

# Pandas

- Pandas는 파이썬에서 사용하는 데이터 분석 라이브러리로 '판다스'라고 읽는다.
- Pandas는 다차원으로 구조화된 데이터를 뜻하는 계량 경제학 용어인 Panel data와 파이썬 데이터 분석인 Python data analysis에서 따온 이름이다.
- Pandas는 안정적으로 대용량의 데이터를 처리하는데 편리한 도구이다.

- Pandas는 NumPy의 고성능 배열 계산 기능과 스프레드시트, SQL과 같은 관계형 데이터베이스의 데이터 조작 기능을 조합한 것이다.
- Pandas는 series와 dataframe 자료구조를 제공한다.
  - ✓ Series : list와 dictionary의 장점을 섞어 놓은 듯한 자료구조
  - ✓ DataFrame : 행과 열로 이루어진 2차원 형태의 자료구조
- Pandas의 기능을 이용해 데이터의 재배치와 집계, 부분집합 구하기 등을 보다 쉽게 할 수 있다.

- 요소들의 모음 ... 집합과 비슷, but 순서 있음
- 예)
  - [1, 2, 3, 4, 5]
  - [5, 4, 3, 2, 1]
  - ['aa', 'b', 'cde', 'fghi']
  - [3, 8, 1, 3, 2]
  - [1, 'a', [1, 2], 'b']

## List (리스트)

### ■ 관련 연산자

연산자	사용 형태	의미	예
+	리스트 + 리스트	두 리스트 연결 시키기	$[1, 2, 3] + ['a', 'b', 'c'] \rightarrow [1, 2, 3, 'a', 'b', 'c']$
*	리스트 * 숫자	리스트를 숫자 만큼 반복하여 연결	$[1, 2] * 4 \rightarrow [1, 2, 1, 2, 1, 2, 1, 2]$

- 우선 순위: \* > +
- \* 사용 시 숫자
  - ✓ 정수형만 가능
  - ✓ 0이나 음수일 경우: [] (비어있는 리스트)

## List (리스트) – 부분 정보 활용하기

- 리스트 인덱싱(indexing)
  - 리스트의 요소 하나를 선택
  - 인덱싱
    - ✓ 왼쪽부터 0, 1, 2, ... 로 증가
    - ✓ 오른쪽에서 -1, -2, -3, ...로 감소

0	1	2	3	4
['a',	'b',	'c',	'd',	'e']
-5	-4	-3	-2	-1

## List (리스트) – 부분 정보 활용하기

### ■ 리스트 인덱싱(indexing)

- 예시 – 왼쪽부터 증가하는 인덱스 사용

```
In [1]: a = [1, 2, 3]
```

```
In [2]: a[0]
```

```
Out[2]: 1
```

```
In [3]: a[1]
```

```
Out[3]: 2
```

```
In [4]: a[2]
```

```
Out[4]: 3
```

```
In [5]: a[3]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<i python-input-5-f75b6be7d8e3> in <module>  
--> 1 a[3]
```

```
IndexError: list index out of range
```

### ■ 리스트 인덱싱(indexing)

- 예시 – 오른쪽부터 감소하는 인덱스 사용

```
In [1]: a = [1, 2, 3]
```

```
In [2]: a[-1]
```

```
Out [2]: 3
```

```
In [3]: a[-2]
```

```
Out [3]: 2
```

```
In [4]: a[-3]
```

```
Out [4]: 1
```



## List (리스트) – 부분 정보 활용하기

- 문자열 인덱싱(indexing)
  - 문자열도 리스트와 같은 방식으로 인덱싱 가능
  - 예시 – 왼쪽부터 증가하는 인덱스 사용

```
In [1]: a = '123'
```

```
In [2]: a[0]
```

```
Out[2]: '1'
```

```
In [3]: a[1]
```

```
Out[3]: '2'
```

```
In [4]: a[2]
```

```
Out[4]: '3'
```

```
In [5]: a[3]
```

```
-----  
IndexError                                Traceback (most recent call last)  
<ipython-input-5-f75b6be7cd8e3> in <module>  
----> 1 a[3]
```

```
IndexError: string index out of range
```

- 문자열 인덱싱(indexing)
  - 문자열도 리스트와 같은 방식으로 인덱싱 가능
  - 예시 – 오른쪽부터 감소하는 인덱스 사용

```
In [1]: a = '123'
```

```
In [2]: a[-1]
```

```
Out[2]: '3'
```

```
In [3]: a[-2]
```

```
Out[3]: '2'
```

```
In [4]: a[-3]
```

```
Out[4]: '1'
```

- 리스트 슬라이싱(slicing)
  - 리스트의 일부분을 잘라 냄
    - ✓ 결과는 리스트
    - ✓ 부분 집합과 비슷
  - 사용하는 방법: **변수**[시작 인덱스:끝 인덱스:스텝]
    - ✓ 시작 인덱스: 범위의 시작, 생략 시 0
    - ✓ 끝 인덱스: 범위의 끝, 생략 시 리스트의 크기
      - ❖ 끝 인덱스는 미포함, 직전 값까지만 포함
    - ✓ 스텝: 자료를 취하는 간격, 생략 시 1

### ■ 리스트 슬라이싱(slicing)

#### • 예시

```
In [1]: a = [0, 1, 2, 3, 4, 5]
```

```
In [2]: a[0:2]
```

```
Out [2]: [0, 1]
```

```
In [3]: a[2:]
```

```
Out [3]: [2, 3, 4, 5]
```

```
In [4]: a[:4:2]
```

```
Out [4]: [0, 2]
```

```
In [5]: a[-3:]
```

```
Out [5]: [3, 4, 5]
```

```
In [6]: a[:-3]
```

```
Out [6]: [0, 1, 2]
```

### ■ 문자열 슬라이싱(slicing)

- 문자열도 리스트와 같은 방식으로 슬라이싱 가능
- 예시

```
In [1]: a = '012345'
```

```
In [2]: a[0:2]
```

```
Out [2]: '01'
```

```
In [3]: a[2:]
```

```
Out [3]: '2345'
```

```
In [4]: a[:3]
```

```
Out [4]: '012'
```

```
In [5]: a[-3:]
```

```
Out [5]: '345'
```

```
In [6]: a[:-3]
```

```
Out [6]: '012'
```

## List (리스트) - 생성하기

### ■ 리스트 생성 관련 함수

- list(), split(), range(끝), range(시작, 끝), range(시작, 끝, 스텝)

```
list('12345') → ['1', '2', '3', '4', '5']
```

```
a = '1 2 3 4 5'  
b = a.split() → b = ['1', '2', '3', '4', '5']
```

```
a = '1:2:3:4:5'  
b = a.split(':') → b = ['1', '2', '3', '4', '5']
```

```
list(range(4)) → [0, 1, 2, 3]
```

```
list(range(3, 5)) → [3, 4]
```

```
list(range(2, 11, 2)) → [2, 4, 6, 8, 10]
```

```
list(range(9, 1, -2)) → [9, 7, 5, 3]
```

## List (리스트) - 수정하기

### ■ 리스트에 추가, 삭제 관련 함수

사용 방법	의미	예시(a = [1, 2, 3]일 때)
리스트.append(요소)	리스트의 마지막에 요소를 추가	a.append(4) → a = [1, 2, 3, 4]
리스트.extend(리스트2)	리스트의 마지막에 리스트2를 추가	a.extend([4, 5]) → a = [1, 2, 3, 4, 5]
리스트.insert(index, 요소)	리스트의 index 위치에 요소를 추가	a.insert(1, 4) → a = [1, 4, 2, 3]
del 리스트[index]	리스트의 index에 위치한 요소를 삭제	del a[1] → a = [1, 3]
리스트.remove(요소)	리스트에서 첫 번째로 나오는 요소를 삭제	a.remove(1) → a = [2, 3]

- 생년월일을 입력 받아 홀수 번째 글자들로만 이루어진 문자열을 출력하는 프로그램을 작성하시오.
  - 문자열 슬라이싱 이용하기

```
In [1]: birthday = input('생년월일 입력(yyyymmdd): ')
        part = birthday[::2]
        print(part)
```

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
생년월일 입력(yyyymmdd): 20190101
2100
```



- 문자열을 입력 받아 거꾸로 출력하는 프로그램을 작성하시오.
  - 문자열 슬라이싱 이용하기

```
In [1]: print('문자열을 입력하시오.')
        myStr = input()
        revStr = myStr[-1::-1]
        print('거꾸로 문자열:')
        print(revStr)
```

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
문자열을 입력하시오.
Hello
거꾸로 문자열:
olleH
```

- Key와 value 쌍들의 모음
- {} 사용
- 예:
  - {'name':'gdhong', 'phone':'0222200001', 'addr':['Seoul', 'Wangsimni']}
- Key
  - 중복 x, list형 불가
- Value
  - 숫자, 문자열, list, dictionary 등 대부분의 자료형 가능

## ■ Value의 선택

- List와 비교하였을 때 index 번호(0부터 시작) 대신 key 값으로 value를 선택

## ■ 예시

```
In [1]: a = {1:'a', 2:'b', 'three':'c'}
```

```
In [2]: a[1]
```

```
Out [2]: 'a'
```

```
In [3]: a[2]
```

```
Out [3]: 'b'
```

```
In [4]: a['three']
```

```
Out [4]: 'c'
```

```
In [5]: a.get(1)
```

```
Out [5]: 'a'
```

```
In [6]: a.get(2)
```

```
Out [6]: 'b'
```

```
In [7]: a.get('three')
```

```
Out [7]: 'c'
```

- 요소 수정, 추가, 삭제
  - 수정: 기존에 있는 쌍에서 value만 수정
  - 추가: 기존에 없는 key에 대한 value를 대입
  - 삭제: `del dictionary[키]`
  - 모두 삭제: `clear()`

## Dictionary - 수정하기

- 요소 수정, 추가, 삭제
  - 예시

```
In [1]: a = {1:'a', 2:'b', 'three':'c'}
```

```
In [2]: a[1] = 'abc'
```

```
In [3]: a
```

```
Out[3]: {1: 'abc', 2: 'b', 'three': 'c'}
```

```
In [4]: a[4] = 'd'
```

```
In [5]: a
```

```
Out[5]: {1: 'abc', 2: 'b', 'three': 'c', 4: 'd'}
```

```
In [6]: del a['three']
```

```
In [7]: a
```

```
Out[7]: {1: 'abc', 2: 'b', 4: 'd'}
```

```
In [8]: a.clear()
```

```
In [9]: a
```

```
Out[9]: {}
```

- Key만 얻기, value만 얻기
  - keys(): dictionary의 key만 모아서 반환
  - values(): dictionary의 value만 모아서 반환
  - 예시

```
In [1]: a = {1:'a', 2:'b', 'three':'c'}
```

```
In [2]: a.keys()
```

```
Out[2]: dict_keys([1, 2, 'three'])
```

```
In [3]: list(a.keys())
```

```
Out[3]: [1, 2, 'three']
```

```
In [4]: a.values()
```

```
Out[4]: dict_values(['a', 'b', 'c'])
```

```
In [5]: list(a.values())
```

```
Out[5]: ['a', 'b', 'c']
```

## 실습

- n, m을 입력 받아 'nXm'을 key로, n\*1, n\*2, ..., n\*m을 요소로 갖는 list를 value로 갖는 dictionary를 출력하시오.

```
In [1]: n = int(input('Input a number: '))
        m = int(input('Input a number: '))
        nList = list(range(n, n*m+1, n))
        mDic = {str(n)+'X'+str(m):nList}
        print(mDic)
```

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
Input a number: 5
Input a number: 8
{'5X8': [5, 10, 15, 20, 25, 30, 35, 40]}
```

```
Input a number: 2
Input a number: 7
{'2X7': [2, 4, 6, 8, 10, 12, 14]}
```

- Dictionary를 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
학생 수: 3  
학번: 11  
이름: 홍길동  
학번: 22  
이름: 김철수  
학번: 33  
이름: 이영미  
입력된 값:  
{'11': '홍길동', '22': '김철수', '33': '이영미'}  
검색할 학번: 22  
학번 22에 해당하는 학생의 이름은 김철수입니다.  
계속 검색하시겠습니까? (y/n) y  
검색할 학번: 11  
학번 11에 해당하는 학생의 이름은 홍길동입니다.  
계속 검색하시겠습니까? (y/n) n
```



```
n = int(input('학생 수: '))
dic = {}
for i in range(n):
    num = input('학번: ')
    name = input('이름: ')
    dic[num] = name;
print('입력된 값:')
print(dic)
while True:
    num = input('검색할 학번: ')
    print('학번 ' + num + '에 해당하는 학생의 이름은 ' + dic.get(num) + '입니다.')
    answer = input('계속 검색하시겠습니까? (y/n) ')
    if answer != 'y':
        break
```

- 배열은 리스트와 비슷하지만 다음과 같은 점에서 다르다.
  - 모든 원소가 같은 자료형이어야 한다.
  - 원소의 개수를 바꿀 수 없다.
- 파이썬은 자체적으로 배열 자료형을 제공하지 않는다.
- 배열은 NumPy 라이브러리에서 제공한다.

# Array - 생성

- List를 array로 만들기
  - 생성방법: 배열 = np.array(리스트)
  - 1차원 vs. 2차원

```
In [1]: import numpy as np
```

```
In [2]: a = np.array([1, 2, 3])
```

```
In [3]: a
```

```
Out [3]: array([1, 2, 3])
```

```
In [1]: import numpy as np
```

```
In [2]: a = np.array([[1, 2, 3], [4, 5, 6]])
```

```
In [3]: a
```

```
Out [3]: array([[1, 2, 3],  
               [4, 5, 6]])
```

## Array - 생성

- 동일 간격으로 등분한 array 생성
  - 배열 = np.linspace(시작, 끝, 숫자개수)
  - 예시

```
In [1]: import numpy as np
```

```
In [2]: a = np.linspace(0, 15, 4)
```

```
In [3]: a
```

```
Out [3]: array([ 0.,  5., 10., 15.])
```

```
In [1]: import numpy as np
```

```
In [2]: b = np.linspace(0, 1, 5)
```

```
In [3]: print(b)
```

```
[0.  0.25 0.5  0.75 1.  ]
```

## Array - 생성

### ■ 수열로 구성된 array 생성

- range + array
- 배열 = np.arange(시작, 끝, 증감)
- 예시

```
In [1]: import numpy as np
```

```
In [4]: a = np.arange(10)
```

```
In [5]: print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
In [6]: b = np.arange(3, 10)
```

```
In [7]: print(b)
```

```
[3 4 5 6 7 8 9]
```

```
In [8]: c = np.arange(3, 10, 2)
```

```
In [9]: print(c)
```

```
[3 5 7 9]
```

## Array - list에 비해 편리한 점

- 다수의 값에 동일한 연산하기
  - List의 기본 연산으로는 해결 x

```
In [1]: a = [1, 2, 3]
```

```
In [2]: b = a * 2
```

```
In [3]: b
```

```
Out [3]: [1, 2, 3, 1, 2, 3]
```



```
In [1]: a = [1, 2, 3]
```

```
In [2]: b = []
```

```
In [3]: for n in a:  
        b.append(n*2)
```

```
In [4]: b
```

```
Out [4]: [2, 4, 6]
```

## Array - list에 비해 편리한 점

- 다수의 값에 동일한 연산하기
  - Array 사용

```
In [1]: a = [1, 2, 3]
```

```
In [2]: b = []
```

```
In [3]: for n in a:  
        b.append(n*2)
```

```
In [4]: b
```

```
Out[4]: [2, 4, 6]
```



```
In [1]: import numpy as np
```

```
In [2]: a = np.array([1, 2, 3])
```

```
In [3]: b = a * 2
```

```
In [4]: print(b)
```

```
[2 4 6]
```

## Array - list에 비해 편리한 점

- 다수의 값에 동일한 연산하기
  - List의 기본 연산으로는 해결 x

```
In [5]: a = [1, 2, 3]
```

```
In [6]: b = [4, 5, 6]
```

```
In [7]: c = a + b
```

```
In [8]: c
```

```
Out [8]: [1, 2, 3, 4, 5, 6]
```



```
In [5]: a = [1, 2, 3]
```

```
In [6]: b = [4, 5, 6]
```

```
In [9]: c = []
```

```
In [10]: for i in range(3):  
         c.append(a[i] + b[i])
```

```
In [11]: c
```

```
Out [11]: [5, 7, 9]
```



## Array - list에 비해 편리한 점

- 다수의 값에 동일한 연산하기
  - Array 사용

```
In [5]: a = [1, 2, 3]
```

```
In [6]: b = [4, 5, 6]
```

```
In [9]: c = []
```

```
In [10]: for i in range(3):  
         c.append(a[i] + b[i])
```

```
In [11]: c
```

```
Out[11]: [5, 7, 9]
```



```
In [1]: import numpy as np
```

```
In [2]: a = np.array([1, 2, 3])
```

```
In [3]: b = np.array([4, 5, 6])
```

```
In [4]: c = a + b
```

```
In [5]: print(c)
```

```
[5 7 9]
```

## Array - list에 비해 편리한 점

### ■ 합 구하기

- List의 기본 연산으로는 해결 x

```
In [1]: a = [[1,2,3], [4,5,6]]
```

```
In [2]: b = []
```

```
In [3]: for i in range(3):  
        b.append(a[0][i] + a[1][i])
```

```
In [4]: b
```

```
Out[4]: [5, 7, 9]
```



```
In [1]: import numpy as np
```

```
In [2]: a = np.array([[1,2,3], [4,5,6]])
```

```
In [3]: b = a.sum(axis=0)
```

```
In [4]: print(b)
```

```
[5 7 9]
```

## Array - list에 비해 편리한 점

### ■ 합 구하기

- List의 기본 연산으로는 해결 x

```
In [1]: a = [[1,2,3], [4,5,6]]
```

```
In [2]: b = []
```

```
In [3]: for i in range(2):  
        b.append(a[i][0] + a[i][1] + a[i][2])
```

```
In [4]: b
```

```
Out[4]: [6, 15]
```



```
In [1]: import numpy as np
```

```
In [2]: a = np.array([[1,2,3], [4,5,6]])
```

```
In [3]: b = a.sum(axis=1)
```

```
In [4]: print(b)
```

```
[ 6 15]
```

# Array - list와의 유사점

- Indexing/slicing 방법이 비슷
  - 예시

```
In [1]: import numpy as np
```

```
In [6]: a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [7]: a
```

```
Out [7]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [8]: a[3]
```

```
Out [8]: 3
```

```
In [9]: a[3:]
```

```
Out [9]: array([3, 4, 5, 6, 7, 8, 9])
```

```
In [10]: a[:3]
```

```
Out [10]: array([0, 1, 2])
```

```
In [11]: a[::2]
```

```
Out [11]: array([0, 2, 4, 6, 8])
```

```
In [12]: a[::-1]
```

```
Out [12]: array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
In [13]: a[-2:-1]
```

```
Out [13]: array([8, 7, 6, 5, 4, 3])
```

## Array - list와의 유사점

- Indexing/slicing 방법이 비슷
  - 예시

```
In [1]: import numpy as np
```

```
In [2]: a = np.array([[3, 0, 5, 5, 1, 0], [9, 7, 0, 3, 5, 8], [1, 1, 9, 7, 8, 0]])
```

```
In [3]: print(a)
```

```
[[3 0 5 5 1 0]
 [9 7 0 3 5 8]
 [1 1 9 7 8 0]]
```

```
In [4]: a[2]
```

```
Out[4]: array([1, 1, 9, 7, 8, 0])
```

```
In [5]: a[2, 3]
```

```
Out[5]: 7
```

```
In [7]: a[1:, 3:]
```

```
Out[7]: array([[3, 5, 8],
               [7, 8, 0]])
```

- Array를 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
학생 수: 3
1번째 학생의 국어,영어,수학 성적(,로 구분): 50,60,80
2번째 학생의 국어,영어,수학 성적(,로 구분): 80,70,90
3번째 학생의 국어,영어,수학 성적(,로 구분): 70,80,80
학생들의 성적:
[[50 60 80]
 [80 70 90]
 [70 80 80]]
각 학생의 총점:
[190 240 230]
각 과목의 평균:
[66.66666667 70.          83.33333333]
```

```
import numpy as np

n = int(input('학생 수: '))
score_list = []
for i in range(1, n+1):
    scores = input('%d번째 학생의 국어,영어,수학 성적(,로 구분): ' % i).split(',')
    scores = [int(i) for i in scores]
    score_list.append(scores)

score_array = np.array(score_list)
print('학생들의 성적: ')
print(score_array)
print('각 학생의 총점: ')
print(score_array.sum(axis=1))
print('각 과목의 평균: ')
print(score_array.sum(axis=0)/n)
```

- Series: 1차원 배열 + index
  - index: values를 선택할 때 주소 역할을 하는 배열(값이 모두 달라야 함)
  - values: 데이터 부분에 해당하는 배열
- 1차원 배열 vs. Series: list vs. dictionary와 비슷
  - 1차원 배열/list: index 번호(자동)로 값 접근
  - Series/Dictionary: key/index명(지정)으로 값 접근



### ■ Values만 입력하는 방법

- 생성 방법: `s = Series(list/array)`
- Index 값은 0, 1, 2, ..., 로 자동 생성
- 예시

```
In [1]: import pandas as pd
```

```
In [2]: score = [84, 21, 87, 100, 59, 46]
```

```
In [3]: s = pd.Series(score)
```

```
In [4]: print(s)
```

```
0    84
1    21
2    87
3   100
4    59
5    46
dtype: int64
```

- Index + values 입력하는 방법
  - 생성 방법: `s = Series(list/array, index = list/array)`
  - Index는 주어진 list나 array로 지정
  - 예시

```
In [1]: import pandas as pd
```

```
In [2]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [3]: score = [84, 21, 87, 100, 59, 46]
```

```
In [4]: s = pd.Series(score, index=names)
```

```
In [5]: print(s)
```

```
철수    84
영이    21
길동    87
미영   100
순이    59
철이    46
dtype: int64
```

### ■ Dictionary를 이용하는 방법

- 생성 방법: `s = Series(dictionary)`
- 예시

```
In [1]: import pandas as pd
```

```
In [2]: dic = {'철수':84, '영이':21, '길동':87, '미영':100, '순이':59, '철이':46}
```

```
In [3]: s = pd.Series(dic)
```

```
In [4]: print(s)
```

```
철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
dtype: int64
```

### ■ 덧셈

- Array간 덧셈:  $\text{score1} + \text{score2} \rightarrow$  순서대로 하나씩 더함
- Series간 덧셈:  $s0 + s1 \rightarrow$  순서와 상관없이 같은 index명을 갖는 값끼리 더함
  - ✓ values만 연산에 관여함
  - ✓ index가 같은 값끼리 연산 수행  $\rightarrow$  데이터 관리에 유리

### ■ 뺄셈, 곱셈 등도 덧셈과 같은 방식으로 처리

## ■ 산술 연산

### • 예시

```
In [1]: import numpy as np
```

```
In [2]: import pandas as pd
```

```
In [3]: names1 = np.array(['철수', '영이', '길동', '미영', '순이', '철이'])
```

```
In [4]: score1 = np.array([84, 21, 87, 100, 59, 46])
```

```
In [5]: names2 = np.array(['길동', '철수', '영이', '철이', '순이', '미영'])
```

```
In [6]: score2 = np.array([99, 87, 87, 84, 77, 15])
```

```
In [7]: s1 = pd.Series(score1, index=names1)
```

```
In [8]: s2 = pd.Series(score2, index=names2)
```

```
In [9]: s1
```

```
Out [9]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
dtype: int32
```

```
In [10]: s2
```

```
Out [10]: 길동      99  
철수      87  
영이      87  
철이      84  
순이      77  
미영      15  
dtype: int32
```

### ■ 산술 연산

#### • 예시

```
In [9]: s1
Out[9]: 철수      84
        영이      21
        길동      87
        미영     100
        순이      59
        철이      46
        dtype: int32
```

```
In [10]: s2
Out[10]: 길동      99
        철수      87
        영이      87
        철이      84
        순이      77
        미영      15
        dtype: int32
```



```
In [11]: s1 + 10
Out[11]: 철수      94
        영이      31
        길동      97
        미영     110
        순이      69
        철이      56
        dtype: int32
```

```
In [12]: s1 + s2
Out[12]: 길동     186
        미영     115
        순이     136
        영이     108
        철수     171
        철이     130
        dtype: int32
```

```
In [15]: s1 - s2
Out[15]: 길동     -12
        미영      85
        순i     -18
        영이     -66
        철수      -3
        철이     -38
        dtype: int32
```

```
In [16]: (s1 + s2) / 2
Out[16]: 길동      93.0
        미영      57.5
        순이      68.0
        영이      54.0
        철수      85.5
        철이      65.0
        dtype: float64
```

## Series - 부분 정보 선택하기

- Index번호를 사용한 부분 정보 선택
  - 예시

```
In [9]: s1
Out[9]: 철수      84
        영이      21
        길동      87
        미영     100
        순이      59
        철이      46
        dtype: int32
```



```
In [17]: s1[2]
Out[17]: 87
```

```
In [18]: s1[2:]
Out[18]: 길동      87
        미영     100
        순이      59
        철이      46
        dtype: int32
```

```
In [19]: s1[:3]
Out[19]: 철수      84
        영이      21
        길동      87
        dtype: int32
```

```
In [20]: s1[:,2]
Out[20]: 철수      84
        길동      87
        순이      59
        dtype: int32
```

```
In [21]: s1[1:3]
Out[21]: 영이      21
        길동      87
        dtype: int32
```

## Series - 부분 정보 선택하기

### ■ Index명을 사용한 부분 정보 선택

#### • 예시

```
In [9]: s1
Out [9]: 철수      84
영이      21
길동      87
미영     100
순이      59
철이      46
dtype: int32
```



```
In [22]: s1['영이']
Out [22]: 21

In [23]: s1['영이':'순이']
Out [23]: 영이      21
길동      87
미영     100
순이      59
dtype: int32
```

```
In [25]: s1['미영':]
Out [25]: 미영     100
순이      59
철이      46
dtype: int32
```

```
In [26]: s1[:, '길동']
Out [26]: 철수      84
영이      21
길동      87
dtype: int32
```

```
In [27]: s1[:, '길동':2]
Out [27]: 철수      84
길동      87
dtype: int32
```

```
In [28]: s1['철이':'길동':-1]
Out [28]: 철이      46
순이      59
미영     100
길동      87
dtype: int32
```



## Series - 값 추가, 수정, 삭제

### ■ 값 추가

- index 명을 사용하여 값 추가
- 예시

```
In [54]: s3 = s1
```

```
In [55]: s3
```

```
Out [55]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
dtype: int32
```



```
In [8]: s3['슬기'] = 98
```

```
In [9]: s3
```

```
Out [9]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
슬기      98  
dtype: int64
```

## Series - 값 추가, 수정, 삭제

### ■ 값 수정

- index 번호와 index 명을 사용하여 값 수정
- 예시

```
In [54]: s3 = s1
```

```
In [55]: s3
```

```
Out [55]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
dtype: int32
```



```
In [14]: s3[2] = 88
```

```
In [15]: s3
```

```
Out [15]: 철수      84  
영이      21  
길동      88  
미영     100  
순이      59  
철이      46  
슬기      98  
dtype: int64
```

```
In [16]: s3['길동'] = 87
```

```
In [17]: s3
```

```
Out [17]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
슬기      98  
dtype: int64
```

## Series - 값 추가, 수정, 삭제

### ■ 값 삭제

- index 명을 사용하여 값 삭제
- 예시

```
In [54]: s3 = s1
```

```
In [55]: s3
```

```
Out [55]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
철이      46  
dtype: int32
```



```
In [18]: del s3['철이']
```

```
In [19]: s3
```

```
Out [19]: 철수      84  
영이      21  
길동      87  
미영     100  
순이      59  
슬기      98  
dtype: int64
```

## Series – 논리 연산과 filtering

### ■ 논리 연산과 filtering 예시

In [48]:

s1

Out [48]:

철수	84
영이	21
길동	87
미영	100
순이	59
철이	46

dtype: int32

In [49]:

s2

Out [49]:

길동	99
철수	87
영이	87
철이	84
순이	77
미영	15

dtype: int32



In [50]:

x = s1 > 85

In [51]:

x

Out [51]:

철수	False
영이	False
길동	True
미영	True
순이	False
철이	False

dtype: bool

In [52]:

s1[x]

Out [52]:

길동	87
미영	100

dtype: int32

In [53]:

s2[x]

Out [53]:

길동	99
미영	15

dtype: int32

- Series를 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```

학생들의 이름 입력(,로 구분): 영희,철수,미나
학생들의 국어성적 입력(,로 구분): 20,20,40
학생들의 영어성적 입력(,로 구분): 40,20,80
학생들의 수학성적 입력(,로 구분): 15,65,95
국어성적
영희    20
철수    20
미나    40
dtype: int64
영어성적
영희    40
철수    20
미나    80
dtype: int64
수학성적
영희    15
철수    65
미나    95
dtype: int64
합계
영희    75
철수    105
미나    215
dtype: int64
    
```

```
import pandas as pd

name = input('학생들의 이름 입력(,로 구분): ').split(',')
score_kor = input('학생들의 국어성적 입력(,로 구분): ').split(',')
score_eng = input('학생들의 영어성적 입력(,로 구분): ').split(',')
score_math = input('학생들의 수학성적 입력(,로 구분): ').split(',')

score_kor = [int(i) for i in score_kor]
score_eng = [int(i) for i in score_eng]
score_math = [int(i) for i in score_math]

s_kor = pd.Series(score_kor, index = name)
s_eng = pd.Series(score_eng, index = name)
s_math = pd.Series(score_math, index = name)
s_tot = s_kor + s_eng + s_math

print('국어성적')
print(s_kor)
print('영어성적')
print(s_eng)
print('수학성적')
print(s_math)
print('합계')
print(s_tot)
```

# DataFrame 개요

- Series: 1차원 배열 + index
- DataFrame: 2차원 배열 + index + columns

	국어	영어	수학
철수	84	87	78
영이	21	87	84
길동	87	99	76
미영	100	15	99
순이	59	77	59
철이	46	84	56

index    Values(1차원)

	국어	영어	수학
철수	84	87	78
영이	21	87	84
길동	87	99	76
미영	100	15	99
순이	59	77	59
철이	46	84	56

index    Values(2차원)

columns

### ■ Series를 이용하여 생성

- 빈 DataFrame 만들기: `d = pd.DataFrame()`
- 열 채우기: `d[column명] = series`
- 예시

```
In [1]: import pandas as pd
```

```
In [2]: names1 = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [3]: score1 = [84, 21, 87, 100, 59, 46]
```

```
In [4]: names2 = ['길동', '철수', '영이', '철이', '순이', '미영']
```

```
In [5]: score2 = [99, 87, 87, 84, 77, 15]
```

```
In [6]: s1 = pd.Series(score1, index=names1)
```

```
In [7]: s2 = pd.Series(score2, index=names2)
```

```
In [10]: d = pd.DataFrame()
```

```
In [11]: d['국어'] = s1
```

```
In [12]: d['영어'] = s2
```

```
In [13]: d['합'] = d.국어 + d.영어
```

```
In [14]: d
```

Out [14]:

	국어	영어	합
철수	84	87	171
영이	21	87	108
길동	87	99	186
미영	100	15	115
순이	59	77	136
철이	46	84	130



- 데이터 직접 넣어 생성
  - 방법1: `d = pd.DataFrame(list/array/dictionary)`
  - 방법2: `d = pd.DataFrame(list/array/dictionary, index = list/array)`
  - 방법3: `d = pd.DataFrame(list/array/dictionary, index = list/array, columns = list/array)`

- 데이터 직접 넣어 생성
  - 예시 – index와 columns 지정 없을 때

```
In [1]: import pandas as pd
```

```
In [2]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [3]: d1 = pd.DataFrame(scores)
```

```
In [4]: d1
```

Out [4]:

	0	1	2
0	84	87	78
1	21	15	84
2	87	84	76
3	100	87	99
4	59	99	59
5	46	77	56

### ■ 데이터 직접 넣어 생성

- 예시 – index와 columns 지정 있을 때

```
In [1]: import pandas as pd
```

```
In [2]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [3]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [4]: lectures = ['국어', '수학', '영어']
```

```
In [5]: d2 = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [6]: d2
```

```
Out [6]:
```

	국어	수학	영어
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

### ■ 데이터 직접 넣어 생성

- 예시 – dictionary를 사용할 때

```
In [1]: import pandas as pd
```

```
In [2]: ScoresWithLectures = {'수학': [84, 21, 87, 100, 59, 46], '국어': [87, 15, 84, 87, 99, 77], '영어': [78, 84, 76, 99, 59, 56]}
```

```
In [3]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [4]: d3 = pd.DataFrame(ScoresWithLectures, index = names)
```

```
In [5]: d3
```

```
Out [5]:
```

	수학	국어	영어
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## ■ index 변경

- DataFrame객체.index = 새로운 index 배열/list
- 예시

In [6]: d1

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

In [7]: d1.index = ['학생1', '학생2', '학생3', '학생4', '학생5', '학생6']

In [8]: d1

Out [8]:

	국어	영어	수학
학생1	84	87	78
학생2	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56

# DataFrame - index, column 변경

## ■ index 변경

- DataFrame객체.rename(index={기존 index : 새 index, ...})

✓ 새로운 DataFrame 객체 반환

## • 예시

```
In [10]: d1
```

```
Out[10]:
```

	국어	영어	수학
학생1	84	87	78
학생2	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56



```
In [11]: d1.rename(index={'학생1':'철수', '학생2':'영이'})
```

```
Out[11]:
```

	국어	영어	수학
철수	84	87	78
영이	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56

```
In [12]: d1
```

```
Out[12]:
```

	국어	영어	수학
학생1	84	87	78
학생2	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56

## DataFrame - index, column 변경

### ■ index 변경

- DataFrame객체.rename(index={기존 index : 새 index, ...}, inplace=True)

✓ 원본 객체 변경

- 예시

In [10]: d1

Out[10]:

	국어	영어	수학
학생1	84	87	78
학생2	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56



In [13]: d1.rename(index={'학생1':'철수', '학생2':'영이'}, inplace=True)

In [14]: d1

Out[14]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
학생3	87	84	76
학생4	100	87	99
학생5	59	99	59
학생6	46	77	56

## DataFrame - index, column 변경

### ■ column 변경

- DataFrame객체.columns = 새로운 columns 배열/list
- 예시

In [16]: d1

Out[16]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [17]: d1.columns = ['과목1', '과목2', '과목3']

In [18]: d1

Out[18]:

	과목1	과목2	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



# DataFrame - index, column 변경

## ■ column 변경

- DataFrame객체.rename(columns={기존 이름 : 새 이름, ...})

✓ 새로운 DataFrame 객체 반환

- 예시

```
In [18]: d1
```

```
Out[18]:
```

	과목1	과목2	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [19]: d1.rename(columns={'과목1':'국어', '과목2':'영어'})
```

```
Out[19]:
```

	국어	영어	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

```
In [20]: d1
```

```
Out[20]:
```

	과목1	과목2	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## DataFrame - index, column 변경

### ■ column 변경

- DataFrame객체.rename(columns={기존 이름 : 새 이름, ...}, inplace=True)
  - ✓ 원본 객체 변경
- 예시

In [18]: d1

Out[18]:

	과목1	과목2	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [21]: d1.rename(columns={'과목1':'국어', '과목2':'영어'}, inplace=True)

In [22]: d1

Out[22]:

	국어	영어	과목3
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

# DataFrame - 행, 열 삭제

## ■ 행 삭제

- DataFrame객체.drop(index/list/배열, [axis=0])

✓ 새로운 DataFrame 객체 반환

- 예시

In [32]: d1

Out[32]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

In [43]: d1.drop('철수', axis=0)

Out[43]:

	국어	영어	수학
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

In [44]: d1

Out[44]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

In [45]: d1.drop(['영이', '미영'], axis=0)

Out[45]:

	국어	영어	수학
철수	84	87	78
길동	87	84	76
순이	59	99	59
철이	46	77	56

In [46]: d1

Out[46]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

# DataFrame - 행, 열 삭제

## ■ 행 삭제

- DataFrame객체.drop(index/list/배열, [axis=0], inplace=True)

✓ 원본 객체 변경

- 예시

In [32]: d1

Out [32]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [47]: d1.drop('영이', axis=0, inplace=True)

In [48]: d1

Out [48]:

	국어	영어	수학
철수	84	87	78
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

# DataFrame - 행, 열 삭제

## ■ 열 삭제

- DataFrame 객체.drop(column명/list/배열, axis=1)

✓ 새로운 DataFrame 객체 반환

- 예시

In [50]: d1

Out[50]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [51]: d1.drop('국어', axis=1)

Out[51]:

	영어	수학
철수	87	78
영이	15	84
길동	84	76
미영	87	99
순이	99	59
철이	77	56

In [52]: d1

Out[52]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

In [53]: d1.drop(['영어', '수학'], axis=1)

Out[53]:

	국어
철수	84
영이	21
길동	87
미영	100
순이	59
철이	46

In [54]: d1

Out[54]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

# DataFrame - 행, 열 삭제

## ■ 열 삭제

- DataFrame객체.drop(column명/list/배열, axis=1, inplace=True)
  - ✓ 원본 객체 변경
- 예시

In [50]: d1

Out[50]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [55]: d1.drop('영어', axis=1, inplace=True)

In [56]: d1

Out[56]:

	국어	수학
철수	84	78
영이	21	84
길동	87	76
미영	100	99
순이	59	59
철이	46	56

## DataFrame - 행, 열 추가

### ■ 행 추가

- DataFrame객체.loc[새index명] = 데이터
- 예시 - 합계 추가

In [69]: d1

Out [69]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [75]: d1.loc['합계'] = d1.sum(axis=0)

In [76]: d1

Out [76]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56
합계	397	449	452

## DataFrame - 행, 열 추가

### ■ 열 추가

- DataFrame객체[새column명] = 데이터
- 예시 - 합계 추가

In [77]: d1

Out [77]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56
합계	397	449	452



In [78]: d1['합계'] = d1.sum(axis=1)

In [79]: d1

Out [79]:

	국어	영어	수학	합계
철수	84	87	78	249
영이	21	15	84	120
길동	87	84	76	247
미영	100	87	99	286
순이	59	99	59	217
철이	46	77	56	179
합계	397	449	452	1298



## DataFrame - 값 수정

### ■ 행의 값 수정

- DataFrame객체.loc[index명] = 데이터
- 예시

In [89]: d1

Out[89]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [90]: d1.loc['철수'] = [80, 90, 80]

In [91]: d1

Out[91]:

	국어	영어	수학
철수	80	90	80
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## DataFrame - 값 수정

### ■ 행의 값 수정

- DataFrame객체.iloc[index번호] = 데이터
- 예시

In [91]: d1

Out[91]:

	국어	영어	수학
철수	80	90	80
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [92]: d1.iloc[0] = [84, 87, 78]

In [93]: d1

Out[93]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## DataFrame - 값 수정

### ■ 열의 값 수정

- DataFrame객체[column명] = 데이터
- 예시

In [93]: d1

Out[93]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [94]: d1['국어'] = [80, 30, 90, 100, 60, 50]

In [95]: d1

Out[95]:

	국어	영어	수학
철수	80	87	78
영이	30	15	84
길동	90	84	76
미영	100	87	99
순이	60	99	59
철이	50	77	56

- DataFrame을 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```

학생들의 이름 입력 (,로 구분) : 철수,영희,미영
학생들의 국어성적 입력 (,로 구분) : 90,80,80
학생들의 영어성적 입력 (,로 구분) : 80,80,90
학생들의 수학성적 입력 (,로 구분) : 70,89,88
      국어  영어  수학  합계
철수   90   80   70   240
영희   80   80   89   249
미영   80   90   88   258
    
```

```
import pandas as pd

name = input('학생들의 이름 입력(,로 구분): ').split(',')
score_kor = input('학생들의 국어성적 입력(,로 구분): ').split(',')
score_eng = input('학생들의 영어성적 입력(,로 구분): ').split(',')
score_math = input('학생들의 수학성적 입력(,로 구분): ').split(',')

score_kor = [int(i) for i in score_kor]
score_eng = [int(i) for i in score_eng]
score_math = [int(i) for i in score_math]

s_kor = pd.Series(score_kor, index = name)
s_eng = pd.Series(score_eng, index = name)
s_math = pd.Series(score_math, index = name)

d = pd.DataFrame()
d['국어'] = s_kor
d['영어'] = s_eng
d['수학'] = s_math
d['합계'] = d.국어 + d.영어 + d.수학

print(d)
```

## DataFrame - 부분 정보 선택

### ■ Index, columns, values 접근

- Index: DataFrame객체.index
- Columns: DataFrame객체.columns
- Values: DataFrame객체.values
- 예시

```
In [58]: d1
```

```
Out[58]:
```

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [59]: d1.index
```

```
Out[59]: Index(['철수', '영이', '길동', '미영', '순이', '철이'], dtype='object')
```

```
In [60]: d1.columns
```

```
Out[60]: Index(['국어', '영어', '수학'], dtype='object')
```

```
In [61]: d1.values
```

```
Out[61]: array([[ 84,  87,  78],
                [ 21,  15,  84],
                [ 87,  84,  76],
                [100,  87,  99],
                [ 59,  99,  59],
                [ 46,  77,  56]], dtype=int64)
```

## DataFrame - 부분 정보 선택

### ■ []을 이용한 선택

- 열 indexing: DataFrame객체[column명] 또는 DataFrame객체.column명
- 예시

In [6]:

```
dl
```

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [7]:

```
dl['국어']
```

Out [7]:

```
철수    84
영이     21
길동    87
미영   100
순이     59
철이     46
Name: 국어, dtype: int64
```

In [8]:

```
dl.수학
```

Out [8]:

```
철수     78
영이     84
길동     76
미영     99
순이     59
철이     56
Name: 수학, dtype: int64
```

## DataFrame - 부분 정보 선택

### ■ []을 이용한 선택

- 행 slicing: DataFrame객체[행번호:행번호] 또는 DataFrame객체[index명:index명]
- 예시

In [6]: dl

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [9]: dl[1:3]

Out [9]:

	국어	영어	수학
영이	21	15	84
길동	87	84	76

In [10]: dl['길동':'순이']

Out [10]:

	국어	영어	수학
길동	87	84	76
미영	100	87	99
순이	59	99	59



## DataFrame - 부분 정보 선택

### ■ []을 이용한 선택

#### • 열과 행 같이 선택

- ✓ DataFrame객체[column명][행번호:행번호]
- ✓ DataFrame객체[column명][index명:index명]
- ✓ DataFrame객체.column명[행번호:행번호]
- ✓ DataFrame객체.column명[index명 : index명]

#### • 예시

In [6]:

```
dl
```

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [12]:

```
dl['국어'][1:3]
```

Out [12]:

영이	21
길동	87

Name: 국어, dtype: int64

In [13]:

```
dl['수학']['철수':'길동']
```

Out [13]:

철수	78
영이	84
길동	76

Name: 수학, dtype: int64

In [14]:

```
dl.영어[1:3]
```

Out [14]:

영이	15
길동	84

Name: 영어, dtype: int64

In [15]:

```
dl.수학['철수':'길동']
```

Out [15]:

철수	78
영이	84
길동	76

Name: 수학, dtype: int64

## DataFrame - 부분 정보 선택

- loc를 이용한 행 선택
  - index명 사용
  - 열 추가 시 comma 사용
  - 열 추가 시 column명 사용
  - 예시

In [6]: d1

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [17]: d1.loc['철수', '국어': '수학']

Out [17]:  
국어 84  
영어 87  
수학 78  
Name: 철수, dtype: int64

In [18]: d1.loc['영이']

Out [18]:  
국어 21  
영어 15  
수학 84  
Name: 영이, dtype: int64

In [19]: d1.loc['영이': '순이', '영어']

Out [19]:  
영이 15  
길동 84  
미영 87  
순이 99  
Name: 영어, dtype: int64

## DataFrame - 부분 정보 선택

- iloc를 이용한 행 선택
  - 번호 사용
  - 열 추가 시 comma 사용
  - 예시

```
In [6]: dl
```

```
Out [6]:
```

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [21]: dl.iloc[0]
```

```
Out [21]: 국어    84  
          영어    87  
          수학    78  
          Name: 철수, dtype: int64
```

```
In [22]: dl.iloc[0,1]
```

```
Out [22]: 87
```

```
In [23]: dl.iloc[0, 1:3]
```

```
Out [23]: 영어    87  
          수학    78  
          Name: 철수, dtype: int64
```

```
In [24]: dl.iloc[1:3]
```

```
Out [24]:
```

	국어	영어	수학
영이	21	15	84
길동	87	84	76

```
In [25]: dl.iloc[[1,2]]
```

```
Out [25]:
```

	국어	영어	수학
영이	21	15	84
길동	87	84	76

```
In [26]: dl.iloc[1, [0,1]]
```

```
Out [26]: 국어    21  
          영어    15  
          Name: 영이, dtype: int64
```

# DataFrame - 산술/논리 연산 및 filtering

## ■ 산술 연산

- Array, series와 비슷
- 예시

```
In [6]: dl
```

```
Out [6]:
```

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [27]: dl.iloc[0] + dl.iloc[1]
```

```
Out [27]: 국어    105  
          영어    102  
          수학    162  
          dtype: int64
```

```
In [28]: dl.수학 + dl.영어
```

```
Out [28]: 철수    165  
          영이     99  
          길동    160  
          미영    186  
          순이    158  
          철이    133  
          dtype: int64
```

```
In [29]: dl.sum()
```

```
Out [29]: 국어    397  
          영어    449  
          수학    452  
          dtype: int64
```

```
In [30]: dl.sum(axis=1)
```

```
Out [30]: 철수    249  
          영이    120  
          길동    247  
          미영    286  
          순이    217  
          철이    179  
          dtype: int64
```

# DataFrame - 산술/논리 연산 및 filtering

## ■ 논리 연산과 filtering

- Array, series와 비슷
- 예시

In [6]: d1

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [32]: x = d1.수학 > 60

In [33]: x

Out [33]:

철수	True
영이	True
길동	True
미영	True
순이	False
철이	False

Name: 수학, dtype: bool

In [34]: d1[x]

Out [34]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99

In [38]: x = d1 > 80

In [39]: x

Out [39]:

	국어	영어	수학
철수	True	True	False
영이	False	False	True
길동	True	True	False
미영	True	True	True
순이	False	True	False
철이	False	False	False

In [40]: d1[x]

Out [40]:

	국어	영어	수학
철수	84.0	87.0	NaN
영이	NaN	NaN	84.0
길동	87.0	84.0	NaN
미영	100.0	87.0	99.0
순이	NaN	99.0	NaN
철이	NaN	NaN	NaN

## DataFrame - 정렬

### ■ 열 기준 정렬하기

- index 기준: DataFrame객체.sort\_index(ascending = True/False)
- 특정 열 기준: DataFrame객체.sort\_values(by = 'column명', ascending = True/False)
- 예시

In [6]:

dl

Out [6]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [41]:

dl.sort\_index()

Out [41]:

	국어	영어	수학
길동	87	84	76
미영	100	87	99
순이	59	99	59
영이	21	15	84
철수	84	87	78
철이	46	77	56

In [42]:

dl.sort\_values(by='국어', ascending=False)

Out [42]:

	국어	영어	수학
미영	100	87	99
길동	87	84	76
철수	84	87	78
순이	59	99	59
철이	46	77	56
영이	21	15	84

- DataFrame을 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```
학생들의 이름 입력(,로 구분): 영희,철수,미미
학생들의 국어성적 입력(,로 구분): 30,50,80
학생들의 영어성적 입력(,로 구분): 40,50,70
학생들의 수학성적 입력(,로 구분): 30,60,90
      국어  영어  수학  합계
영희   30   40   30   100
철수   50   50   60   160
미미   80   70   90   240
평균   53   53   60   166
검색할 학생 이름은? 철수
국어        50
영어        50
수학        60
합계       160
Name: 철수, dtype: int64
```

```
import pandas as pd

names = input('학생들의 이름 입력(,로 구분): ').split(',')
score_kor = input('학생들의 국어성적 입력(,로 구분): ').split(',')
score_eng = input('학생들의 영어성적 입력(,로 구분): ').split(',')
score_math = input('학생들의 수학성적 입력(,로 구분): ').split(',')

score_kor = [int(i) for i in score_kor]
score_eng = [int(i) for i in score_eng]
score_math = [int(i) for i in score_math]

s_kor = pd.Series(score_kor, index = names)
s_eng = pd.Series(score_eng, index = names)
s_math = pd.Series(score_math, index = names)

d = pd.DataFrame()
d['국어'] = s_kor
d['영어'] = s_eng
d['수학'] = s_math
d['합계'] = d.sum(axis=1)
d.loc['평균'] = d.sum() // len(names)

print(d)

name = input('검색할 학생 이름은? ')
print(d.loc[name])
```



- DataFrame을 이용하여 다음과 같이 실행되는 프로그램을 작성하시오.

실행결과(밑줄 친 부분은 키보드로 입력 받는 부분)

```

학생들의 이름 입력(,로 구분): 철수,영미,길동
학생들의 국어성적 입력(,로 구분): 50,70,90
학생들의 영어성적 입력(,로 구분): 80,75,80
학생들의 수학성적 입력(,로 구분): 95,85,90
    국어  영어  수학  합계
철수  50   80   95   225
영미  70   75   85   230
길동  90   80   90   260
정렬 기준 선택(1:국어, 2:영어, 3:수학, 4:합계): 4
정렬 방법 선택(1:오름차순, 2:내림차순): 2
    국어  영어  수학  합계
길동  90   80   90   260
영미  70   75   85   230
철수  50   80   95   225
    
```

## 실습 - 답안

```
import pandas as pd

name = input('학생들의 이름 입력(,로 구분): ').split(',')
score_kor = input('학생들의 국어성적 입력(,로 구분): ').split(',')
score_eng = input('학생들의 영어성적 입력(,로 구분): ').split(',')
score_math = input('학생들의 수학성적 입력(,로 구분): ').split(',')

score_kor = [int(i) for i in score_kor]
score_eng = [int(i) for i in score_eng]
score_math = [int(i) for i in score_math]

s_kor = pd.Series(score_kor, index = name)
s_eng = pd.Series(score_eng, index = name)
s_math = pd.Series(score_math, index = name)

d = pd.DataFrame()
d['국어'] = s_kor
d['영어'] = s_eng
d['수학'] = s_math
d['합계'] = d.국어 + d.영어 + d.수학

print(d)
```

```
sort_no = input('정렬 기준 선택(1:국어, 2:영어, 3:수학, 4:합계): ')
sort_type = input('정렬 방법 선택(1:오름차순, 2:내림차순): ')

if sort_no == '1':
    if sort_type == '1':
        d_new = d.sort_values(by = '국어', ascending = True)
    else:
        d_new = d.sort_values(by = '국어', ascending = False)
elif sort_no == '2':
    if sort_type == '1':
        d_new = d.sort_values(by = '영어', ascending = True)
    else:
        d_new = d.sort_values(by = '영어', ascending = False)
elif sort_no == '3':
    if sort_type == '1':
        d_new = d.sort_values(by = '수학', ascending = True)
    else:
        d_new = d.sort_values(by = '수학', ascending = False)
elif sort_no == '4':
    if sort_type == '1':
        d_new = d.sort_values(by = '합계', ascending = True)
    else:
        d_new = d.sort_values(by = '합계', ascending = False)

print(d_new)
```

- Pandas의 데이터 입출력

파일 형식	읽기	쓰기
MS Excel	read_excel	to_excel
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
Local clipboard	read_clipboard	to_clipboard
HDF5 Format	read_hdf	to_hdf
SQL	read_sql	to_sql

## ■ 엑셀 파일로 저장하기

- DataFrame객체.to\_excel('경로/파일명', encoding = '코딩방식')
  - ✓ 한글 encoding 방식: encoding = 코딩방식(utf-8이나 euc-kr 사용)
- 예시

```
In [1]: import pandas as pd
```

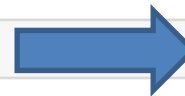
```
In [2]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [3]: lectures = ['국어', '영어', '수학']
```

```
In [4]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [5]: dl = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [6]: dl.to_excel("c:/data/scores.xls", encoding = 'utf-8')
```



	A	B	C	D
1		국어	영어	수학
2	철수	84	87	78
3	영이	21	15	84
4	길동	87	84	76
5	미영	100	87	99
6	순이	59	99	59
7	철이	46	77	56

<c:/data/scores.xls>

## ■ 엑셀 파일 읽어오기

- DataFrame 객체 = `pd.read_excel('경로/파일명', sheet_name='시트명', encoding = '코딩방식')`
  - ✓ 시트명을 넣어주지 않으면 1번째 시트에서 읽어옴
  - ✓ index 열 지정: `index_col = 열번호`
- 예시 - index 열 지정하지 않은 경우

	A	B	C	D
1		국어	영어	수학
2	철수	84	87	78
3	영이	21	15	84
4	길동	87	84	76
5	미영	100	87	99
6	순이	59	99	59
7	철이	46	77	56



<c:/data/scores.xls>

```
In [1]: import pandas as pd
```

```
In [2]: d2 = pd.read_excel('c:/data/scores.xls')
```

```
In [3]: d2
```

Out [3]:

Unnamed: 0	국어	영어	수학	
0	철수	84	87	78
1	영이	21	15	84
2	길동	87	84	76
3	미영	100	87	99
4	순이	59	99	59
5	철이	46	77	56

## ■ 엑셀 파일 읽어오기

- DataFrame 객체 = `pd.read_excel('경로/파일명', sheet_name='시트명', encoding = '코딩방식')`
  - ✓ 시트명을 넣어주지 않으면 1번째 시트에서 읽어옴
  - ✓ index열 지정: `index_col = 열번호`
- 예시 - index열 지정한 경우

	A	B	C	D
1		국어	영어	수학
2	철수	84	87	78
3	영이	21	15	84
4	길동	87	84	76
5	미영	100	87	99
6	순이	59	99	59
7	철이	46	77	56

<c:/data/scores.xls>



```
In [4]: d3 = pd.read_excel('c:/data/scores.xls', index_col=0)
```

```
In [5]: d3
```

Out [5]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## ■ CSV(comma-separated values) 파일로 저장하기

- 줄바꿈으로 행, comma 등으로 열을 구분하여 데이터를 저장하는 파일 형식
- DataFrame객체.to\_csv('경로/파일명.csv', encoding = '코딩방식')
  - ✓ 한글 encoding 방식은 euc-kr 권장
- 예시

```
In [6]: import pandas as pd
```

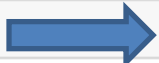
```
In [7]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [8]: lectures = ['국어', '영어', '수학']
```

```
In [9]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [10]: d1 = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [11]: d1.to_csv('c:/data/scores.csv')
```



```
,국어,영어,수학  
철수,84,87,78  
영이,21,15,84  
길동,87,84,76  
미영,100,87,99  
순이,59,99,59  
철이,46,77,56
```

<c:/data/scores.csv>

## ■ CSV 파일 읽어오기

- DataFrame 객체 = `pd.read_csv('경로/파일명', encoding = '코딩방식')`
  - ✓ index 열 지정: `index_col = 열번호`
  - ✓ comma(,) 외의 구별 문자 지정: `sep = 구별문자(보통 '\t' 탭)`
- 예시 - index 열 지정하지 않은 경우

```
,국어,영어,수학
철수,84,87,78
영이,21,15,84
길동,87,84,76
미영,100,87,99
순이,59,99,59
철이,46,77,56
```

<c:/data/scores.csv>



```
In [12]: d2 = pd.read_csv('c:/data/scores.csv')
```

```
In [13]: d2
```

Out [13]:

	Unnamed: 0	국어	영어	수학
0	철수	84	87	78
1	영이	21	15	84
2	길동	87	84	76
3	미영	100	87	99
4	순이	59	99	59
5	철이	46	77	56



## ■ CSV 파일 읽어오기

- DataFrame 객체 = `pd.read_csv('경로/파일명', encoding = '코딩방식')`
  - ✓ index 열 지정: `index_col = 열번호`
  - ✓ comma(,) 외의 구별 문자 지정: `sep = 구별문자(보통 '\t' 탭)`
- 예시 - index 열 지정한 경우

```
,국어,영어,수학
철수,84,87,78
영이,21,15,84
길동,87,84,76
미영,100,87,99
순이,59,99,59
철이,46,77,56
```

<c:/data/scores.csv>



```
In [14]: d3 = pd.read_csv('c:/data/scores.csv', index_col=0)
```

```
In [15]: d3
```

Out [15]:

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

## ■ JSON 파일로 저장하기

- DataFrame객체.to\_json('경로/파일명')
- 예시

```
In [6]: import pandas as pd
```

```
In [21]: names = ['stu1', 'stu2', 'stu3', 'stu4', 'stu5', 'stu6']
```

```
In [22]: lectures = ['kor', 'eng', 'math']
```

```
In [23]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [24]: d1 = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [26]: d1.to_json('c:/data/scores.json')
```



```
{  
  "kor":{"stu1":84,"stu2":21,"stu3":87,"stu4":100,"stu5":59,"stu6":46},  
  "eng":{"stu1":87,"stu2":15,"stu3":84,"stu4":87,"stu5":99,"stu6":77},  
  "math":{"stu1":78,"stu2":84,"stu3":76,"stu4":99,"stu5":59,"stu6":56}  
}
```

<c:/data/scores.json>

### ■ JSON 파일 읽어오기

- DataFrame 객체 = `pd.read_json('경로/파일명')`
- 예시

```
{  
  "kor":{"stu1":84,"stu2":21,"stu3":87,"stu4":100,"stu5":59,"stu6":46},  
  "eng":{"stu1":87,"stu2":15,"stu3":84,"stu4":87,"stu5":99,"stu6":77},  
  "math":{"stu1":78,"stu2":84,"stu3":76,"stu4":99,"stu5":59,"stu6":56}  
}
```

<c:/data/scores.json>



```
In [27]: d2 = pd.read_json('c:/data/scores.json')
```

```
In [28]: d2
```

```
Out [28]:
```

	kor	eng	math
stu1	84	87	78
stu2	21	15	84
stu3	87	84	76
stu4	100	87	99
stu5	59	99	59
stu6	46	77	56

## ■ HTML 파일로 저장하기

- DataFrame은 HTML 페이지에 표 형식(<table>)으로 저장됨
- DataFrame객체.to\_html('경로/파일명')
- 예시

```
In [6]: import pandas as pd
```

```
In [21]: names = ['stu1', 'stu2', 'stu3', 'stu4', 'stu5', 'stu6']
```

```
In [22]: lectures = ['kor', 'eng', 'math']
```

```
In [23]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [30]: df = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [31]: df.to_html('c:/data/scores.html')
```



<c:/data/scores.html>

```
<table border="1" class="dataframe">
<thead>
  <tr style="text-align: right;">
    <th></th>
    <th>kor</th>
    <th>eng</th>
    <th>math</th>
  </tr>
</thead>
```

```
<tbody>
  <tr>
    <th>stu1</th>
    <td>84</td>
    <td>87</td>
    <td>78</td>
  </tr>
  <tr>
    <th>stu2</th>
    <td>21</td>
    <td>15</td>
    <td>84</td>
  </tr>
  ...
</tbody>
</table>
```

## ■ HTML 파일 읽어오기

- HTML 웹 페이지에 있는 <table> 태그에서 표 형식의 데이터를 읽어옴
- DataFrame객체리스트 = pd.read\_html('경로/파일명', encoding = '코딩방식')
- DataFrame객체리스트 = pd.read\_html('웹주소(url)', encoding = '코딩방식')
- 예시

	kor	eng	math
stu1	84	87	78
stu2	21	15	84
stu3	87	84	76
stu4	100	87	99
stu5	59	99	59
stu6	46	77	56
	국어	영어	수학
철수	84	87	78
영희	21	15	84
미영	87	84	76



<c:/data/scores.html>

```
In [38]: tables = pd.read_html('c:/data/scores.html', index_col = 0, encoding='euc-kr')
```

```
In [39]: print(len(tables))
```

2

```
In [40]: for i in range(len(tables)):
          print("tables[%d]" % i)
          print(tables[i])
          print()
```

```
tables[0]
      kor  eng  math
stu1   84   87   78
stu2   21   15   84
stu3   87   84   76
stu4  100   87   99
stu5   59   99   59
stu6   46   77   56
```

```
tables[1]
      국어  영어  수학
철수   84   87   78
영희   21   15   84
미영   87   84   76
```

## ■ 데이터베이스에 저장하기: SQLite DB 사용

- DataFrame객체.to\_sql(name, con, flavor='sqlite', schema=None, if\_exists='fail', index=True, index\_label=None, chunksize=None, dtype=None)
- 예시

```
import sqlite3
```

```
import pandas as pd
```

```
con = sqlite3.connect("test.db")
```

```
names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
lectures = ['국어', '영어', '수학']
```

```
scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
d1 = pd.DataFrame(scores, index = names, columns = lectures)
```

```
d1.to_sql('scores', con, index_label='이름')
```

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

### ■ 데이터베이스에서 읽어오기: SQLite DB 사용

- DataFrame 객체 = `pd.read_sql('쿼리', con)`
- 예시

```
import sqlite3
```

```
import pandas as pd
```

```
con = sqlite3.connect("test.db")
```

```
d2 = pd.read_sql('SELECT * FROM scores', con)
```

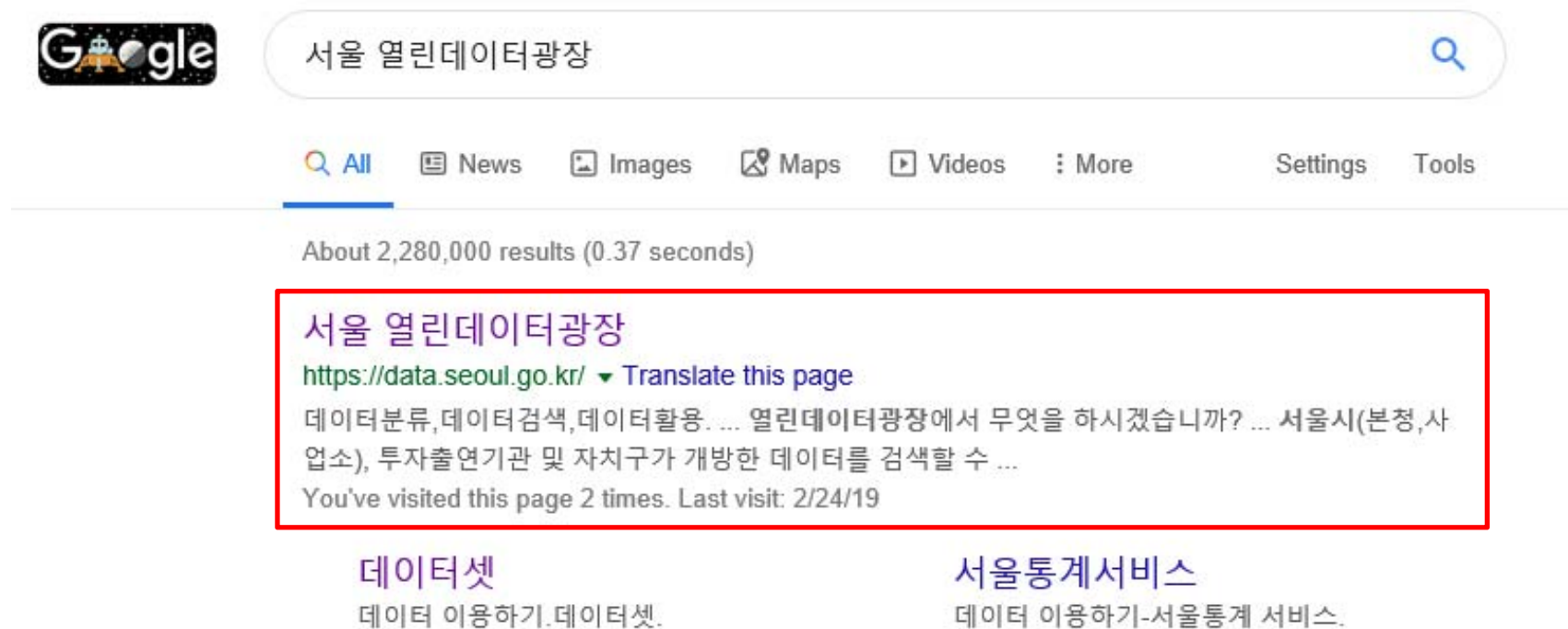
```
d2
```

	이름	국어	영어	수학
0	철수	84	87	78
1	영이	21	15	84
2	길동	87	84	76
3	미영	100	87	99
4	순이	59	99	59
5	철이	46	77	56

## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

### ■ 데이터 취득

- ✓ 서울시열린데이터광장 (<https://data.seoul.go.kr>) => 서울시 주민등록인구 (구별) 통계

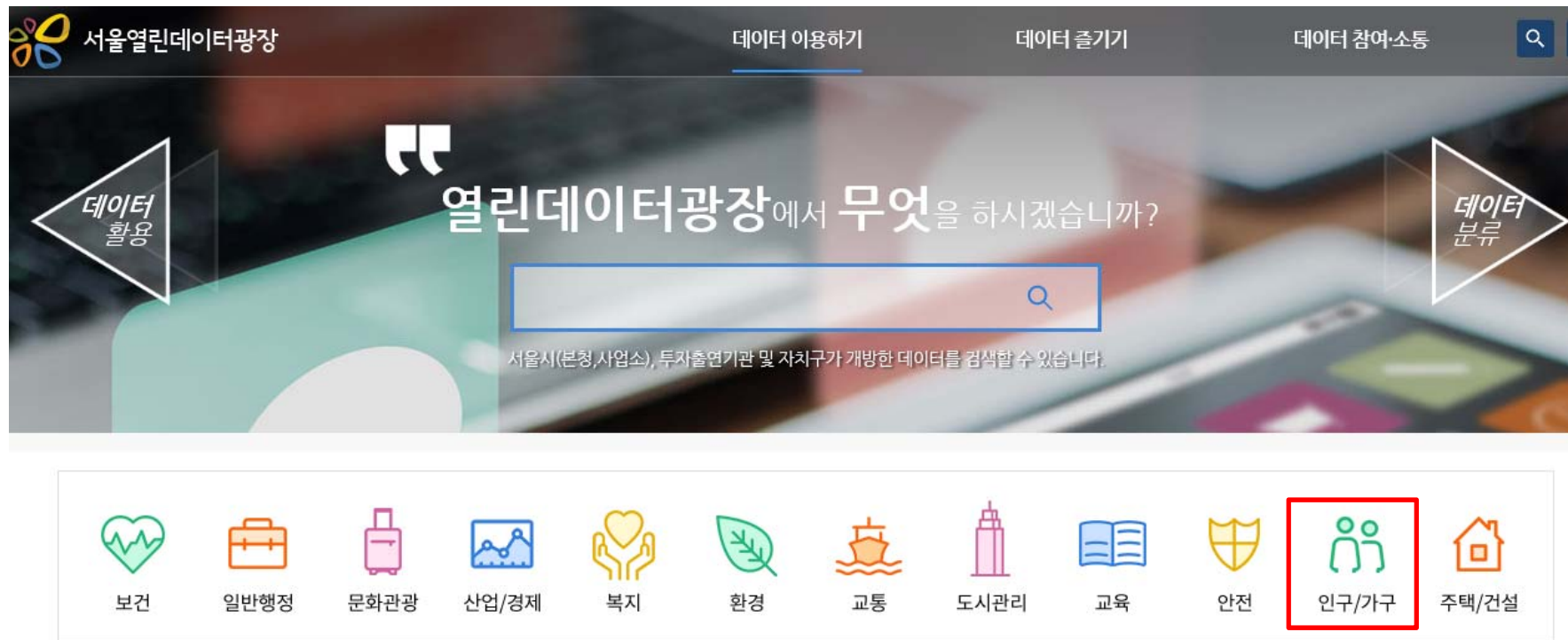




## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

### ■ 데이터 취득

- 서울시열린데이터광장 (<https://data.seoul.go.kr>) => 서울시 주민등록인구 (구별) 통계



# 실습 - 공공데이터 분석: 구별 인구 데이터 분석

## ■ 데이터 취득

- 서울시열린데이터광장 (<https://data.seoul.go.kr>) => 서울시 주민등록인구 (구별) 통계

The screenshot shows the Seoul Open Data Platform interface. At the top, there's a navigation bar with '데이터 이용하기' (Use Data), '데이터 즐기기' (Enjoy Data), and '데이터 참여소통' (Participate and Communicate with Data). Below this is a '데이터목록' (Data List) section with a grid of categories. The '인구/가구' (Population/Household) category is highlighted. A search bar on the right contains the text '검색어' (Search term) and a search button. Below the search bar, it says '234건이 검색되었습니다.' (234 items were searched). A table of results is displayed, with the first result highlighted in red. This result is for '[인구/가구] 서울시 주민등록인구 (구별) 통계' (Population/Household Seoul Resident Population (by District) Statistics). It includes a '통계' (Statistics) tab and a 'SHEET' button. The description states: '서울시 통계정보 조회수 : 256276 서울시 통계정보시스템에서 제공하는 주민등록인구(구별)에 대한 통계정보입니다. 주민등록인구수를 자치구...' (Seoul City Statistics Information Search Count: 256,276. This is statistical information about the resident population (by district) provided by the Seoul City Statistics Information System. It is the resident population count by district...

전체	보건	일반행정	문화/관광	산업/경제	복지	환경	가구주이 가족인식 가족일반 가족형성 결혼인식 사회인식 아동 여성권익 여성일반 외국인인구 인구구조 인구이동 인구주이 주민등록인구 청소년 출생사망
	교통	도시관리	교육	안전	인구/가구	주택/건설	

Sheet Open API Chart Map File Link LOD

제공기관을 선택하세요 검색어

결과내 검색

234건이 검색되었습니다.

전체 제목순 최신순 조회순 15개씩

**[인구/가구] 서울시 주민등록인구 (구별) 통계**

통계 SHEET OPEN API CHART

서울시 통계정보  
조회수 : 256276  
서울시 통계정보시스템에서 제공하는 주민등록인구(구별)에 대한 통계정보입니다. 주민등록인구수를 자치구...

[인구/가구] 서울시 주민등록인구 (동별) 통계

통계 SHEET OPEN API CHART

서울시 통계정보  
조회수 : 106856  
서울시 통계정보시스템에서 제공하는 주민등록인구(동별)에 대한 통계정보입니다. 주민등록인구수를 동 단...

[인구/가구] 서울시 고령자현황 (동별) 통계

통계 SHEET OPEN API CHART

서울시 통계정보  
조회수 : 65485  
서울시 통계정보시스템에서 제공하는 고령자현황(동별)에 대한 통계정보입니다. 서울시 전체인구중 고령인...

# 실습 - 공공데이터 분석: 구별 인구 데이터 분석

## ■ 데이터 취득

- 서울시 열린데이터광장 (<https://data.seoul.go.kr>) => 서울시 주민등록인구 (구별) 통계



통계 서울시 주민등록인구 (구별) 통계 ★★★★★



평가

활용갤러리등록

URL복사

상세정보 닫기

목록

분류 인구/가구	원본시스템 <b>바도가기</b>	저작권자 서울특별시 스마트도시정책관 빅데이터담당관 (02-2133-4285)	제공기관 서울특별시	제3저작권자 없음
담당자 황재일 (02-2133-4285)	원본형태 DB	데이터공개일자 2017.12.26	갱신주기 정기(분기, 3, 6, 9, 12 기준)	제공부서 스마트도시정책관 빅데이터담당관
태그 세대, 세대당인구, 외국인, 인구밀도, 한국인, 인구, 등록외국인			이용허락조건 저작권자표시(BY) 이용이나 변경 및 2차적 저작물의 작성을 포함한 자유이용을 허락합니다.	

\* 통계명 : 주민등록인구(구별)

\* 통계종류 : 주민등록인구수를 자치구별로 제공하는 일반·보고통계

\* 근거법령 : 「주민등록법」에 따른 주민등록표에 등록된 서울시민 및 세대와 「출입국관리법」에 따른 외국인등록표에 등록된 서울시 거주 외국인

c:/data/pop\_in\_seoul.xls  
로 저장

Sheet

Chart

Open Api

자료분석

**XLS**

CSV

HWP

언어

한국어

소수점

기간

분기

2019.1/4 분기

~ 2019.1/4 분기

자료검색

# 실습 - 공공데이터 분석: 구별 인구 데이터 분석

## ■ 데이터 읽기

```
import pandas as pd
```

```
pop_seoul = pd.read_excel('c:/data/pop_in_seoul.xls')
```

```
pop_seoul
```

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6	인구.7	인구.8	세대당인구	65세이상고령자
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인	등록외국인	등록외국인	세대당인구	65세이상고령자
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계	남자	여자	세대당인구	65세이상고령자
2	2019.1/4	합계	4290922	10054979	4909387	5145592	9770216	4772134	4998082	284763	137253	147510	2.28	1436125
3	2019.1/4	종로구	73914	162913	78963	83950	152778	74536	78242	10135	4427	5708	2.07	26981
4	2019.1/4	중구	61800	135836	66720	69116	125942	61992	63950	9894	4728	5166	2.04	22421
5	2019.1/4	용산구	109413	245139	119597	125542	229168	110626	118542	15971	8971	7000	2.09	38049
6	2019.1/4	성동구	137247	314608	154011	160597	306404	150287	156117	8204	3724	4480	2.23	43076
7	2019.1/4	광진구	163460	370658	179162	191496	354873	172361	182512	15785	6801	8984	2.17	46288
8	2019.1/4	동대문구	162228	363262	179100	184162	346750	172784	173966	16512	6316	10196	2.14	57570
9	2019.1/4	종랑구	181182	407211	201808	205403	402203	199730	202473	5008	2078	2930	2.22	62789
10	2019.1/4	성북구	188670	450021	217400	232621	438245	212830	225415	11776	4570	7206	2.32	68612
11	2019.1/4	강북구	143663	321151	156525	164626	317386	155075	162311	3765	1450	2315	2.21	58858

### ■ 데이터 읽기

- 특정 열 가져오고, column 설정

```
pop_seoul = pd.read_excel('c:/data/pop_in_seoul.xls', usecols=[1,3,6,9,13], header=2)
```

pop\_seoul

	자치구	계	계.1	계.2	65세이상고령자
0	합계	10054979	9770216	284763	1436125
1	종로구	162913	152778	10135	26981
2	중구	135836	125942	9894	22421
3	용산구	245139	229168	15971	38049
4	성동구	314608	306404	8204	43076
5	광진구	370658	354873	15785	46288
6	동대문구	363262	346750	16512	57570
7	종랑구	407211	402203	5008	62789
8	성북구	450021	438245	11776	68612
9	강북구	321151	317386	3765	58858
10	도봉구	340089	337820	2269	56742

### ■ 데이터 읽기

- column명 변경

```
pop_seoul.columns = ['구별', '인구수', '한국인', '외국인', '고령자']
```

```
pop_seoul
```

	구별	인구수	한국인	외국인	고령자
0	합계	10054979	9770216	284763	1436125
1	종로구	162913	152778	10135	26981
2	중구	135836	125942	9894	22421
3	용산구	245139	229168	15971	38049
4	성동구	314608	306404	8204	43076
5	광진구	370658	354873	15785	46288
6	동대문구	363262	346750	16512	57570
7	중랑구	407211	402203	5008	62789
8	성북구	450021	438245	11776	68612

## ■ 데이터 읽기

- index 설정

```
pop_seoul.set_index('구별', inplace=True)
```

```
pop_seoul
```

	인구수	한국인	외국인	고령자
구별				
합계	10054979	9770216	284763	1436125
종로구	162913	152778	10135	26981
중구	135836	125942	9894	22421
용산구	245139	229168	15971	38049
성동구	314608	306404	8204	43076
광진구	370658	354873	15785	46288
동대문구	363262	346750	16512	57570
중랑구	407211	402203	5008	62789
성북구	450021	438245	11776	68612
강북구	321151	317386	3765	58858

- 데이터 읽기
  - 합계 데이터(1번째 행) 삭제

```
pop_seoul.drop('합계', inplace=True)
```

```
pop_seoul.head()
```

	인구수	한국인	외국인	고령자
구별				
종로구	162913	152778	10135	26981
중구	135836	125942	9894	22421
용산구	245139	229168	15971	38049
성동구	314608	306404	8204	43076
광진구	370658	354873	15785	46288



- 데이터 분석하기
  - 한국인 비율 구하기

```
pop_seoul['한국인비율(%)'] = (pop_seoul.한국인 / pop_seoul.인구수) * 100
```

```
pop_seoul
```

	인구수	한국인	외국인	고령자	한국인비율(%)
구별					
종로구	162913	152778	10135	26981	93.778888
중구	135836	125942	9894	22421	92.716217
용산구	245139	229168	15971	38049	93.484921
성동구	314608	306404	8204	43076	97.392310
광진구	370658	354873	15785	46288	95.741357
동대문구	363262	346750	16512	57570	95.454520
중랑구	407211	402203	5008	62789	98.770171
성북구	450021	438245	11776	68612	97.383233

- 데이터 분석하기
  - 외국인 비율 구하기

```
pop_seoul['외국인비율(%)'] = (pop_seoul.외국인 / pop_seoul.인구수) * 100
```

```
pop_seoul
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)
구별						
종로구	162913	152778	10135	26981	93.778888	<a href="#">6.221112</a>
중구	135836	125942	9894	22421	92.716217	7.283783
용산구	245139	229168	15971	38049	93.484921	6.515079
성동구	314608	306404	8204	43076	97.392310	<a href="#">2.607690</a>
광진구	370658	354873	15785	46288	95.741357	<a href="#">4.258643</a>
동대문구	363262	346750	16512	57570	95.454520	4.545480
중랑구	407211	402203	5008	62789	98.770171	1.229829
성북구	450021	438245	11776	68612	97.383233	<a href="#">2.616767</a>

## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

- 데이터 분석하기
  - 고령자 비율 구하기

```
pop_seoul['고령자비율(%)'] = (pop_seoul.고령자 / pop_seoul.인구수) * 100
```

```
pop_seoul
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
종로구	162913	152778	10135	26981	93.778888	<a href="#">6.221112</a>	<a href="#">16.561600</a>
중구	135836	125942	9894	22421	92.716217	7.283783	<a href="#">16.505934</a>
용산구	245139	229168	15971	38049	93.484921	6.515079	15.521398
성동구	314608	306404	8204	43076	97.392310	<a href="#">2.607690</a>	13.691960
광진구	370658	354873	15785	46288	95.741357	<a href="#">4.258643</a>	12.488062
동대문구	363262	346750	16512	57570	95.454520	4.545480	15.848066
중랑구	407211	402203	5008	62789	98.770171	1.229829	15.419279
성북구	450021	438245	11776	68612	97.383233	<a href="#">2.616767</a>	15.246400

## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

- 데이터 분석하기
  - 인구수 합계 내림차순 정렬하기

```
pop_seoul.sort_values(by = '인구수', ascending = False)
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
송파구	685361	678521	6840	83492	99.001986	0.998014	12.182193
강서구	602886	596287	6599	80903	98.905432	<a href="#">1.094568</a>	13.419287
강남구	546875	541854	5021	68104	99.081874	0.918126	12.453303
노원구	545486	541174	4312	78170	99.209512	0.790488	14.330340
관악구	520645	502615	18030	73005	96.536988	3.463012	14.022030
은평구	488713	484274	4439	78406	99.091696	0.908304	<a href="#">16.043363</a>
양천구	466622	462599	4023	58930	99.137846	0.862154	12.629066
성북구	450021	438245	11776	68612	97.383233	<a href="#">2.616767</a>	15.246400
구로구	438889	404726	34163	63017	92.216027	7.783973	14.358300

## ■ 데이터 분석하기

- 인구수가 적은 순서대로 5개 구 출력하기

```
sorted_pop = pop_seoul.sort_values(by = '인구수', ascending = True)
```

```
sorted_pop.head()
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
중구	135836	125942	9894	22421	92.716217	7.283783	<a href="#">16.505934</a>
종로구	162913	152778	10135	26981	93.778888	<a href="#">6.221112</a>	<a href="#">16.561600</a>
용산구	245139	229168	15971	38049	93.484921	6.515079	15.521398
금천구	254244	233981	20263	36301	92.030097	7.969903	14.278016
성동구	314608	306404	8204	43076	97.392310	<a href="#">2.607690</a>	13.691960

- 데이터 분석하기
  - 외국인수 내림차순 정렬하기

```
pop_seoul.sort_values(by = '외국인', ascending = False)
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
영등포구	404556	368824	35732	56463	91.167601	8.832399	13.956782
구로구	438889	404726	34163	63017	92.216027	7.783973	14.358300
금천구	254244	233981	20263	36301	92.030097	7.969903	14.278016
관악구	520645	502615	18030	73005	96.536988	3.463012	14.022030
동대문구	363262	346750	16512	57570	95.454520	4.545480	15.848066
용산구	245139	229168	15971	38049	93.484921	6.515079	15.521398
광진구	370658	354873	15785	46288	95.741357	<a href="#">4.258643</a>	12.488062
동작구	412031	398886	13145	60462	96.809706	<a href="#">3.190294</a>	14.674139

## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

- 데이터 분석하기
  - 외국인 비율 내림차순 정렬하기

```
pop_seoul.sort_values(by = '외국인비율(%)', ascending = False)
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
영등포구	404556	368824	35732	56463	91.167601	8.832399	13.956782
금천구	254244	233981	20263	36301	92.030097	7.969903	14.278016
구로구	438889	404726	34163	63017	92.216027	7.783973	14.358300
중구	135836	125942	9894	22421	92.716217	7.283783	<a href="#">16.505934</a>
용산구	245139	229168	15971	38049	93.484921	6.515079	15.521398
종로구	162913	152778	10135	26981	93.778888	<a href="#">6.221112</a>	<a href="#">16.561600</a>
동대문구	363262	346750	16512	57570	95.454520	4.545480	15.848066
광진구	370658	354873	15785	46288	95.741357	<a href="#">4.258643</a>	12.488062
서대문구	324604	311771	12833	51085	96.046568	3.953432	15.737637

## 실습 - 공공데이터 분석: 구별 인구 데이터 분석

- 데이터 분석하기
  - 고령자 비율 오름차순 정렬하기

```
pop_seoul.sort_values(by = '고령자비율(%)', ascending = True)
```

	인구수	한국인	외국인	고령자	한국인비율(%)	외국인비율(%)	고령자비율(%)
구별							
송파구	685361	678521	6840	83492	99.001986	0.998014	12.182193
강남구	546875	541854	5021	68104	99.081874	0.918126	12.453303
광진구	370658	354873	15785	46288	95.741357	<a href="#">4.258643</a>	12.488062
양천구	466622	462599	4023	58930	99.137846	0.862154	12.629066
서초구	437007	432762	4245	55366	99.028620	0.971380	12.669362
마포구	386571	375106	11465	51293	97.034180	<a href="#">2.965820</a>	13.268714
강서구	602886	596287	6599	80903	98.905432	<a href="#">1.094568</a>	13.419287
성동구	314608	306404	8204	43076	97.392310	<a href="#">2.607690</a>	13.691960
강동구	429601	425267	4334	59742	98.991157	<a href="#">1.008843</a>	13.906392



## ■ 평균값

- 모든 열의 평균값: DataFrame객체.mean()
- 특정 열의 평균값: DataFrame객체['column명'].mean()
- 예시

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [54]: import pandas as pd
```

```
In [55]: names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
In [56]: lectures = ['국어', '영어', '수학']
```

```
In [57]: scores = [[84, 87, 78], [21, 15, 84], [87, 84, 76], [100, 87, 99], [59, 99, 59], [46, 77, 56]]
```

```
In [67]: d1 = pd.DataFrame(scores, index = names, columns = lectures)
```

```
In [68]: d1.mean()
```

```
Out [68]: 국어    66.166667
          영어    74.833333
          수학    75.333333
          dtype: float64
```

```
In [69]: d1['국어'].mean()
```

```
Out [69]: 66.16666666666667
```

### ■ 중간값

- 주어진 값들을 크기의 순서대로 정렬했을 때 가장 중앙에 위치하는 값
- 값이 짝수개일 때에는 중앙에 있는 두 값의 평균
- 모든 열의 중간값: DataFrame객체.median()
- 특정 열의 중간값: DataFrame객체['column명'].median()
- 예시

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [70]: d1.median()
```

```
Out [70]: 국어    71.5  
          영어    85.5  
          수학    77.0  
          dtype: float64
```

```
In [71]: d1['영어'].median()
```

```
Out [71]: 85.5
```

### ■ 최대값

- 모든 열의 최대값: DataFrame객체.max()
- 특정 열의 최대값: DataFrame객체['column명'].max()
- 예시

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [72]: d1.max()
```

```
Out [72]: 국어    100  
          영어     99  
          수학     99  
          dtype: int64
```

```
In [73]: d1['수학'].max()
```

```
Out [73]: 99
```

### ■ 최소값

- 모든 열의 최소값: DataFrame객체.min()
- 특정 열의 최소값: DataFrame객체['column명'].min()
- 예시

dl

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



```
In [74]: dl.min()
```

```
Out[74]: 국어    21  
영어    15  
수학    56  
dtype: int64
```

```
In [77]: dl[['국어', '수학']].min()
```


```
Out[77]: 국어    21  
수학    56  
dtype: int64
```

### ■ 표준편차

- 표준 편차(standard deviation)는 분산을 제곱근한 것
  - ✓ 분산(variance)은 관측값에서 평균을 뺀 값을 제곱하고, 그것을 모두 더한 후 전체 개수로 나눠서 구한다.
- 모든 열의 표준편차: DataFrame객체.std()
- 특정 열의 표준편차 : DataFrame객체['column명']. std()
- 예시

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [78]: d1.std()

Out [78]:

국어	29.647372
영어	30.162339
수학	16.020820
dtype:	float64

In [79]: d1['국어'].std()

Out [79]: 29.64737200270315

## ■ 상관계수

- 두 변수간의 관계 강도를 나타냄
- 모든 열의 상관계수: DataFrame객체.corr()
- 특정 열의 상관계수 : DataFrame객체[column명 리스트].corr()
- 예시

dl

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56



In [80]: dl.corr()

Out [80]:

	국어	영어	수학
국어	<u>1.000000</u>	0.739217	<u>0.411671</u>
영어	0.739217	<u>1.000000</u>	<u>-0.260196</u>
수학	<u>0.411671</u>	<u>-0.260196</u>	<u>1.000000</u>

In [81]: dl[['국어', '영어']].corr()

Out [81]:

	국어	영어
국어	<u>1.000000</u>	0.739217
영어	0.739217	<u>1.000000</u>

- 데이터 분석하기
  - 평균 구하기

```
pop_seoul.mean()
```

```
인구수          402199.160000
한국인          390808.640000
외국인           11390.520000
고령자           57445.000000
한국인비율(%)    96.686855
외국인비율(%)    3.313145
고령자비율(%)    14.568931
dtype: float64
```

### ■ 데이터 분석하기

- 인구수 최대값 구하기

```
pop_seoul['인구수'].max()
```

685361

- 인구수 최소값 구하기

```
pop_seoul['인구수'].min()
```

135836



- 데이터 분석하기
  - 인구수와 고령자수 간의 상관 관계

```
pop_seoul[['인구수', '고령자']].corr()
```

	인구수	고령자
인구수	1.000000	0.942086
고령자	0.942086	1.000000

## ■ 누락 데이터 확인

- isnull() : 누락 데이터이면 True를 반환하고, 유효한 데이터가 존재하면 False를 반환
- notnull() : 유효한 데이터가 존재하면 True를 반환하고, 누락 데이터이면 False를 반환
- 예시

d1

	국어	영어	수학
철수	84	87	78
영이	21	15	84
길동	87	84	76
미영	100	87	99
순이	59	99	59
철이	46	77	56

d2

	국어	영어	수학
철수	80	80	90
영이	50	60	70
길동	80	90	80
미영	100	90	99
순이	77	95	70
슬기	55	80	65

```
d3 = d1 + d2
```

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	NaN	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	NaN	NaN	NaN

```
d3.isNull()
```

	국어	영어	수학
길동	False	False	False
미영	False	False	False
순이	False	False	False
슬기	True	True	True
영이	False	False	False
철수	False	False	False
철이	True	True	True

```
d3.notNull()
```

	국어	영어	수학
길동	True	True	True
미영	True	True	True
순이	True	True	True
슬기	False	False	False
영이	True	True	True
철수	True	True	True
철이	False	False	False

### ■ 누락 데이터 제거

- 행 제거: DataFrame객체.dropna(subset=column명 리스트, how='any'/'all', axis=0)
- 열 제거: DataFrame객체.dropna(axis=1, thresh=개수)
- 예시 - 행 제거

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	NaN	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	NaN	NaN	NaN



```
d4 = d3.dropna(subset=['국어'])
```

d4

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0

## ■ 누락 데이터 치환

- DataFrame객체['column명'].fillna(값)
  - ✓ 새로운 객체 반환
- DataFrame객체['column명'].fillna(값, inplace=True)
  - ✓ 원본 객체 변경
- 예시

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	NaN	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	NaN	NaN	NaN



```
min_kor = d3['국어'].min()
```

```
d3['국어'].fillna(min_kor, inplace=True)
```

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	71.0	NaN	NaN

### 중복 데이터 확인

- 행 중복 확인: DataFrame객체.duplicated()
- 열 중복 확인: DataFrame객체['column명'].duplicated()
- 예시

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	71.0	NaN	NaN



d3.duplicated()

```
길동    False
미영    False
순이    False
슬기    False
영이    False
철수    False
철이     True
dtype: bool
```

d3['국어'].duplicated()

```
길동    False
미영    False
순이    False
슬기    False
영이     True
철수    False
철이     True
Name: 국어, dtype: bool
```

## ■ 중복 데이터 제거

- 중복 행 제거: DataFrame객체.drop\_duplicates(inplace=False)
- 중복 열 제거: DataFrame객체.drop\_duplicates(subset=column명 리스트, inplace=False)
- 예시

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	71.0	NaN	NaN



```
d4 = d3.drop_duplicates()
```

d4

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0

```
d5 = d3.drop_duplicates(subset=['국어', '영어'])
```

d5

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0

## ■ 자료형 변환

- DataFrame객체['column명'].astype(자료형)
- 예시

d3

	국어	영어	수학
길동	167.0	174.0	156.0
미영	200.0	177.0	198.0
순이	136.0	194.0	129.0
슬기	71.0	NaN	NaN
영이	71.0	75.0	154.0
철수	164.0	167.0	168.0
철이	71.0	NaN	NaN



d3['영어']

```
길동    174.0
미영    177.0
순이    194.0
슬기      NaN
영이     75.0
철수    167.0
철이      NaN
Name: 영어, dtype: float64
```

d3['영어'].astype('str')

```
길동    174.0
미영    177.0
순이    194.0
슬기      nan
영이     75.0
철수    167.0
철이      nan
Name: 영어, dtype: object
```

## ■ 다른 자료형을 시계열 객체로 변환

### • 문자열을 Timestamp로 변환

✓ to\_datetime() 함수 사용

### • 예시

d1

	생년월일	점수
철수	1990-03-02	90
영이	1991-06-08	95
길동	1990-11-22	80
미영	1991-01-05	88

d1.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, 철수 to 미영
Data columns (total 2 columns):
생년월일      4 non-null object
점수          4 non-null int64
dtypes: int64(1), object(1)
memory usage: 96.0+ bytes
```



```
d1['생년월일'] = pd.to_datetime(d1['생년월일'])
```

d1

	생년월일	점수
철수	1990-03-02	90
영이	1991-06-08	95
길동	1990-11-22	80
미영	1991-01-05	88

d1.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, 철수 to 미영
Data columns (total 2 columns):
생년월일      4 non-null datetime64[ns]
점수          4 non-null int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 96.0+ bytes
```



## ■ 다른 자료형을 시계열 객체로 변환

- Timestamp를 Period로 변환

- ✓ to\_period() 함수 사용

- 예시

```
dates = ['2019-01-01', '2020-03-02', '2021-12-31']
```

```
ts_dates = pd.to_datetime(dates)
```

```
ts_dates
```

```
DatetimeIndex(['2019-01-01', '2020-03-02', '2021-12-31'], dtype='datetime64[ns]', freq=None)
```

```
pr_day = ts_dates.to_period(freq='D')
```

```
pr_day
```

```
PeriodIndex(['2019-01-01', '2020-03-02', '2021-12-31'], dtype='period[D]', freq='D')
```

```
pr_month = ts_dates.to_period(freq='M')
```

```
pr_month
```

```
PeriodIndex(['2019-01', '2020-03', '2021-12'], dtype='period[M]', freq='M')
```

```
pr_year = ts_dates.to_period(freq='A')
```

```
pr_year
```

```
PeriodIndex(['2019', '2020', '2021'], dtype='period[A-DEC]', freq='A-DEC')
```

## ■ 시계열 데이터 만들기

- Timestamp 배열
  - ✓ `date_range()` 함수 사용
- 예시

```
ts_ms = pd.date_range(start='2019-01-01',  
                      end=None,  
                      periods=6,  
                      freq='MS',  
                      tz='Asia/Seoul')
```

```
ts_ms
```

```
DatetimeIndex(['2019-01-01 00:00:00+09:00', '2019-02-01 00:00:00+09:00',  
               '2019-03-01 00:00:00+09:00', '2019-04-01 00:00:00+09:00',  
               '2019-05-01 00:00:00+09:00', '2019-06-01 00:00:00+09:00'],  
              dtype='datetime64[ns, Asia/Seoul]', freq='MS')
```

```
ts_m = pd.date_range(start='2019-01-01',  
                    end=None,  
                    periods=6,  
                    freq='2M',  
                    tz='Asia/Seoul')
```

```
ts_m
```

```
DatetimeIndex(['2019-01-31 00:00:00+09:00', '2019-03-31 00:00:00+09:00',  
               '2019-05-31 00:00:00+09:00', '2019-07-31 00:00:00+09:00',  
               '2019-09-30 00:00:00+09:00', '2019-11-30 00:00:00+09:00'],  
              dtype='datetime64[ns, Asia/Seoul]', freq='2M')
```

## ■ 시계열 데이터 만들기

- Period 배열
  - ✓ period\_range() 함수 사용
- 예시

```
pr_m = pd.period_range(start='2019-01-01',  
                        end=None,  
                        periods=3,  
                        freq='2M')
```

```
pr_m
```

```
PeriodIndex(['2019-01', '2019-03', '2019-05'], dtype='period[2M]', freq='2M')
```

```
pr_h = pd.period_range(start='2019-01-01',  
                        end=None,  
                        periods=3,  
                        freq='H')
```

```
pr_h
```

```
PeriodIndex(['2019-01-01 00:00', '2019-01-01 01:00', '2019-01-01 02:00'], dtype='period[H]', freq='H')
```

## 데이터 사전 처리

### ■ 시계열 데이터 활용

- 날짜 데이터 분리

- ✓ 연-월-일 정보에서 연,월,일 추출: dt.year, dt.month, dt.day를 사용

- 예시

d1

	생년월일	점수
철수	1990-03-02	90
영이	1991-06-08	95
길동	1990-11-22	80
미영	1991-01-05	88



```
d1['년'] = d1['생년월일'].dt.year
```

```
d1['월'] = d1['생년월일'].dt.month
```

```
d1['일'] = d1['생년월일'].dt.day
```

d1

	생년월일	점수	년	월	일
철수	1990-03-02	90	1990	3	2
영이	1991-06-08	95	1991	6	8
길동	1990-11-22	80	1990	11	22
미영	1991-01-05	88	1991	1	5

# 데이터 사전 처리

## ■ 시계열 데이터 활용

- 날짜 데이터 분리

- ✓ 연-월-일 정보에서 연-월 등 추출: dt.to\_period()를 사용

- 예시

d1

	생년월일	점수	년	월	일
철수	1990-03-02	90	1990	3	2
영이	1991-06-08	95	1991	6	8
길동	1990-11-22	80	1990	11	22
미영	1991-01-05	88	1991	1	5



```
d1['년월'] = d1['생년월일'].dt.to_period(freq='M')
```

d1

	생년월일	점수	년	월	일	년월
철수	1990-03-02	90	1990	3	2	1990-03
영이	1991-06-08	95	1991	6	8	1991-06
길동	1990-11-22	80	1990	11	22	1990-11
미영	1991-01-05	88	1991	1	5	1991-01

# 데이터 사전 처리

## ■ 시계열 데이터 활용

- 날짜 인덱스 활용
- 예시

d1

	생년월일	점수	년	월	일	년월
철수	1990-03-02	90	1990	3	2	1990-03
영이	1991-06-08	95	1991	6	8	1991-06
길동	1990-11-22	80	1990	11	22	1990-11
미영	1991-01-05	88	1991	1	5	1991-01



d2 = d1.set\_index('생년월일')

d2

점수	년	월	일	년월
	생년월일			
90	1990	3	2	1990-03
95	1991	6	8	1991-06
80	1990	11	22	1990-11
88	1991	1	5	1991-01

d2['1990']

점수	년	월	일	년월
	생년월일			
90	1990	3	2	1990-03
80	1990	11	22	1990-11

d2['1990-03']

점수	년	월	일	년월
	생년월일			
90	1990	3	2	1990-03

d2['1990-01':'1990-12']

점수	년	월	일	년월
	생년월일			
90	1990	3	2	1990-03
80	1990	11	22	1990-11

today = pd.to\_datetime('2019-01-01')

d2['날짜\_차이'] = today - d2.index

d2

점수	년	월	일	년월	날짜_차이
	생년월일				
90	1990	3	2	1990-03	10532 days
95	1991	6	8	1991-06	10069 days
80	1990	11	22	1990-11	10267 days
88	1991	1	5	1991-01	10223 days

- 열 순서 변경
  - 예시

d1

	생년월일	점수	년	월	일	년월
철수	1990-03-02	90	1990	3	2	1990-03
영이	1991-06-08	95	1991	6	8	1991-06
길동	1990-11-22	80	1990	11	22	1990-11
미영	1991-01-05	88	1991	1	5	1991-01



```
column_customed = ['점수', '생년월일', '년월', '년', '월', '일']
```

```
d1[column_customed]
```

	점수	생년월일	년월	년	월	일
철수	90	1990-03-02	1990-03	1990	3	2
영이	95	1991-06-08	1991-06	1991	6	8
길동	80	1990-11-22	1990-11	1990	11	22
미영	88	1991-01-05	1991-01	1991	1	5

```
columns_sorted = sorted(d1.columns)
```

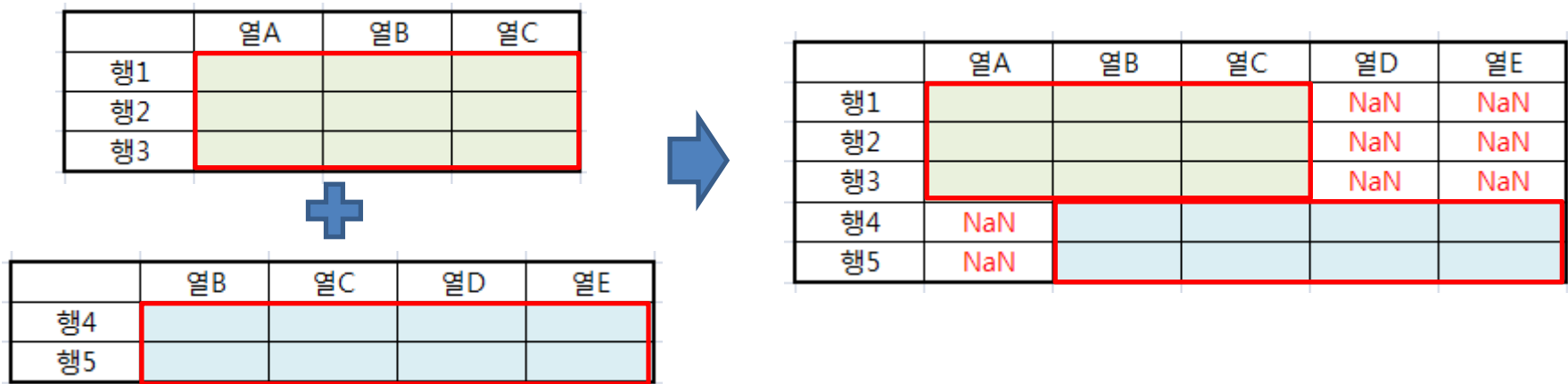
```
d1[columns_sorted]
```

	년	년월	생년월일	월	일	점수
철수	1990	1990-03	1990-03-02	3	2	90
영이	1991	1991-06	1991-06-08	6	8	95
길동	1990	1990-11	1990-11-22	11	22	80
미영	1991	1991-01	1991-01-05	1	5	88

## ■ DataFrame 연결

- `pandas.concat(DataFrame 리스트, axis=0, join='outer', ignore_index=False, sort=False)`

✓ axis = 0 일때





## ■ DataFrame 연결

- 예시 - axis = 0

```
import pandas as pd
```

```
d1 = pd.DataFrame({'a': ['a0', 'a1', 'a2', 'a3'],
                   'b': ['b0', 'b1', 'b2', 'b3'],
                   'c': ['c0', 'c1', 'c2', 'c3']},
                  index=[0, 1, 2, 3])
```

```
d2 = pd.DataFrame({'a': ['a2', 'a3', 'a4', 'a5'],
                   'b': ['b2', 'b3', 'b4', 'b5'],
                   'c': ['c2', 'c3', 'c4', 'c5'],
                   'd': ['d2', 'd3', 'd4', 'd5']},
                  index=[2, 3, 4, 5])
```

d1

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

d2

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

```
r1 = pd.concat([d1, d2], sort=False)
```

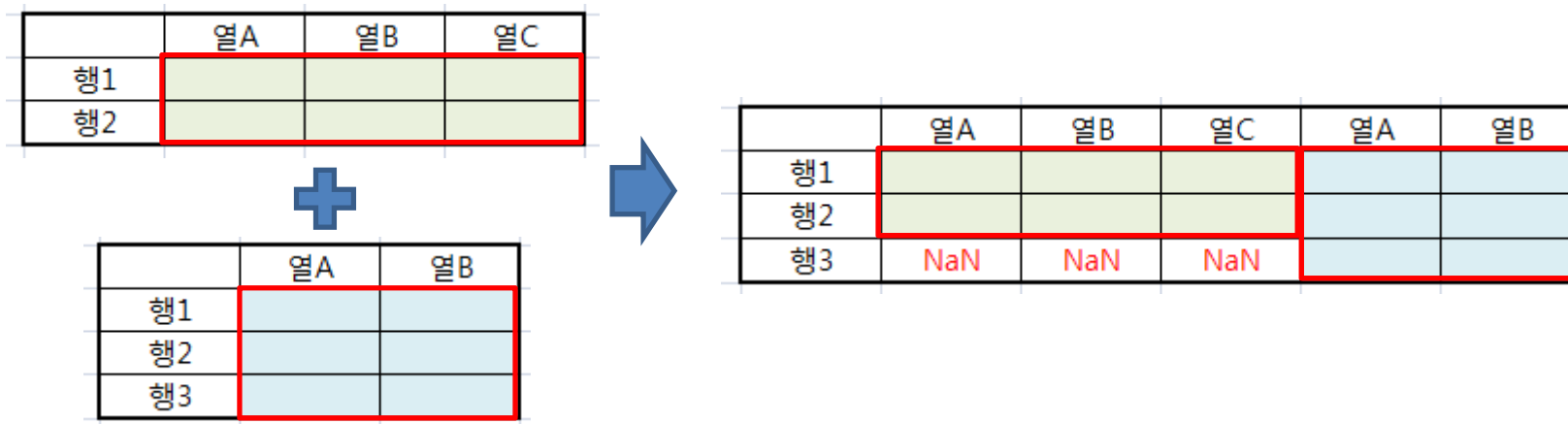
r1

	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

## ■ DataFrame 연결

- `pandas.concat(DataFrame 리스트, axis=0, join='outer', ignore_index=False, sort=False)`

✓ axis = 1 일때



## ■ DataFrame 연결

- 예시 - axis = 1

```
import pandas as pd
```

```
d1 = pd.DataFrame({'a': ['a0', 'a1', 'a2', 'a3'],
                   'b': ['b0', 'b1', 'b2', 'b3'],
                   'c': ['c0', 'c1', 'c2', 'c3']},
                  index=[0, 1, 2, 3])
```

```
d2 = pd.DataFrame({'a': ['a2', 'a3', 'a4', 'a5'],
                   'b': ['b2', 'b3', 'b4', 'b5'],
                   'c': ['c2', 'c3', 'c4', 'c5'],
                   'd': ['d2', 'd3', 'd4', 'd5']},
                  index=[2, 3, 4, 5])
```

d1

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

d2

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5



```
r2 = pd.concat([d1, d2], axis=1)
```

r2

	a	b	c	a	b	c	d
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	NaN	NaN	NaN	NaN
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3
4	NaN	NaN	NaN	a4	b4	c4	d4
5	NaN	NaN	NaN	a5	b5	c5	d5

# DataFrame 응용

## ■ DataFrame 병합

- `pandas.merge(df_left, df_right, how='inner', on=None)`

	A	B	C
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2



	A	B	E
0	0a	0b	0e
1	a1	b1	e1



(how='outer', on='A')

	A	B_x	C	B_y	E
0	a0	b0	c0	NaN	NaN
1	a1	b1	c1	b1	e1
2	a2	b2	c2	NaN	NaN
3	0a	NaN	NaN	0b	0e

(how='inner', on='A')

	A	B_x	C	B_y	E
0	a1	b1	c1	b1	e1

## ■ DataFrame 병합

### • 예시

d1

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

d2

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5



```
r3 = pd.merge(d1, d2, how='outer', on='a')
```

r3

	a	b_x	c_x	b_y	c_y	d
0	a0	b0	c0	NaN	NaN	NaN
1	a1	b1	c1	NaN	NaN	NaN
2	a2	b2	c2	b2	c2	d2
3	a3	b3	c3	b3	c3	d3
4	a4	NaN	NaN	b4	c4	d4
5	a5	NaN	NaN	b5	c5	d5

```
r4 = pd.merge(d1, d2, how='inner', on='a')
```

r4

	a	b_x	c_x	b_y	c_y	d
0	a2	b2	c2	b2	c2	d2
1	a3	b3	c3	b3	c3	d3

```
r5 = pd.merge(d1, d2)
```

r5

	a	b	c	d
0	a2	b2	c2	d2
1	a3	b3	c3	d3

# DataFrame 응용

## ■ DataFrame 결합

- merge() 함수와 작동 방식 비슷
- 두 DataFrame의 행 index를 기준으로 결합
- DataFrame1.join(DataFrame2, how='left')
- 예시

d1			d2			
a	b	c	aa	bb	cc	dd
0	a0	b0	c0			
1	a1	b1	c1			
2	a2	b2	c2			
3	a3	b3	c3			

aa	bb	cc	dd
2	a2	b2	c2
3	a3	b3	c3
4	a4	b4	c4
5	a5	b5	c5



```
d1.join(d2, how='outer')
```

	a	b	c	aa	bb	cc	dd
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	NaN	NaN	NaN	NaN
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3
4	NaN	NaN	NaN	a4	b4	c4	d4
5	NaN	NaN	NaN	a5	b5	c5	d5

```
d1.join(d2, how='left')
```

	a	b	c	aa	bb	cc	dd
0	a0	b0	c0	NaN	NaN	NaN	NaN
1	a1	b1	c1	NaN	NaN	NaN	NaN
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3

```
d1.join(d2, how='inner')
```

	a	b	c	aa	bb	cc	dd
2	a2	b2	c2	a2	b2	c2	d2
3	a3	b3	c3	a3	b3	c3	d3

## ■ 그룹화

- DataFrame객체.groupby(기준이 되는 열의 리스트)
- 예시

```
import pandas as pd
```

```
names = ['철수', '영이', '길동', '미영', '순이', '철이']
```

```
columns = ['성별', '점수']
```

```
scores = [['남', 90], ['여', 90], ['남', 85], ['여', 99], ['여', 76], ['남', 98]]
```

```
d1 = pd.DataFrame(scores, index = names, columns = columns)
```

```
d1
```

	성별	점수
철수	남	90
영이	여	90
길동	남	85
미영	여	99
순이	여	76
철이	남	98

```
grouped = d1.groupby(['성별'])
```

```
for key, group in grouped:
    print('* key:', key)
    print('* number:', len(group))
    print(group)
    print()
```

```
* key: 남
* number: 3
   성별  점수
철수  남   90
길동  남   85
철이  남   98
```

```
* key: 여
* number: 3
   성별  점수
영이  여   90
미영  여   99
순이  여   76
```

## ■ 그룹 연산

### • 데이터 집계

- ✓ group객체.함수()
- ✓ 함수: mean(), max(), min(), sum(), count(), std(), 등등

### • 예시

d1

	성별	점수
철수	남	90
영이	여	90
길동	남	85
미영	여	99
순이	여	76
철이	남	98



```
grouped = d1.groupby(['성별'])
```

```
for key, group in grouped:
    print('* key:', key)
    print('* number:', len(group))
    print(group)
    print()
```

```
* key: 남
* number: 3
   성별  점수
철수   남   90
길동   남   85
철이   남   98
```

```
* key: 여
* number: 3
   성별  점수
영이   여   90
미영   여   99
순이   여   76
```



```
mean_all = grouped.mean()
```

```
mean_all
```

	점수
성별	
남	91.000000
여	88.333333



## ■ 그룹 연산

### • 데이터 변환

✓ group객체.transform(매핑 함수)

d1

	성별	점수
철수	남	90
영이	여	90
길동	남	85
미영	여	99
순이	여	76
철이	남	98



```
grouped = d1.groupby(['성별'])
```

```
for key, group in grouped:
    print('* key:', key)
    print('* number:', len(group))
    print(group)
    print()
```

```
* key: 남
* number: 3
  성별  점수
철수   남   90
길동   남   85
철이   남   98

* key: 여
* number: 3
  성별  점수
영이   여   90
미영   여   99
순이   여   76
```



```
def diff(x):
    return (x - x.mean())
```

```
score_diff = grouped.transform(diff)
```

score\_diff

	점수
철수	-1.000000
영이	1.666667
길동	-6.000000
미영	10.666667
순이	-12.333333
철이	7.000000

## ■ 그룹 연산

### • 그룹 객체 필터링

✓ group객체.filter(조건식 함수)

```
grouped = d1.groupby(['성별'])
```

```
for key, group in grouped:
    print('* key:', key)
    print('* number:', len(group))
    print(group)
    print()
```

```
* key: 남
* number: 3
  성별  점수
철수   남   90
길동   남   85
철이   남   98
```

```
* key: 여
* number: 3
  성별  점수
영이   여   90
미영   여   99
순이   여   76
```



```
grouped_filter = grouped.filter(lambda x: x.점수.mean() > 90)
```

```
grouped_filter
```

	성별	점수
철수	남	90
길동	남	85
철이	남	98

## 실습 - 공공데이터 분석: CCTV와 인구수

### ■ CCTV 데이터 취득



A screenshot of a Google search interface. The search bar contains the text '서울시 자치구 연도별 cctv 설치 현황'. Below the search bar, there are tabs for '전체' (All), '뉴스' (News), '이미지' (Images), '지도' (Maps), '동영상' (Videos), and '더보기' (More). The search results show approximately 30,300 results in 0.38 seconds. The first result is highlighted with a red box and is a file from the Seoul Open Government Data portal. The second result is a chart from the same portal. Both results include the URL 'https://opengov.seoul.go.kr/data/2813904' and 'https://opengov.seoul.go.kr/data/2813901' respectively.

Google

서울시 자치구 연도별 cctv 설치 현황

전체 뉴스 이미지 지도 동영상 더보기 설정 도구

검색결과 약 30,300개 (0.38초)

**[FILE]**서울시 자치구 연도별 CCTV 설치 현황 | 서울시 정보소통광장 ...  
<https://opengov.seoul.go.kr/data/2813904> ▼  
[FILE]서울시 자치구 연도별 CCTV 설치 현황 - 문서정보. 관리번호, D0000020245367, 등록일, 2019-07-18. 분류, 기타. 원본시스템, 공공데이터, 제공부서, 스마트 ...

**[CHART]**서울시 자치구 목적별 CCTV 설치 현황 | 서울시 정보소통광장 ...  
<https://opengov.seoul.go.kr/data/2813901> ▼  
... 지하철 노선별 물품보관함 문의처 · · 차상위계층 의료급여 선정방법과 절차는 어떻게 되나요? · 주  
거정비사업(재개발·재건축·주거환경개선사업)의 시행절차는?

## ■ CCTV 데이터 취득


[로그인](#) [맞춤](#)

[원문정보](#) [회의정보](#) [사전공표](#) [정보공개청구](#) [이용안내](#)

[홈](#) > [더보기](#) > [서울시의 다양한 아카이브 정보](#) > [공공데이터](#)

### [FILE]서울시 자치구 년도별 CCTV 설치 현황



[내용 바로가기](#)

[\[FILE\]서울시 자치구 년도별 CCTV 설치 현황](#) → [내용보기](#)

# 실습 - 공공데이터 분석: CCTV와 인구수

## ■ CCTV 데이터 취득

데이터셋

HOME > 데이터 이용하기 > 데이터셋

원천 서울시 자치구 년도별 CCTV 설치 현황 ★★★★★

★★★★★ ▼ 평가 활용갤러리등록 URL복사 상세정보 닫기 목록

분류 안전	원본시스템 <b>바로가기</b> 서울시 자치구 CCTV	저작권자 자치구 (02-2133-2883)	제공기관 서울특별시	제3저작권자 없음
담당자 오용근 (02-2133-2890)	원본형태 File	데이터공개일자 2014.04.02	경신주기 비정기(자료변경시)	제공부서 스마트도시정책관 정보통신보안담당관
태그 씨씨티비, 감시카메라	데이터수정일자 2018.06.08	이용허락조건 저작권자표시(BY)  이용이나 변경 및 2차적 저작물의 작성을 포함한 자유이용을 허락합니다.		

서울특별시 각 자치구의 설치 년도별 CCTV 설치 현황입니다.

Chart File

서울시 자치구 년도별 CCTV 설치 현황관련 파일을 제공합니다. 다운로드 하세요.

NO	파일명	파일크기(KB)	마지막수정일	최초공개일	다운로드
1	서울시 자치구 년도별 CCTV 설치 현황.xlsx	12.37	2018.06.08	2018.06.08	<b>DOWN</b>
2	서울시 자치구 년도별 CCTV 설치 현황(2011년 이전 ~2018년).xlsx	13.03	2019.06.26	2019.06.26	DOWN

c:/data/cctv\_in\_seoul.xlsx  
로 저장

## 실습 - 공공데이터 분석: CCTV와 인구수

### ■ CCTV 데이터 취득

- c:/data/cctv\_in\_seoul.xlsx 파일을 열어서 '기관명' 부분에 공백 없애고 저장

A	B	C	D
기관명	소계	2013년이전	2014년
강남구	4,758	1,979	474
강동구	1,493	1,028	73
강북구	946	472	70
강서구	1,202	634	52
관악구	3,223	1,511	406



A	B	C	D
기관명	소계	2013년이전	2014년
강남구	4,758	1,979	474
강동구	1,493	1,028	73
강북구	946	472	70
강서구	1,202	634	52
관악구	3,223	1,511	406
광진구	1,228	1,025	85
구로구	2,746	1,434	189

# 실습 - 공공데이터 분석: CCTV와 인구수

## 인구수 데이터 읽기

```
import pandas as pd
```

```
pop_seoul = pd.read_excel('c:/data/pop_in_seoul.xls')
```

```
pop_seoul
```

	기간	자치구	세대	인구	인구.1	인구.2	인구.3	인구.4	인구.5	인구.6	인구.7	인구.8	세대당인구	65세이상고령자
0	기간	자치구	세대	합계	합계	합계	한국인	한국인	한국인	등록외국인	등록외국인	등록외국인	세대당인구	65세이상고령자
1	기간	자치구	세대	계	남자	여자	계	남자	여자	계	남자	여자	세대당인구	65세이상고령자
2	2019.1/4	합계	4290922	10054979	4909387	5145592	9770216	4772134	4998082	284763	137253	147510	2.28	1436125
3	2019.1/4	종로구	73914	162913	78963	83950	152778	74536	78242	10135	4427	5708	2.07	26981
4	2019.1/4	중구	61800	135836	66720	69116	125942	61992	63950	9894	4728	5166	2.04	22421
5	2019.1/4	용산구	109413	245139	119597	125542	229168	110626	118542	15971	8971	7000	2.09	38049
6	2019.1/4	성동구	137247	314608	154011	160597	306404	150287	156117	8204	3724	4480	2.23	43076
7	2019.1/4	광진구	163460	370658	179162	191496	354873	172361	182512	15785	6801	8984	2.17	46288
8	2019.1/4	동대문구	162228	363262	179100	184162	346750	172784	173966	16512	6316	10196	2.14	57570
9	2019.1/4	종랑구	181182	407211	201808	205403	402203	199730	202473	5008	2078	2930	2.22	62789
10	2019.1/4	성북구	188670	450021	217400	232621	438245	212830	225415	11776	4570	7206	2.32	68612
11	2019.1/4	강북구	143663	321151	156525	164626	317386	155075	162311	3765	1450	2315	2.21	58858

- 인구수 데이터 읽기
  - 특정 열 가져오고, column 설정

```
pop_seoul = pd.read_excel('c:/data/pop_in_seoul.xls', usecols=[1,3,6,9,13], header=2)
```

pop\_seoul

	자치구	계	계.1	계.2	65세이상고령자
0	합계	10054979	9770216	284763	1436125
1	종로구	162913	152778	10135	26981
2	중구	135836	125942	9894	22421
3	용산구	245139	229168	15971	38049
4	성동구	314608	306404	8204	43076
5	광진구	370658	354873	15785	46288
6	동대문구	363262	346750	16512	57570
7	종랑구	407211	402203	5008	62789
8	성북구	450021	438245	11776	68612
9	강북구	321151	317386	3765	58858
10	도봉구	340089	337820	2269	56742



- 인구수 데이터 읽기
  - column명 변경

```
pop_seoul.columns = ['구별', '인구수', '한국인', '외국인', '고령자']
```

```
pop_seoul
```

	구별	인구수	한국인	외국인	고령자
0	합계	10054979	9770216	284763	1436125
1	종로구	162913	152778	10135	26981
2	중구	135836	125942	9894	22421
3	용산구	245139	229168	15971	38049
4	성동구	314608	306404	8204	43076
5	광진구	370658	354873	15785	46288
6	동대문구	363262	346750	16512	57570
7	중랑구	407211	402203	5008	62789
8	성북구	450021	438245	11776	68612

- 인구수 데이터 읽기
  - 합계 데이터(1번째 행) 삭제

```
pop_seoul.drop(0, inplace=True)
```

```
pop_seoul
```

	구별	인구수	한국인	외국인	고령자
1	종로구	162913	152778	10135	26981
2	중구	135836	125942	9894	22421
3	용산구	245139	229168	15971	38049
4	성동구	314608	306404	8204	43076
5	광진구	370658	354873	15785	46288
6	동대문구	363262	346750	16512	57570
7	중랑구	407211	402203	5008	62789
8	성북구	450021	438245	11776	68612
9	강북구	321151	317386	3765	58858
10	도봉구	340089	337820	2269	56742
11	노원구	545486	541174	4312	78170

### ■ CCTV 데이터 읽기

```
cctv_seoul = pd.read_excel('c:/data/cctv_in_seoul.xlsx')
```

```
cctv_seoul
```

	기관명	소계	2013년이전	2014년	2015년	2016년	2017년
0	강남구	4758	1979	474	760	770	775
1	강동구	1493	1028	73	142	240	10
2	강북구	946	472	70	147	257	0
3	강서구	1202	634	52	177	168	171
4	관악구	3223	1511	406	593	352	361
5	광진구	1228	1025	85	62	19	37
6	구로구	2746	1434	189	256	326	541
7	금천구	1526	922	91	305	109	99
8	노원구	1576	627	80	461	298	110
9	도봉구	899	480	185	49	102	83
10	동대문구	1555	1046	29	111	233	136
11	동작구	1792	781	341	103	314	253

- CCTV 데이터 읽기
  - column명 변경

```
cctv_seoul.columns = ['구별', 'CCTV수', '2013년 이전', '2014년', '2015년', '2016년', '2017년']
```

```
cctv_seoul
```

	구별	CCTV수	2013년 이전	2014년	2015년	2016년	2017년
0	강남구	4758	1979	474	760	770	775
1	강동구	1493	1028	73	142	240	10
2	강북구	946	472	70	147	257	0
3	강서구	1202	634	52	177	168	171
4	관악구	3223	1511	406	593	352	361
5	광진구	1228	1025	85	62	19	37
6	구로구	2746	1434	189	256	326	541
7	금천구	1526	922	91	305	109	99

## 실습 - 공공데이터 분석: CCTV와 인구수

### ■ CCTV와 인구수 데이터 합병

```
pop_cctv = pd.merge(pop_seoul, cctv_seoul, how='outer', on='구별')
```

```
pop_cctv
```

	구별	인구수	한국인	외국인	고령자	CCTV수	2013년 이전	2014년	2015년	2016년	2017년
0	종로구	162913	152778	10135	26981	1925	1324	167	163	129	142
1	중구	135836	125942	9894	22421	1260	786	40	191	123	120
2	용산구	245139	229168	15971	38049	2379	2071	97	76	77	58
3	성동구	314608	306404	8204	43076	2554	1953	159	98	39	305
4	광진구	370658	354873	15785	46288	1228	1025	85	62	19	37
5	동대문구	363262	346750	16512	57570	1555	1046	29	111	233	136
6	중랑구	407211	402203	5008	62789	1053	751	64	102	75	61
7	성북구	450021	438245	11776	68612	2221	1155	208	263	357	238
8	강북구	321151	317386	3765	58858	946	472	70	147	257	0
9	도봉구	340089	337820	2269	56742	899	480	185	49	102	83
10	노원구	545486	541174	4312	78170	1576	627	80	461	298	110

- CCTV와 인구수 데이터 합병
  - index 설정

```
pop_cctv.set_index('구별', inplace=True)
```

```
pop_cctv
```

	인구수	한국인	외국인	고령자	CCTV수	2013년 이전	2014년	2015년	2016년	2017년
구별										
종로구	162913	152778	10135	26981	1925	1324	167	163	129	142
중구	135836	125942	9894	22421	1260	786	40	191	123	120
용산구	245139	229168	15971	38049	2379	2071	97	76	77	58
성동구	314608	306404	8204	43076	2554	1953	159	98	39	305
광진구	370658	354873	15785	46288	1228	1025	85	62	19	37
동대문구	363262	346750	16512	57570	1555	1046	29	111	233	136
종랑구	407211	402203	5008	62789	1053	751	64	102	75	61
성북구	450021	438245	11776	68612	2221	1155	208	263	357	238
강북구	321151	317386	3765	58858	946	472	70	147	257	0

## 실습 - 공공데이터 분석: CCTV와 인구수

### ■ CCTV와 인구수 데이터 분석

- 인구수 대비 CCTV 비율, 외국인 대비 CCTV 비율, 고령자 대비 CCTV 비율을 계산하여 데이터프레임에 추가

```
pop_cctv['CCTV/인구수'] = (pop_cctv['CCTV수'] / pop_cctv['인구수']) * 100
pop_cctv['CCTV/외국인'] = (pop_cctv['CCTV수'] / pop_cctv['외국인']) * 100
pop_cctv['CCTV/고령자'] = (pop_cctv['CCTV수'] / pop_cctv['고령자']) * 100
```

```
pop_cctv
```

	인구수	한국인	외국인	고령자	CCTV수	2013년 이전	2014년	2015년	2016년	2017년	CCTV/인구수	CCTV/외국인	CCTV/고령자
구별													
종로구	162913	152778	10135	26981	1925	1324	167	163	129	142	1.181612	18.993587	7.134650
중구	135836	125942	9894	22421	1260	786	40	191	123	120	0.927589	12.734991	5.619732
용산구	245139	229168	15971	38049	2379	2071	97	76	77	58	0.970470	14.895749	6.252464
성동구	314608	306404	8204	43076	2554	1953	159	98	39	305	0.811804	31.131156	5.929056
광진구	370658	354873	15785	46288	1228	1025	85	62	19	37	0.331303	7.779538	2.652955
동대문구	363262	346750	16512	57570	1555	1046	29	111	233	136	0.428066	9.417393	2.701060
중랑구	407211	402203	5008	62789	1053	751	64	102	75	61	0.258588	21.026358	1.677045
성북구	450021	438245	11776	68612	2221	1155	208	263	357	238	0.493533	18.860394	3.237043
강북구	321151	317386	3765	58858	946	472	70	147	257	0	0.294565	25.126162	1.607258
도봉구	340089	337820	2269	56742	899	480	185	49	102	83	0.264343	39.620978	1.584364
노원구	545486	541174	4312	78170	1576	627	80	461	298	110	0.288917	36.549165	2.016119

## 실습 - 공공데이터 분석: CCTV와 인구수

- CCTV와 인구수 데이터 분석
  - 데이터프레임 엑셀 파일로 저장

```
pop_cctv.to_excel('c:/data/cctv_pop.xls', encoding = 'euc-kr')
```



A	B	C	D	E	F	G	H	I	J	K	L	M	N
구별	인구수	한국인	외국인	고령자	CCTV수	2013년 이전	2014년	2015년	2016년	2017년	CCTV/인구수	CCTV/외국인	CCTV/고령자
종로구	162913	152778	10135	26981	1925	1324	167	163	129	142	1.181612272	18.99358658	7.134650309
중구	135836	125942	9894	22421	1260	786	40	191	123	120	0.927589152	12.7349909	5.619731502
용산구	245139	229168	15971	38049	2379	2071	97	76	77	58	0.970469815	14.89574854	6.252463928
성동구	314608	306404	8204	43076	2554	1953	159	98	39	305	0.811803896	31.13115553	5.929055623
광진구	370658	354873	15785	46288	1228	1025	85	62	19	37	0.33130271	7.779537536	2.65295541
동대문구	363262	346750	16512	57570	1555	1046	29	111	233	136	0.428065694	9.417393411	2.70105958
중랑구	407211	402203	5008	62789	1053	751	64	102	75	61	0.258588299	21.02635783	1.677045342
성북구	450021	438245	11776	68612	2221	1155	208	263	357	238	0.493532524	18.86039402	3.237043083
강북구	321151	317386	3765	58858	946	472	70	147	257	0	0.294565485	25.12616202	1.607258147
도봉구	340089	337820	2269	56742	899	480	185	49	102	83	0.264342569	39.6209784	1.584364316
노원구	545486	541174	4312	78170	1576	627	80	461	298	110	0.288916672	36.54916512	2.016118716
은평구	488713	484274	4439	78406	2505	1351	343	210	358	243	0.512570773	56.43162875	3.194908553
서대문구	324604	311771	12833	51085	2705	1808	114	109	266	408	0.833323064	21.07846957	5.295096408
마포구	386571	375106	11465	51293	1743	838	65	164	334	342	0.450887418	15.2027911	3.3981245
양천구	466622	462599	4023	58930	2498	1701	164	178	338	117	0.535336954	62.09296545	4.238927541

c:/data/cctv\_pop.xls