fillna(method='bfill', limit=1)**
  - method='bfill'
    - 데이터프레임에서 바로 열의 이후에 있는 내용으로 채우려면
    - 즉, 결측값을 채우기 위해 다음 유효한 관찰 값을 사용하여 결측값을 채움
  - 키워드 인자 limit=1로 이후 값으로 최대 하나만 채움

```
df.fillna(method='ffill')
✓ 0.0s
C:\Users\PC\AppData\Local\Temp\ipykernel_42
df.fillna(method='ffill')
```

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	3.0	4.0	NaN	1.0
3	3.0	3.0	NaN	4.0

```
df.fillna(method='ffill', limit=1)
✓ 0.0s
C:\Users\PC\AppData\Local\Temp\ipykernel_42
df.fillna(method='ffill', limit=1)
```

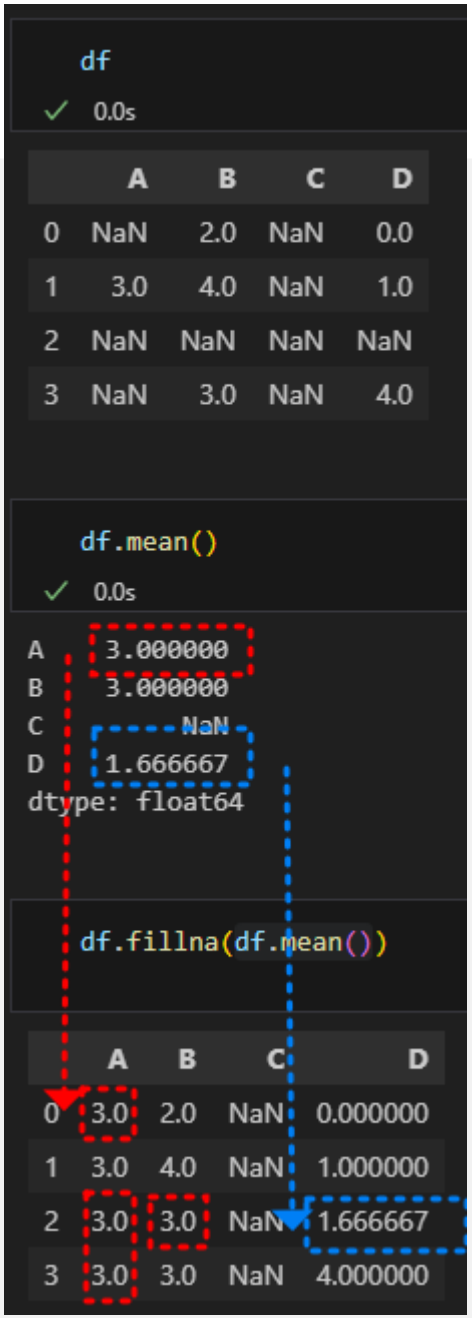
	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	3.0	4.0	NaN	1.0
3	NaN	3.0	NaN	4.0

```
df.fillna(method='bfill', limit=1)
✓ 0.2s
C:\Users\PC\AppData\Local\Temp\ipykernel_42
df.fillna(method='bfill', limit=1)
```

	A	B	C	D
0	3.0	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	3.0	NaN	4.0
3	NaN	3.0	NaN	4.0

df.fillna(df.mean())

- **대체할 값으로 각 열의 평균값을 넣는 방법**
  - C 열은 모두 결측값으로 대체하지 못함



	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

	A	B	C	D
0	3.0	2.0	NaN	0.000000
1	3.0	4.0	NaN	1.000000
2	3.0	3.0	NaN	1.666667
3	3.0	3.0	NaN	4.000000

df.fillna(df.mean()['A':'B'])

- **df.fillna(df.mean()['A':'B'])**
  - 대체할 값으로 각각 열 A와 B의 평균값을 넣는 방법
  - C 열은 모두 결측값으로 대체하지 못하며
  - 열 D는 지정하지 않았으므로 대체하지 못함

- **df.fillna(df.mean()['D'])**
  - 모든 결측값을 열 D의 평균값으로 채움

```
df.fillna(df.mean()['A':'B'])
```
✓ 0.0s

	A	B	C	D
0	3.0	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	3.0	3.0	NaN	NaN
3	3.0	3.0	NaN	4.0

```
df
```
✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

```
df.fillna(df.mean()['D'])
```
✓ 0.0s

	A	B	C	D
0	1.666667	2.000000	1.666667	0.000000
1	3.000000	4.000000	1.666667	1.000000
2	1.666667	1.666667	1.666667	1.666667
3	1.666667	3.000000	1.666667	4.000000

df.fillna(df.mean()['D'], limit=1)

- df.fillna(df.mean()['D'], limit=1)
  df.fillna(df.mean()['D'], limit=1, axis=0)
  - 인자 limit=1로 각 열에서 1개만 채움
- df.fillna(df.mean()['D'], limit=1, axis=1)
  - 인자 axis=1로 각 행에서 최대 1개만 채움

```
df
```

✓ 0.0s

	A	B	C	D
0	NaN	2.0	NaN	0.0
1	3.0	4.0	NaN	1.0
2	NaN	NaN	NaN	NaN
3	NaN	3.0	NaN	4.0

```
df.fillna(df.mean()['D'], limit=1)
```

	A	B	C	D
0	1.666667	2.000000	1.666667	0.000000
1	3.000000	4.000000	NaN	1.000000
2	NaN	1.666667	NaN	1.666667
3	NaN	3.000000	NaN	4.000000

```
df.fillna(df.mean()['D'], limit=1, axis=0)
```

	A	B	C	D
0	1.666667	2.000000	1.666667	0.000000
1	3.000000	4.000000	NaN	1.000000
2	NaN	1.666667	NaN	1.666667
3	NaN	3.000000	NaN	4.000000

```
df.fillna(df.mean()['D'], limit=1, axis=1)
```

	A	B	C	D
0	1.666667	2.0	NaN	0.0
1	3.000000	4.0	1.666667	1.0
2	1.666667	NaN	NaN	NaN
3	1.666667	3.0	NaN	4.0