

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

▼ 기술 통계 계산과 요약

```
df = DataFrame([[1.4,np.nan],[7.1,-4.5],[np.nan,np.nan],[0.75,-13.]],
                index = ['a','b','c','d'],
                columns = ['one','two'])
```

df



	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-13.0

df.sum()



```
one      9.25
two     -17.50
dtype: float64
```

df.sum(axis=1)



```
a      1.40
b      2.60
c      0.00
d     -12.25
dtype: float64
```

df.idxmax()



```
one      b
two      b
dtype: object
```

#skipna : 누락된 값을 제외할 것인지 정하는 옵션
df.mean(axis=1, skipna = False)



```
a      NaN
b      1.300
c      NaN
d     -6.125
dtype: float64
```

```
df.idxmax()    # 가장 값이 큰 인덱스 출력
```

```
one      b
two      b
dtype: object
```

```
df
```

```
      one  two
a  1.40  NaN
b  7.10 -4.5
c  NaN  NaN
d  0.75 -13.0
```

```
df.cumsum()    #누적
```

```
      one  two
a  1.40  NaN
b  8.50 -4.5
c  NaN  NaN
d  9.25 -17.5
```

```
obj = Series(['a', 'a', 'c', 'd'])
```

```
obj
```

```
0    a
1    a
2    c
3    d
dtype: object
```

```
obj.describe() #한 번에 통계 결과를 여러개 만들어냄
```



```
count    4
unique    3
top       a
freq      2
dtype: object
```

```
obj = Series(['a', 'a', 'c', 'd'] * 4)
```

```
obj.describe()
```

```
count    16
unique     3
top       a
freq      8
dtype: object
```

▼ 유일 값, 값 세기, 멤버십

```
obj = Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
uniques = obj.unique() # 유일한 값(유니크값) 세기
uniques
```

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

```
obj.value_counts() # 값 세기
```

```
c    3
a    3
b    2
d    1
dtype: int64
```

```
obj.value_counts(sort=False)
```

```
a    3
b    2
c    3
d    1
dtype: int64
```

```
pd.value_counts(obj, sort=True)
```



```
c    3
a    3
b    2
d    1
dtype: int64
```

```
obj.value_counts(sort=True)
```

```
c    3
a    3
b    2
d    1
dtype: int64
```

```
mask = obj.isin(['b', 'c'])
mask
```

```
0    True
1    False
2    False
3    False
4    False
5    True
6    True
7    True
8    True
dtype: bool
```

```
obj[mask]
```

```
0    c
5    b
6    b
7    c
8    c
dtype: object
```

```
data = DataFrame({'Que1' : [1,1,1,2,4],
                  'Que2' : [2,3,1,2,3],
                  'Que3' : [1,5,2,2,4]})
```

```
data
```



	Que1	Que2	Que3
0	1	2	1
1	1	3	5
2	1	1	2
3	2	2	2
4	4	3	4

```
result = data.apply(pd.value_counts).fillna(9)
result
```



	Que1	Que2	Que3
1	3.0	1.0	1.0
2	1.0	2.0	2.0
3	9.0	2.0	9.0
4	1.0	9.0	1.0
5	9.0	9.0	1.0

?data.apply # apply 함수 설명 보기

```
result = data.apply(pd.value_counts).cumsum()
result
```



	Que1	Que2	Que3
1	3.0	1.0	1.0
2	4.0	3.0	3.0
3	NaN	5.0	NaN
4	5.0	NaN	4.0
5	NaN	NaN	5.0

▼ 누락된 데이터 처리하기

- pandas는 누락된 데이터를 실수든 아니든 모두 NaN(Not a Number)으로 취급한다.

```
string_data = Series(['aardvark', 'artichoke', np.nan, 'avocado'])
string_data
```



```
0    aardvark
1    artichoke
2         NaN
3     avocado
dtype: object
```

```
string_data.isnull()
```

```
0    False
1    False
2     True
3    False
dtype: bool
```

```
string_data[0] = None
string_data.isnull()
```

```
0     True
1    False
2     True
3    False
dtype: bool
```

▼ 누락된 데이터 골라내기

- Series에 대해 `dropna` 메소드를 적용하면, 실제 데이터가 들어있는 색인값과 Series값으로 반환한다.

```
from numpy import nan as NA
data = Series([1,NA,3.5,NA,7])
data.dropna()
```

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

```
data[data.notnull()]
```

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

```
data
```



```
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
```

```
data = DataFrame([[1., 6.5, 3.], [1, NA, NA], [NA, NA, NA], [NA, 6.5, 3.]])
data
```



	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
data.isnull()
```



	0	1	2
0	False	False	False
1	False	True	True
2	True	True	True
3	True	False	False

```
cleaned_data = data.dropna()
```

```
cleaned_data
```



	0	1	2
0	1.0	6.5	3.0

```
# 모든 값이 NA인 로우만 제외
```

```
data.dropna(how='all')
```



	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

```
data[4] = NA
data
```



	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

```
data.dropna(axis=1,how='all')
```



	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
df = DataFrame(np.random.randn(7,3))
```

df



	0	1	2
0	-0.293766	-0.559905	1.420368
1	0.028130	-0.039620	-1.016603
2	-0.714908	-0.305489	0.126543
3	0.898104	-0.139994	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

```
df.iloc[:4,1] = NA
df.iloc[:2,2] = NA
```

df



	0	1	2
0	-0.293766	NaN	NaN
1	0.028130	NaN	NaN
2	-0.714908	NaN	0.126543
3	0.898104	NaN	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

thresh : 몇 개 이상의 값이 들어있든로우만 살펴보고 싶을 때

```
df.dropna(thresh=3)
```



	0	1	2
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

```
df.dropna(thresh=2) # thresh=2 :: NA가 2개 이상인 경우만 삭제
```



	0	1	2
2	-0.714908	NaN	0.126543
3	0.898104	NaN	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

누락된 값 채우기

```
df
```



	0	1	2
0	-0.293766	NaN	NaN
1	0.028130	NaN	NaN
2	-0.714908	NaN	0.126543
3	0.898104	NaN	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

```
df.fillna(0)
```



	0	1	2
0	-0.293766	0.000000	0.000000
1	0.028130	0.000000	0.000000
2	-0.714908	0.000000	0.126543
3	0.898104	0.000000	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

fillna에 사전값을 넣어 각 칼럼마다 다른 값을 넣을 수도 있음
df.fillna({1: 5, 2: -1})



	0	1	2
0	-0.293766	5.000000	-1.000000
1	0.028130	5.000000	-1.000000
2	-0.714908	5.000000	0.126543
3	0.898104	5.000000	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

fillna는 값을 채워 넣은 객체의 참조를 반환
_ = df.fillna(0, inplace=True)
df



	0	1	2
0	-0.293766	0.000000	0.000000
1	0.028130	0.000000	0.000000
2	-0.714908	0.000000	0.126543
3	0.898104	0.000000	1.292484
4	0.791428	-0.270550	0.473339
5	1.101897	-3.383951	-0.273394
6	0.439860	0.606735	0.687068

inplace=True :: 또 다른 객체를 반환하지 않고 기존 객체를 수정

```
df = DataFrame(np.random.randn(6,3))
df.iloc[2:,1] = NA
df.iloc[4:,2] = NA
df
```



	0	1	2
0	0.624992	-0.374923	-0.225250
1	0.946028	-2.475911	-0.647615
2	-0.224139	NaN	0.621481
3	1.533411	NaN	1.517053
4	-0.528929	NaN	NaN
5	-0.795105	NaN	NaN

```
df.fillna(method = 'ffill')
```



	0	1	2
0	1.964181	-1.862370	-0.181858
1	-0.240739	0.250955	-1.014885
2	-0.714474	0.250955	-0.568680
3	0.298564	0.250955	1.503525
4	-1.338149	0.250955	1.503525
5	1.740365	0.250955	1.503525

```
df.fillna(method='bfill')
```



	0	1	2
0	-0.004916	-1.590970	0.916494
1	1.127411	0.186797	-2.124222
2	-0.427043	NaN	-0.000379
3	-1.464211	NaN	-1.259124
4	1.472739	NaN	NaN
5	0.826697	NaN	NaN

```
df.fillna(method='ffill', limit=3)
```



	0	1	2
0	0.624992	-0.374923	-0.225250
1	0.946028	-2.475911	-0.647615
2	-0.224139	-2.475911	0.621481
3	1.533411	-2.475911	1.517053
4	-0.528929	-2.475911	1.517053
5	-0.795105	NaN	1.517053

```
data = Series([1., NA, 3.5, NA, 7])
data.fillna(data.mean())
```



```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

▼ Merge

```
df = pd.DataFrame(np.random.randn(10, 4))
```

```
df
```



	0	1	2	3
0	-1.080419	1.358693	-0.961668	0.734180
1	0.722622	-0.109833	0.284602	-0.019367
2	-0.295452	0.297437	0.280276	-0.123358
3	-0.895981	0.205292	-0.757292	1.357538
4	-1.287096	-0.409108	-0.894718	-1.697985
5	2.500054	0.008133	1.107169	-0.923730
6	-1.846901	-0.118254	-1.162602	-0.727689
7	0.734065	-1.181757	-0.809964	-0.991737
8	0.894451	-0.550830	-0.546563	1.385450
9	0.047948	-1.259092	0.926986	0.005700

```
pieces = [df[:3], df[3:7], df[7:]]
```

```
pieces
```

```
[
      0      1      2      3
0 -1.080419  1.358693 -0.961668  0.734180
1  0.722622 -0.109833  0.284602 -0.019367
2 -0.295452  0.297437  0.280276 -0.123358,
      0      1      2      3
3 -0.895981  0.205292 -0.757292  1.357538
4 -1.287096 -0.409108 -0.894718 -1.697985
5  2.500054  0.008133  1.107169 -0.923730
6 -1.846901 -0.118254 -1.162602 -0.727689,
      0      1      2      3
7  0.734065 -1.181757 -0.809964 -0.991737
8  0.894451 -0.550830 -0.546563  1.385450
9  0.047948 -1.259092  0.926986  0.005700]
```

```
pd.concat(pieces)      #default로 axis가 0
```



	0	1	2	3
0	-1.080419	1.358693	-0.961668	0.734180
1	0.722622	-0.109833	0.284602	-0.019367
2	-0.295452	0.297437	0.280276	-0.123358
3	-0.895981	0.205292	-0.757292	1.357538
4	-1.287096	-0.409108	-0.894718	-1.697985
5	2.500054	0.008133	1.107169	-0.923730
6	-1.846901	-0.118254	-1.162602	-0.727689
7	0.734065	-1.181757	-0.809964	-0.991737
8	0.894451	-0.550830	-0.546563	1.385450
9	0.047948	-1.259092	0.926986	0.005700

```
?pd.concat
```

▼ Join

```
left = pd.DataFrame({'key' : ['foo','foo'], 'lval' : [1,2]})
right = pd.DataFrame({'key': ['foo','foo'], 'rval' : [4,5]})
```

left



	key	lval
0	foo	1
1	foo	2

right



	key	rval
0	foo	4
1	foo	5

```
pd.merge(left, right, on='key')
```



	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

```
left = DataFrame({'key' : ['foo','bar'], 'lval' : [1,2]})
```

```
right = DataFrame({'key' : ['foo','bar'], 'rval' : [4,50]})
```

```
pd.merge(left,right,on = 'key')
```



	key	lval	rval
0	foo	1	4
1	bar	2	50

▼ Grouping (groupby)

- Splitting
- Applying
- Combining
- 동일한 키값끼리 그룹핑

```
df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar',  
                        'foo', 'bar', 'foo', 'foo'],  
                  'B': ['one', 'one', 'two', 'three',  
                        'two', 'two', 'one', 'three'],  
                  'C': np.random.randn(8),  
                  'D': np.random.randn(8)})
```

df



	A	B	C	D
0	foo	one	-0.621497	-1.530697
1	bar	one	1.397674	-0.242016
2	foo	two	-0.958995	-0.695016
3	bar	three	-0.197460	-0.692142
4	foo	two	1.186806	0.299109
5	bar	two	-0.331972	-0.000568
6	foo	one	-0.944805	0.146293
7	foo	three	0.763906	-1.931550

```
df.groupby('A').mean()
```



	A	C	D
bar	0.289414	-0.311575	
foo	-0.114917	-0.742372	

```
df.groupby(['B', 'A']).sum()
```



		B	A	C	D
one	bar	1.397674	-0.242016		
	foo	-1.566302	-1.384404		
three	bar	-0.197460	-0.692142		
	foo	0.763906	-1.931550		
two	bar	-0.331972	-0.000568		
	foo	0.227810	-0.395907		

```
df.groupby(['A', 'B']).sum()
```



		C	D
A	B		
bar	one	1.397674	-0.242016
	three	-0.197460	-0.692142
	two	-0.331972	-0.000568
foo	one	-1.566302	-1.384404
	three	0.763906	-1.931550
	two	0.227810	-0.395907