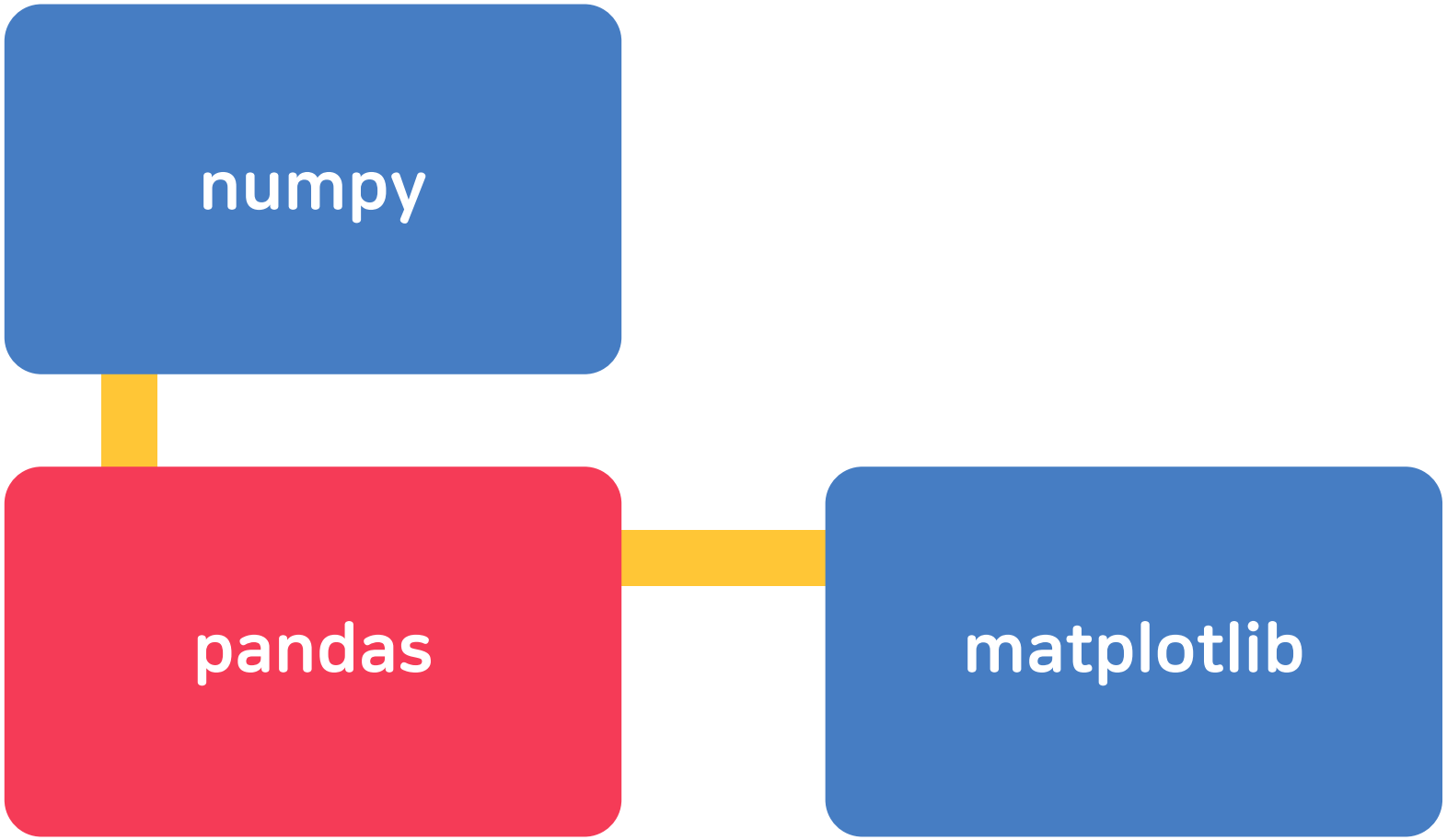
python

스마트인재개발원
Smart Human Resources Development

이 상 준 연구원

python

수업 진행방향



스마트인재개발원
Smart Human Resources Development

분석에 특화된 모듈(라이브러리)

Numpy

- 고성능 과학계산을 위한 데이터분석 라이브러리

Pandas

- 행과 열로 구성된 표 형식의 데이터를 지원하는 라이브러리

Matplotlib

- 2D 그래프로 시각화가 가능한 라이브러리

python Pandas 모듈이란?


```
score = np.array([[75,25,95],
                  [70,55,100],
                  [100,65,85],
                  [80,100,95]])
score
```

```
array([[ 75,  25,  95],
       [ 70,  55, 100],
       [100,  65,  85],
       [ 80, 100,  95]])
```

	A	B	C	D
1	이름	Java	Arduino	Python
2	상준	75	25	95
3	세연	70	55	100
4	운비	100	65	85
5	예호	80	100	95

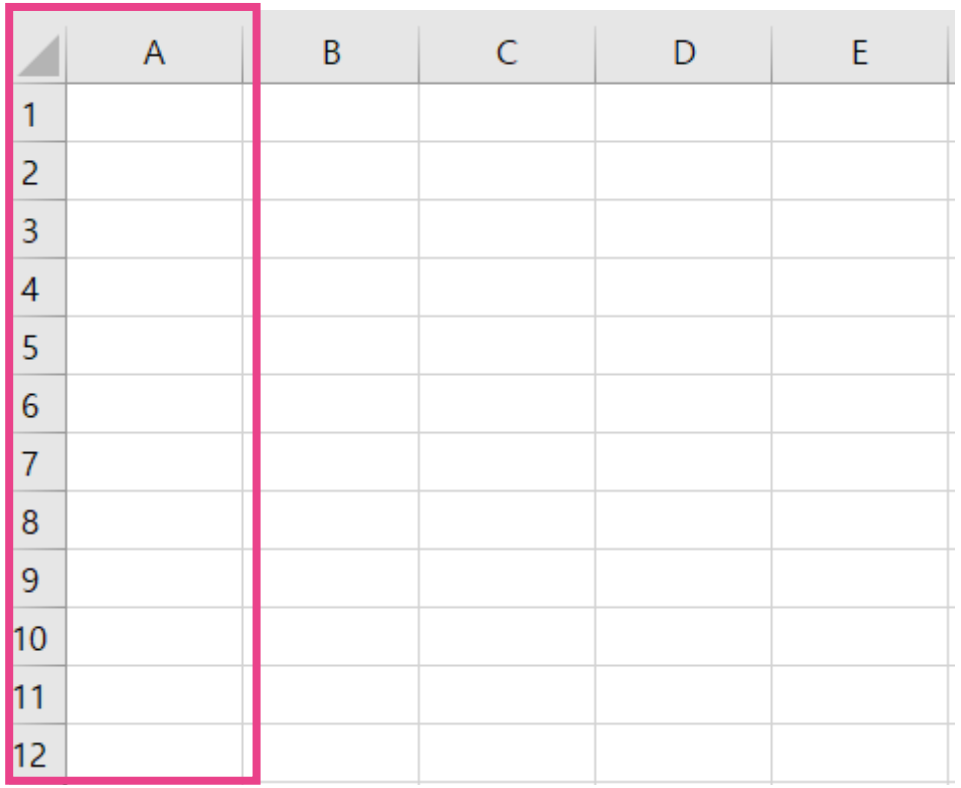
Pandas

데이터 조작 및 분석을 위한 라이브러리

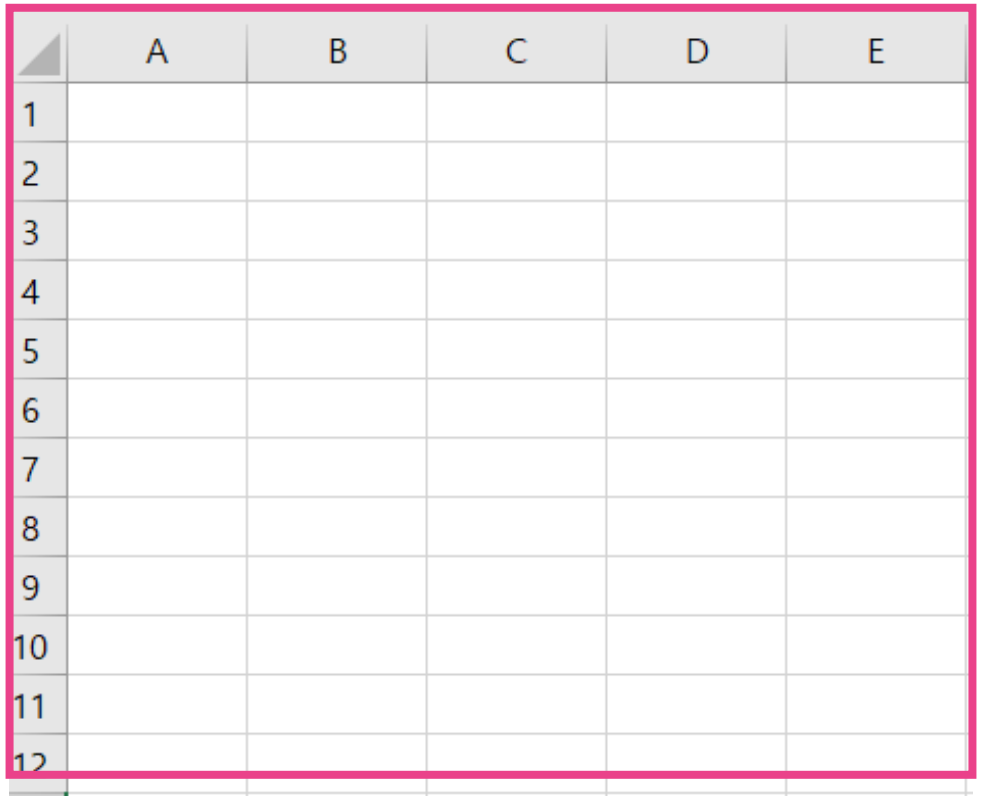
- Series Class : 1차원
  : 인덱스(index) + 값(value)

- DataFrame Class : 2차원
  : 행과 열을 가지는 표와 같은 형태
  : 서로 다른 종류의 자료형을 저장할 수 있음

- **Series Class** : 1차원
  : 인덱스(index) + 값(value)



- **DataFrame Class** : 2차원
  : 행과 열을 가지는 표와 같은 형태



index

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S
14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S

Series

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S
14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S

DataFrame

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S
14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S

Columns

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S
13	0	3	Saundercock, Mr. William Henry	male	20.0	0	0	A/5. 2151	8.0500	NaN	S
14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S

Pandas 모듈 import 하기

- import pandas as pd
  : pandas모듈(라이브러리)를 import하고 앞으로 pd라는 이름으로 부르겠다.

Series 사용

Series 생성

```
1 population = pd.Series([9904312,3448737,2890451,266052])
2 population
```

```
0    9904312
1    3448737
2    2890451
3     266052
dtype: int64
```

인덱스 지정하여 생성하기

```
1  population = pd.Series([9904312,3448737,2890451,266052],
2                         index = ['서울','부산','인천','대구'])
3  population
```

```
서울     9904312
부산     3448737
인천     2890451
대구      266052
dtype: int64
```

1. Series값 확인

```
1 population.values
```

```
array([9904312, 3448737, 2890451,  266052], dtype=int64)
```

2. Series 인덱스 확인

```
1 population.index
```

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
```

3. Series 타입 확인

```
1 population.dtype
```

```
dtype('int64')
```

Series에 이름 지정

```
1  population.name = "인구"
2  population.index.name = "도시"
3  population
```

```
도시
서울      9904312
부산      3448737
인천      2890451
대구       266052
Name: 인구, dtype: int64
```

Series 연산

```
1 population / 1000000
```

```
도시
서울     9.904312
부산     3.448737
인천     2.890451
대구     0.266052
Name: 인구, dtype: float64
```

Series 인덱싱

```
1 population[1],population["부산"]
```

```
(3448737, 3448737)
```

```
1 population[3],population["대구"]
```

```
(266052, 266052)
```

Series 인덱싱

```
1 population[[0,3,1]]
```

```
도시
서울     9904312
대구      266052
부산     3448737
Name: 인구, dtype: int64
```

```
1 population[['서울','대구','부산']]
```

```
도시
서울     9904312
대구      266052
부산     3448737
Name: 인구, dtype: int64
```

Series Boolean 인덱싱

```
1  population >= 2500000
```

```
도시
서울      True
부산      True
인천      True
대구     False
Name: 인구, dtype: bool
```

```
1  population[population >= 2500000]
```

```
도시
서울    9904312
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

Q1. 인구수가 500만 이상의 도시는?

```
1 population[population >= 5000000]
```

```
도시
서울     9904312
Name: 인구, dtype: int64
```

Q2. 인구수가 250만 이상 500만 이하의 도시는?

```
1 population[(population >= 2500000)&(population <= 5000000)]
```

```
도시
부산     3448737
인천     2890451
Name: 인구, dtype: int64
```

Series 슬라이싱

```
1  population[1:3]
```

```
도시
부산    3448737
인천    2890451
Name: 인구, dtype: int64
```

```
1  population["부산":"대구"]
```

```
도시
부산    3448737
인천    2890451
대구     266052
Name: 인구, dtype: int64
```

Dictionary객체로 Series 생성

Dictionary객체로 Series 생성

```
1  data = {"아구몬":9631482, "피요몬":3393191,
2          "텐토몬":1490158, "가부몬":2632035}
3  digimon = pd.Series(data)
4  digimon
```

```
아구몬    9631482
피요몬    3393191
텐토몬    1490158
가부몬    2632035
dtype: int64
```

Dictionary객체로 Series 생성

```
1 data2 = {"아구몬":9904312, "피요몬":3448737,
2         "텐토몬":2466052, "테일몬":2890451}
3 digimon_levelup = pd.Series(data2)
4 digimon_levelup
```

```
아구몬     9904312
피요몬     3448737
텐토몬     2466052
테일몬     2890451
dtype: int64
```

레벨업해서 올라간 공격력을 계산

```
1 attack = digimon_levelup - digimon
2 attack
```

```
가부몬          NaN
아구몬     272830.0
테일몬          NaN
텐토몬     975894.0
피요몬      55546.0
dtype: float64
```

비어있지 않은 데이터들만 보려면?

```
1 attack.notnull()
```

```
가부몬     False
아구몬      True
테일몬     False
텐토몬      True
피요몬      True
dtype: bool
```

```
1 attack[attack.notnull()]
```

```
아구몬     272830.0
텐토몬     975894.0
피요몬      55546.0
dtype: float64
```

비어있는 데이터들만 보려면?

```
1  attack.isnull()
```

```
가부몬     True
아구몬    False
테일몬     True
텐토몬    False
피요몬    False
dtype: bool
```

```
1  attack[attack.isnull()]
```

```
가부몬   NaN
테일몬   NaN
dtype: float64
```

레벨업해서 올라간 공격력의 증가율(%)을 계산

```
1  rp = (digimon_levelup - digimon) / digimon*100
2  rp[rp.notnull()]
```

```
아구몬      2.832690
텐토몬     65.489297
피요몬      1.636984
dtype: float64
```

Series 데이터 갱신, 추가, 삭제

```
1  rp['아구몬'] = 1.6
2  rp['텐토몬'] = 1.41
3  del rp['피요몬']
4  rp[rp.notnull()]
```

```
아구몬    1.60
텐토몬    1.41
dtype: float64
```

DataFrame 사용

DataFrame 생성하기

```
1  data = {"2015":[9904312,3448737,2890451,2466052],
2          "2010":[9631482,3393191,2632035,2431774]}
3
4  df = pd.DataFrame(data)
5  df
```

	2015	2010
0	9904312	9631482
1	3448737	3393191
2	2890451	2632035
3	2466052	2431774

DataFrame 인덱스 수정

```
1 df.index = ['서울','부산','인천','대구']
2 df
```

	2015	2010
서울	9904312	9631482
부산	3448737	3393191
인천	2890451	2632035
대구	2466052	2431774

DataFrame 인덱스 지정하여 생성

```
1  data = [
2      [9904312,3448737,2890451,2466052],
3      [9631482,3393191,2632035,2431774]
4  ]
5
6  ind = ['2015','2010']
7  col = ['서울','부산','인천','대구']
8  df2 = pd.DataFrame(data, index = ind, columns = col)
9
10 df2
```

	서울	부산	인천	대구
2015	9904312	3448737	2890451	2466052
2010	9631482	3393191	2632035	2431774

DataFrame 전치하기

```
1 df2.T
```

	2015	2010
서울	9904312	9631482
부산	3448737	3393191
인천	2890451	2632035
대구	2466052	2431774

1. DataFrame 값 확인

```
1 df.values
```

```
array([[9904312, 9631482],
       [3448737, 3393191],
       [2890451, 2632035],
       [2466052, 2431774]], dtype=int64)
```

2. DataFrame 인덱스 확인

```
1 df.index
```

```
Index(['서울', '부산', '인천', '대구'], dtype='object')
```

3. DataFrame 컬럼 확인

```
1 df.columns
```

```
Index(['2015', '2010'], dtype='object')
```

DataFrame 열 인덱스 확인하기

```
1 df['2015']
```

```
서울      9904312
부산      3448737
인천      2890451
대구      2466052
Name: 2015, dtype: int64
```

```
1 df[['2015']]
```

	2015
서울	9904312
부산	3448737
인천	2890451
대구	2466052

```
1 df[['2010','2015']]
```

	2010	2015
서울	9631482	9904312
부산	3393191	3448737
인천	2632035	2890451
대구	2431774	2466052

DataFrame 새로운 column 추가하기

```
1  df['2005'] = [9762546,3512547,2517680,2456016]
2  df
```

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547
인천	2890451	2632035	2517680
대구	2466052	2431774	2456016

DataFrame 행 인덱싱

```
1  df[0:2]
```

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547

```
1  df["서울":"인천"]
```

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547
인천	2890451	2632035	2517680

loc[] 인덱서

- 실제 인덱스를 사용하여 행을 가지고 올 때 사용

```
1  df.loc['서울':'부산','2015':'2010']
```

	2015	2010
서울	9904312	9631482
부산	3448737	3393191

iloc[] 인덱서

-numpy의 array인덱싱 방식으로 행을 가지고 올때 사용

```
1  df.iloc[3]
```

```
2015    2466052
2010    2431774
2005    2456016
Name: 대구, dtype: int64
```

Pandas Boolean 인덱싱

```
1 df['2010']>=2500000
```

```
서울      True
부산      True
인천      True
대구     False
Name: 2010, dtype: bool
```

```
1 df[df['2010']>=2500000]
```

	2015	2010	2005
서울	9904312	9631482	9762546
부산	3448737	3393191	3512547
인천	2890451	2632035	2517680

csv파일 불러오기

```
1 population_number = pd.read_csv('population_number.csv', encoding = 'euc-kr')
2 population_number
```

	도시	지역	2015	2010	2005	2000
0	서울	수도권	9904312	9631482.0	9762546.0	9853972
1	부산	경상권	3448737	NaN	NaN	3655437
2	인천	수도권	2890451	2632035.0	NaN	2466338
3	대구	경상권	2466052	2431774.0	2456016.0	2473990

csv파일 불러오기

```
1 population_number = pd.read_csv('population_number.csv', index_col='도시', encoding = 'euc-kr')
2 population_number
```

도시	지역	2015	2010	2005	2000
서울	수도권	9904312	9631482.0	9762546.0	9853972
부산	경상권	3448737	NaN	NaN	3655437
인천	수도권	2890451	2632035.0	NaN	2466338
대구	경상권	2466052	2431774.0	2456016.0	2473990

value_counts함수

- 값이 숫자, 문자열, 카테고리 값인 경우에 각각의 **값이 나온 횟수를 세는 기능**

```
1  s2.tail()
```

```
95    0
96    3
97    4
98    3
99    1
dtype: int32
```

```
1  s2.value_counts()
```

```
0     23
4     20
3     16
2     14
1     14
5     13
dtype: int64
```

value_counts함수

- 값이 숫자, 문자열, 카테고리 값인 경우에 각각의 **값이 나온 횟수를 세는 기능**

```
1  population_number['2015'].value_counts()
```

```
2466052    1
2890451    1
3448737    1
9904312    1
Name: 2015, dtype: int64
```

```
1  population_number['2010'].value_counts()
```

```
9631482.0    1
2431774.0    1
2632035.0    1
Name: 2010, dtype: int64
```

정렬

sort_index함수 : 인덱스 값을 기준으로 정렬하는 방법

```
1 population_number.sort_index()
```

	지역	2015	2010	2005	2000
도시					
대구	경상권	2466052	2431774.0	2456016.0	2473990
부산	경상권	3448737	NaN	NaN	3655437
서울	수도권	9904312	9631482.0	9762546.0	9853972
인천	수도권	2890451	2632035.0	NaN	2466338

정렬

sort_value함수 : 데이터 값을 기준으로 정렬하는 방법

```
1  population_number['2010'].sort_values()
```

```
도시
대구     2431774.0
인천     2632035.0
서울     9631482.0
부산           NaN
Name: 2010, dtype: float64
```

정렬

sort_value함수 : 데이터 값을 기준으로 정렬하는 방법

```
1  population_number['2010'].sort_values(ascending=False)
```

```
도시
서울     9631482.0
인천     2632035.0
대구     2431774.0
부산           NaN
Name: 2010, dtype: float64
```

1. DataFrame을 2010년 인구수 기준으로 정렬해라

```
1 population_number.sort_values(by='2010')
```

	지역	2015	2010	2005	2000
도시					
대구	경상권	2466052	2431774.0	2456016.0	2473990
인천	수도권	2890451	2632035.0	NaN	2466338
서울	수도권	9904312	9631482.0	9762546.0	9853972
부산	경상권	3448737	NaN	NaN	3655437

2. DataFrame을 지역, 2010년 인구수 기준으로 정렬해라

```
1 population_number.sort_values(by = ['지역','2010'])
```

도시	지역	2015	2010	2005	2000
대구	경상권	2466052	2431774.0	2456016.0	2473990
부산	경상권	3448737	NaN	NaN	3655437
인천	수도권	2890451	2632035.0	NaN	2466338
서울	수도권	9904312	9631482.0	9762546.0	9853972

score.csv파일 불러오기

```
1 score
```

과목	1반	2반	3반	4반
파이썬	45	44	73	39
DB	76	92	45	69
자바	47	92	45	69
크롤링	92	81	85	40
Web	11	79	47	26

학급별 총계

```
1 score.sum()
```

```
1반     271
2반     388
3반     295
4반     243
dtype: int64
```

학급별 순위

```
1 score.sum().sort_values(ascending = False)
```

```
2반     388
3반     295
1반     271
4반     243
dtype: int64
```

과목별 합계 구하기

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

```
과목
파이썬    201
DB      282
자바      253
크롤링     298
Web     163
dtype: int64
```

과목별 합계를 DataFrame에 추가하기

```
1 score['합계'] = score.sum(axis = 1)
```

```
1 score
```

과목	1반	2반	3반	4반	합계
파이썬	45	44	73	39	201
DB	76	92	45	69	282
자바	47	92	45	69	253
크롤링	92	81	85	40	298
Web	11	79	47	26	163

Q1. 과목별 평균을 계산하여 column을 추가해보자

```
1  score['평균'] = score.loc[:,:'4반'].mean(axis = 1)
```

```
1  score
```

과목	1반	2반	3반	4반	합계	평균
파이썬	45	44	73	39	201	50.25
DB	76	92	45	69	282	70.50
자바	47	92	45	69	253	63.25
크롤링	92	81	85	40	298	74.50
Web	11	79	47	26	163	40.75

Q2. 반 평균을 계산하여 row를 추가해보자

```
1 score.loc['반평균'] = score.loc[:,:].mean()
```

```
1 score
```

과목	1반	2반	3반	4반	합계	평균
파이썬	45.0	44.0	73.0	39.0	201.0	50.25
DB	76.0	92.0	45.0	69.0	282.0	70.50
자바	47.0	92.0	45.0	69.0	253.0	63.25
크롤링	92.0	81.0	85.0	40.0	298.0	74.50
Web	11.0	79.0	47.0	26.0	163.0	40.75
반평균	54.2	77.6	59.0	48.6	239.4	59.85

max(), min()함수

```
1 score.max()
```

```
1반      92.0
2반      92.0
3반      85.0
4반      69.0
합계     298.0
평균      74.5
dtype: float64
```

```
1 score.min()
```

```
1반      11.00
2반      44.00
3반      45.00
4반      26.00
합계     163.00
평균      40.75
dtype: float64
```

```
1 score.max(axis = 1)
```

```
과목
파이썬     201.0
DB      282.0
자바      253.0
크롤링     298.0
Web     163.0
반평균     239.4
dtype: float64
```

```
1 score.min(axis = 1)
```

```
과목
파이썬      39.0
DB       45.0
자바       45.0
크롤링      40.0
Web      11.0
반평균      48.6
dtype: float64
```

Q1. 전체 반에서 과목별 가장 높은 점수만 뽑아와라

```
1  maxArr = score.loc[:'Web',:'4반'].max(axis = 1)
```

```
1  maxArr
```

```
과목
파이썬     73.0
DB      92.0
자바      92.0
크롤링     92.0
Web     79.0
dtype: float64
```

Q2. 전체 반에서 과목별 가장 낮은 점수만 뽑아와라

```
1 minArr = score.loc[:'Web',:'4반'].min(axis = 1)
```

```
1 minArr
```

```
과목
파이썬     39.0
DB       45.0
자바       45.0
크롤링      40.0
Web      11.0
dtype: float64
```

Q3. 과목별 가장 큰 값과 작은 값의 차이를 구하시오.

```
1 maxArr - minArr
```

```
과목
파이썬     34.0
DB      47.0
자바      47.0
크롤링     52.0
Web     68.0
dtype: float64
```

df.apply(func, axis = 0 or 1)

- 행이나 열 단위로 더 복잡한 처리를 하고 싶을 때 사용!
- Pandas의 객체에 다른 라이브러리 함수를 적용하는 방법
- Pandas 객체에 열 혹은 행에 대해 함수를 적용하게 해주는 함수

Q. 과목별 가장 큰 값과 작은 값의 차이를 구하시오.

```
1 score.loc[:'Web',:'4반'].apply(max_min, axis = 1)
```

```
과목
파이썬     34.0
DB      47.0
자바      47.0
크롤링     52.0
Web     68.0
dtype: float64
```

아래와 같은 DataFrame을 만들어보자.

```
1  data_dic = {'A':[1,3,3,4,4],'B':[1,2,2,3,3],'C':[1,2,4,4,5]}
2  df3 = pd.DataFrame(data_dic)
3  df3
```

	A	B	C
0	1	1	1
1	3	2	2
2	3	2	4
3	4	3	4
4	4	3	5

apply를 사용하여 각 열에 대해 어떤 값이 얼마나 사용되었는 지 알아보자.

```
1  df4 = df3.apply(pd.value_counts)
2  df4
```

	A	B	C
1	1.0	1.0	1.0
2	NaN	2.0	1.0
3	2.0	2.0	NaN
4	2.0	NaN	2.0
5	NaN	NaN	1.0

fillna()

- 결측치(NaN)를 원하는 값으로 바꾸는 기능

```
1  df4.fillna(0)
```

	A	B	C
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

카테고리 생성하기

카테고리 만들기

나이	1~15	16~25	26~35	36~60	61~99
구분	미성년자	청년	중년	장년	노년

```
ages = [0, 2, 10, 21, 23, 37, 31, 61, 20, 41, 32, 100]

bins = [0, 15, 25, 35, 60, 99]

labels = ["미성년자", "청년", "중년", "장년", "노년"]

cats = pd.cut( ages, bins, labels=labels)
```

```
1 cats
```

```
[NaN, 미성년자, 미성년자, 청년, 청년, ..., 노년, 청년, 장년, 중년, NaN]
Length: 12
Categories (5, object): [미성년자 < 청년 < 중년 < 장년 < 노년]
```

```
1 type(cats)
```

```
pandas.core.arrays.categorical.Categorical
```

```
1 cats.categories
```

```
Index(['미성년자', '청년', '중년', '장년', '노년'], dtype='object')
```

Q1. age를 데이터프레임으로 만들어라.

```
1 ageArr = pd.DataFrame(ages, columns = ['ages'])
2 ageArr
```

	ages
0	0
1	2
2	10
3	21
4	23
5	37
6	31
7	61

Q2.각 연령에 맞게 카테고리를 추가하자.

```
1 ageArr['age_cat'] = cats
2 ageArr
```

	ages	age_cat
0	0	NaN
1	2	미성년자
2	10	미성년자
3	21	청년
4	23	청년
5	37	장년
6	31	중년
7	61	노년

Q3. 카테고리별로 나이 개수를 확인해보자.

```
1  ageArr['age_cat'].value_counts()
```

```
청년        3
장년        2
중년        2
미성년자      2
노년        1
Name: age_cat, dtype: int64
```

DataFrame 병합

concat함수

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index=[4, 5, 6, 7])

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8, 9, 10, 11])
```

concat함수

```
1 result = pd.concat([df1,df2,df3])
2 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

Keys 속성 부여 시 다중 index가 된다.

```
1 result = pd.concat([df1,df2,df3], keys = ['x','y','z'])
2 result
```

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

다중 index 확인

```
1 result.index
```

```
MultiIndex([('x',  0),
            ('x',  1),
            ('x',  2),
            ('x',  3),
            ('y',  4),
            ('y',  5),
            ('y',  6),
            ('y',  7),
            ('z',  8),
            ('z',  9),
            ('z', 10),
            ('z', 11)],
           )
```

```
1 result.index.get_level_values(0)
```

```
Index(['x', 'x', 'x', 'x', 'y', 'y', 'y', 'y', 'z', 'z', 'z', 'z'], dtype='object')
```

행 방향으로 병합하기

```
1  df4 = pd.DataFrame({'B': ['B2', 'B3', 'B6', 'B7'],
2                      'D': ['D2', 'D3', 'D6', 'D7'],
3                      'F': ['F2', 'F3', 'F6', 'F7']},
4                     index=[2, 3, 6, 7])
5  df4
```

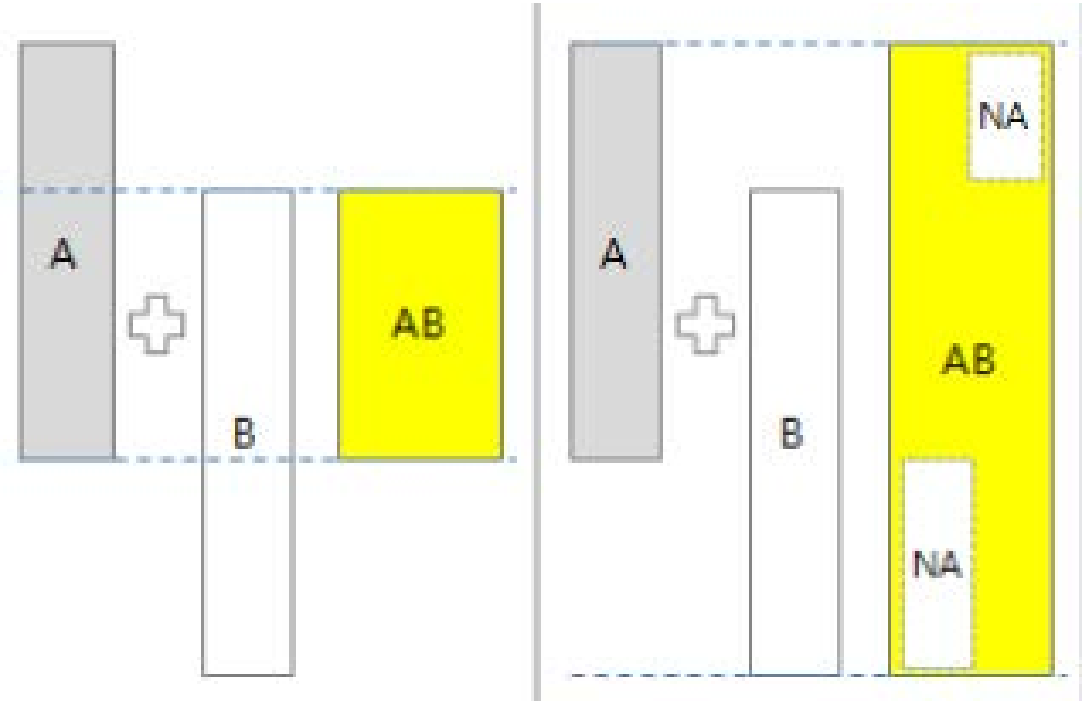
	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

행 방향으로 병합하기

```
1 result = pd.concat([df1,df4],axis = 1)
2 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

concat함수의 join속성



Innerjoin

Fulljoin

(Outerjoin)

concat함수의 join속성

```
1 result = pd.concat([df1,df4],axis = 1, join = 'inner')
2 result
```

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

concat함수의 ignore_index 속성

- 기존 인덱스를 무시하고 새로운 인덱스 부여하는 기능

```
1  result = pd.concat([df1,df4], ignore_index = True)
2  result
```

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

merge 함수 사용하기

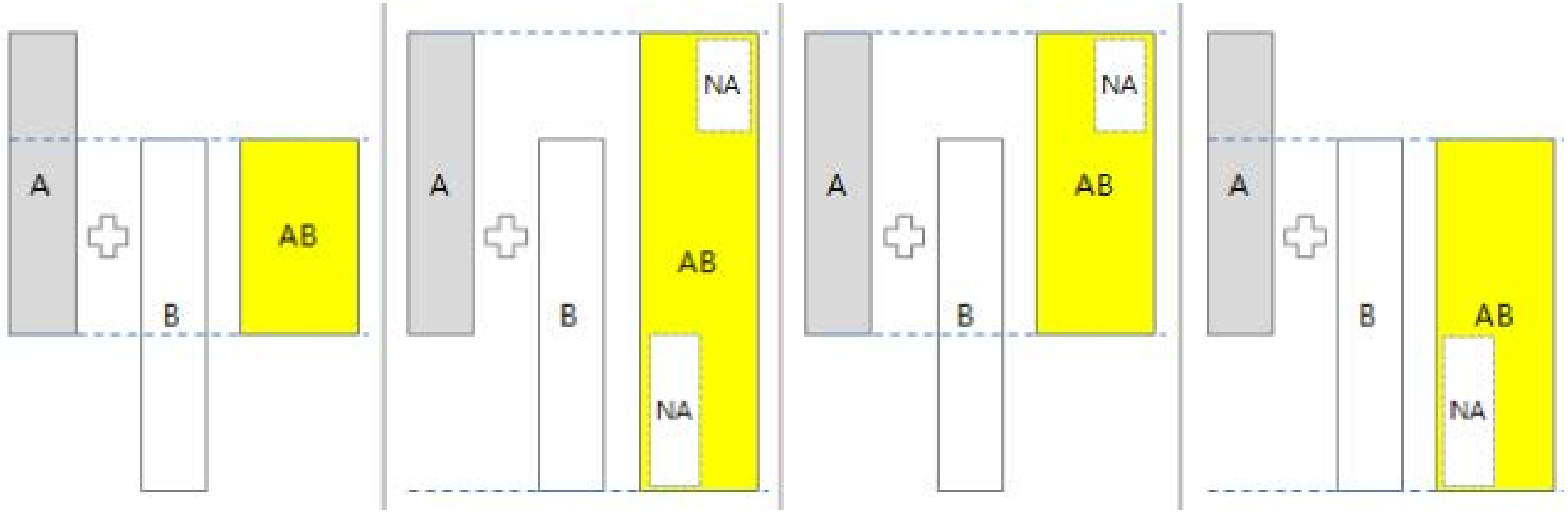
```
df5 = pd.DataFrame({'key': ['K0','K2','K3','K4'],
'A':['A0','A1','A2','A3'],
'B':['B0','B1','B2','B3']})


df6 = pd.DataFrame({'key': ['K0','K1','K2','K3'],
'C':['C0','C1','C2','C3'],
'D':['D0','D1','D2','D3']})

```

merge 함수 사용하기

```
1  result = pd.merge(df5,df6, on ='key')
2  result
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A1	B1	C2	D2
2	K3	A2	B2	C3	D3

merge함수의 how속성



Innerjoin

Fulljoin
(Outerjoin)

leftjoin

rightjoin

how 속성 이용해서 병합하기

```
1 result = pd.merge(df5,df6,how = 'left',on = 'key')
2 result
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A1	B1	C2	D2
2	K3	A2	B2	C3	D3
3	K4	A3	B3	NaN	NaN

```
1 result = pd.merge(df5,df6,how = 'right',on = 'key')
2 result
```

	key	A	B	C	D
0	K0	A0	B0	C0	D0
1	K2	A1	B1	C2	D2
2	K3	A2	B2	C3	D3
3	K1	NaN	NaN	C1	D1

최종 실습문제

-범죄현황 데이터-

2015-2017년 광주광역시 범죄현황 데이터를 이용해
전년 대비 지역별 범죄 증감율을 구해보자!

$$\text{범죄 증감율} = \frac{\text{금년} - \text{작년}}{\text{작년}} * 100$$

결과화면

관서명	2015총계	2015-2016 증감율	2016총계	2016-2017 증감율	2017총계
광주지방경찰청계	18830	-18.130643	15416	-9.516087	13949
광주동부경찰서	2355	-12.186837	2068	-13.007737	1799
광주서부경찰서	4720	-17.542373	3892	-6.526208	3638
광주남부경찰서	2117	-11.903637	1865	-17.050938	1547
광주북부경찰서	5466	-24.112697	4148	-4.893925	3945
광주광산경찰서	4172	-17.473634	3443	-12.285797	3020

데이터 읽어오기

```
1 df2015 = pd.read_csv('2015.csv',encoding='euc-kr',index_col ='관서명')
2 df2016 = pd.read_csv('2016.csv',encoding='euc-kr',index_col ='관서명')
3 df2017 = pd.read_csv('2017.csv',encoding='euc-kr',index_col ='관서명')
```

2017년도에만 있는 데이터 삭제하기

```
1  df2017 = df2017.drop('광주지방경찰청')
```

데이터 총합 구하기

```
df2015['총계'] = df2015.sum(axis = 1)
df2016['총계'] = df2016.sum(axis = 1)
df2017['총계'] = df2017.sum(axis = 1)
```

발생건수와 총계를 기준으로 Series형태로 데이터 자르기

```
1  s1 = df2015[df2015['구분']=='발생건수'].loc[:,'총계']
2  s1.name = '2015총계'
3  s2 = df2016[df2016['구분']=='발생건수'].loc[:,'총계']
4  s2.name = '2016총계'
5  s3 = df2017[df2017['구분']=='발생건수'].loc[:,'총계']
6  s3.name = '2017총계'
```

전년대비 증감율 계산하기

```
1  s4 = (s2-s1)/s1*100
2  s4.name = '2015-2016 증감율'
3
4  s5 = (s3-s2)/s2*100
5  s5.name = '2016-2017 증감율'
```

행 기준 데이터 합치기

```
1 total = pd.concat([s1,s4,s2,s5,s3],axis = 1)
2 total
```

관서명	2015총계	2015-2016 증감율	2016총계	2016-2017 증감율	2017총계
광주지방경찰청계	18830	145.608072	46248	-9.516087	41847
광주동부경찰서	2355	163.439490	6204	-13.007737	5397
광주서부경찰서	4720	147.372881	11676	-6.526208	10914
광주남부경찰서	2117	164.289088	5595	-17.050938	4641
광주북부경찰서	5466	127.661910	12444	-4.893925	11835
광주광산경찰서	4172	147.579099	10329	-12.285797	9060

python

다음시간에는?

Matplotlib