

# 데이터 편집

# concat

## ❖concat()

- ✓ 축을 이용해서 이어 붙이는 함수
- ✓ Series 객체는 이 함수를 이용하면 값과 인덱스를 연결해줍니다.
- ✓ Series 객체끼리 concat를 할 때 axis는 1을 대입하면 컬럼 단위로 이어 붙이기를 해서 DataFrame을 만들어 줍니다.
- ✓ join 옵션에 inner를 적용하면 양쪽에 존재하는 인덱스의 데이터만 합치는데 기본 값은 outer
- ✓ join\_axes를 이용해서 join에 참여할 인덱스를 직접 설정하는 것이 가능
- ✓ keys를 이용해서 컬럼의 이름을 붙이는 것이 가능

# concat

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

s1 = Series([84900, 818000, 1756], index=['다음', "네이버", "넥슨"])
s2 = Series([86100, 871000, 1776, 295000], index=['다음', "네이버", "넥슨", "NC"])
print(pd.concat([s1, s2]))
print("=====")
print(pd.concat([s1, s2], axis=1))
print("=====")
print(pd.concat([s1, s2], axis=1, join='inner'))
print("=====")
print(pd.concat([s1, s2], axis=1, keys=['2017-03-19', '2017-03-20'], join_axes=[['다음', "네이버", "넥슨", "NC"]]))
```

# concat

```
다음      84900
네이버    818000
넥슨      1756
다음      86100
네이버    871000
넥슨      1776
NC        295000
dtype: int64
```

```
=====
      0      1
NC      NaN 295000
네이버  818000.0 871000
넥슨    1756.0   1776
다음    84900.0  86100
```

```
=====
      0      1
다음    84900  86100
네이버  818000 871000
넥슨    1756   1776
```

```
=====
      2017-03-19 2017-03-20
다음    84900.0    86100
네이버  818000.0    871000
넥슨    1756.0     1776
NC      NaN      295000
```

# concat

## ❖concat()

- ✓ DataFrame은 기본적으로 동일한 컬럼을 기준으로 해서 행 단위로 결합
- ✓ 존재하지 않는 컬럼의 값은 None
- ✓ axis는 합치는 축의 방향으로 기본 값은 0
- ✓ keys 옵션에 리스트를 대입하면 그룹별 상위 인덱스를 생성할 수 있습니다.
- ✓ join 옵션에 inner를 대입해서 동일한 컬럼이나 인덱스를 기준으로 결합 가능
- ✓ join\_axes 옵션을 이용해서 한쪽에만 존재하는 인덱스를 가지고 join 가능

# concat

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

stock1 = {
    '2017-03-20': [84900, 818000, 1756,292000],
    '2017-03-21': [86100, 871000, 1776,295000]}
stock2 = {
    '2017-04-20': [90800, 796000, 1695],
    '2017-04-21': [90600, 806000, 1703]}
df1 = pd.DataFrame(stock1, columns=['2017-03-20', '2017-03-21'], index=['다음',"
네이버", "넥슨", "NC"])
df2 = pd.DataFrame(stock2, columns=['2017-04-20', '2017-04-21'], index=['다음',"
네이버", "넥슨"])
print(pd.concat([df1, df2], axis=1).fillna('-'))
```

# concat

	2017-03-20	2017-03-21	2017-04-20	2017-04-21
NC	292000	295000	-	-
네이버	818000	871000	796000	806000
벅스	1756	1776	1695	1703
다음	84900	86100	90800	90600

# concat

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

stock1 = {
    '2017-03-20': [84900, 818000, 1756, 292000],
    '2017-03-21': [86100, 871000, 1776, 295000]}
stock2 = {
    '2017-04-20': [90800, 796000, 1695, 8000],
    '2017-04-21': [90600, 806000, 1703, 8500]}
df1 = pd.DataFrame(stock1, columns=['2017-03-20', '2017-03-21'], index=['다음', "네이버", "넥슨", "NC"])
df2 = pd.DataFrame(stock2, columns=['2017-04-20', '2017-04-21'], index=['다음', "네이버", "넥슨", '위메이드'])
print(pd.concat([df1, df2], axis=1, join='inner'))
print('=====')
print(pd.concat([df1, df2], axis=1, join_axes=[df1.index]).fillna('-'))
```



# concat

	2017-03-20	2017-03-21	2017-04-20	2017-04-21
다음	84900	86100	90800	90600
네이버	818000	871000	796000	806000
넥슨	1756	1776	1695	1703

=====

	2017-03-20	2017-03-21	2017-04-20	2017-04-21
다음	84900	86100	90800	90600
네이버	818000	871000	796000	806000
넥슨	1756	1776	1695	1703
NC	292000	295000	-	-

# join

## ❖merge()

- ✓ DataFrame 이나 Series를 합치기 위한 함수로 첫번째와 두번째 매개변수는 합치기 위한 Series 나 DataFrame : left와 right 옵션과 함께 대입해도 됨
- ✓ 하나 이상의 **key**를 가지고 연산
- ✓ 데이터베이스의 JOIN 연산과 유사
- ✓ 옵션 없이 호출하면 동일한 컬럼 이름을 기준으로 join을 수행하고 **on** 옵션에 **컬럼 이름을 명시** 적으로 설정하면 설정된 컬럼 이름으로 join을 수행합니다.
- ✓ on 절에 여러 개의 컬럼을 지정할 수 있습니다.
- ✓ 양쪽 프레임의 컬럼 이름이 다른 경우에는 **left\_on** 에 호출하는 프레임의 컬럼이름을 **right\_on** 에 대상이 되는 프레임의 컬럼 이름을 대입하면 됩니다.

# join

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

stock1 = {'name': ['다음', "네이버", "넥슨", "NC"],
'2017-03-20': [84900, 818000, 1756,292000],
'2017-03-21': [86100, 871000, 1776,295000]}

stock2 = {'name': ['다음', "네이버", "넥슨", "NC"],
'2017-04-20': [90800, 796000, 1695,366500],
'2017-04-21': [90600, 806000, 1703,358500]}
```

# join

```
df1 = pd.DataFrame(stock1, columns=['name', '2017-03-20', '2017-03-21'])
print(df1)
df2 = pd.DataFrame(stock2, columns=['name', '2017-04-20', '2017-04-21'])
print('=====')
print(df2)
result = df1.merge(df2)
print('=====')
print(result)
```

# join

	name	2017-03-20	2017-03-21
0	다음	84900	86100
1	네이버	818000	871000
2	빅스	1756	1776
3	NC	292000	295000

	name	2017-04-20	2017-04-21
0	다음	90800	90600
1	네이버	796000	806000
2	빅스	1695	1703
3	NC	366500	358500

	name	2017-03-20	2017-03-21	2017-04-20	2017-04-21
0	다음	84900	86100	90800	90600
1	네이버	818000	871000	796000	806000
2	빅스	1756	1776	1695	1703
3	NC	292000	295000	366500	358500

# join

## ❖merge()

- ✓ how 옵션에 inner 나 left, right, outer를 대입해서 outer join을 수행할 수 있습니다.
- ✓ suffixes 옵션에 튜플로 컬럼 이름을 2개 대입하면 양쪽에 동일한 이름의 컬럼이 존재하는 경우 컬럼 이름 뒤에 튜플의 값을 추가해주는데 기본 값은 \_x, \_y 입니다.
- ✓ join 하려는 속성이 컬럼이 아니고 index 인 경우는 left\_index=True 또는 right\_index는 True를 지정해서 수행할 수 있습니다.
- ✓ sort: 기본값은 True로 설정되어 있는데 조인하는 key를 기준으로 정렬 수행 여부 설정

# join

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

stock1 = {'name': ['다음', "네이버", "넥슨", "NC"],
'2017-03-20': [84900, 818000, 1756, 292000],
'2017-03-21': [86100, 871000, 1776, 295000]}

stock2 = {'name': ['다음', "네이버", "넥슨"],
'2017-04-20': [90800, 796000, 1695],
'2017-04-21': [90600, 806000, 1703]}
```

# join

```
df1 = pd.DataFrame(stock1, columns=['name', '2017-03-20', '2017-03-21'])
df2 = pd.DataFrame(stock2, columns=['name', '2017-04-20', '2017-04-21'])
result = df1.merge(df2, on='name')
print('=====')
print(result)
result = df1.merge(df2, on='name', how='outer')
print('=====')
print(result)
```



# join

=====

	name	2017-03-20	2017-03-21	2017-04-20	2017-04-21
0	다음	84900	86100	90800	90600
1	네이버	818000	871000	796000	806000
2	빅스	1756	1776	1695	1703

=====

	name	2017-03-20	2017-03-21	2017-04-20	2017-04-21
0	다음	84900	86100	90800.0	90600.0
1	네이버	818000	871000	796000.0	806000.0
2	빅스	1756	1776	1695.0	1703.0
3	NC	292000	295000	NaN	NaN

# join

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

stock1 = {'name': ['다음', "네이버", "넥슨", "NC"],
'2017-03-20': [84900, 818000, 1756, 292000],
'2017-03-21': [86100, 871000, 1776, 295000]}

stock2 = {'name': ['다음', "네이버", "넥슨"],
'2017-04-20': [90800, 796000, 1695],
'2017-04-21': [90600, 806000, 1703]}
```

# join

```
df1 = pd.DataFrame(stock1, columns=['name', '2017-03-20', '2017-03-21'])
df2 = pd.DataFrame(stock2, columns=['name', '2017-04-20', '2017-04-21'])
result = df1.merge(df2, on='name', how='outer')
result.index = result["name"]
result = result.drop("name", axis=1)
result = result.T
print(result)
print("=====")
print(result["넥슨"].corr(result["네이버"]))
```

# join

name	다음	네이버	빅스	NC
2017-03-20	84900.0	818000.0	1756.0	292000.0
2017-03-21	86100.0	871000.0	1776.0	295000.0
2017-04-20	90800.0	798000.0	1695.0	NaN
2017-04-21	90600.0	806000.0	1703.0	NaN

=====

0.878620924052

# concat

## ❖ append()

- ✓ 인덱스가 별 의미없는 숫자인 경우 데이터를 무조건 행 방향으로 결합하는 함수

## ❖ Combine\_first()

- ✓ 데이터를 합칠 때 인덱스가 겹치는 경우 호출하는 데이터의 것으로 채워서 결합하는 함수



# concat

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

stock1 = {
    '2017-03-20': [84900, 1756, 292000],
    '2017-03-21': [86100, 1776, 295000]}
stock2 = {
    '2017-03-20': [90800, 796000, 1695],
    '2017-03-21': [90600, 806000, 1703]}
df1 = pd.DataFrame(stock1, columns=['2017-03-20', '2017-03-21'], index=['다음',
    넥슨", "NC"])
df2 = pd.DataFrame(stock2, columns=['2017-03-20', '2017-03-21'], index=['다음',
    네이버", "넥슨"])
print(df1.combine_first(df2))
```

# concat

	2017-03-20	2017-03-21
NC	292000.0	295000.0
네이버	796000.0	806000.0
넥슨	1756.0	1776.0
다음	84900.0	86100.0



# pivot

## ❖ stack()

- ✓ DataFrame에서 컬럼을 하위 인덱스로 해서 Series 객체를 생성해주는 함수
- ✓ Nan 데이터는 제외
- ✓ dropna에 False를 대입하면 Nan 데이터도 참여

## ✓ unstack()

- ✓ 계층적 인덱스를 가진 Series 객체의 경우는 하위 레벨의 인덱스를 컬럼으로 하는 DataFrame을 생성
- ✓ DataFrame의 경우는 컬럼을 상위 인덱스로 하는 Series 객체를 생성




# pivot

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

stock1 = {
    '2017-03-20': [84900, 1756, 292000],
    '2017-03-21': [86100, np.nan, 295000]}

df1 = pd.DataFrame(stock1, index=['다음', '넥슨', 'NC'])
print(df1)
print("=====")
print(df1.stack())
print("=====")
print(df1.unstack())
```



```
2017-03-20 2017-03-21
다음      84900    86100.0
넥슨      1756     NaN
NC      292000    295000.0
=====
다음 2017-03-20    84900.0
      2017-03-21    86100.0
넥슨 2017-03-20    1756.0
NC 2017-03-20    292000.0
      2017-03-21    295000.0
dtype: float64
=====
2017-03-20 다음    84900.0
      넥슨    1756.0
      NC    292000.0
2017-03-21 다음    86100.0
      넥슨    NaN
      NC    295000.0
dtype: float64
```

# pivot

## ❖pivot()

- ✓ DataFrame을 원하는 형식으로 그룹화하기 위한 함수
- ✓ 첫번째 매개변수는 행의 인덱스로 사용할 컬럼 이름
- ✓ 두번째 매개변수는 열의 인덱스로 사용할 컬럼 이름
- ✓ 세번째 매개변수는 출력될 데이터로 사용할 컬럼 이름

# pivot

**select \* from stock**

	num	date	name	price	count
1	7	2017-03-20	다음	84900	500
2	8	2017-03-20	넥슨	1756	300
3	9	2017-03-20	NC	292000	200
4	10	2017-03-21	다음	86100	450
5	11	2017-03-21	넥슨	1787	220
6	12	2017-03-21	NC	295000	320

# pivot

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
import pymysql, sys
con = pymysql.connect(host='211.183.2.253', port=3306,
                      user='root', passwd='wnddkd',
                      db='user30', charset = 'utf8')

list = []
try:
    cursor = con.cursor()
    cursor.execute("select date, name, price, count from stock")
    data = cursor.fetchall()
    for imsi in data:
        list.append(imsi)
```

# pivot

except:

```
print('exception:', sys.exc_info())
```

finally:

```
con.close()
```

```
frame = DataFrame(list, columns=['date', 'name', 'price', 'count'])
```

```
print(frame)
```

```
print('=====')
```

```
print(frame.pivot('date', 'name'))
```

# pivot

```
date name price count
0 2017-03-20 다음 84900 500
1 2017-03-20 넥슨 1756 300
2 2017-03-20 NC 292000 200
3 2017-03-21 다음 86100 450
4 2017-03-21 넥슨 1787 220
5 2017-03-21 NC 295000 320
```

```
=====
           price      count
name      NC 넥슨 다음 NC 넥슨 다음
date
2017-03-20 292000 1756 84900 200 300 500
2017-03-21 295000 1787 86100 320 220 450
```

# 데이터 가공

## ❖ duplicated()

- ✓ 데이터의 중복을 Boolean Series 객체로 리턴해주는 함수

## ❖ drop\_duplicates()

- ✓ 매개변수를 대입하지 않으면 모든 컬럼의 값이 동일한 경우 제거
- ✓ 매개변수로 컬럼의 리스트를 대입하면 대입된 컬럼의 값이 일치하는 데이터를 제거하고 중복된 데이터의 첫번째 값을 보존
- ✓ keep 옵션에 last를 대입하면 마지막 데이터의 값을 보존

## ❖ map()

- ✓ 함수 이름이나 dict를 대입해서 함수를 수행한 결과나 매핑이 되는 dict의 값을 리턴해서 적용하는 함수



# 데이터 가공

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

def func(s):
    if len(s) < 5:
        return s
    else :
        return s[0:3] + '..'
loc = {'한국':'korea', '중국':'china', '일본':'japan'}
```

# 데이터 가공

```
df = DataFrame(['안녕하세요','니하오', '곤니찌와', '안녕하세요'], ['한국','중국','일본',  
'한국'])
```

```
df = df.T
```

```
df.columns = ['인사말', '국가']
```

```
df = df.drop_duplicates()#중복 제거
```

```
df['nation'] = df['국가'].map(loc)#nation 컬럼 생성 - 국가를 loc 맵에 매핑한 결과
```

```
df['인사말'] = df['인사말'].map(func)
```

```
print(df)
```

# 데이터 가공

- 인사말 국가 nation
- 0 안녕하.. 한국 korea
  - 1 니하오 중국 china
  - 2 곤니찌와 일본 japan



# 데이터 가공

## ❖ replace()

- ✓ 데이터의 값을 변경해주는 함수
- ✓ 2개의 매개변수를 대입하는데 첫번째 매개변수의 값을 두번째 매개변수의 값으로 변경해줍니다.
- ✓ list를 대입해서 여러 개의 값을 치환하는 것 가능
- ✓ 2개의 매개변수 대신에 하나의 dict를 대입해서 치환하는 것도 가능

## ❖ rename()

- ✓ 인덱스나 컬럼의 이름을 변경할 때 사용하는 함수
- ✓ index 나 columns 옵션에 함수를 대입하면 함수의 결과를 이용해서 변경
- ✓ dict를 대입하면 dict의 key에 해당하는 이름을 value로 수정

# 데이터 가공

- ❖ 조건을 만족하는 데이터를 다른 데이터로 치환: `DataFrame[조건] = 값`
- ❖ 행의 데이터 중에서 하나라도 조건을 만족하는 데이터만 추출: `DataFrame[(조건).any(1)]`
- ❖ Series의 데이터의 순서를 임의로 변경하고자 하는 경우에는 `numpy.random.permutation(개수)`를 이용
- ❖ DataFrame에서 특정 개수만큼 데이터 추출  
`DataFrame.take(numpy.random.permutation(개수))`

# 데이터 가공

```
import pandas as pd
import numpy as np
from pandas import Series, DataFrame
stock = {
    '2017-03-20': [84900, 1756,292000],
    '2017-03-21': [86100, np.nan,295000]}
df = pd.DataFrame(stock, index=['다음','넥슨', "NC"])
df = df.rename(index={'다음':'DAUM', '넥슨':'NEXON'})
print(df.replace({np.nan:'-'}))
```

# 데이터 가공

	2017-03-20	2017-03-21
DAUM	84900	86100
NEXON	1756	-
NC	292000	295000



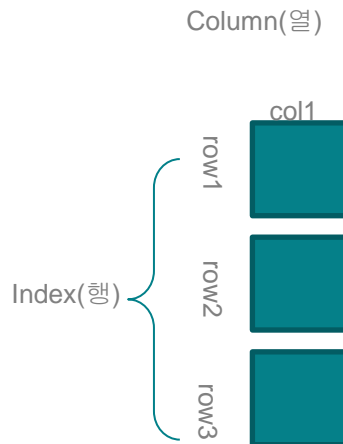
# Group by



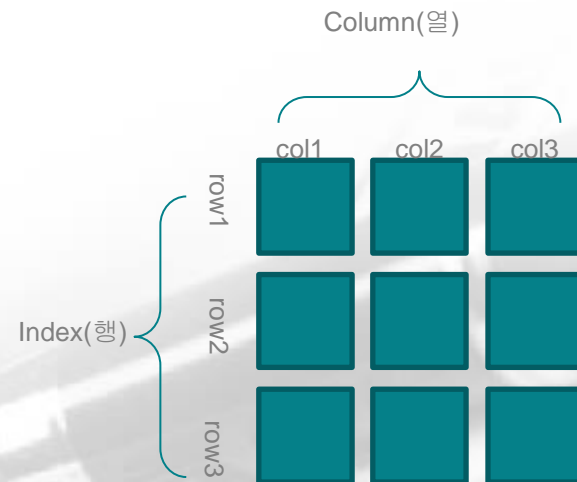
# GroupBy

- ❖ DataFrame이나 Series에서 데이터를 그룹 별로 묶는데 사용
- ❖ groupby(키가 되는 컬럼)을 호출해서 컬럼 단위로 묶을 수 있습니다.
- ❖ groupby 메소드로 묶으면 다른 클래스의 인스턴스가 생성

## SeriesGroupby(1차원)



## DataFrameGroupby(2차원)



# GroupBy

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

df = pd.read_csv('tips.csv')
gp = df['tip'].groupby(df['time'])
print(gp)
```



# GroupBy

❖ SeriesGroupby 객체도 iterable 이므로 행과 데이터를 for를 이용해서 접근할 수 있습니다.

```
for name, group in gp:  
    # print the name of the regiment  
    print(name)  
    # print the data of that regiment  
    print(group)
```

# GroupBy

❖ DataFrame에 groupby를 적용한 DataFrameGroupBy 객체도 SeriesGroupBy의 함수

- ✓ count: NA가 아닌 데이터의 개수
- ✓ min, max: 최소값 과 최대값
- ✓ Sum: NA가 아닌 값의 합계
- ✓ mean: NA가 아닌 값들의 평균
- ✓ median: 산술 중간 값
- ✓ var, std: 분산과 표준편차
- ✓ prod: NA가 아닌 값들의 곱
- ✓ first, last: NA가 아닌 값 중에서 첫번째 와 마지막 값
- ✓ describe: 요약

# GroupBy

- ❖ DataFrame에 groupby를 적용한 DataFrameGroupBy 객체도 SeriesGroupBy와 동일
- ❖ 여러 개의 컬럼으로 그룹화 가능

```
result = df['tip'].groupby([df['time'],df['sex']])  
print(result)  
print("=====  
result = result.mean().unstack()  
result = result.replace(np.nan, '-')  
print(result)
```

# GroupBy

❖group by에 함수 이름을 넘겨주면 인덱스를 가지고 함수를 수행한 결과를 가지고 그룹화 해서 결과를 리턴합니다.

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

```
df = pd.read_csv('tips.csv', index_col='sex')
print(df.groupby(len).mean()) #sex의 글자 수 별로 묶어서 평균 구하기
```

# GroupBy

- ❖ 그룹화 한 후 원하는 집계를 추출하기 위해서는 agg 나 aggregate 함수에 함수의 이름을 넘겨주면 됩니다.
- ❖ 함수는 사용자 정의 함수 가능
- ❖ 여러 개의 함수도 가능
- ❖ 튜플의 형태로 대입하면 튜플의 첫번째 데이터가 컬럼 이름이 됩니다.

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

```
def cha(ar):
    return ar.max() - ar.min()
```

```
df = pd.read_csv('tips.csv')
grouped = df.groupby('sex')
print(grouped['tip'].aggregate(cha)) #성별로 Tip의 최대값과 최소값의 차이
```

# GroupBy

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

```
def cha(ar):  
    return ar.max() - ar.min()
```

```
df = pd.read_csv('tips.csv')  
grouped = df.groupby('sex')  
print(grouped['tip'].agg(['mean', cha]))
```





# GroupBy

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

```
def cha(ar):  
    return ar.max() - ar.min()
```

```
df = pd.read_csv('tips.csv')  
grouped = df.groupby('sex')  
print(grouped.agg([('평균', 'mean'), ('합계', 'sum')]))
```

# GroupBy

❖ 각각의 컬럼에 다른 함수를 적용하고자 할 때는 dict 형태로 컬럼 이름을 키로 하고 적용하고자 하는 함수의 이름이나 함수 이름의 리스트를 대입하면 됩니다.

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

def cha(ar):
    return ar.max() - ar.min()

df = pd.read_csv('tips.csv')
grouped = df.groupby('sex')
print(grouped.agg({'tip':['mean','sum'],'size':'mean'}))
```

# apply

- ❖ apply 메소드는 내부 함수를 모든 원소에 대해 계산을 처리함
- ❖ 매개변수는 ,를 하고 대입

df.apply(func)



# apply

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

df = pd.read_csv('tips.csv')
print(df[['total_bill', 'tip', 'size']].apply(sum))
```



# apply

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

def descsort(df, column='tip'):
    return df.sort_values(by=column, ascending=False)

df = pd.read_csv('tips.csv')
print(df.groupby('sex').apply(descsort, column='total_bill'))
```



# pivot\_table

❖ 데이터를 그룹화하기 위한 함수

`pandas.pivot_table(데이터프레임, values=[연산을 수행할 컬럼 나열], index=[인덱스 나열], columns=[컬럼의 인덱스 나열], margins=전체데이터출력 여부, argfunc=수행할 함수, fill_value=NA일 때의 값)`

❖ `aggfunc`를 생략하면 평균

```
from pandas import Series, DataFrame
```

```
import pandas as pd
```

```
import numpy as np
```

```
df = pd.read_csv('tips.csv')
```

```
print(pd.pivot_table(df, values=['tip','total_bill'], index=['sex', 'day'],  
columns='smoker'))
```

# pivot\_table

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

df = pd.read_csv('tips.csv')
print(pd.pivot_table(df, values=['tip','total_bill'], index=['sex', 'day'],
columns='smoker', aggfunc=[sum, len]))
```

# crosstab

❖ 그룹의 빈도를 계산하기 위한 피벗테이블 함수

pandas.crosstab(인덱스, 컬럼, 전체 데이터의 출력 여부)

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

df = pd.read_csv('tips.csv')
print(pd.crosstab(df.sex, df.smoker, margins=True))
```