Pandas

Pandas

구조화된 데이터의 처리를 지원하는 Python 라이브러리

Python계의 엑셀!

Pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

Series

In [2]:
```
list_data = [1,2,3,4,5]
example_obj = Series(data = list_data)
example_obj
```

Out[2]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Index

data

Data type

Series

In [3]:
```
list_data = [1,2,3,4,5]
list_name = ["a","b","c","d","e"]
example_obj = Series(data = list_data, index=list_name)
example_obj
```

index 이름을 지정

Out[3]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

Series

Data와 index 이름을 지정

```
In [4]: dict_data = {"a":1, "b":2, "c":3, "d":4, "e":5}
        example_obj = Series(dict_data, dtype=np.float32, name="example_data")
        example_obj
```

data type 설정

series 이름 설정

```
Out[4]: a    1.0
        b    2.0
        c    3.0
        d    4.0
        e    5.0
        Name: example_data, dtype: float32
```

Series

data index에 접근하기

```
In [5]: example_obj["a"]

Out[5]: 1.0
```

data index에 값 할당하기

```
In [6]: example_obj["a"] = 3.2
        example_obj

Out[6]: a    3.2
        b    2.0
        c    3.0
        d    4.0
        e    5.0
        Name: example_data, dtype: float32
```

Series

```
example_obj.values
```

값 리스트만

```
array([ 3.20000005,  2.        ,  3.        ,  4.        ,  5.        ],
 dtype=float32)
```

```
example_obj.index
```

Index 리스트만

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

```
example_obj.name = "number"
example_obj.index.name = "alphabet"
example_obj
```

Data에 대한 정보를 저장

```
alphabet
a    3.2
b    2.0
c    3.0
d    4.0
e    5.0
Name: number, dtype: float32
```

Series

```
dict_data_1 = {"a":1, "b":2, "c":3, "d":4, "e":5}
indexes = ["a","b","c","d","e","f","g","h"]
series_obj_1 = Series(dict_data_1, index=indexes)
series_obj_1
```

index 값을 기준으로 series 생성

```
a    1.0
b    2.0
c    3.0
d    4.0
e    5.0
f    NaN
g    NaN
h    NaN
dtype: float64
```

Pandas의 구성

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	weight_0
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	1
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	1

Series

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

DataFrame

In [1]:
```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

column_name : data

In [2]:
```
# Example from - https://chrisalbon.com/python/pandas_map_values_to_values.html
raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze'],
        'age': [42, 52, 36, 24, 73],
        'city': ['San Francisco', 'Baltimore', 'Miami', 'Douglas', 'Boston']}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'city'])
df
```

Out[2]:

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

DataFrame

```
In [3]: DataFrame(rav_data, columns = ["age", 'city"])
```

Out[3]:

column 선택

	age	city
0	42	San Francisco
1	52	Baltimore
2	36	Miami
3	24	Douglas
4	73	Boston

```
In [4]: DataFrame(rav_data, columns = ["first_name","last_name","age", 'city", "debt"])
```

Out[4]:

새로운 column 추가

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Cooze	73	Boston	NaN

DataFrame

```
df = DataFrame(raw_data, columns = ["first_name","last_name","age", "city", "debt"])
df.first_name
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

column 선택 – series 추출

```
df["first_name"]
```

```
0    Jason
1    Molly
2     Tina
3     Jake
4      Amy
Name: first_name, dtype: object
```

column 선택 – series 추출

DataFrame

```
df
```

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	NaN
1	Molly	Jacobson	52	Baltimore	NaN
2	Tina	Ali	36	Miami	NaN
3	Jake	Milner	24	Douglas	NaN
4	Amy	Cooze	73	Boston	NaN

loc – index location

```
df.loc[1]
```

```
first_name        Molly
last_name      Jacobson
age                  52
city           Baltimore
debt                 NaN
Name: 1, dtype: object
```

iloc – index position

```
df["age"].iloc[1:]
```

```
1    52
2    36
3    24
4    73
Name: age, dtype: int64
```

DataFrame

Column에 새로운 데이터 할당

```
df.debt = df.age > 40
df
```

	first_name	last_name	age	city	debt
0	Jason	Miller	42	San Francisco	True
1	Molly	Jacobson	52	Baltimore	True
2	Tina	Ali	36	Miami	False
3	Jake	Milner	24	Douglas	False
4	Amy	Cooze	73	Boston	True

```
df.T
```

DataFrame

In [13]:
```
df.T
```

Out[13]:

	0	1	2	3	4
first_name	Jason	Molly	Tina	Jake	Amy
last_name	Miller	Jacobson	Ali	Milner	Cooze
age	42	52	36	24	73
city	San Francisco	Baltimore	Miami	Douglas	Boston
debt	True	True	False	False	True

transpose

In [14]:
```
df.values
```

Out[14]:
```
array([['Jason', 'Miller', 42, 'San Francisco', True],
       ['Molly', 'Jacobson', 52, 'Baltimore', True],
       ['Tina', 'Ali', 36, 'Miami', False],
       ['Jake', 'Milner', 24, 'Douglas', False],
       ['Amy', 'Cooze', 73, 'Boston', True]], dtype=object)
```

값 출력

In [20]:
```
df.to_csv()
```

Out[20]:
```
',first_name,last_name,age,city,debt\n0,Jason,Miller,42,San Francisco,True\n1,Molly,Jacobson,52,Baltimore,True\n2,Tina,Ali,36,Miami,False\n
3,Jake,Milner,24,Douglas,False\n4,Amy,Cooze,73,Boston,True\n'
```

csv 변환

DataFrame

Column을 삭제함

```
del df["debt"]
```

```
df
```

	first_name	last_name	age	city
0	Jason	Miller	42	San Francisco
1	Molly	Jacobson	52	Baltimore
2	Tina	Ali	36	Miami
3	Jake	Milner	24	Douglas
4	Amy	Cooze	73	Boston

DataFrame

```
# Example from Python for data analyis

pop = {'Nevada': {2001: 2.4, 2002: 2.9},
 'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
DataFrame(pop)
```

Nested dict에서는

Column 값

Index 값

	Nevada	Ohio
2000	NaN	1.5
2001	2.4	1.7
2002	2.9	3.6

Selection with column names

```
df["account"].head(3)
```

한개의 column 선택시

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

1개 이상의 column 선택

```
df[["account", "street", "state"]].head(3)
```

	account	street	state
0	211829	34456 Sean Highway	Texas
1	320563	1311 Alvis Tunnel	NorthCarolina
2	648336	62184 Schamberger Underpass Apt. 231	Iowa

Series selection

```
account_serires = df["account"]
account_serires[:3]
```

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

```
account_serires[[0,1,2]]
```

```
0    211829
1    320563
2    648336
Name: account, dtype: int64
```

1개 이상의 index

```
account_serires[account_serires<250000]
```

```
0     211829
3     109996
4     121213
5     132971
6     145068
7     205217
8     209744
9     212303
10    214098
11    231907
12    242368
Name: account, dtype: int64
```

Boolean index

Basic, loc, iloc selection

Column 과 index number

```
df[["name","street"]][:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

Column number와 index number

```
df.iloc[:2,:2]
```

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

```
df.loc[[211829,320563],["name","street"]]
```

Column 과 index name

	name	street
account		
211829	Kerluke, Koepp and Hilpert	34456 Sean Highway
320563	Walter-Trantow	1311 Alvis Tunnel

Data drop

```
df.drop(1)
```
Index number로 drop

	name	street	city	s
0	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	T
2	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	lc

Series operation

```
s1 = Series(
    range(1,6), index=list("abced"))
s1
```

```
a    1
b    2
c    3
e    4
d    5
dtype: int64
```

```
s2 = Series(
    range(5,11), index=list("bcedef"))
s2
```

```
b     5
c     6
e     7
d     8
e     9
f    10
dtype: int64
```

```
s1.add(s2)
```

```
a      NaN
b      7.0
c      9.0
d     13.0
e     11.0
e     13.0
f      NaN
dtype: float64
```

```
s1 + s2
```

```
a      NaN
b      7.0
c      9.0
d     13.0
e     11.0
e     13.0
f      NaN
dtype: float64
```

index 으로 기준으로 연산수행

겹치는 index가 없을 경우 NaN값으로 반환

Dataframe operation

```
df1 = DataFrame(
    np.arange(9).reshape(3,3),
    columns=list("abc"))
df1
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

```
df2 = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df2
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
df1 + df2
```

	a	b	c	d
0	0.0	2.0	4.0	NaN
1	7.0	9.0	11.0	NaN
2	14.0	16.0	18.0	NaN
3	NaN	NaN	NaN	NaN

```
df1.add(df2,fill_value=0)
```

	a	b	c	d
0	0.0	2.0	4.0	3.0
1	7.0	9.0	11.0	7.0
2	14.0	16.0	18.0	11.0
3	12.0	13.0	14.0	15.0

df는 column과 index를 모두 고려

add operation을 쓰면 NaN값 0으로 변환
Operation types: add, sub, div, mul

Series + Dataframe

```
df = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

+

10 11 12 13
10 11 12 13
10 11 12 13
10 11 12 13

```
s = Series(
    np.arange(10,14),
    index=list("abcd"))
s
```

```
a    10
b    11
c    12
d    13
dtype: int64
```

```
df + s
```

	a	b	c	d
0	10	12	14	16
1	14	16	18	20
2	18	20	22	24
3	22	24	26	28

column을 기준으로
broadcasting이 발생함

Series + Dataframe

```
df = DataFrame(
    np.arange(16).reshape(4,4),
    columns=list("abcd"))
df
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

```
s2 = Series(np.arange(10,14))
s2
```

```
0    10
1    11
2    12
3    13
dtype: int64
```

```
df + s2
```

	a	b	c	d	0	1	2	3
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
df.add(s2, axis=0)
```

	a	b	c	d
0	10	11	12	13
1	15	16	17	18
2	20	21	22	23
3	25	26	27	28

axis를 기준으로 row broadcasting 실행

apply for dataframe

- 내장 연산 함수를 사용할 때도 똑같은 효과를 거둘 수 있음
- mean, std 등 사용가능

```
df_info.sum()
```

```
earn       4.474344e+07
height     9.183125e+04
age        6.250800e+04
dtype: float64
```

```
df_info.apply(sum)
```

```
earn       4.474344e+07
height     9.183125e+04
age        6.250800e+04
dtype: float64
```

apply for dataframe

- scalar 값 이외에 series값의 반환도 가능함

```
def f(x):
    return Series([x.min(), x.max()], index=["min", "max"])
df_info.apply(f)
```

	earn	height	age
min	-98.580489	57.34	22
max	317949.127955	77.21	95

applymap for dataframe

- series 단위가 아닌 element 단위로 함수를 적용함
- series 단위에 apply를 적용시킬 때와 같은 효과

```
f = lambda x : -x
df_info.applymap(f).head(5)
```

	earn	height	age
0	-79571.299011	-73.89	-49
1	-96396.988643	-66.23	-62
2	-48710.666947	-63.77	-33
3	-80478.096153	-63.22	-95
4	-82089.345498	-63.08	-43

```
f = lambda x : -x
df_info["earn"].apply(f).head(5)
```

```
0   -79571.299011
1   -96396.988643
2   -48710.666947
3   -80478.096153
4   -82089.345498
Name: earn, dtype: float64
```

describe

- Numeric type 데이터의 요약 정보를 보여줌

```
df = pd.read_csv("wages.csv")
df.head()
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	male	white	16	49
1	96396.988643	66.23	female	white	16	62
2	48710.666947	63.77	female	white	16	33
3	80478.096153	63.22	female	other	16	95
4	82089.345498	63.08	female	white	17	43

```
df.describe()
```

	earn	height	ed	age
count	1379.000000	1379.000000	1379.000000	1379.000000
mean	32446.292622	66.592640	13.354605	45.328499
std	31257.070006	3.818108	2.438741	15.789715
min	-98.580489	57.340000	3.000000	22.000000
25%	10538.790721	63.720000	12.000000	33.000000
50%	26877.870178	66.050000	13.000000	42.000000
75%	44506.215336	69.315000	15.000000	55.000000
max	317949.127955	77.210000	18.000000	95.000000

unique

- series data의 유일한 값을 list를 반환함

```
df.race.unique()
```

유일한 인종의 값 list

```
array(['white', 'other', 'hispanic', 'black'], dtype=object)
```

```
np.array(dict(enumerate(df["race"].unique())))
```

dict type으로 index

```
array({0: 'white', 1: 'other', 2: 'hispanic', 3: 'black'}, dtype=object)
```

```
value = list(map(int, np.array(list(enumerate(df["race"].unique())))[:, 0].tolist()))
key = np.array(list(enumerate(df["race"].unique())), dtype=str)[:, 1].tolist()

value, key
```

label index 값과 label 값 각각 추출

```
([0, 1, 2, 3], ['white', 'other', 'hispanic', 'black'])
```

unique

label str → index 값으로 변환

```
df["race"].replace(to_replace=key, value=value, inplace=True)
```

```
value = list(map(int, np.array(list(enumerate(df["sex"].unique())))[:, 0].tolist()))
key = np.array(list(enumerate(df["sex"].unique())), dtype=str)[:, 1].tolist()

value, key
```

성별에 대해서도 동일하게 적용

```
([0, 1], ['male', 'female'])
```

```
df["sex"].replace(to_replace=key, value=value, inplace=True)
df.head(5)
```

	earn	height	sex	race	ed	age
0	79571.299011	73.89	0	0	16	49
1	96396.988643	66.23	1	0	16	62

"sex"와 "race" column의 index labelling

sum

- 기본적인 column 또는 row 값의 연산을 지원
- sub, mean, min, max, count, median, mad, var 등

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

column 별

```
earn      4.474344e+07
height    9.183125e+04
sex       8.590000e+02
race      5.610000e+02
ed        1.841600e+04
age       6.250800e+04
dtype: float64
```

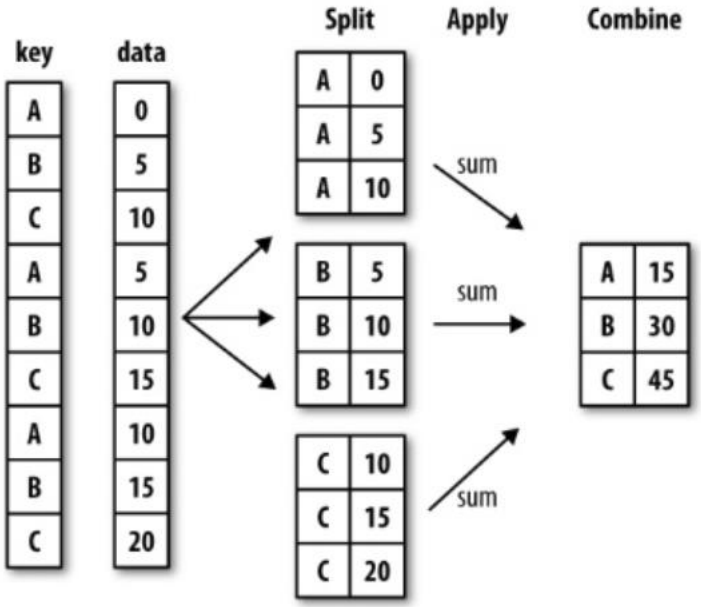
```
df.sum(axis=1)
```

row 별

```
0        79710.189011
1        96542.218643
2        48824.436947
3        80654.316153
4        82213.425498
5        15423.882901
6        47231.711821
```

Groupby

- SQL groupby 명령어와 같음
- split → apply → combine
- 과정을 거쳐 연산함



Groupby

적용받는 연산

```
df.groupby("Team")["Points"].sum()
```

묶음의 기준이 되는 컬럼

적용받는 컬럼

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014

```
Team
Devils     1536
Kings      2285
Riders     3049
Royals     1505
kings       812
Name: Points, dtype: int64
```

결과
TEAM을 기준으로
Points을 Sum

Groupby

- 한 개이상의 column을 묶을 수 있음

```
df.groupby(["Team", "Year"])["Points"].sum()
```

```
Team     Year
Devils   2014    863
         2015    673
Kings    2014    741
         2016    756
         2017    788
Riders   2014    876
         2015    789
         2016    694
```

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
2	863	2	Devils	2014
3	673	3	Devils	2015
4	741	3	Kings	2014

Hierarchical index

- Groupby 명령의 결과물도 결국은 dataframe
- 두 개의 column으로 groupby를 할 경우, index가 두개 생성

```
h_index.index
```

```
MultiIndex(levels=[['Devils', 'Kings', 'Riders', 'Royals', 'kings'], [2014, 2015, 2016, 201
7]],
           labels=[[0, 0, 1, 1, 1, 2, 2, 2, 2, 3, 3, 4], [0, 1, 0, 2, 3, 0, 1, 2, 3, 0, 1,
 1]],
           names=['Team', 'Year'])
```

```
h_index["Devils":"Kings"]
```

```
Team    Year
Devils  2014    863
        2015    673
Kings   2014    741
        2016    756
        2017    788
Name: Points, dtype: int64
```

Hierarchical index – unstack()

- Group으로 묶여진 데이터를 matrix 형태로 전환해줌

```
Team     Year
Devils   2014     863
         2015     673
Kings    2014     741
         2016     756
         2017     788
Riders   2014     876
         2015     789
         2016     694
         2017     690
Royals   2014     701
         2015     804
kings    2015     812
```

```
h_index.unstack()
```

Year	2014	2015	2016	2017
Team				
Devils	863.0	673.0	NaN	NaN
Kings	741.0	NaN	756.0	788.0
Riders	876.0	789.0	694.0	690.0
Royals	701.0	804.0	NaN	NaN
kings	NaN	812.0	NaN	NaN

Hierarchical index – operations

- Index level을 기준으로 기본 연산 수행 가능

```
h_index.sum(level=0)
```

```
Team
Devils     1536
Kings      2285
Riders     3049
Royals     1505
kings       812
Name: Points, dtype: int64
```

```
h_index.sum(level=1)
```

```
Year
2014     3181
2015     3078
2016     1450
2017     1478
Name: Points, dtype: int64
```

Groupby – gropued

- 특정 key값을 가진 그룹의 정보만 추출 가능

```
grouped.get_group("Devils")
```

	Points	Rank	Team	Year
2	863	2	Devils	2014
3	673	3	Devils	2015

Groupby – gropued

- 추출된 group 정보에는 세 가지 유형의 apply가 가능함
- Aggregation: 요약된 통계정보를 추출해 줌
- Transformation: 해당 정보를 변환해줌
- Filtration: 특정 정보를 제거 하여 보여주는 필터링 기능

Groupby – aggregation

```
grouped.agg(sum)
```

	Points	Rank	Year
Team			
Devils	1536	5	4029
Kings	2285	5	6047
Riders	3049	7	8062
Royals	1505	5	4029
kings	812	4	2015

```
import numpy as np
grouped.agg(np.mean)
```

	Points	Rank	Year
Team			
Devils	768.000000	2.500000	2014.500000
Kings	761.666667	1.666667	2015.666667
Riders	762.250000	1.750000	2015.500000
Royals	752.500000	2.500000	2014.500000
kings	812.000000	4.000000	2015.000000

Groupby – aggregation

```
grouped['Points'].agg([np.sum, np.mean, np.std])
```

	sum	mean	std
Team			
Devils	1536	768.000000	134.350288
Kings	2285	761.666667	24.006943
Riders	3049	762.250000	88.567771
Royals	1505	752.500000	72.831998
kings	812	812.000000	NaN

특정 컬럼에
여러개의 function을
Apply 할 수 도 있음

Data

- 시간과 데이터 종류가 정리된 통화량 데이터

```
import dateutil

df_phone = pd.read_csv("phone_data.csv")
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)
df_phone.head()
```

	index	date	duration	item	month	network	network_type
0	0	2014-10-15 06:58:00	34.429	data	2014-11	data	data
1	1	2014-10-15 06:58:00	13.000	call	2014-11	Vodafone	mobile
2	2	2014-10-15 14:46:00	23.000	call	2014-11	Meteor	mobile
3	3	2014-10-15 14:48:00	4.000	call	2014-11	Tesco	mobile
4	4	2014-10-15 17:27:00	4.000	call	2014-11	Tesco	mobile

https://www.shanelynn.ie/wp-content/uploads/2015/06/phone_data.csv

```
df_phone.groupby('month')['duration'].sum()
```

```
month
2014-11    26639.441
2014-12    14641.870
2015-01    18223.299
2015-02    15522.299
2015-03    22750.441
Name: duration, dtype: float64
```

```
df_phone[df_phone['item'] == 'call'].groupby('network')['duration'].sum()
```

```
network
Meteor        7200.0
Tesco        13828.0
Three        36464.0
Vodafone     14621.0
landline     18433.0
voicemail     1775.0
Name: duration, dtype: float64
```

```python
df_phone.groupby(['month', 'item'])['date'].count()
```

```
month    item
2014-11  call    107
         data     29
         sms      94
2014-12  call     79
         data     30
         sms      48
2015-01  call     88
         data     31
         sms      86
2015-02  call     67
         data     31
         sms      39
2015-03  call     47
         data     29
         sms      25
Name: date, dtype: int64
```

```
df_phone.groupby(['month', 'item'])['date'].count().unstack()
```

item	call	data	sms
month			
2014-11	107	29	94
2014-12	79	30	48
2015-01	88	31	86
2015-02	67	31	39
2015-03	47	29	25

```
df_phone.groupby('month', as_index=False).agg({"duration": "sum"})
```

	month	duration
0	2014-11	26639.441
1	2014-12	14641.870
2	2015-01	18223.299
3	2015-02	15522.299
4	2015-03	22750.441

```
df_phone.groupby(['month', 'item']).agg({'duration':sum,
                                         'network_type': "count",
                                         'date': 'first'})    # g
```

		network_type	date	duration
month	**item**			
2014-11	**call**	107	2014-10-15 06:58:00	25547.000
	data	29	2014-10-15 06:58:00	998.441
	sms	94	2014-10-16 22:18:00	94.000
2014-12	**call**	79	2014-11-14 17:24:00	13561.000
	data	30	2014-11-13 06:58:00	1032.870
	sms	48	2014-11-14 17:28:00	48.000
2015-01	**call**	88	2014-12-15 20:03:00	17070.000
	data	31	2014-12-13 06:58:00	1067.299

```
df_phone.groupby(['month', 'item']).agg({'duration': [min, max, sum],      # find the min
                                         'network_type': "count", # find the number of networ
                                         'date': [min, 'first', 'nunique']})    # get the min
```

		network_type	date			duration		
		count	min	first	nunique	min	max	sum
month	item							
2014-11	call	107	2014-10-15 06:58:00	2014-10-15 06:58:00	104	1.000	1940.000	25547.000
	data	29	2014-10-15 06:58:00	2014-10-15 06:58:00	29	34.429	34.429	998.441
	sms	94	2014-10-16 22:18:00	2014-10-16 22:18:00	79	1.000	1.000	94.000
2014-12	call	79	2014-11-14 17:24:00	2014-11-14 17:24:00	76	2.000	2120.000	13561.000
	data	30	2014-11-13 06:58:00	2014-11-13 06:58:00	30	34.429	34.429	1032.870
	sms	48	2014-11-14 17:28:00	2014-11-14 17:28:00	41	1.000	1.000	48.000

```
grouped = df_phone.groupby('month').agg( {"duration" : [min, max, np.mean]})

grouped.columns = grouped.columns.droplevel(level=0)
grouped.rename(columns={"min": "min_duration", "max": "max_duration", "mean": "mean_duration"})
```

	min_duration	max_duration	mean_duration
month			
2014-11	1.0	1940.0	115.823657
2014-12	1.0	2120.0	93.260318
2015-01	1.0	1859.0	88.894141
2015-02	1.0	1863.0	113.301453
2015-03	1.0	10528.0	225.251891

Pivot Table

- 우리가 Excel에서 보던 그 것!
- Index 축은 groupby와 동일함
- Column에 추가로 labelling 값을 추가하여,
- Value에 numeric type 값을 aggregation 하는 형태

Pivot Table

```
df_phone = pd.read_csv("phone_data.csv")
df_phone['date'] = df_phone['date'].apply(dateutil.parser.parse, dayfirst=True)
df_phone.head()
```

	index	date	duration	item	month	network	network_type
0	0	2014-10-15 06:58:00	34.429	data	2014-11	data	data
1	1	2014-10-15 06:58:00	13.000	call	2014-11	Vodafone	mobile
2	2	2014-10-15 14:46:00	23.000	call	2014-11	Meteor	mobile
3	3	2014-10-15 14:48:00	4.000	call	2014-11	Tesco	mobile
4	4	2014-10-15 17:27:00	4.000	call	2014-11	Tesco	mobile

값 가로축 세로축

```
df_phone.pivot_table(["duration"],
                     index=[df_phone.month,df_phone.item],
                     columns=df_phone.network, aggfunc="sum", fill_value=0)
```

		duration								
	network	Meteor	Tesco	Three	Vodafone	data	landline	special	voicemail	world
month	item									
2014-11	call	1521	4045	12458	4316	0.000	2906	0	301	0
	data	0	0	0	0	998.441	0	0	0	0
	sms	10	3	25	55	0.000	0	1	0	0
2014-12	call	2010	1819	6316	1302	0.000	1424	0	690	0
	data	0	0	0	0	1032.870	0	0	0	0
	sms	12	1	13	18	0.000	0	0	0	4

Crosstab

- 특허 두 칼럼에 교차 빈도, 비율, 덧셈 등을 구할 때 사용
- Pivot table의 특수한 형태
- User-Item Rating Matrix 등을 만들 때 사용가능함

Crosstab

```
df_movie = pd.read_csv("./movie_rating.csv")
df_movie.head()
```

	critic 세로축	title 가로축	rating 값
0	Jack Matthews	Lady in the Water	3.0
1	Jack Matthews	Snakes on a Plane	4.0
2	Jack Matthews	You Me and Dupree	3.5
3	Jack Matthews	Superman Returns	5.0
4	Jack Matthews	The Night Listener	3.0

```
pd.crosstab(index=df_movie.critic,columns=df_movie.title,values=df_movie.rating,
            aggfunc="first").fillna(0)
```

title	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

```
df_movie.pivot_table(["rating"], index=df_movie.critic, columns=df_movie.title,
                     aggfunc="sum", fill_value=0)
```

	rating					
title	**Just My Luck**	**Lady in the Water**	**Snakes on a Plane**	**Superman Returns**	**The Night Listener**	**You Me and Dupree**
critic						
Claudia Puig	3.0	0.0	3.5	4.0	4.5	2.5
Gene Seymour	1.5	3.0	3.5	5.0	3.0	3.5
Jack Matthews	0.0	3.0	4.0	5.0	3.0	3.5
Lisa Rose	3.0	2.5	3.5	3.5	3.0	2.5
Mick LaSalle	2.0	3.0	4.0	3.0	3.0	2.0
Toby	0.0	0.0	4.5	4.0	0.0	1.0

Merge

- SQL에서 많이 사용하는 Merge와 같은 기능
- 두 개의 데이터를 하나로 합침

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Merge

subject_id 기준으로 merge

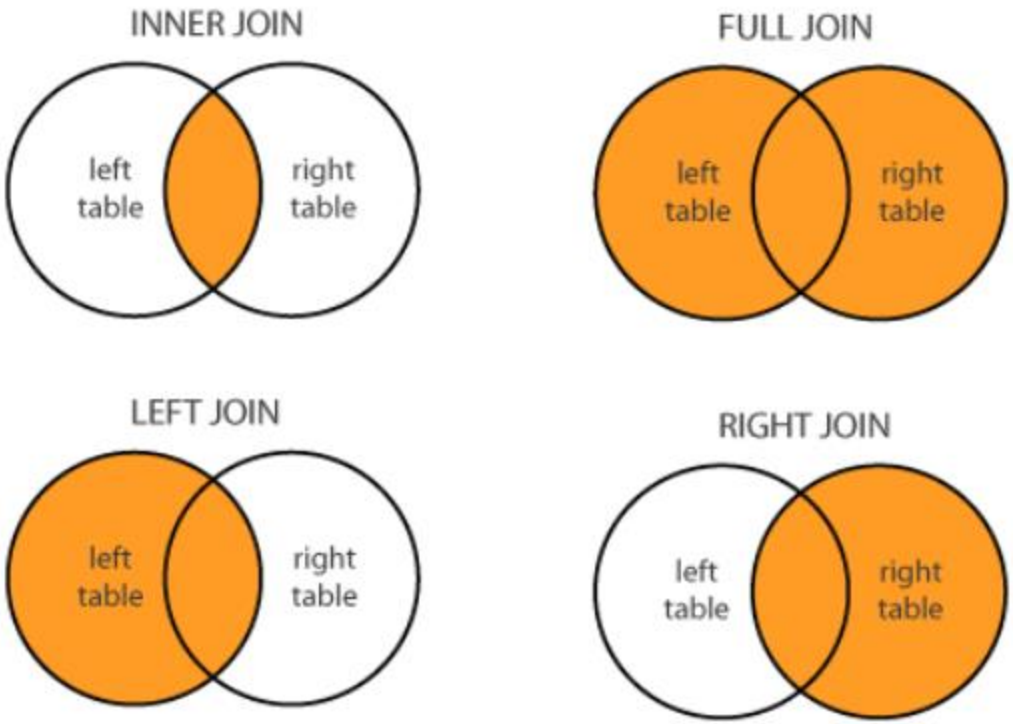
pd.merge(df_a, df_b, on='subject_id')

	subject_id	test_score
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14

	subject_id	test_id	first_name	last_nam
0	4	61	Billy	Bond
1	5	16	Brian	Blac
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Join method


INNER JOIN
left table
right table
FULL JOIN
left table
right table
LEFT JOIN
left table
right table
RIGHT JOIN
left table
right table

Data

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Left join

```
pd.merge(df_a, df_b, on='subject_id', how='left')
```

click to scroll output; double click to hide

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black

Right join

```
pd.merge(df_a, df_b, on='subject_id', how='right')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black
2	6	NaN	NaN	Bran	Balwner
3	7	NaN	NaN	Bryce	Brice
4	8	NaN	NaN	Betty	Btisan

Full(outer) join

```
pd.merge(df_a, df_b, on='subject_id', how='outer')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

Inner join

```
pd.merge(df_a, df_b, on='subject_id', how='inner')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

Index based join

```
pd.merge(df_a, df_b, right_index=True, left_index=True)
```

	subject_id_x	first_name_x	last_name_x	subject_id_y	first_name_y	last_name_y
0	1	Alex	Anderson	4	Billy	Bonder
1	2	Amy	Ackerman	5	Brian	Black
2	3	Allen	Ali	6	Bran	Balwner
3	4	Alice	Aoni	7	Bryce	Brice
4	5	Ayoung	Atiches	8	Betty	Btisan