데이터 처리 프로그래밍

박소현

다섯 번째 주제:

데이터
Reshaping, combine, grouping

1. Reshaping Data

1) .melt( ) – 열 이름을 행 값으로(즉, value)로 보내는 매서드

```
pd.melt( frame, id_vars=None, value_vars=None, var_name=None, value_name='value', col_level=None, )
```

- id_vars= : 위치를 그대로 유지할 열의 이름을 지정
- value_vars= : 행으로 위치를 변경할 열의 이름 지정
- var_name= : value_vars로 위치를 변경한 열의 이름 지정
- value_name= : var_name으로 위치를 변경한 열의 데이터를 저장한 열의 이름을 지정

```
df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
                   'B': {0: 1, 1: 3, 2: 5},
                   'C': {0: 2, 1: 4, 2: 6}})
df
```

	A	B	C
0	a	1	2
1	b	3	4
2	c	5	6

```
pd.melt(df, id_vars=['A'], value_vars=['B'])
```

	A	variable	value
0	a	B	1
1	b	B	3
2	c	B	5

```
df
```

	A	B	C
0	a	1	2
1	b	3	4
2	c	5	6

```
pd.melt(df, id_vars=['A'], value_vars=['B', 'C'])
```

	A	variable	value
0	a	B	1
1	b	B	3
2	c	B	5
3	a	C	2
4	b	C	4
5	c	C	6

```
df
```

	A	B	C
0	a	1	2
1	b	3	4
2	c	5	6

```
pd.melt(df, value_vars=['A', 'B', 'C'])
```

	variable	value
0	A	a
1	A	b
2	A	c
3	B	1
4	B	3
5	B	5
6	C	2
7	C	4
8	C	6

"variable"과 "value" 열이름

바꾸려면?

```
.rename(columns = {'variable' : '가', 'value': '값'})
```

```
pd.melt(df, value_vars=['A', 'B', 'C']).rename(columns = {'variable' : '가', 'value': '값'})
```

df

	A	B	C
0	a	1	2
1	b	3	4
2	c	5	6

	가	값
0	A	a
1	A	b
2	A	c
3	B	1
4	B	3
5	B	5
6	C	2
7	C	4
8	C	6

```
pew = pd.read_csv('pew.csv')
```

```
pew.head()
```

	religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don't know/refused	15	14	15	11	10	35	21	17	18	116

종교와 소득구간 정보를 담고있는 pew

```
pew2 = pd.melt(pew, id_vars = 'religion')
pew2.head(7)
```

	religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don't know/refused	15	14	15	11	10	35	21	17	18	116

```
pew2 = pd.melt(pew, id_vars = 'religion')
pew2.head(7)
```

	religion	variable	value
0	Agnostic	<$10k	27
1	Atheist	<$10k	12
2	Buddhist	<$10k	27
3	Catholic	<$10k	418
4	Don't know/refused	<$10k	15
5	Evangelical Prot	<$10k	575
6	Hindu	<$10k	1

id_vars에서 지정한 열(religion)을 제외한,
나머지 열이 variabl열로 정리되고
소득정보 열(<$10K, $10-20K, ...)의 행 데이터도
value열로 정리
➔이 과정을 "religion열을 고정하여 피벗했다"
고 말함

1. Reshaping Data 1)pd.melt( )

	religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don't know/refused	15	14	15	11	10	35	21	17	18	116

```
pew2 = pd.melt(pew, id_vars = 'religion')
```

```
pew2.iloc[[0, 1, 2, 18, 19, 20], :]
```

	religion	variable	value
0	Agnostic	<$10k	27
1	Atheist	<$10k	12
2	Buddhist	<$10k	27
18	Agnostic	$10-20k	34
19	Atheist	$10-20k	27
20	Buddhist	$10-20k	21

```
pew3 = pd.melt(pew, id_vars = 'religion', var_name='income', value_name='count')
pew3.head()
```

	religion	income	count
0	Agnostic	<$10k	27
1	Atheist	<$10k	12
2	Buddhist	<$10k	27
3	Catholic	<$10k	418
4	Don't know/refused	<$10k	15

pd.melt(frame, id_vars=None, value_vars=None, var_name=None, value_name='value',

- id_vars= : 위치를 그대로 유지할 열의 이름을 지정
- value_vars= : 행으로 위치를 변경할 열의 이름 지정
- var_name= : value_vars로 위치를 변경한 열의 이름 지정
- value_name= : var_name으로 위치를 변경한 열의 데이터를 저장한 열의 이름을 지정

```
billb = pd.read_csv('billboard.csv')
billb.head(3)
```

고정 열로 어떤걸로 하는게 좋을까?

행 값으로 보내면 좋을 애는?

	year	artist	track	time	date.entered	wk1	wk2	wk3	wk4	wk5	...	wk74	wk75	wk76
0	2000	2 Pac	Baby Don't Cry (Keep...	4:22	2000-02-26	87	82.0	72.0	77.0	87.0	...	NaN	NaN	NaN
1	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87.0	92.0	NaN	NaN	...	NaN	NaN	NaN
2	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70.0	68.0	67.0	66.0	...	NaN	NaN	NaN

3 rows × 81 columns

```
billb_2 = pd.melt(billb,
                  id_vars=['year', 'artist', 'track', 'time', 'date.entered'],
                  var_name='week', value_name='rationg')
billb_2.head()
```

	year	artist	track	time	date.entered	week	rationg
0	2000	2 Pac	Baby Don't Cry (Keep...	4:22	2000-02-26	wk1	87.0
1	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	wk1	91.0
2	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	wk1	81.0
3	2000	3 Doors Down	Loser	4:24	2000-10-21	wk1	76.0
4	2000	504 Boyz	Wobble Wobble	3:35	2000-04-15	wk1	57.0

2) .pivot( ) – 기존의 데이터프레임을 이용, 지정한 행과 열 기준으로 데이터를 배치하는 매서드

```
df.pivot?
```

```
df.pivot(index=None, columns=None, values=None)
```

```
df2 = pd.DataFrame({'foo': ['one', 'one', 'one', 'two', 'two','two'],
                    'bar': ['A', 'B', 'C', 'A', 'B', 'C'],
                    'baz': [1, 2, 3, 4, 5, 6]})
```

	foo	bar	baz
0	one	A	1
1	one	B	2
2	one	C	3
3	two	A	4
4	two	B	5
5	two	C	6

```
df2
```

	foo	bar	baz
0	one	A	1
1	one	B	2
2	one	C	3
3	two	A	4
4	two	B	5
5	two	C	6

```
ef = df2.pivot(index='foo', columns='bar', values='baz')
ef
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

```
ef.index
```

```
Index(['one', 'two'], dtype='object', name='foo')
```

```
ef.index.name
```

```
'foo'
```

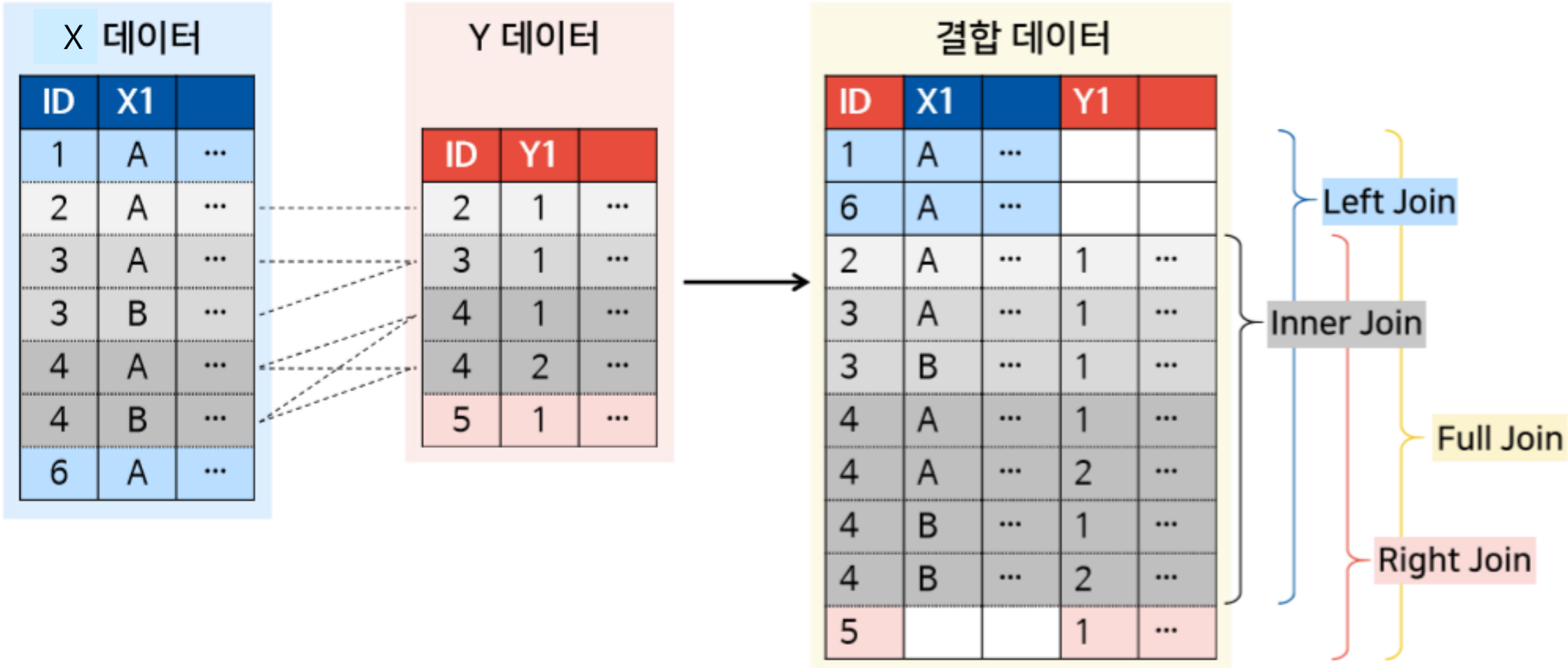
```
ef.columns.name
```

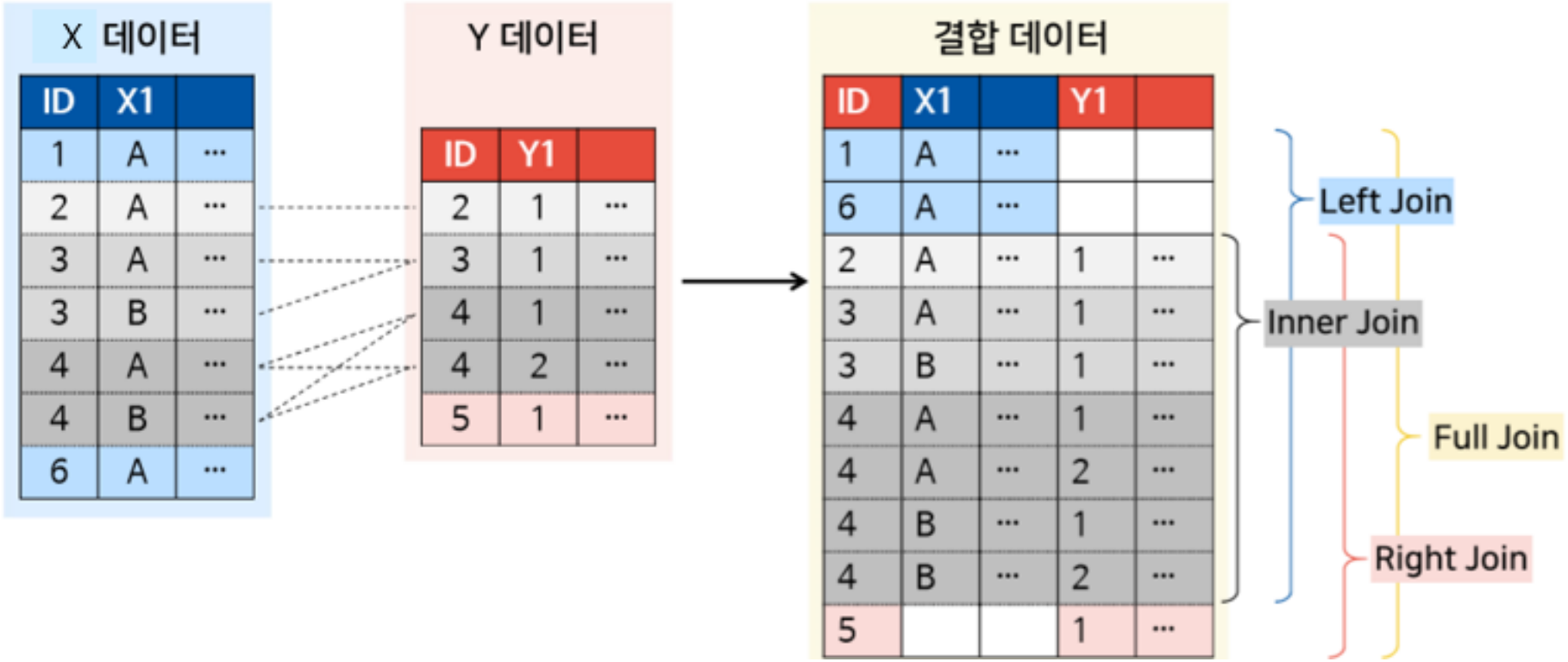
```
'bar'
```

```
ef.columns
```

```
Index(['A', 'B', 'C'], dtype='object', name='bar')
```

3) .join( ) – index를 기준으로 데이터 세트를 합쳐 주는 매서드





join

- inner : 공통행만 골라서 연결
- outer : 두개의 데이터프레임 중 누굴 기준으로 할것인지에 따라 left와 right, outer로 나뉨
  - left : 왼쪽 데이터 프레임을 모두 포함해 연결
  - right : 오른쪽 데이터프레임을 모두 포함해 연결
  - outer : 왼쪽과 오른쪽 데이터 프레임 모두를 포함해 연결

```
import numpy as np
dfa = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                   index = ['a', 'c', 'e'], columns=['Ohio', 'Nevada'])
```

```
dfb = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                   index = ['e', 'c', 'd', 'b'], columns=['Misouri', 'Alabama'])
```

```
dfa
```

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
dfb
```

	Misouri	Alabama
e	7.0	8.0
c	9.0	10.0
d	11.0	12.0
b	13.0	14.0

```
dfa
```

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
dfb
```

	Misouri	Alabama
e	7.0	8.0
c	9.0	10.0
d	11.0	12.0
b	13.0	14.0

```
join1 = dfa.join(dfb, how = 'outer')
join1
```

	Ohio	Nevada	Misouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	13.0	14.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	7.0	8.0

```
join2 = dfa.join(dfb, how = 'inner')
join2
```

	Ohio	Nevada	Misouri	Alabama
c	3.0	4.0	9.0	10.0
e	5.0	6.0	7.0	8.0

```
dfa
```

	Ohio	Nevada
a	1.0	2.0
c	3.0	4.0
e	5.0	6.0

```
dfb
```

	Misouri	Alabama
e	7.0	8.0
c	9.0	10.0
d	11.0	12.0
b	13.0	14.0

```
join3 = dfa.join(dfb, how = 'left')
join3
```

	Ohio	Nevada	Misouri	Alabama
a	1.0	2.0	NaN	NaN
c	3.0	4.0	9.0	10.0
e	5.0	6.0	7.0	8.0

```
join4 = dfa.join(dfb, how = 'right')
join4
```

	Ohio	Nevada	Misouri	Alabama
e	5.0	6.0	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
b	NaN	NaN	13.0	14.0

```
join5 = dfa.join(dfb, how = 'right', sort = True)
join5
```

	Ohio	Nevada	Misouri	Alabama
b	NaN	NaN	13.0	14.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	7.0	8.0

4) .concat( ) – 시리즈 or 데이터프레임을 합쳐 주는 매서드

Series와 concat

```
s1 = pd.Series(['a', 'b'])
s1
```

```
0    a
1    b
dtype: object
```

```
s2 = pd.Series(['c', 'd'])
s2
```

```
0    c
1    d
dtype: object
```

```
pd.concat([s1, s2])
```

```
0    a
1    b
0    c
1    d
dtype: object
```

```
pd.concat([s1, s2], ignore_index=True)
```

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

Series와 concat

```
s1 = pd.Series(['a', 'b'])
s1
```

```
0    a
1    b
dtype: object
```

```
s2 = pd.Series(['c', 'd'])
s2
```

```
0    c
1    d
dtype: object
```

```
pd.concat([s1, s2], axis = 0)
```

```
0    a
1    b
0    c
1    d
dtype: object
```

행 기준 연결
결과: 시리즈

```
pd.concat([s1, s2], axis = 1)
```

	0	1
0	a	c
1	b	d

열 기준 연결
결과: 데이터프레임

DataFrame과 concat

```
df1 = pd.DataFrame([['a', 1], ['b', 2]],
                   columns=['letter', 'number'])
df1
```

	letter	number
0	a	1
1	b	2

```
df2 = pd.DataFrame([['c', 3], ['d', 4]],
                   columns=['letter', 'number'])
df2
```

	letter	number
0	c	3
1	d	4

```
pd.concat([df1, df2])
```

	letter	number
0	a	1
1	b	2
0	c	3
1	d	4

```
pd.concat([df1, df2])
```

	letter	number
0	a	1
1	b	2
0	c	3
1	d	4

왼쪽의 결과를

- gf변수에 담고,
- index 기준으로 정렬 하려면?

```
gf = pd.concat([df1, df2])
gf.sort_index()
```

	letter	number
0	a	1
0	c	3
1	b	2
1	d	4

```
gf = pd.concat([df1, df2])
gf.sort_index()
```

	letter	number
0	a	1
0	c	3
1	b	2
1	d	4

- **.reset_index( )**

```
ef = pd.concat([df1, df2]).reset_index()
ef
```

	index	letter	number
0	0	a	1
1	1	b	2
2	0	c	3
3	1	d	4

"index"라는 열 없애려면?

```
ef = ef.drop(columns = 'index')
ef
```

	letter	number
0	a	1
1	b	2
2	c	3
3	d	4

```
df3 = pd.DataFrame([['c', 3, 'cat'], ['d', 4, 'dog']],
                   columns=['letter', 'number', 'animal'])
df3
```

	letter	number	animal
0	c	3	cat
1	d	4	dog

```
df1
```

	letter	number
0	a	1
1	b	2

```
pd.concat([df1, df3], sort=False)
```

	letter	number	animal
0	a	1	NaN
1	b	2	NaN
0	c	3	cat
1	d	4	dog

```
pd.concat([df1, df3], sort=True)
```

	animal	letter	number
0	NaN	a	1
1	NaN	b	2
0	cat	c	3
1	dog	d	4

```
df3 = pd.DataFrame([['c', 3, 'cat'], ['d', 4, 'dog']],
                   columns=['letter', 'number', 'animal'])
df3
```

	letter	number	animal
0	c	3	cat
1	d	4	dog

```
df1
```

	letter	number
0	a	1
1	b	2

```
pd.concat([df1, df3], join="inner")
```

	letter	number
0	a	1
1	b	2
0	c	3
1	d	4

```
pd.concat([df1, df3], join="outer", sort = True)
```

	animal	letter	number
0	NaN	a	1
1	NaN	b	2
0	cat	c	3
1	dog	d	4

- concat에서는 inner와 outer만 사용 가능, left, right 은 지원 안 함

1. Reshaping Data 4) .concat( )

```
df4 = pd.DataFrame([['bird', 'polly'], ['monkey', 'george']],
                   columns=['animal', 'name'])
df4
```

	animal	name
0	bird	polly
1	monkey	george

```
df1
```

	letter	number
0	a	1
1	b	2

```
pd.concat([df1, df4], axis = 0)
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-p
A future version
of pandas will change to not sort b

To accept the future behavior, pass

To retain the current behavior and

  """Entry point for launching an
```

	animal	letter	name	number
0	NaN	a	NaN	1.0
1	NaN	b	NaN	2.0
0	bird	NaN	polly	NaN
1	monkey	NaN	george	NaN

```
pd.concat([df1, df4], axis = 1)
```

	letter	number	animal	name
0	a	1	bird	polly
1	b	2	monkey	george

```
pd.concat([df1, df4], axis = 0, sort = True)
```

	animal	letter	name	number
0	NaN	a	NaN	1.0
1	NaN	b	NaN	2.0
0	bird	NaN	polly	NaN
1	monkey	NaN	george	NaN

```
pd.concat([df1, df4], axis = 0, sort = False)
```

	letter	number	animal	name
0	a	1.0	NaN	NaN
1	b	2.0	NaN	NaN
0	NaN	NaN	bird	polly
1	NaN	NaN	monkey	george

```
pd.concat?
```

```
pd.concat( objs, axis=0, join='outer', join_axes=None, ignore_index=False, keys=None,
levels=None, names=None, verify_integrity=False, sort=None, copy=True,
```

- axis = 1, 열을 추가하는 방식으로 데이터 세트 합침
- join = 'outer' 또는 'inner'만 가능 (right, left join은 지원 안 함)
- ignore_index = True 기존의 index는 무시하고 concat 결과에 새 index 부여
- Keys, levels 는 multi index일때 사용하는 옵션
- verify_integrity = True 면 중복되는 index가 있는지 검사, 중복 index 있으면 에러 메시지
- sort = True concat 결과 열들을 정렬

```
s3 = pd.concat([s1, s2], keys=['s1', 's2'])
s3
```

```
s1  0    a
    1    b
s2  0    c
    1    d
dtype: object
```

```
s3.index
```

```
MultiIndex(levels=[['s1', 's2'], [0, 1]],
           codes=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

```
s3.keys()
```

```
MultiIndex(levels=[['s1', 's2'], [0, 1]],
           codes=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

```
pd.concat([s1, s2], keys=['s1', 's2'],names=['Series name', 'Row ID'] )
```

```
Series name  Row ID
s1           0         a
             1         b
s2           0         c
             1         d
dtype: object
```

```
df5 = pd.DataFrame([1], index=['a'])
df5
```

	0
a	1

```
df6 = pd.DataFrame([2], index=['a'])
df6
```

	0
a	2

```
pd.concat([df5, df6])
```

	0
a	1
a	2

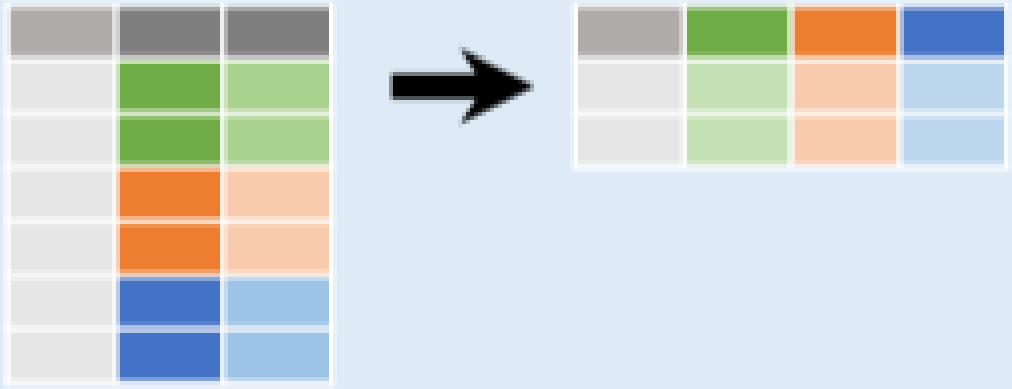
```
pd.concat([df5, df6], verify_integrity=True)
```

...

위의 코드가 에러나는 이유: 중복되는 인덱스 값이 있으면 verify_integrity=True 옵션 주면 검증을 위함

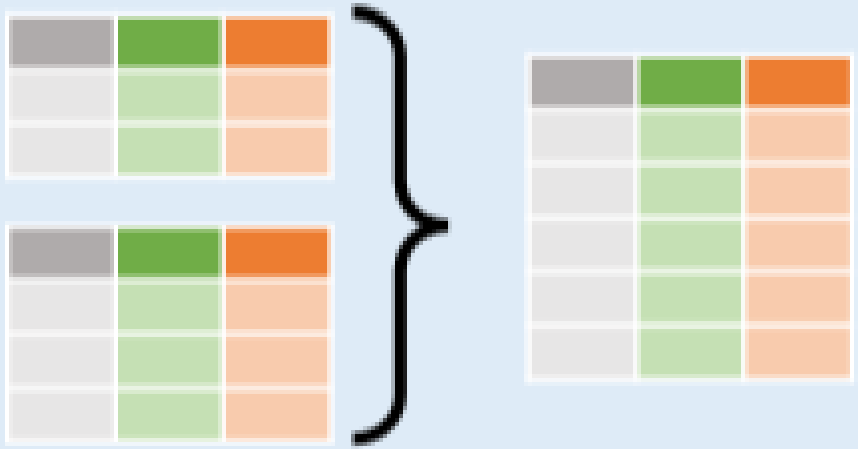
`pd.melt(df)`

Gather columns into rows.



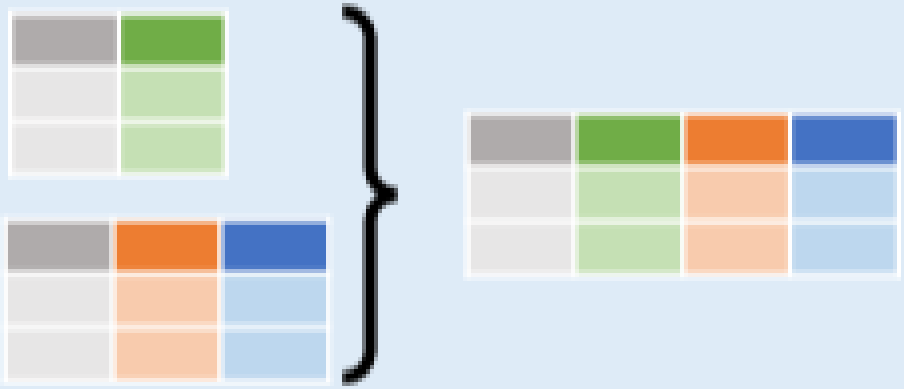
`df.pivot(columns='var', values='val')`

Spread rows into columns.



`pd.concat([df1,df2])`

Append rows of DataFrames



`pd.concat([df1,df2], axis=1)`

Append columns of DataFrames

2. Combine Data Set

.merge( ) – 공통된 열을 기준으로 데이터 세트를 합침

adf

	x1	x2
0	A	1
1	B	2
2	C	3

bdf

	x1	x3
0	A	T
1	B	F
2	D	T

- x1을 기준으로 adf와 bdf를 merge

	x1	x2	x3
0	A	1	T
1	B	2	F
2	C	3	NaN

```
pd.merge(adf, bdf, how='left', on='x1')
```

	x1	x2	x3
0	A	1.0	T
1	B	2.0	F
2	D	NaN	T

```
pd.merge(adf, bdf, how='right', on='x1')
```

```
pd.merge(adf, bdf, how='inner', on='x1')
```

	x1	x2	x3
0	A	1	T
1	B	2	F

adf

	x1	x2
0	A	1
1	B	2
2	C	3

bdf

	x1	x3
0	A	T
1	B	F
2	D	T

```
pd.merge(adf, bdf, how='outer', on='x1')
```

	x1	x2	x3
0	A	1.0	T
1	B	2.0	F
2	C	3.0	NaN
3	D	NaN	T

- Filtering Join

adf

	x1	x2
0	A	1
1	B	2
2	C	3

bdf

	x1	x3
0	A	T
1	B	F
2	D	T

```
bdf.x1
```

왼쪽 코드 예상 결과는?

```
0    A
1    B
2    D
```

```
adf.x1.isin(bdf.x1)
```

```
0     True
1     True
2    False
```

위의 것 무슨 뜻?

'adf 데이터 프레임의 x1열에 bdf.x1의 값이 있나'

True에 해당하는 행들만 보고 싶으면?

```
adf[adf.x1.isin(bdf.x1)]
```

	x1	x2
0	A	1
1	B	2

반대에 해당하는 행들만 보고 싶으면?

```
adf[~adf.x1.isin(bdf.x1)]
```

	x1	x2
2	C	3

ydf

	x1	x2
0	A	1
1	B	2
2	C	3

zdf

	x1	x2
0	B	2
1	C	3
2	D	4

```
pd.merge(ydf, zdf)
```

	x1	x2
0	B	2
1	C	3

기본값= inner join, 2개의 데이터 세트에 공통된 값만 join

```
pd.merge(ydf, zdf, how='outer')
```

	x1	x2
0	A	1
1	B	2
2	C	3
3	D	4

ydf

	x1	x2
0	A	1
1	B	2
2	C	3

zdf

	x1	x2
0	B	2
1	C	3
2	D	4

```
pd.merge(ydf, zdf, how='outer', indicator=True)
```

	x1	x2	_merge	
0	A	1	left_only	왼쪽세트만 값이 있다
1	B	2	both	둘 다 값 있다
2	C	3	both	
3	D	4	right_only	오른쪽 세트만 값 있다

```
pd.merge(ydf, zdf, how='outer', indicator=True) .query('_merge == "left_only"')
```

	x1	x2	_merge
0	A	1	left_only

```
pd.merge?
```

```
pd.merge( left, right, how='inner', on=None, left_on=None, right_on=None,
          left_index=False, right_index=False,
          sort=False, suffixes=('x', 'y'), copy=True, indicator=False, validate=None)
```

데이터 세트를 아래로 붙일 때 주로 concat
데이터 세트를 옆으로 붙일 때 주로 merge, 단 공통으로 갖는 열이 꼭 있어야 함

apply(function)

```
df.apply?
```

```
df['species'].apply(lambda x : x[:3])
```

```
df['species_3'] = df['species'].apply(lambda x : x[:3])
df
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_3
0	5.1	3.5	1.4	0.2	setosa	set
1	4.9	3.0	1.4	0.2	setosa	set
2	4.7	3.2	1.3	0.2	setosa	set
3	4.6	3.1	1.5	0.2	setosa	set

```
import pandas as pd
```

```
import seaborn as sns
```

```
df = sns.load_dataset("iris")
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

2. Combine Data Set cf) .apply(function)

```
def smp (x):
    x = x[-3:]
    return x
```

```
df['species'].apply(smp)
```

```
0       osa
1       osa
2       osa
3       osa
4       osa
5       osa
6       osa
7       osa
8       osa
9       osa
10      osa
11      ...
```

```
df['species-3'] = df['species'].apply(smp)
df
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_3	species-3
0	5.1	3.5	1.4	0.2	setosa	set	osa
1	4.9	3.0	1.4	0.2	setosa	set	osa
2	4.7	3.2	1.3	0.2	setosa	set	osa
3	4.6	3.1	1.5	0.2	setosa	set	osa
4	5.0	3.6	1.4	0.2	setosa	set	osa
5	5.4	3.9	1.7	0.4	setosa	set	osa
6	4.6	3.4	1.4	0.3	setosa	set	osa

3. Group Data

.groupby( ) – 지정한 열을 각 그룹으로 만들어 각종계산 수행

```
import pandas as pd
import seaborn as sns
```

```
df = sns.load_dataset("mpg")
df.shape
```

```
(398, 9)
```

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst

3. Group Data .groupby( ).agg( )

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

```
df.groupby(by="origin").min()
```

origin	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	name
europe	16.2	4	68.0	46.0	1825	12.2	70	audi 100 ls
japan	18.0	3	70.0	52.0	1613	11.4	70	datsun 1200
usa	9.0	4	85.0	52.0	1800	8.0	70	amc ambassador brougham

3. Group Data .groupby( ).agg( )

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

```
df.groupby(by="origin")['cylinders'].mean()
```

```
origin
europe    4.157143
japan     4.101266
usa       6.248996
Name: cylinders, dtype: float64
```

3. Group Data .groupby( ).agg( )

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	usa
1	15.0	8	350.0	165.0	3693	11.5	70	usa
2	18.0	8	318.0	150.0	3436	11.0	70	usa
3	16.0	8	304.0	150.0	3433	12.0	70	usa
4	17.0	8	302.0	140.0	3449	10.5	70	usa

```
df.groupby(['model_year','origin'])['cylinders'].median()
```

model_year	origin	
70	europe	4.0
	japan	4.0
	usa	8.0
71	europe	4.0
	japan	4.0
	usa	6.0
72	europe	4.0
	japan	4.0
	usa	8.0
73	europe	4.0
	japan	4.0
	usa	8.0
74	europe	4.0
	japan	4.0
	usa	6.0
75	europe	4.0
	japan	4.0
	usa	6.0
76	europe	4.0
	japan	4.0
	usa	6.0
77	europe	4.0

3. Group Data .groupby( ).agg( )

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	usa
1	15.0	8	350.0	165.0	3693	11.5	70	usa
2	18.0	8	318.0	150.0	3436	11.0	70	usa
3	16.0	8	304.0	150.0	3433	12.0	70	usa
4	17.0	8	302.0	140.0	3449	10.5	70	usa

```
df.groupby(['model_year','origin'])['cylinders'].median()
```

```
pd.DataFrame(df.groupby(['model_year','origin'])['cylinders'].median())
```

		cylinders
model_year	origin	
70	europe	4.0
	japan	4.0
	usa	8.0
71	europe	4.0
	japan	4.0
	usa	6.0
72	europe	4.0
	japan	4.0
	usa	8.0
73	europe	4.0
	japan	4.0
	usa	8.0
74	europe	4.0
	japan	4.0

3. Group Data .groupby( ).agg( )와 pivot( ) 비교

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	18.0	8	307.0	130.0	3504	12.0	70	usa
1	15.0	8	350.0	165.0	3693	11.5	70	usa
2	18.0	8	318.0	150.0	3436	11.0	70	usa
3	16.0	8	304.0	150.0	3433	12.0	70	usa
4	17.0	8	302.0	140.0	3449	10.5	70	usa

```
df.groupby(['model_year','origin'])['cylinders'].median()
```

```
pd.DataFrame(df.groupby(['model_year','origin'])['cylinders'].median())
```

origin/model_year 별로 cylinders 보려면?

		cylinders
model_year	origin	
70	europe	4.0
	japan	4.0
	usa	8.0
71	europe	4.0
	japan	4.0
	usa	6.0
72	europe	4.0
	japan	4.0
	usa	8.0
73	europe	4.0
	japan	4.0
	usa	8.0
74	europe	4.0
	japan	4.0

3. Group Data .groupby( ).agg( )와 pivot( ) 비교

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

즉, 아래 결과 처럼 origin/model_year 별로 cylinders 보려면?

model_year	70	71	72	73	74	75	76	77	78	79	80	81	82
origin													
europe	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.5	4.0	4.0	4.0	4.0
japan	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
usa	8.0	6.0	8.0	8.0	6.0	6.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0

예전에 배운 매서드 중에...

```
df2
```

	foo	bar	baz
0	one	A	1
1	one	B	2
2	one	C	3
3	two	A	4
4	two	B	5
5	two	C	6

```
ef = df2.pivot(index='foo', columns='bar', values='baz')
ef
```

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

```
ef.index
```

```
Index(['one', 'two'], dtype='object', name='foo')
```

```
ef.index.name
```

```
'foo'
```

```
ef.columns.name
```

```
'bar'
```

```
ef.columns
```

```
Index(['A', 'B', 'C'], dtype='object', name='bar')
```

3. Group Data .groupby( ).agg( )와 pivot( ) 비교

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

model_year	70	71	72	73	74	75	76	77	78	79	80	81	82
origin													
europe	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.5	4.0	4.0	4.0	4.0
japan	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
usa	8.0	6.0	8.0	8.0	6.0	6.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0

index=origin, columns=model_year, values =cylinders

df.pivot(index='origin', columns= 'model_year', values='cylinders') 실행… ??...!!!!!!

3. Group Data .groupby( ).agg( )와 pivot( ) 비교

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

model_year	70	71	72	73	74	75	76	77	78	79	80	81	82
origin													
europe	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.5	4.0	4.0	4.0	4.0
japan	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
usa	8.0	6.0	8.0	8.0	6.0	6.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0

index=origin, columns=model_year, values =cylinders

pivot이 안되는 경우 ㅠㅠ

- 중복값이 있는 경우
- columns가 2개 이상인 경우
- index가 2개 이상인 경우

model_year	70	71	72	73	74	75	76	77	78	79	80	81	82
origin													
europe	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.5	4.0	4.0	4.0	4.0
japan	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
usa	8.0	6.0	8.0	8.0	6.0	6.0	6.0	6.0	6.0	6.0	4.0	4.0	4.0

pivot이 안되는 경우
- 중복값이 있는 경우
- columns가 2개 이상인 경우
- index가 2개 이상인 경우

index=origin, columns=model_year, values =cylinders

➔ pivot_table로 가능!!

- 중복값

```
df.pivot_table(index= ' origin ' , columns=  ' model_year ' , values= ' cylinders')
df.pivot_table(index= ' origin ' , columns=  ' model_year ' , values= ' cylinders ' , aggfunc= ' median')
```

- 열 2개 이상

```
df.pivot_table(index='origin', columns= ['model_year', 'cylinders'], values = 'mpg')
```

- 인덱스 2개 이상(multi-index)

```
df.pivot_table(index=['origin', 'model_year'], columns= 'cylinders', values = 'mpg')
df.pivot_table(index=['origin', 'model_year'], columns= 'cylinders', values = 'mpg', aggfunc='median')
```

참고 주제:

시계열 데이터
(Time Series Data)

1. 시계열(Time Series) 데이터

- 시간의 변화에 따른 관측 데이터
  - 예: 주식, 금융, 날짜별 판매량, 월별 취업률 등등
  - 추세 파악이 필요한 경우에 많이 사용
- 데이터의 Index가 시간(timestamp)인 데이터

```
import datetime as dt
```

- dt.datetime.today( )
- dt.datetime.now( )
- dt.datetime.today( ).weekday( )
  - 0:월, 1:화, 2:수, 3:목, 4:금, 5:토, 6:일
- dt.datetime(1995, 12, 5)

```
from datetime import datetime
```

- datetime.today( )
- datetime.now( )
- datetime.today( ).weekday( )
  - datetime.today( ).hour
- datetime(1995, 12, 5)
  - datetime(1995,12,5).month

1. 시계열(Time Series) 데이터

- 시계열변수.strftime('형식'): 날짜 관련 형태를 지정된 형식으로 반환

```
date1 = datetime.now()
date1
```

```
datetime.datetime(2020, 11, 27, 9, 35, 17
```

```
date1.strftime('%Y')
```

```
'2020'
```

```
date1.strftime('%b')
```

```
'Nov'
```

```
date1.strftime('%y')
```

```
'20'
```

```
date1.strftime('%d')
```

```
'27'
```

```
date1.strftime('%B')
```

```
'November'
```

```
date1.strftime('%A')
```

```
'Friday'
```

```
print(date1.strftime('%I'), date1.strftime('%p'))
```

```
09 AM
```

형식	의미	예
%Y (%y)	네자리(두자리)연도	2010(10), 2018(18), 1999(99)
%B (%b)	국가별 월 이름	January(Jan), February(Feb)
%d	날짜	01, 02, …, 30, 31
%A (%a)	국가별 요일 이름	Sunday(Sun), Monday(Mon)
%H	24시간 단위 시간	00, 01, …, 23
%I %p	12시간 단위 시간, 오전/오후	01, 02, …, 12 AM PM
%M %S	분 초	00, 01, …, 59
%x	국가별 날짜 표현	08/16/88
%c	국가별 날짜, 시간 표현	Tue Aug 16 21:30:00 1988

1. 시계열(Time Series) 데이터

- 날짜를 일일이 입력하는 방식

```
ts1 = [datetime(2020,1,1), datetime(2021,1,2)]
ts1
```

```
[datetime.datetime(2020, 1, 1, 0, 0), datetime.datetime(2021, 1, 2, 0, 0)]
```

```
ts_ind = pd.DatetimeIndex(ts1)
ts_ind
```

```
DatetimeIndex(['2020-01-01', '2021-01-02'], dtype='datetime64[ns]', freq=None)
```

```
pd.DataFrame(np.random.randn(2, 2), ts_ind, columns=['a', 'b'])
```

	a	b
2020-01-01	-0.379624	0.580391
2021-01-02	-0.244201	-0.127505

1. 시계열(Time Series) 데이터

- 날짜 범위로 입력하는 방식 : pd.date_range(start=, end=, periods=, freq=)

```
pd.date_range('2021-12-01', '2021-12-10')
```

```
DatetimeIndex(['2021-12-01', '2021-12-02', '2021-12-03', '2021-12-04',
               '2021-12-05', '2021-12-06', '2021-12-07', '2021-12-08',
               '2021-12-09', '2021-12-10'],
              dtype='datetime64[ns]', freq='D')
```

start	시작 날짜(시간) 지정
end	끝 날짜(시간) 지정
periods	생성할 날짜(시간) 수 지정
freq	날짜 단위, 시간 단위 등에 대한 지정 ('y' 'm' 'd' 'h' 'min' 'w' 등, 기본값 : 'D')

```
aa= pd.date_range('2021-01-01', '2021-06-30')
pd.DataFrame(np.random.randn(len(aa),2), aa, ['a', 'b'])
```

	a	b
2021-01-01	0.182764	0.836223
2021-01-02	-0.951761	-0.040994
2021-01-03	0.455606	0.416059
2021-01-04	-0.777953	-1.465658
2021-01-05	2.048241	2.437984
2021-01-06	0.446145	0.562055

```
pd.date_range('2020-12-10', periods=4, freq='d')
```

```
DatetimeIndex(['2020-12-10', '2020-12-11', '2020-12-12', '2020-12-13'],
dtype='datetime64[ns]', freq='D')
```

```
pd.date_range(end='2020-12-10', periods=4, freq='d')
```

```
DatetimeIndex(['2020-12-07', '2020-12-08', '2020-12-09', '2020-12-10'],
dtype='datetime64[ns]', freq='D')
```

```
pd.date_range('2020-12-10', periods=4, freq='m')
```

```
DatetimeIndex(['2020-12-31', '2021-01-31', '2021-02-28', '2021-03-31'],
dtype='datetime64[ns]', freq='M')
```

1. 시계열(Time Series) 데이터

- 날짜 범위로 입력하는 방식2 : pd.period_range(start=, end=, periods=, freq=)

```
pd.date_range('2021-12-01', '2021-12-10')
```

```
DatetimeIndex(['2021-12-01', '2021-12-02', '2021-12-03', '2021-12-04',
               '2021-12-05', '2021-12-06', '2021-12-07', '2021-12-08',
               '2021-12-09', '2021-12-10'],
              dtype='datetime64[ns]', freq='D')
```

```
pd.period_range('2021-12-01', '2021-12-10')
```

```
PeriodIndex(['2021-12-01', '2021-12-02', '2021-12-03', '2021-12-04',
             '2021-12-05', '2021-12-06', '2021-12-07', '2021-12-08',
             '2021-12-09', '2021-12-10'],
            dtype='period[D]', freq='D')
```

- start 기간 시작 날짜(시간) 지정
- end 기간 끝 날짜(시간) 지정
- periods 생성할 날짜(시간) 수 지정
- freq 날짜 단위, 시간 단위 등에 대한 지정

```
pd.date_range(end='2020-12-10', periods=4, freq='m')
```

```
DatetimeIndex(['2020-08-31', '2020-09-30', '2020-10-31', '2020-11-30'],
dtype='datetime64[ns]', freq='M')
```

```
pd.period_range(end='2020-12-10', periods=4, freq='m')
```

```
PeriodIndex(['2020-09', '2020-10', '2020-11', '2020-12'], dtype='period
[M]', freq='M')
```

1. 시계열(Time Series) 데이터

- 외부 데이터 불러 올 때 종종 발생하는 상황 : 날짜를 날짜로 인식 안 하는;;

```
df2 = pd.read_csv('walmart_stock.csv')
df2.head()
```

	Date	Open	High	Low	C
0	2012-01-03	59.970001	61.060001	59.869999	60.33
1	2012-01-04	60.209999	60.349998	59.470001	59.70
2	2012-01-05	59.349998	59.619999	58.369999	59.41
3	2012-01-06	59.419998	59.450001	58.869999	59.00
4	2012-01-09	59.029999	59.549999	58.919998	59.18

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 7 columns):
Date          1258 non-null object
Open          1258 non-null float64
High          1258 non-null float64
Low           1258 non-null float64
Close         1258 non-null float64
Volume        1258 non-null int64
Adj Close     1258 non-null float64
dtypes: float64(5), int64(1), object(1)
memory usage: 68.9+ KB
```

- 위의 예에서 Date를 object라 인식하고 있음..

1. 시계열(Time Series) 데이터

- 외부 데이터 불러 올 때 종종 발생하는 상황 : 날짜를 날짜로 인식 안 하는;;
- 해결 방법 1 : pd.to_datetime(해당 열)/ .set_index('해당열', inplace=True)

```
df2 = pd.read_csv('walmart_stock.csv')
df2.head()
```

	Date	Open	High	Low	C
0	2012-01-03	59.970001	61.060001	59.869999	60.33
1	2012-01-04	60.209999	60.349998	59.470001	59.70

```
df2.Date = pd.to_datetime(df2.Date)
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 7 columns):
Date          1258 non-null datetime64[ns]
Open          1258 non-null float64
```

```
df2.set_index('Date', inplace=True)
df2.head()
```

Date	Open	High	Low	Close	Volu
2012-01-03	59.970001	61.060001	59.869999	60.330002	126688
2012-01-04	60.209999	60.349998	59.470001	59.709999	95933
2012-01-05	59.349998	59.619999	58.369999	59.419998	127682
2012-01-06	59.419998	59.450001	58.869999	59.000000	80694
2012-01-09	59.029999	59.549999	58.919998	59.180000	66793

1. 시계열(Time Series) 데이터

- 외부 데이터 불러 올 때 종종 발생하는 상황 : 날짜를 날짜로 인식 안 하는;;
- 해결 방법 2 : read_csv에서 옵션 주기 index_col='해당 열' parse_dates=True

```
df4 = pd.read_csv('walmart_stock.csv', index_col='Date', parse_dates=True)
df4.head()
```

parse_dates=True 날짜로 볼 수 있는 것은 날짜로 인식하라는 옵션

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001	61.060001	59.869999	60.330002	12668800	52.619235
2012-01-04	60.209999	60.349998	59.470001	59.709999	9593300	52.078475
2012-01-05	59.349998	59.619999	58.369999	59.419998	12768200	51.825539
2012-01-06	59.419998	59.450001	58.869999	59.000000	8069400	51.459220
2012-01-09	59.029999	59.549999	58.919998	59.180000	6679300	51.616215

1. 시계열(Time Series) 데이터

- 연월일시분초를 각각 떼어내서 쓰고 싶을 때.. (시계열 데이터가 index아님)

```
print(train.shape)
train.head()
```

```
(10886, 12)
```

	datetime	season	holiday	workingday	wea
0	2011-01-01 00:00:00	1	0	0	
1	2011-01-01 01:00:00	1	0	0	
2	2011-01-01 02:00:00	1	0	0	
3	2011-01-01 03:00:00	1	0	0	
4	2011-01-01 04:00:00				

```
train["datetime-year"] = train["datetime"].dt.year
train["datetime-month"] = train["datetime"].dt.month
train["datetime-day"] = train["datetime"].dt.day
train["datetime-hour"] = train["datetime"].dt.hour
train["datetime-minute"] = train["datetime"].dt.minute
train["datetime-second"] = train["datetime"].dt.second
```

	datetime	datetime-year	datetime-month	datetime-day	datetime-hour	datetime-minute	datetime-second
0	2011-01-01 00:00:00	2011	1	1	0	0	0
1	2011-01-01 01:00:00	2011	1	1	1	0	0
2	2011-01-01 02:00:00	2011	1	1	2	0	0

1. 시계열(Time Series) 데이터

- 시계열데이터의 resampling: 시간 유형 변경으로 데이터를 재 조정

시계열 데이터 리샘플링 함수

```
df변수.resample(rule).집계함수()          # 업 샘플링
df변수.resample(rule).채우기함수()        # 다운 샘플링
```

- 업 샘플링 예: 연월일시 ➔ 연월일
- 다운 샘플링 예: 연월일 ➔ 연월일시

- *rule* 리샘플링 기간 설정 (예: 'm' '2d' '5min' 등)
- *집계함수* 업 샘플링 사용할 함수 (max() min() sum() mean() ohlc() 등)
  ohlc – open high low closed
- *채우기함수* 다운 샘플링시 사용할 보간법 함수 (bfill() ffill() 등)

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001	61.060001	59.869999	60.330002	12668800	52.619235
2012-01-04	60.209999	60.349998	59.470001	59.709999	9593300	52.078475
2012-01-05	59.349998	59.619999	58.369999	59.419998	12768200	51.825539
2012-01-06	59.419998	59.450001	58.869999	59.000000	8069400	51.459220
2012-01-09	59.029999	59.549999	58.919998	59.180000	6679300	51.616215

1. 시계열(Time Series) 데이터

- 시계열데이터의 resampling: 시간 유형 변경으로 데이터를 재 조정

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001	61.060001	59.869999	60.330002	12668800	52.619235
2012-01-04	60.209999	60.349998	59.4			
2012-01-05	59.349998	59.619999	58.3			
2012-01-06	59.419998	59.450001	58.8			
2012-01-09	59.029999	59.549999	58.9			

```
df4.resample(rule='A').mean()
```

Date	Open	High	Lo
2012-12-31	67.158680	67.602120	66.78652
2013-12-31	75.264048	75.729405	74.84305
2014-12-31	77.274524	77.740040	76.86440
2015-12-31	72.569405	73.064167	72.03480
2016-12-31	69.481349	70.019643	69.02349

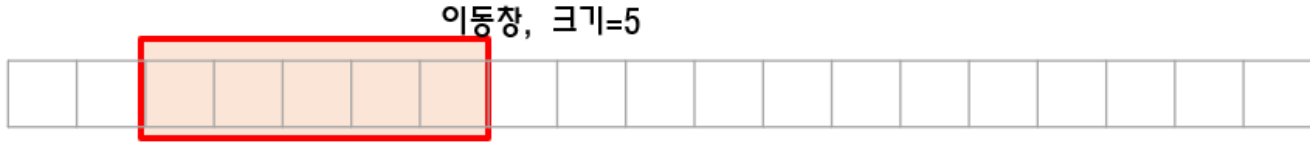
```
df4.resample(rule='Q').mean()
```

Date	Open	High	Lo
2012-03-31	60.462903	60.807258	60.16048
2012-06-30	62.888889	63.400159	62.59222
2012-09-30	73.081587	73.549682	72.71761
2012-12-31	72.174678	72.623226	71.64774
2013-03-31	70.898834	71.393000	70.55266
2013-06-30	76.873906	77.417813	76.41328

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM

  - 이동 창(moving window)



  - 단순 이동 창 생성

```
이동창변수 = Series변수.rolling(size, min_periods=1)
이동창변수 = df변수.rolling(size)
```

    - *size* 이동 창의 크기
    - min_periods 이동창의 최소 기간

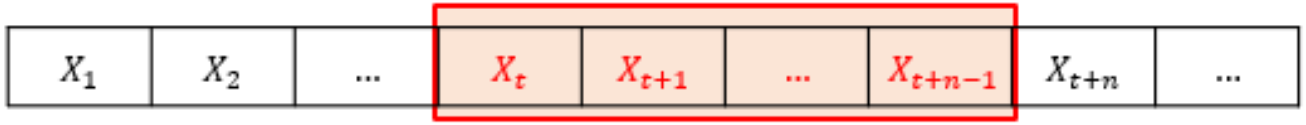
  - 지수 가중 이동 창 생성

```
이동창변수 = df변수.ewm(span=None, alpha=None, min_periods=1)
```

    - span 이동창의 크기
    - min_periods 이동 창의 최소 기간
    - alpha 평활계수

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM



- **이동 평균(moving average)**
  - 시간의 흐름에 따라 가장 오래된 변수의 값을 빼고, 새로운 변수의 값을 추가하여 구하는 평균

- **단순 이동 평균(n-period moving average)**
  - 대상 기간 데이터의 중요도를 동일하게 취급하여 구해진 평균

$$MA_t = \frac{\sum_{i=0}^{n-1} X_i}{n}, t = 1, 2, \dots$$

- **지수 가중 이동 평균(exponential weighted moving average)**
  - 최근의 데이터에 더 높은 가중치를 부여하여 구해진 평균

$$EMA_t = \alpha X_{t+n-1} + (1-\alpha) EWA_{t-1}, t = 1, 2, \dots \quad (\text{단, } EMA_1 = MA_1)$$

  - α 평활 계수

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001 ①	61.060001	59.869999	60.330002	12668800	52.6
2012-01-04	60.209999 ②	60.349998	59.470001	59.709999	9593300	52.0
2012-01-05	59.349998 ③	59.619999	58.369999	59.419998	12768200	51.8
2012-01-06	59.419998 ④	59.450001	58.869999	59.000000	8069400	51.4
2012-01-09	59.029999 ⑤	59.549999	58.919998	59.180000	6679300	51.6

```
df2.rolling(3, min_periods=2).sum()
```

Date	Open	High	Low	
2012-01-03	NaN	NaN	NaN	
2012-01-04	120.180000	121.409999	119.340000	①+②
2012-01-05	179.529998	181.029998	177.709999	①+②+③
2012-01-06	178.979995	179.419998	176.709999	②+③+④
2012-01-09	177.799995	178.619999	176.159996	③+④+⑤

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001 ①	61.060001	59.869999	60.330002	12668800	52.6
2012-01-04	60.209999 ②	60.349998	59.470001	59.709999	9593300	52.0
2012-01-05	59.349998 ③	59.619999	58.369999	59.419998	12768200	51.8
2012-01-06	59.419998 ④	59.450001	58.869999	59.000000	8069400	51.4
2012-01-09	59.029999 ⑤	59.549999	58.919998	59.180000	6679300	51.6

```
df2.rolling(2, min_periods=2).sum()
```

Date	Open	High	Low	
2012-01-03	NaN	NaN	NaN	
2012-01-04	120.180000	121.409999	119.340000	①+②
2012-01-05	119.559997	119.969997	117.840000	②+③
2012-01-06	118.769996	119.070000	117.239998	③+④
2012-01-09	118.449997	119.000000	117.789997	④+⑤

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001 ①	61.060001	59.869999	60.330002	12668800	52
2012-01-04	60.209999 ②	60.349998	59.470001	59.709999	9593300	52
2012-01-05	59.349998 ③	59.619999	58.369999	59.419998	12768200	51
2012-01-06	59.419998 ④	59.450001	58.869999	59.000000	8069400	51
2012-01-09	59.029999 ⑤	59.549999	58.919998	59.180000	6679300	51

```
df2.rolling(3, min_periods=3).sum()
```

Date	Open	High	Low	
2012-01-03	NaN	NaN	NaN	
2012-01-04	NaN	NaN	NaN	
2012-01-05	179.529998	181.029998	177.709999	①+②+③
2012-01-06	178.979995	179.419998	176.709999	②+③+④
2012-01-09	177.799995	178.619999	176.159996	③+④+⑤

1. 시계열(Time Series) 데이터

- 이동창 - Rolling과 EWM

Date	Open	High	Low	Close	Volume	Adj Close
2012-01-03	59.970001 ①	61.060001	59.869999	60.330002		
2012-01-04	60.209999 ②	60.349998	59.470001	59.709999		
2012-01-05	59.349998 ③	59.619999	58.369999	59.419998		
2012-01-06	59.419998 ④	59.450001	58.869999	59.000000		
2012-01-09	59.029999 ⑤	59.549999	58.919998	59.180000		

```
df2.ewm(alpha=0.3, min_periods=2).mean()
```

Date	Open	High	Low
2012-01-03	NaN	NaN	NaN
2012-01-04	60.111176	60.642352	59.634706
2012-01-05	59.763606	60.175524	59.057214
2012-01-06	59.627954	59.889096	58.983304
2012-01-09	59.412327	59.766815	58.960475

①,② 평균

①,②,③ 평균

①,②,③,④ 평균

①,②,③,④,⑤ 평균