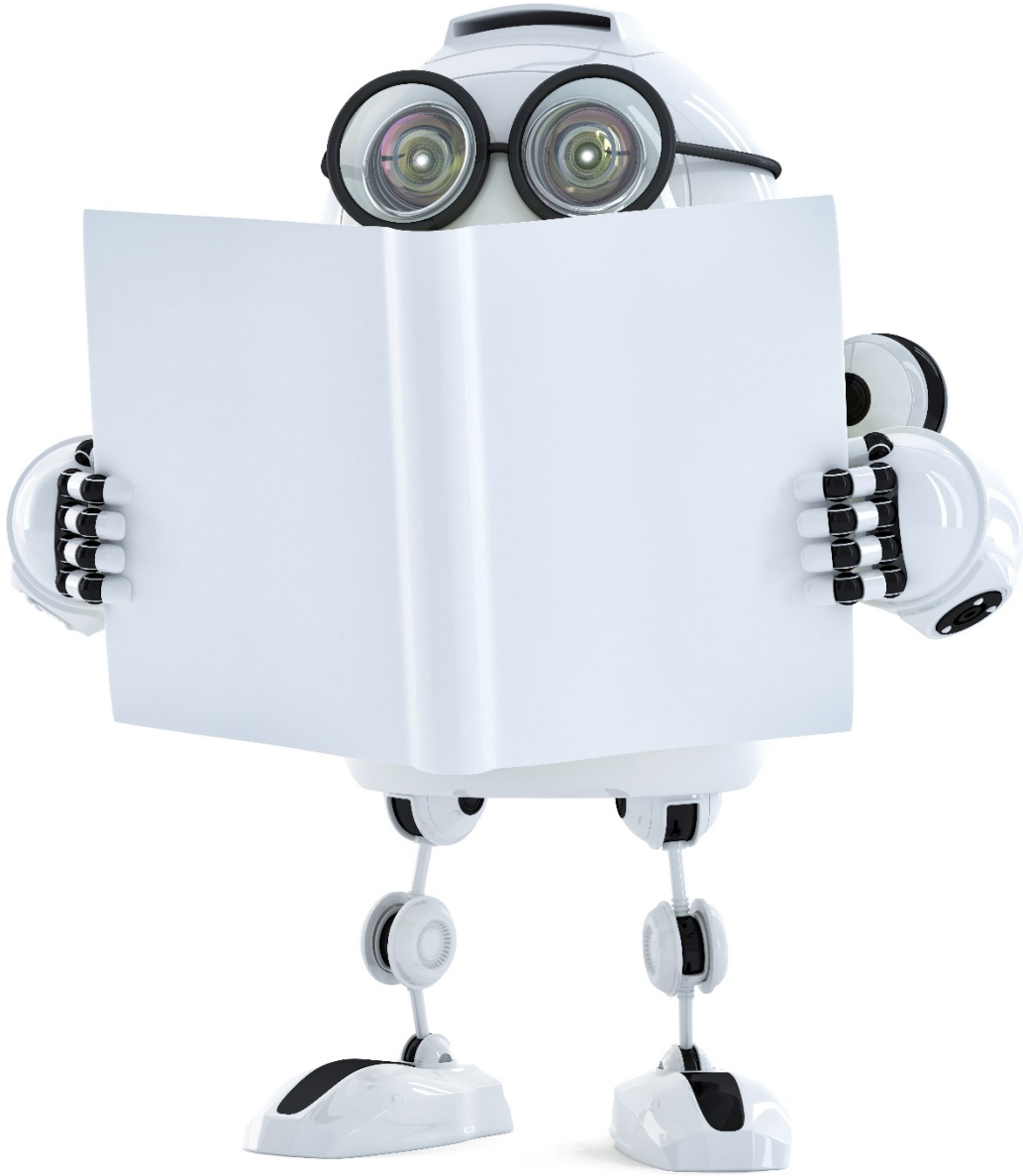
Pandas #2

Data handling

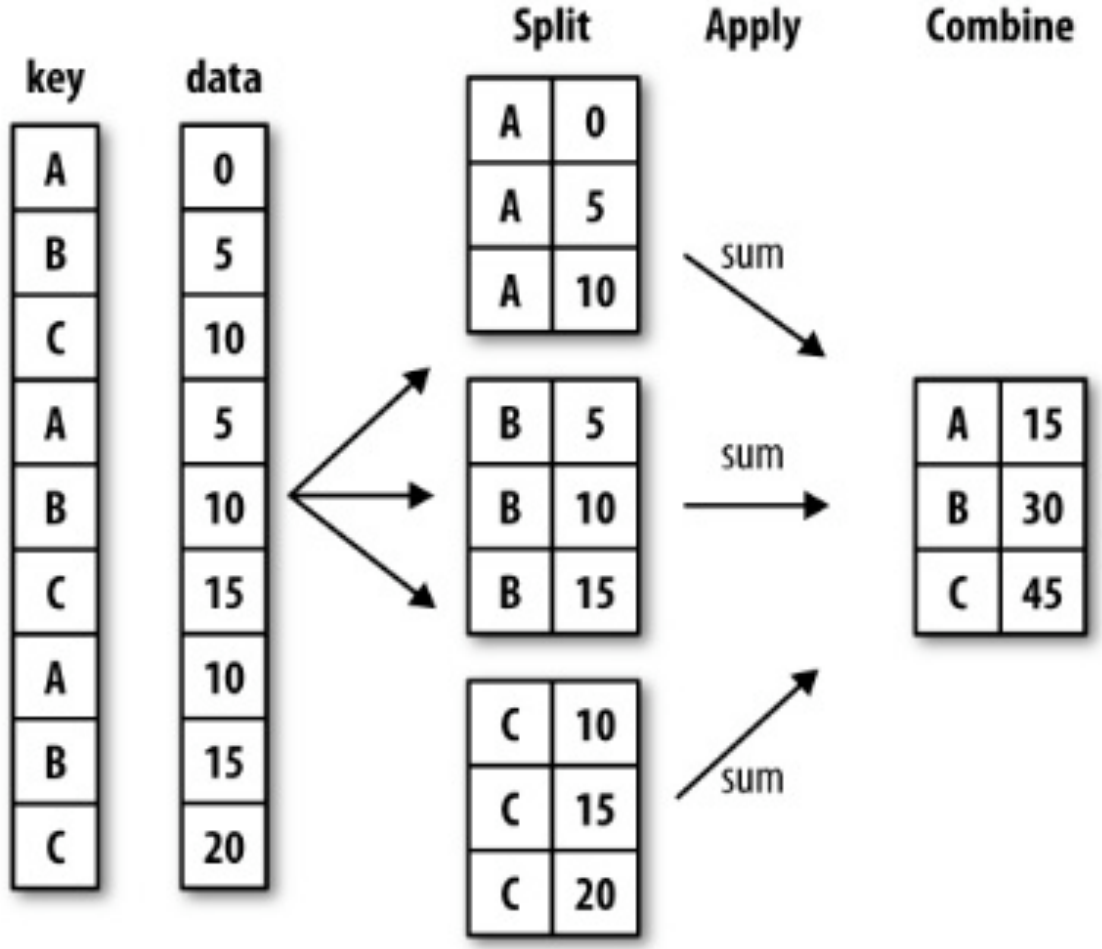


Director of TEAMLAB
Sungchul Choi

Groupby I

Groupby

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



https://goo.gl/EGuoiA

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'])
```

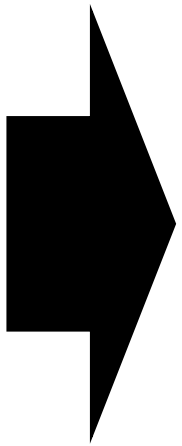
```
h_index["Devils":"Kings"]
```

```
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 – swaplevel

- Index level을 변경할 수 있음

```
h_index.swaplevel()
```

```
Year  Team
2014  Devils    863
2015  Devils    673
2014  Kings     741
2016  Kings     756
2017  Kings     788
2014  Riders    876
2015  Riders    789
2016  Riders    694
2017  Riders    690
2014  Royals    701
2015  Royals    804
      kings     812
Name: Points, dtype: int64
```

```
h_index.swaplevel().sortlevel(0)
```

```
Year  Team
2014  Devils    863
      Kings     741
      Riders    876
      Royals    701
2015  Devils    673
      Riders    789
      Royals    804
      kings     812
2016  Kings     756
      Riders    694
2017  Kings     788
      Riders    690
Name: Points, dtype: int64
```

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 II

Groupby – gropued

- Groupby에 의해 Split된 상태를 추출 가능함

```
grouped = df.groupby("Team")

for name,group in grouped:
    print (name)
    print (group)
```

Tuple 형태로 그룹의 key 값
Value값이 추출됨

```
Devils
    Points  Rank     Team  Year
2      863     2  Devils  2014
3      673     3  Devils  2015
Kings
    Points  Rank    Team  Year
4      741     3  Kings  2014
6      756     1  Kings  2016
7      788     1  Kings  2017
```

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

Groupby – transformation

- Aggregation과 달리 key값 별로 요약된 정보가 아님
- 개별 데이터의 변환을 지원함

```
df
```

	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
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x)
grouped.transform(score)
```

	Points	Rank	Year
0	876	1	2014
1	789	2	2015
2	863	2	2014
3	673	3	2015
4	741	3	2014
5	812	4	2015
6	756	1	2016
7	788	1	2017
8	694	2	2016
9	701	4	2014

```
df
```

	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
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x.max())
grouped.transform(score)
```

	Points	Rank	Year
0	876	2	2017
1	876	2	2017
2	863	3	2015
3	863	3	2015
4	788	3	2017
5	812	4	2015
6	788	3	2017
7	788	3	2017
8	876	2	2017
9	804	4	2015

단 max나 min 처럼 Series 데이터에 적용되는 데이터 들은 Key값을 기준으로 Grouped된 데이터 기준

```
df
```

	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
5	812	4	kings	2015
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
9	701	4	Royals	2014

```
score = lambda x: (x - x.mean()) / x.std()
grouped.transform(score)
```

	Points	Rank	Year
0	1.284327	-1.500000	-1.161895
1	0.302029	0.500000	-0.387298
2	0.707107	-0.707107	-0.707107
3	-0.707107	0.707107	0.707107
4	-0.860862	1.154701	-1.091089
5	NaN	NaN	NaN
6	-0.236043	-0.577350	0.218218
7	1.096905	-0.577350	0.872872
8	-0.770596	0.500000	0.387298
9	-0.707107	0.707107	-0.707107

$$z_i = \frac{x_i - \mu}{\sigma}$$

Groupby – filter

- 특정 조건으로 데이터를 검색할 때 사용

```
df.groupby('Team').filter(lambda x: len(x) >= 3)
```

	Points	Rank	Team	Year
0	876	1	Riders	2014
1	789	2	Riders	2015
4	741	3	Kings	2014
6	756	1	Kings	2016
7	788	1	Kings	2017
8	694	2	Riders	2016
11	690	2	Riders	2017

- filter안에는 boolean 조건이 존재해야함
- len(x)는 grouped된 dataframe 개수

```
df.groupby('Team').filter(lambda x: x["Rank"].sum() > 2)
```

```
df.groupby('Team').filter(lambda x: x["Points"].sum() > 1000)
```

```
df.groupby('Team').filter(lambda x: x["Rank"].mean() > 1)
```

Case study

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
```

```
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
Crosstab

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
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

값

```
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 & Concat

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	Bonde
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

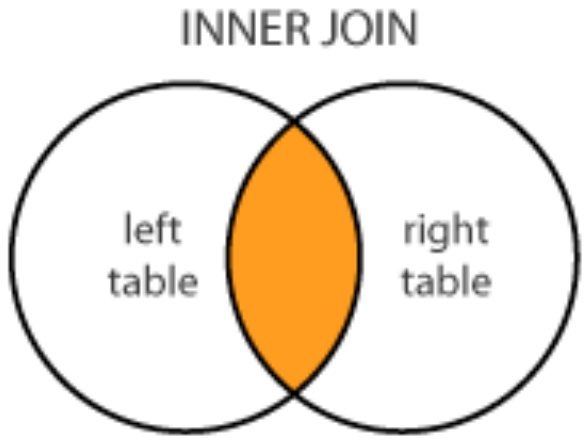
Merge

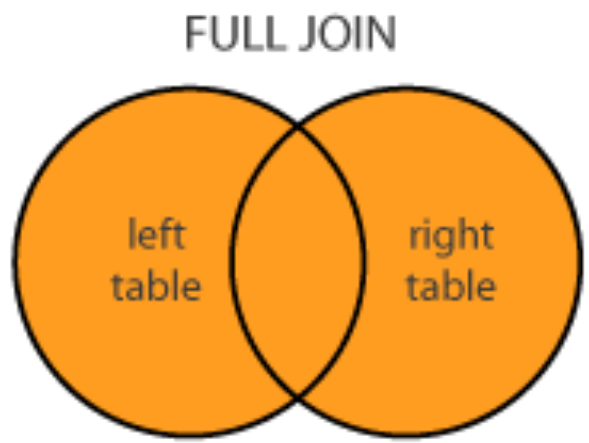
두 dataframe이 column이름이 다를 때

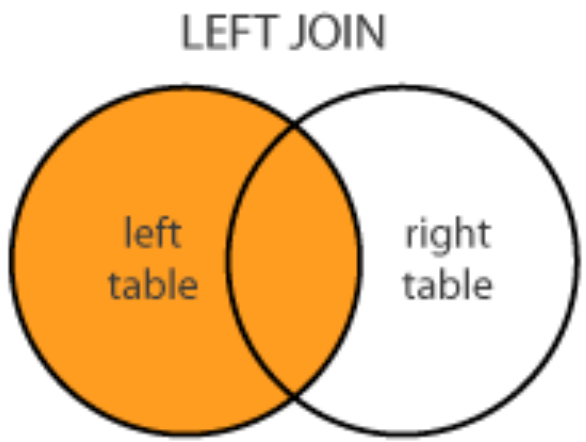
```
pd.merge(df_a, df_b, left_on='subject_id', right_on='subject_id')
```

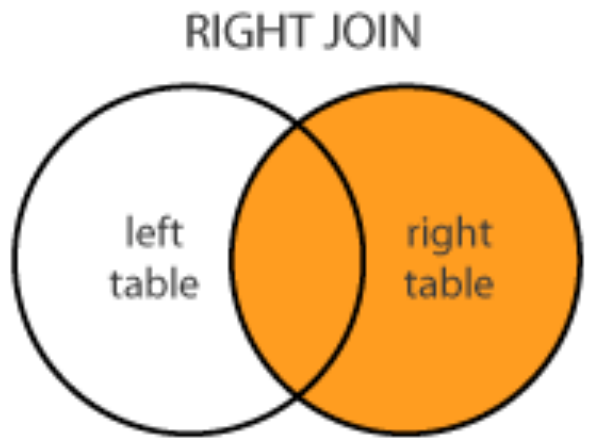
	subject_id	test_score	first_name	last_name
0	4	61	Billy	Bonder
1	5	16	Brian	Black
2	7	14	Bryce	Brice
3	8	15	Betty	Btisan

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

Concat

- 같은 형태의 데이터를 붙이는 연산작업

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

https://pandas.pydata.org/pandas-docs/stable/merging.html

Concat

```
df_new = pd.concat([df_a, df_b])
df_new.reset_index()
```

	index	subject_id	first_name	last_name
0	0	1	Alex	Anderson
1	1	2	Amy	Ackerman
2	2	3	Allen	Ali
3	3	4	Alice	Aoni
4	4	5	Ayoung	Atiches
5	0	4	Billy	Bonder
6	1	5	Brian	Black
7	2	6	Bran	Balwner

```
df_a.append(df_b)
```

	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
0	4	Billy	Bonder
1	5	Brian	Black

Concat

```
df_new = pd.concat([df_a, df_b], axis=1)
df_new.reset_index()
```

	index	subject_id	first_name	last_name	subject_id	first_name	last_name
0	0	1	Alex	Anderson	4	Billy	Bonder
1	1	2	Amy	Ackerman	5	Brian	Black
2	2	3	Allen	Ali	6	Bran	Balwner

DB
Persistence

Database connection

- Data loading시 db connection 기능을 제공함

Database 연결 코드

```
import sqlite3

conn = sqlite3.connect("./data/flights.db")
cur = conn.cursor()
cur.execute("select * from airlines limit 5;")
results = cur.fetchall()
results
```

db 연결 conn을 사용하여 dataframe 생성

```
df_airplines = pd.read_sql_query("select * from airlines;", conn)
df_airports = pd.read_sql_query("select * from airports;", conn)
df_routes = pd.read_sql_query("select * from routes;", conn)
```

XLS persistence

- Dataframe의 엑셀 추출 코드
- Xls 엔진으로 openpyxls 또는 XlsxWrite 사용

```
writer = pd.ExcelWriter('./data/df_routes.xlsx', engine='xlsxwriter')
df_routes.to_excel(writer, sheet_name='Sheet1')
```

Pickle persistence

- 가장 일반적인 python 파일 persistence
- to_pickle, read_pickle 함수 사용

```
df_routes.to_pickle("./data/df_routes.pickle")
```

```
df_routes_pickle = pd.read_pickle("./data/df_routes.pickle")
df_routes_pickle.head()
```

	index	airline	airline_id	source	source_id	dest	dest_id	codeshare	stops	equipment
0	0	2B	410	AER	2965	KZN	2990	None	0	CR2
1	1	2B	410	ASF	2966	KZN	2990	None	0	CR2
2	2	2B	410	ASF	2966	MRV	2962	None	0	CR2

Human knowledge belongs to the world.