

6

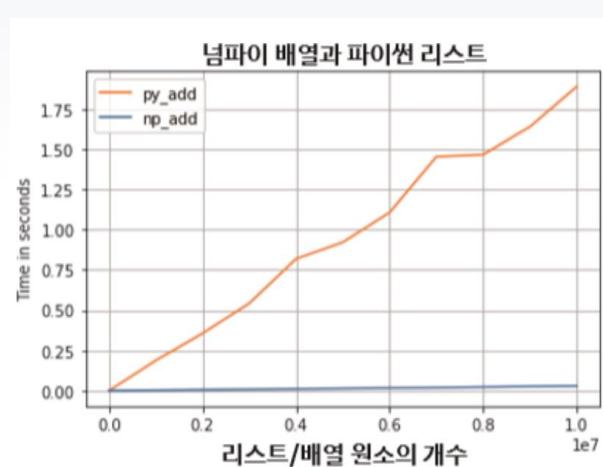
Numpy

Numpy 란?

- 행렬 연산에 다양한 기능을 제공하는 라이브러리 (<https://numpy.org/>)
- 내부가 C로 작성되어 있어 속도가 매우 빠름
- 파이썬 외부 라이브러리
- 파이썬을 활용한 과학 컴퓨팅 전용 모듈
- 복잡한 행렬계산, 선형대수, 통계등의 기능 제공
- 선형대수학을 빠르게 연산
- pip install numpy 로 설치

Numpy 사용

```
1 import numpy as np
```



넘파이와 리스트의 연산 속도비교
[출처 : <https://dkswnk.tistory.com/323>]

Numpy 배열

- numpy 배열은 동일한 type (ndarray)
- list 는 고성능 수치계산이 어려워 numpy 배열로 수치계산 수행

numpy 배열생성 및 타입확인

```
1 import numpy as np  
2 data = np.array( [ 1, 2, 3, 4, 5 ] )  
3 data  
  
array([1, 2, 3, 4, 5])
```

```
1 type( data )
```

numpy.ndarray

(ndarray : n-dimensional array , N차원 배열)

ndim 차원확인

```
1 import numpy as np  
2 data1 = np.array( [ 1, 2, 3, 4, 5 ] )  
3 data2 = np.array( [ [ 1, 2, 3 ], [ 4, 5, 6 ] ] )  
4 print( data1.ndim )  
5 print( data2.ndim )
```

```
1  
2
```

(data : 1차원 , data1 : 2차원)

Numpy 명령어

shape - 크기확인

```
1 data = np.array([1,2,3,4,5])
2 data1 = np.array([[1,2,3],[4,5,6]])
3 print(data.shape)
4 print(data1.shape)
```

```
(5,)
(2, 3)
```

dtype - 타입확인

```
1 data = np.array([[1,2,3],[4,5,6]])
2 print(data.dtype)
```

```
int64
```

- **bool**: 불리언 (True, False)
- **int**: 정수 (부호 있는 정수)
- **uint**: 부호 없는 정수
- **float**: 부동소수점 수 (단정도, 배정도, 배열 단정도)
- **complex**: 복소수 (배열 단정도 복소수, 배열 배정도 복소수)
- **string**: 문자열
- **unicode**: 유니코드 문자열
- **void**: 사용자 정의 데이터 타입

Numpy 명령어

dtype지정 및 변경 (astype)

```
1 data = np.array([[1,2,3],[4,5,6]])
2 print(data.dtype)
3
4 data = np.array([[1,2,3],[4,5,6]],dtype='float')
5 print(data.dtype)
6
7 data = data.astype('int')
8 print(data.dtype)
```

```
int64
float64
int64
```

```
1 data = np.array([('1','2','3'),('4','5','6')],dtype='int')
2 print(data.dtype)
3 data
```

```
int64
array([[1, 2, 3],
       [4, 5, 6]])
```

(문자형 데이터를 int형으로 변형)

size - 총 갯수

```
1 import numpy as np
2 data = np.array([1,2,3,4,5])
3 data1 = np.array([[1,2,3],[4,5,6]])
4 print(data.size)
5 print(data1.size)
```

```
5
6
```

Numpy 명령어

T - 행, 열의 교환 (transpose)

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

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

```
1 data.T
```

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

arange() - array생성 : 리스트 range()와 비슷

- np.arange(start , end , step)

```
1 data = np.arange(10)
2 data
```

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

```
1 data = np.arange(0,10,2)
2 data
```

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

linspace()

- np.linspace(start , end , count)

```
1 data = np.linspace(1,10,10)
```

```
2 data
```

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

```
1 data = np.linspace(1,10,20)
```

```
2 data
```

```
array([ 1.          ,  1.47368421,  1.94736842,  2.42105263,  2.89473684,
       3.36842105,  3.84210526,  4.31578947,  4.78947368,  5.26315789,
       5.73684211,  6.21052632,  6.68421053,  7.15789474,  7.63157895,
       8.10526316,  8.57894737,  9.05263158,  9.52631579, 10.        ])
```

Numpy 명령어

zeros() : 0으로 채워진 행렬

```
1 data = np.zeros((3,3))  
2 data
```

```
array([[0., 0., 0.],  
       [0., 0., 0.],  
       [0., 0., 0.]])
```

ones() : 1로 채워진 행렬

```
1 data = np.ones((5,3))  
2 data
```

```
array([[1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.],  
       [1., 1., 1.]])
```

full() : 지정값으로 채워진 행렬

```
1 data = np.full((5,3),fill_value=10)  
2 data
```

```
array([[10, 10, 10],  
       [10, 10, 10],  
       [10, 10, 10],  
       [10, 10, 10],  
       [10, 10, 10]])
```

eye() : 대각으로 1로 채워진 행렬

```
1 data = np.eye(3)  
2 data
```

```
array([[1., 0., 0.],  
       [0., 1., 0.],  
       [0., 0., 1.]])
```

Numpy 명령어

난수 생성 방법 및 관련 함수

- `np.random.rand()` : 0부터 1사이의 균일 분포로 난수를 생성
- `np.random.randn()` : 가우시안 정규 분포로 난수를 생성
- `np.random.randint()` : 지정한 범위 내에서 균일 분포로 정수 난수를 생성
- `np.random.random()` : 0부터 1사이의 균일 분포로 난수를 생성
- `np.random.choice()` : 주어진 1차원 배열에서 임의의 요소를 선택
- `np.random.shuffle()` : 주어진 배열을 무작위로 섞기
- `np.random.seed()` : 난수 발생 시드를 설정
- `np.random.permutation()` : 주어진 배열을 무작위로 섞은 후 반환
- `np.random.beta()` : 베타 분포로부터 난수를 생성
- `np.random.binomial()` : 이항 분포로부터 난수를 생성
- `np.random.chisquare()` : 카이 제곱 분포로부터 난수를 생성
- `np.random.exponential()` : 지수 분포로부터 난수를 생성
- `np.random.gamma()` : 감마 분포로부터 난수를 생성
- `np.random.normal()` : 정규 분포로부터 난수를 생성
- `np.random.poisson()` : 포아송 분포로부터 난수를 생성
- `np.random.uniform()` : 균일 분포로부터 난수를 생성

```
1 np.random.rand(3)
```

```
array([ 0.07465943,  0.91769073,  0.79559678])
```

```
1 np.random.randn(3)
```

```
array([-0.05734587, -2.00335685, -0.29712669])
```

```
1 np.random.randint(1,10,3)
```

```
array([4, 3, 3])
```

Numpy 명령어

reshape : 형태변경

```
1 data = np.arange(1,10)
2 print(data)
3 data.reshape(3,3)
```

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

numpy연산 : 더하기, 빼기 , 곱하기 , 나누기

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data1 = np.array([[10,20,30],[40,50,60]])
3 data + data1
```

```
array([[11, 22, 33],
       [44, 55, 66]])
```

```
1 data - data1
```

```
array([[ -9, -18, -27],
       [-36, -45, -54]])
```

```
1 data * data1
```

```
array([[ 10,  40,  90],
       [160, 250, 360]])
```

```
1 data / data1
```

```
array([[ 0.1,  0.1,  0.1],
       [ 0.1,  0.1,  0.1]])
```

Numpy 통계

sum(): 합계

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.sum()
```

21

sum(axis=0) : 열의 합계)

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.sum(axis = 0)

array([5, 7, 9])
```

sum(axis=1 : 행의 합계)

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.sum(axis = 1)

array([ 6, 15])
```

mean(): 평균

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.mean()
```

3.5

std(): 표준편차

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.std()
```

1.707825127659933

var(): 분산

- 데이터가 평균으로부터 얼마나 떨어져 있는지 표
- 관측 값에서 평균을 뺀 값을 제곱하고, 그 것을 모두 더한 후 전체 개수로 나눠서 구한 값

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.var()
```

2.9166666666666665

max(): 최대값

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.max()
```

6

min(): 최소값

```
1 data = np.array([[1,2,3],[4,5,6]])
2 data.min()
```

1

Numpy 예제

문제 : 다음 주어진 data.txt 파일의 자료를 불러와 총합과 평균을 구하세요.

```
1 import numpy as np      numpy 임포트
```

```
1 f = open('data.txt', 'r')  data.txt 파일을 읽기 모드로 불러오기
```

```
1 data = f.readlines()      파일의 내용을 모두 불러와서 데이터의 내용과 데이터의 길이를 확인
2 print(data)
3 print(len(data))
```

```
['20', '97', '66', '0', '1', '52', '18', '6', '20', '21', '17', '33', '16', '4', '98',
1
```

```
1 data = data[0].split(',')
2 print(data)              데이터가 한 개 있음으로 맨 처음 데이터를 불러와, 로 데이터를 분리
```

```
['20', '97', '66', '0', '1', '52', '18', '6', '20',
```

Numpy 예제

```
1 data = np.array(data)  
2 data
```

```
array(['20', '97', '66', '0', '1', '52', '18', '6', '20', '21', '17',  
       '33', '16', '4', '98', '46', '33', '80', '64', '72', '70', '27',  
       '68', '70', '13', '36', '60', '56', '47', '27', '2', '70', '27',  
       '67', '80', '54', '13', '64', '55', '46', '41', '19', '63', '14',  
       '79', '4', '22', '7', '81', '26', '36', '95', '75', '47', '28',  
       '76', '84', '27', '90', '16', '33', '90', '20', '87', '29', '84',  
       '66', '98', '18', '62', '40', '23', '97', '4', '80', '8', '20',  
       '36', '41', '87', '79', '12', '81', '51', '29', '78', '39', '32',  
       '84', '59', '7', '48', '26', '83', '46', '34', '38', '63', '69',  
       '10', ''], dtype='<U2')
```

변환된 데이터를 numpy array로 변환후 데이터 확인

```
1 data = data.astype('int')
```

유니코드 데이터를 정수형 데이터로 변환

```
1 data.sum()
```

numpy sum() 함수를 이용하여 합계 계산

4637

```
1 data.mean()
```

numpy mean() 함수를 이용하여 합계 계산

46.37

Numpy 예제

문제 : 다음 주어진 data_weed.txt 파일은 4주간의 데이터가 저장된 데이터입니다.

각 주단위의 평균을 구하세요. 1주평균, 2주평균, 3주평균, 4주평균

```
1 import numpy as np
```

numpy 임포트

```
1 f = open('data_week.txt', 'r')
```

data_week.txt 파일을 불러옴

```
1 data = f.readlines()  
2 data
```

파일안의 데이터를 모두 불러옴

```
['40,52,11,95,52,61,21\n',  
'53,17,83,34,29,95,51\n',  
'3,2,3,35,62,8,42\n',  
'30,10,8,5,39,49,82\n']
```

```
1 result = []  
2 for temp in data:  
3     result.append( temp.strip().split(',') )  
4 result
```

리스트 안의 4개의 데이터의 공백 (\n)을 제거하고, 로 구분해서 저장

```
[['40', '52', '11', '95', '52', '61', '21'],  
 ['53', '17', '83', '34', '29', '95', '51'],  
 ['3', '2', '3', '35', '62', '8', '42'],  
 ['30', '10', '8', '5', '39', '49', '82']]
```

Numpy 예제

```
1 result = np.array(result,dtype='int')      데이터를 numpy array로 저장하고 타입을 정수형으로 변환  
2 result
```

```
array([[40, 52, 11, 95, 52, 61, 21],  
       [53, 17, 83, 34, 29, 95, 51],  
       [ 3,  2,  3, 35, 62,  8, 42],  
       [30, 10,  8,  5, 39, 49, 82]])
```

```
1 week_mean = result.mean(axis=1)  
2 week_mean
```

행별 평균 데이터를 구함

```
array([47.42857143, 51.71428571, 22.14285714, 31.85714286])
```

```
1 count = 0  
2 for temp in week_mean:  
3     count += 1  
4     print(count,'주 평균 : ',temp)
```

행별 평균 데이터를 화면에 출력

```
1 주 평균 : 47.42857142857143  
2 주 평균 : 51.714285714285715  
3 주 평균 : 22.142857142857142  
4 주 평균 : 31.857142857142858
```

7

Pandas

Pandas란?

- 데이터 분석에 관련된 기능을 제공하는 파이썬 라이브러리
- 큰데이터를 빠르게 처리 (indexing, slicing, sorting 등)
- 데이터들의 연산
- 외부 데이터 (csv, txt, excel 등) 의 처리
- 설치 : pip install pandas

series 생성 - tuple

```
1 data = pd.Series((1,2,3))  
2 data
```

```
0    1  
1    2  
2    3  
dtype: int64
```

series : pandas를 구성하는 자료형 (1차원 자료형)

series 생성 - list

```
1 import pandas as pd  
2  
3 data = pd.Series(['서울', '부산', '대전'])  
4 data  
  
0  
서울  
1    부산  
2    대전  
dtype: object
```

series 생성 - numpy

```
1 data = pd.Series(np.array([1,2,3]))  
2 data  
  
0    1  
1    2  
2    3  
dtype: int64
```

series 생성 - dict

```
1 data = pd.Series({'서울':1, '부산':2})  
2 data  
  
서울    1  
부산    2  
dtype: int64
```

Series

Series 데이터 타입

- object: 일반 문자열 타입
- float: 실수
- int: 정수
- category: 카테고리
- datetime: 시간

series 데이터 확인 : info()

```
1 data.info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 5 entries, 0 to 4
Series name: None
Non-Null Count Dtype
-----
5 non-null    int64
dtypes: int64(1)
memory usage: 168.0 bytes
```

```
1 data = pd.Series(np.array([1,2,3,4,5]))
2 data
```

| | | |
|---|---|-------|
| 0 | 1 | |
| 1 | 2 | |
| 2 | 3 | value |
| 3 | 4 | |
| 4 | 5 | |

dtype: int64

index

Series

index . value 값 확인

```
1 data = pd.Series([1,2,3])
2 print(data.index)
3 print(data.values)

RangeIndex(start=0, stop=3, step=1)
[1 2 3]
```

dtype : 데이터 타입확인

```
1 data = pd.Series([1,2,3])
2 print(data.dtypes)
3
4 data1 = pd.Series([0.1,0.2])
5 print(data1.dtypes)

int64
float64
```

index 설정

```
1 data = pd.Series([10,15,30])
2 data

0    10
1    15
2    30
dtype: int64
```

```
1 data.index = ['서울', '부산', '대전']
2 data
```

```
서울    10
부산    15
대전    30
dtype: int64
```

index 조회

```
1 data = pd.Series([10,15,30],
2                   index = ['서울', '부산', '대전'])
3 print( data['서울'] )
4 print( data.at['서울'] )
```

```
10
10
```

Series

Series 연산 - 합

```
1 data = pd.Series([1,2,3,4,5])
2 data = data + 10
3 data
```

```
0    11
1    12
2    13
3    14
4    15
dtype: int64
```

```
1 data = pd.Series([1,2,3,4,5])
2 data1 = pd.Series([10,20,30,40,50])
3 data = data + data1
4 data
```

```
0    11
1    22
2    33
3    44
4    55
dtype: int64
```

Series 연산 - 차

```
1 data = pd.Series([1,2,3,4,5])
2 data = data - 10
3 data
```

```
0    -9
1    -8
2    -7
3    -6
4    -5
dtype: int64
```

```
1 data = pd.Series([1,2,3,4,5])
2 data1 = pd.Series([10,20,30,40,50])
3 data = data - data1
4 data
```

```
0    -9
1   -18
2   -27
3   -36
4   -45
dtype: int64
```

Series 연산 - 곱

```
1 data = pd.Series([1,2,3,4,5])
2 data = data * 10
3 data
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

Series 연산 - 나누기

```
1 data = pd.Series([1,2,3,4,5])
2 data = data / 10
3 data
```

```
0    0.1
1    0.2
2    0.3
3    0.4
4    0.5
dtype: float64
```

Series - 결측치 처리

NaN 처리 - fillna(값) 값으로 데이터 채우기

```
1 data = pd.Series([1,2,3], index = ['강남','노원','서초'])
2 data1 = pd.Series([4,5,6], index = ['강남','마포','동작'])
3 total = data + data1
4 total
```

```
강남      5.0
노원      NaN
동작      NaN
마포      NaN
서초      NaN
dtype: float64
```

```
1 total = total.fillna(0)
2 total
```

```
강남      5.0
노원      0.0
동작      0.0
마포      0.0
서초      0.0
dtype: float64
```

NaN 처리 - dropna() 삭제하기

```
1 data = pd.Series([1,2,3], index = ['강남','노원','서초'])
2 data1 = pd.Series([4,5,6], index = ['강남','마포','동작'])
3 total = data + data1
4 total
```

```
강남      5.0
노원      NaN
동작      NaN
마포      NaN
서초      NaN
dtype: float64
```

```
1 total = total.dropna()
2 total
```

```
강남      5.0
dtype: float64
```

Series - 통계

describe() : 통계정보

```
1 data = pd.Series([1,2,3,4,5])
2 data1 = pd.Series([10,20,30,40,50])
3 data = data + data1
4 data.describe()
```

```
count      5.000000
mean      33.000000
std       17.392527
min       11.000000
25%      22.000000
50%      33.000000
75%      44.000000
max       55.000000
dtype: float64
```

```
1 print('평균 : ',result.mean()) #평균
2 print('표준편차 : ',result.std()) #표준편차
3 print('최소값 : ',result.min()) #최소값
4 print('최대값 : ',result.max()) #최대값
```

```
평균 : 33.0
표준편차 : 17.392527130926087
최소값 : 11
최대값 : 55
```

Pandas Series의 describe() 메서드는 Series의 간단한 통계 요약을 제공합니다.

- count : 데이터수
- mean : 평균
- std : 표준편차
- min : 최소값
- 25% : 1/4 분위수
- 50% : 2/4 분위수
- 75% : 3/4 분위수
- max : 최대값

위 값들을 통해 데이터의 분포와 이상치 여부 등을 확인합니다.

DataFrame

데이터프레임(DataFrame)은 테이블 형태로 구성된 2차원 데이터 구조입니다.
행(row)과 열(column)로 이루어져 있으며, 각 열은 서로 다른 데이터 타입을 가질 수 있습니다.

- 2차원 데이터
- Series 자료형을 테이블 형태로 표현
- from pandas import Series, DataFrame

DataFrame 생성방법 - 리스트를 활용한 생성

```
1 import pandas as pd  
2  
3 df = pd.DataFrame([[1,2,3],[4,5,6]])  
4 df
```

| 0 | 1 | 2 | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |

DataFrame 생성방법 - dict를 활용한 생성

```
1 import pandas as pd  
2  
3 df = pd.DataFrame({'강남구':[1,2,3], '영등포구':[4,5,6]  
4 , '서초구':[7,8,9]})  
5 df
```

| | 강남구 | 영등포구 | 서초구 | |
|---|-----|------|-----|--|
| 0 | 1 | 4 | 7 | |
| 1 | 2 | 5 | 8 | |
| 2 | 3 | 6 | 9 | |

DataFrame

index, column 설정

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                     index=['A','B'],columns=['서울','부산'])  
3 df
```

| | 서울 | 부산 |
|---|----|----|
| A | 1 | 2 |
| B | 3 | 4 |

index, column 변경

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                     index=['A','B'],columns=['서울','부산'])  
3 df.index = ['1주','2주']  
4 df.columns = ['서울시','부산시']  
5 df
```

| | 서울시 | 부산시 |
|----|-----|-----|
| 1주 | 1 | 2 |
| 2주 | 3 | 4 |

컬럼명으로 데이터 조회

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                     index=['A','B'],columns=['서울','부산'])  
3  
4 print( df['서울'] )  
5 print( df.서울 )  
6 print( df[['서울','부산']] )  
7
```

```
A    1  
B    3  
Name: 서울, dtype: int64  
A    1  
B    3  
Name: 서울, dtype: int64  
서울  부산  
A    1    2  
B    3    4
```

- df [컬럼명]
- df.컬럼명
- df [[컬럼명1, 컬럼명2]]

DataFrame

column 이름 변경 : rename()

```
1 df = pd.DataFrame({'서울':[1,2], '부산시':[3,4], '대전시':[5,6]})  
2 df
```

| | 서울 | 부산시 | 대전시 |
|---|----|-----|-----|
| 0 | 1 | 3 | 5 |
| 1 | 2 | 4 | 6 |

```
1 df.rename(columns={'서울': '서울시'})
```

| | 서울시 | 부산시 | 대전시 |
|---|-----|-----|-----|
| 0 | 1 | 3 | 5 |
| 1 | 2 | 4 | 6 |

column 제거 : drop()

```
1 df = pd.DataFrame({'서울':[1,2], '부산':[3,4], '대전':[5,6]})  
2 df.drop(columns=['서울', '대전'])
```

| | 부산 |
|---|----|
| 0 | 3 |
| 1 | 4 |

index 초기화 : reset_index()

```
1 df = pd.DataFrame([10,20,30], index = ['A','B','C'])  
2 df
```

| | 0 |
|---|----|
| A | 10 |
| B | 20 |
| C | 30 |

```
1 df.reset_index()
```

| index | 0 |
|-------|------|
| 0 | A 10 |
| 1 | B 20 |
| 2 | C 30 |

DataFrame

데이터조회 : 인덱스

- df.loc[인덱스 이름]
- df.loc[[인덱스이름1, 인덱스이름2]]
- df.iloc[인덱스 번호]
- df.iloc[시작 : 끝]

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                     index=['A','B'],columns=['서울','부산'])  
3  
4 df.loc['A']
```

```
서울      1  
부산      2  
Name: A, dtype: int64
```

```
1 df.loc[['A','B']]
```

| | 서울 | 부산 |
|---|----|----|
| A | 1 | 2 |
| B | 3 | 4 |

```
1 df.iloc[0]
```

```
서울      1  
부산      2  
Name: A, dtype: int64
```

```
1 df.iloc[0:]
```

| | 서울 | 부산 |
|---|----|----|
| A | 1 | 2 |
| B | 3 | 4 |

DataFrame

데이터조회 : 인덱스,컬럼 동시사용

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                 index=['A','B'],columns=['서울','부산'])  
3  
4 df.loc['A']
```

```
서울    1  
부산    2  
Name: A, dtype: int64
```

```
1 df.loc['A', '서울']
```

(A인덱스의 서울컬럼의 value)

```
1
```

```
1 df.loc['B', '부산']
```

(B인덱스의 부산컬럼의 value)

```
4
```

```
1 df.loc['A'][:]
```

(A인덱스의 모든 컬럼의 value)

```
서울    1  
부산    2  
Name: A, dtype: int64
```

```
1 df.loc['A'][['서울','부산']]
```

(A인덱스의 서울,부산 컬럼의 value)

```
서울    1  
부산    2  
Name: A, dtype: int64
```

```
1 df = pd.DataFrame([[1,2],[3,4]],  
2                 index=['A','B'],columns=['서울','부산'])  
3  
4 df.iloc[0]
```

```
서울    1  
부산    2  
Name: A, dtype: int64
```

```
1 df.iloc[0,0]
```

(0번 인덱스의 0번 컬럼의 value)

```
1
```

```
1 df.iloc[1,1]
```

(1번 인덱스의 1번 컬럼의 value)

```
4
```

```
1 df.iloc[0][:]
```

(0번 인덱스의 모든 컬럼의 value)

```
서울    1  
부산    2  
Name: A, dtype: int64
```

```
1 df.iloc[0][[0,1]]
```

(0번 인덱스의 0번,1번 컬럼의 value)

```
서울    1  
부산    2  
Name: A, dtype: int64
```

DataFrame

조건으로 열의 값 추출 - 서울 컬럼의 값이 10보다 큰 데이터 추출

```
1 df = pd.DataFrame({'서울':[10,20,30], '부산':[2,3,4], '대전':[5,6,7]})  
2 df[df['서울']>10]
```

| 서울 | 부산 | 대전 |
|----|----|----|
| 1 | 20 | 3 |
| 2 | 30 | 4 |

서울 컬럼의 값이 20이 아닌 데이터 추출

```
1 df[df['서울']!=20]
```

| 서울 | 부산 | 대전 |
|----|----|----|
| 0 | 10 | 2 |
| 2 | 30 | 4 |

조건으로 열의 값 추출 - 서울 컬럼의 값이 10인 데이터 추출

```
1 df = pd.DataFrame({'서울':[10,20,30], '부산':[2,3,4], '대전':[5,6,7]})  
2 df[df['서울']==10]
```

| 서울 | 부산 | 대전 |
|----|----|----|
| 0 | 10 | 2 |

DataFrame

데이터 샘플링 - 비율

```
1 df = pd.DataFrame({'서울':[10,20,30,40,50,60],  
2                 '부산':[1,2,3,4,5,6],  
3                 '대전':[5,6,7,8,9,10]})  
4 df
```

| | 서울 | 부산 | 대전 |
|---|----|----|----|
| 0 | 10 | 1 | 5 |
| 1 | 20 | 2 | 6 |
| 2 | 30 | 3 | 7 |
| 3 | 40 | 4 | 8 |
| 4 | 50 | 5 | 9 |
| 5 | 60 | 6 | 10 |

```
1 df.sample(frac=0.5)
```

| | 서울 | 부산 | 대전 |
|---|----|----|----|
| 1 | 20 | 2 | 6 |
| 3 | 40 | 4 | 8 |
| 2 | 30 | 3 | 7 |

데이터 샘플링 - 갯수

```
1 df = pd.DataFrame({'서울':[10,20,30,40,50,60],  
2                 '부산':[1,2,3,4,5,6],  
3                 '대전':[5,6,7,8,9,10]})  
4 df
```

| | 서울 | 부산 | 대전 |
|---|----|----|----|
| 0 | 10 | 1 | 5 |
| 1 | 20 | 2 | 6 |
| 2 | 30 | 3 | 7 |
| 3 | 40 | 4 | 8 |
| 4 | 50 | 5 | 9 |
| 5 | 60 | 6 | 10 |

```
1 df.sample(n=4)
```

| | 서울 | 부산 | 대전 |
|---|----|----|----|
| 1 | 20 | 2 | 6 |
| 0 | 10 | 1 | 5 |
| 2 | 30 | 3 | 7 |
| 4 | 50 | 5 | 9 |

DataFrame 합치기

merge 함수

- pd.merge(left, right, how , on)
- left : 합칠 왼쪽 데이터프레임
- right : 합칠 오른쪽 데이터프레임
- how : inner , left, right, outer
- on : 두 데이터를 합칠 기준 컬럼

how 옵션

- Inner : left, right에서 둘다 존재하는 컬럼 데이터만 합침
- left : left 데이터 프레임 기준으로 합침
- right : right 데이터프레임 기준으로 합침
- outer : left,right의 모든 data합침

```
1 data = pd.DataFrame([[ '강남',1,2],[ '서초',4,5],[ '노원',5,6]])
2 data.columns = [ '지역명', '번호', '매출']
3 data
```

| | 지역명 | 번호 | 매출 |
|---|-----|----|----|
| 0 | 강남 | 1 | 2 |
| 1 | 서초 | 4 | 5 |
| 2 | 노원 | 5 | 6 |

```
1 data1 = pd.DataFrame([[ '강남',10,20],[ '도봉',40,50],[ '노원',50,60]])
2 data1.columns = [ '지역명', '번호', '매출']
3 data1
```

| | 지역명 | 번호 | 매출 |
|---|-----|----|----|
| 0 | 강남 | 10 | 20 |
| 1 | 도봉 | 40 | 50 |
| 2 | 노원 | 50 | 60 |

```
1 pd.merge(left=data , right=data1, how='inner' , on='지역명')
```

| | 지역명 | 번호_x | 매출_x | 번호_y | 매출_y |
|---|-----|------|------|------|------|
| 0 | 강남 | 1 | 2 | 10 | 20 |
| 1 | 노원 | 5 | 6 | 50 | 60 |

(data, data1을 지역명 컬럼기준으로 둘다 존재하는 데이터만 결합)

DataFrame 합치기 - column 기준

```
1 pd.merge(left=data , right=data1, how='left', on='지역명')
```

| | 지역명 | 번호_x | 매출_x | 번호_y | 매출_y |
|---|-----|------|------|------|------|
| 0 | 강남 | 1 | 2 | 10.0 | 20.0 |
| 1 | 서초 | 4 | 5 | NaN | NaN |
| 2 | 노원 | 5 | 6 | 50.0 | 60.0 |

(left데이터 지역명 기준으로 right데이터 결합)

```
1 pd.merge(left=data , right=data1, how='outer', on='지역명')
```

| | 지역명 | 번호_x | 매출_x | 번호_y | 매출_y |
|---|-----|------|------|------|------|
| 0 | 강남 | 1.0 | 2.0 | 10.0 | 20.0 |
| 1 | 서초 | 4.0 | 5.0 | NaN | NaN |
| 2 | 노원 | 5.0 | 6.0 | 50.0 | 60.0 |
| 3 | 도봉 | NaN | NaN | 40.0 | 50.0 |

(left,right 모든 데이터를 결합)

```
1 pd.merge(left=data , right=data1, how='right', on='지역명')
```

| | 지역명 | 번호_x | 매출_x | 번호_y | 매출_y |
|---|-----|------|------|------|------|
| 0 | 강남 | 1.0 | 2.0 | 10 | 20 |
| 1 | 도봉 | NaN | NaN | 40 | 50 |
| 2 | 노원 | 5.0 | 6.0 | 50 | 60 |

(right데이터 지역명 기준으로 left데이터 결합)

DataFrame 합치기 - index 기준

```
1 data = pd.DataFrame([['강남',1,2],['서초',4,5],['노원',5,6]])
2 data.columns = ['지역명', '번호', '매출']
3 data.index = ['1월', '2월', '3월']
4 data
```

| | 지역명 | 번호 | 매출 |
|----|-----|----|----|
| 1월 | 강남 | 1 | 2 |
| 2월 | 서초 | 4 | 5 |
| 3월 | 노원 | 5 | 6 |

```
1 data1 = pd.DataFrame([['강남',10,20],['도봉',40,50],['노원',50,60]])
2 data1.columns = ['지역명', '번호', '매출']
3 data1.index = ['1월', '2월', '5월']
4 data1
```

| | 지역명 | 번호 | 매출 |
|----|-----|----|----|
| 1월 | 강남 | 10 | 20 |
| 2월 | 도봉 | 40 | 50 |
| 5월 | 노원 | 50 | 60 |

```
1 pd.merge(left=data , right=data1, left_index=True, right_index=True, how='inner')
```

| | 지역명_x | 번호_x | 매출_x | 지역명_y | 번호_y | 매출_y |
|----|-------|------|------|-------|------|------|
| 1월 | 강남 | 1 | 2 | 강남 | 10 | 20 |
| 2월 | 서초 | 4 | 5 | 도봉 | 40 | 50 |

DataFrame 연산

DataFrame 연산

```
1 data = pd.DataFrame([[1,2,3],[4,5,6]])
2 data.index = ['1월','2월']
3 data.columns = ['서울','대전','부산']
4 data
```

| | 서울 | 대전 | 부산 |
|----|----|----|----|
| 1월 | 1 | 2 | 3 |
| 2월 | 4 | 5 | 6 |

```
1 data1 = pd.DataFrame([[10,20,30],[40,50,60]])
2 data1.index = ['1월','2월']
3 data1.columns = ['서울','대전','부산']
4 data1
```

| | 서울 | 대전 | 부산 |
|----|----|----|----|
| 1월 | 10 | 20 | 30 |
| 2월 | 40 | 50 | 60 |

더하기

```
1 data+data1
```

| | 서울 | 대전 | 부산 |
|----|----|----|----|
| 1월 | 11 | 22 | 33 |
| 2월 | 44 | 55 | 66 |

곱하기

```
1 data*data1
```

| | 서울 | 대전 | 부산 |
|----|-----|-----|-----|
| 1월 | 10 | 40 | 90 |
| 2월 | 160 | 250 | 360 |

빼기

```
1 data-data1
```

| | 서울 | 대전 | 부산 |
|----|-----|-----|-----|
| 1월 | -9 | -18 | -27 |
| 2월 | -36 | -45 | -54 |

나누기

```
1 data/data1
```

| | 서울 | 대전 | 부산 |
|----|-----|-----|-----|
| 1월 | 0.1 | 0.1 | 0.1 |
| 2월 | 0.1 | 0.1 | 0.1 |

DataFrame 파일처리

- CSV(Comma-Separated Values) 파일은 구분자 콤마(,)로 구분된 텍스트 파일입니다.
- 각 줄은 레코드(record)이고, 각 레코드는 콤마(,)로 구분된 필드(field)로 이루어져 있습니다.
- CSV 파일은 데이터를 저장하고 공유하기 위한 가장 일반적인 파일 형식 중 하나입니다.
- CSV 파일은 데이터베이스, 스프레드시트, 과학 실험 결과 등 다양한 형태의 데이터를 저장하기에 적합합니다.

CSV 파일의 특징

1. 간단하고 가벼운 파일 형식이기 때문에 처리 속도가 빠릅니다.
2. 거의 모든 프로그램에서 지원하고, 데이터베이스나 스프레드시트 프로그램에서 쉽게 가져올 수 있습니다.
3. 콤마(,) 외에도 다른 구분자(delimiter)를 사용할 수 있습니다.
4. 문자열 필드에 콤마(,)가 포함될 경우, 해당 필드를 따옴표(")로 묶어 구분합니다.

DataFrame 파일처리

csv 파일 불러오기

- pd.read_csv(filePath, sep, header, names, index_col, skiprows, nrows, encoding)
- filePath : 파일경로 , sep : 구분자 , header(컬럼명) : 없을 경우 none
- names : header가 없을시 컬럼명 입력 가능 , skiprows : 파일에서 행을 건너뛰고 불러옴
- index_col : 컬럼을 index로 사용 , nrows : 입력한 개수만큼의 데이터만 읽음
- encoding : 인코딩 타입 입력, 데이터가 한글이면 'CP949'

```
1 df = pd.read_csv('cdata.csv')
2 df

-----
UnicodeDecodeError                                 Traceback (most recent call last)
<ipython-input-151-fe0a75ed0f10> in <module>
----> 1 df = pd.read_csv('cdata.csv')
      2 df

          ^ 9 frames 
/usr/local/lib/python3.9/dist-packages/pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xb1 in position 25: invalid start byte

SEARCH STACK OVERFLOW
```

(csv 파일에 한글 데이터가 존재할 때 encoding 없이 불러오면
생기는 오류)

```
1 df = pd.read_csv('cdata.csv',encoding='cp949')
2 df
```

| | Unnamed: 0 | Index | No | Age | City | edit |
|---|------------|-------|-----|-----|------|------|
| 0 | 0 | 1 | 홍길동 | 25 | 서울 | |
| 1 | 1 | 2 | 김준기 | 21 | 서울 | |
| 2 | 2 | 9 | 이명식 | 22 | 부산 | |
| 3 | 3 | 32 | 방준혁 | 24 | 광주 | |
| 4 | 4 | 47 | 최명기 | 31 | 부산 | |

(csv 파일에 한글 데이터가 존재할 때 encoding 옵션 선택
후 불러오기)

DataFrame 파일처리

column이 없는 파일 처리하기 - header = None

```
1 df = pd.read_csv('cdata_nohead.csv')
2 df
```

| 0 | 1 | 총길동 | 25 | 서울 |
|---|---|-----|----|----|
| 0 | 1 | 김준기 | 21 | 서울 |
| 1 | 2 | 이명식 | 22 | 부산 |
| 2 | 3 | 방준혁 | 24 | 광주 |
| 3 | 4 | 최명기 | 31 | 부산 |

```
1 df = pd.read_csv('cdata_nohead.csv',header=None)
2 df
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|----|-----|----|
| 0 | 0 | 1 | 총길동 | 25 |
| 1 | 1 | 2 | 김준기 | 21 |
| 2 | 2 | 9 | 이명식 | 22 |
| 3 | 3 | 32 | 방준혁 | 24 |
| 4 | 4 | 47 | 최명기 | 31 |

column이 없는 파일 처리하기 - names로 컬럼생성

```
1 df = pd.read_csv('cdata_nohead.csv',header=None
2 ,names=['index','no','이름','나이','지역'])
3 df
```

| index | no | 이름 | 나이 | 지역 |
|-------|----|----|-----|----|
| 0 | 0 | 1 | 총길동 | 25 |
| 1 | 1 | 2 | 김준기 | 21 |
| 2 | 2 | 9 | 이명식 | 22 |
| 3 | 3 | 32 | 방준혁 | 24 |
| 4 | 4 | 47 | 최명기 | 31 |

```
1 df = pd.read_csv('cdata_nohead.csv',header=None)
2 df.columns=['index','no','이름','나이','지역']
3 df
```

| index | no | 이름 | 나이 | 지역 |
|-------|----|----|-----|----|
| 0 | 0 | 1 | 총길동 | 25 |
| 1 | 1 | 2 | 김준기 | 21 |
| 2 | 2 | 9 | 이명식 | 22 |
| 3 | 3 | 32 | 방준혁 | 24 |
| 4 | 4 | 47 | 최명기 | 31 |

DataFrame 파일처리

index_col : 특정 컬럼을 인덱스로 지정

```
1 df = pd.read_csv('cdata_nohead.csv', header=None  
2                 , names=['index', 'no', '이름', '나이', '지역']  
3                 , index_col = 'index')  
4 df
```

| | no | 이름 | 나이 | 지역 | ✎ |
|-------|----|-----|----|----|---|
| index | | | | | |
| 0 | 1 | 홍길동 | 25 | 서울 | |
| 1 | 2 | 김준기 | 21 | 서울 | |
| 2 | 9 | 이명식 | 22 | 부산 | |
| 3 | 32 | 방준혁 | 24 | 광주 | |
| 4 | 47 | 최명기 | 31 | 부산 | |

excel파일 불러오기

```
1 df = pd.read_excel('cdata.xlsx')  
2 df
```

| index | no | 이름 | 나이 | 지역 | ✎ |
|-------|----|----|-----|----|----|
| 0 | 0 | 1 | 홍길동 | 25 | 서울 |
| 1 | 1 | 2 | 김준기 | 21 | 서울 |
| 2 | 2 | 9 | 이명식 | 22 | 부산 |
| 3 | 3 | 32 | 방준혁 | 24 | 광주 |
| 4 | 4 | 47 | 최명기 | 31 | 부산 |

```
df = pd.read_excel('example.xlsx', sheet_name='Sheet1')  
위와 같이 시트 이름도 지정 가능
```

DataFrame 통계

describe() 함수

- 데이터프레임(DataFrame)의 각 열(column)에 대한 요약 통계 정보를 제공

describe() 함수 통계 정보내용

- count : 열의 데이터 개수
- mean : 열의 평균값
- std : 열의 표준편차
- min : 열의 최소값
- 25% : 열의 25% 백분위수 값
- 50% : 열의 50% 백분위수 값(중앙값)
- 75% : 열의 75% 백분위수 값
- max : 열의 최대값

데이터프레임의 각 열에 대한 요약 통계 정보를 쉽게 확인

```
1 df = pd.read_csv('cdata.csv',encoding='cp949')
2           ,index_col = 'Index')
3 del df[ 'Unnamed: 0' ]
4 df
```

| No | Age | City | edit |
|--------------|-----|------|------|
| Index | | | |
| 1 | 홍길동 | 25 | 서울 |
| 2 | 김준기 | 21 | 서울 |
| 9 | 이명식 | 22 | 부산 |
| 32 | 방준혁 | 24 | 광주 |
| 47 | 최명기 | 31 | 부산 |

```
1 df.describe()
```

| Age | edit |
|-------|-----------|
| count | 5.000000 |
| mean | 24.600000 |
| std | 3.911521 |
| min | 21.000000 |
| 25% | 22.000000 |
| 50% | 24.000000 |
| 75% | 25.000000 |
| max | 31.000000 |

DataFrame 통계

sum() : 합계

```
1 df[ 'Age' ].sum()
```

123

var() : 분산

```
1 df[ 'Age' ].var()
```

15.3

min() : 최소값

```
1 df[ 'Age' ].min()
```

21

mean() : 평균

```
1 df[ 'Age' ].mean()
```

24.6

count() : 개수

```
1 df.count()
```

```
No      5  
Age     5  
City    5  
dtype: int64
```

max() : 최대값

```
1 df[ 'Age' ].max()
```

31

std() : 표준편차

```
1 df[ 'Age' ].std()
```

3.9115214431215892

corr() : 상관계수

```
1 df.corr()
```



결측치 처리

결측치 확인 : isnull()

```
1 df = pd.DataFrame({'서울':[10,np.nan,30,40,np.nan,60],  
2                 '부산':[1,2,np.nan,4,5,6],  
3                 '대전':[5,np.nan,7,8,9,np.nan]})  
4 df
```

| | 서울 | 부산 | 대전 |
|---|------|-----|-----|
| 0 | 10.0 | 1.0 | 5.0 |
| 1 | NaN | 2.0 | NaN |
| 2 | 30.0 | NaN | 7.0 |
| 3 | 40.0 | 4.0 | 8.0 |
| 4 | NaN | 5.0 | 9.0 |
| 5 | 60.0 | 6.0 | NaN |

```
1 df.isnull()
```

| | 서울 | 부산 | 대전 |
|---|-------|-------|-------|
| 0 | False | False | False |
| 1 | True | False | True |
| 2 | False | True | False |
| 3 | False | False | False |
| 4 | True | False | False |
| 5 | False | False | True |

null 제거 : dropna()

```
1 df.dropna()
```

| | 서울 | 부산 | 대전 |
|---|------|-----|-----|
| 0 | 10.0 | 1.0 | 5.0 |
| 3 | 40.0 | 4.0 | 8.0 |

null 대체 :fillna()

```
1 df.fillna(10)
```

| | 서울 | 부산 | 대전 |
|---|------|------|------|
| 0 | 10.0 | 1.0 | 5.0 |
| 1 | 10.0 | 2.0 | 10.0 |
| 2 | 30.0 | 10.0 | 7.0 |
| 3 | 40.0 | 4.0 | 8.0 |
| 4 | 10.0 | 5.0 | 9.0 |
| 5 | 60.0 | 6.0 | 10.0 |

```
1 df.fillna(df.mean())
```

| | 서울 | 부산 | 대전 |
|---|------|-----|------|
| 0 | 10.0 | 1.0 | 5.00 |
| 1 | 35.0 | 2.0 | 7.25 |
| 2 | 30.0 | 3.6 | 7.00 |
| 3 | 40.0 | 4.0 | 8.00 |
| 4 | 35.0 | 5.0 | 9.00 |
| 5 | 60.0 | 6.0 | 7.25 |

DataFrame 통계

sort_values() : 정렬

- `ascending` : True - 오름차순, False - 내림차순 정렬
- `inplace` : True 이면 정렬한 값을 DataFrame에 바로 반영
- `by` : 정렬할 기준 변수
- `axis` : 0 - 열방향 정렬, 1 - 행방향 정렬

```
1 df = pd.read_csv('cdata_nohead.csv',header=None
2                 ,names=['index','no','이름','나이','지역']
3                 ,index_col = 'index')
4 df.sort_values(by='나이')
```

| no | 이름 | 나이 | 지역 |
|----|-----|----|----|
| 1 | 김준기 | 21 | 서울 |
| 2 | 이명식 | 22 | 부산 |
| 3 | 방준혁 | 24 | 광주 |
| 0 | 홍길동 | 25 | 서울 |
| 4 | 최명기 | 31 | 부산 |

(매출컬럼 기준으로 오름차순 정렬)

```
1 df.sort_values(by='나이',ascending=False)
```

| no | 이름 | 나이 | 지역 |
|--------------|-----|----|----|
| index | | | |
| 4 | 최명기 | 31 | 부산 |
| 0 | 홍길동 | 25 | 서울 |
| 3 | 방준혁 | 24 | 광주 |
| 2 | 이명식 | 22 | 부산 |
| 1 | 김준기 | 21 | 서울 |

(매출컬럼 기준으로 내림차순 정렬)

DataFrame 통계

sort_index() : index명 기준으로 정

```
1 df = pd.read_csv('cdata_nohead.csv',header=None  
2                 ,names=['index','no','이름','나이','지역']  
3                 ,index_col = 'index')  
4 df.sort_values(by='나이')
```

| no | 이름 | 나이 | 지역 |
|-------|-----|----|----|
| index | | | |
| 1 | 김준기 | 21 | 서울 |
| 2 | 이명식 | 22 | 부산 |
| 3 | 방준혁 | 24 | 광주 |
| 0 | 홍길동 | 25 | 서울 |
| 4 | 최명기 | 31 | 부산 |

```
1 df.sort_index()
```

| no | 이름 | 나이 | 지역 |
|-------|-----|----|----|
| index | | | |
| 0 | 홍길동 | 25 | 서울 |
| 1 | 김준기 | 21 | 서울 |
| 2 | 이명식 | 22 | 부산 |
| 3 | 방준혁 | 24 | 광주 |
| 4 | 최명기 | 31 | 부산 |

컬럼명 기준으로 정렬

```
1 df = pd.read_csv('cdata_nohead.csv',header=None  
2                 ,names=['index','no','이름','나이','지역']  
3                 ,index_col = 'index')  
4 df.sort_values(by='나이')
```

| no | 이름 | 나이 | 지역 |
|-------|-----|----|----|
| index | | | |
| 1 | 김준기 | 21 | 서울 |
| 2 | 이명식 | 22 | 부산 |
| 3 | 방준혁 | 24 | 광주 |
| 0 | 홍길동 | 25 | 서울 |
| 4 | 최명기 | 31 | 부산 |

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

| no | 나이 | 이름 | 지역 |
|-------|----|-----|----|
| index | | | |
| 0 | 25 | 홍길동 | 서울 |
| 1 | 21 | 김준기 | 서울 |
| 2 | 22 | 이명식 | 부산 |
| 3 | 24 | 방준혁 | 광주 |
| 4 | 31 | 최명기 | 부산 |

DataFrame 예제

seoul_wifi_data.csv 파일은 서울시의 공공wifi 설치 현황 데이터입니다. 각 구별 설치 현황을 파악하세요.

```
1 import pandas as pd  
2  
3 df = pd.read_csv('seoul_wifi_data.csv',encoding='cp949')  
4 df.head()
```

| | 구명 | 유형 | 설치주소 | 지역명 | 상세설치장소 | 설치기관(회사) | 설치위치(x좌표) | 설치위치(y좌표) | 콘텐츠아이디 |
|---|-----|-------|--------|----------------|-----------|----------|------------|-----------|----------|
| 0 | 강남구 | 복지센터등 | 아동복지센터 | 강남구 광평로34길 124 | 아동복지센터 1F | 서울시(시스코) | 127.088505 | 37.479804 | WF120001 |
| 1 | 강남구 | 복지센터등 | 아동복지센터 | 강남구 광평로34길 124 | 아동복지센터 1F | 서울시(시스코) | 127.088505 | 37.479804 | WF120002 |
| 2 | 강남구 | 복지센터등 | 아동복지센터 | 강남구 광평로34길 124 | 아동복지센터 1F | 서울시(시스코) | 127.088505 | 37.479804 | WF120003 |
| 3 | 강남구 | 복지센터등 | 아동복지센터 | 강남구 광평로34길 124 | 아동복지센터 1F | 서울시(시스코) | 127.088505 | 37.479804 | WF120004 |
| 4 | 강남구 | 복지센터등 | 아동복지센터 | 강남구 광평로34길 124 | 아동복지센터 2F | 서울시(시스코) | 127.088505 | 37.479804 | WF120005 |

(read_csv 함수를 이용하여 파일을 불러와서 상위 5개의 데이터를 확인)

DataFrame 예제

구명 컬럼을 value_counts()함수를 이용하여 구별 설치 현황을 파악하세요.

```
1 df[ '구명' ].value_counts()
```

```
중구      1248
강남구    895
구로구    805
마포구    756
노원구    712
양천구    654
은평구    605
강서구    591
서초구    555
서대문구  547
종로구    535
관악구    515
성동구    512
동대문구  505
용산구    490
금천구    481
영등포구  474
광진구    463
중랑구    416
송파구    402
성북구    384
도봉구    369
강북구    324
강동구    320
동작구    286
Name: 구명, dtype: int64
```

DataFrame 예제

mpg.csv 파일은 자동차 회사의 연비 데이터를 저장한 파일입니다.

컬럼내용

- manufacturer : 회사명
- cty : 도심연비
- hwy : 고속도로 연비

위 컬럼을 이용하여 다음문제를 해결하세요.

1. 몇 개의 회사데이터가 있는지 알아보세요.
2. 회사별로 참여한 자동차가 몇 대인지 파악하세요. (현대 : 20 , 도요타 : 30)
3. 도심연비와 고속도로 연비를 평균낸 total 연비를 구하세요.
4. total 연비 상위 10개 회사의 회사별 개수를 구하세요.
5. 평균 total 연비보다 높은 자동차는 PASS, 낮은 자동차는 FAIL로 구분하세요.
6. PASS 자동차 대수와 FAIL자동차 대수를 구하세요.

DataFrame 예제

```
1 import pandas as pd  
2 data = pd.read_csv('mpg.csv')  
3 data.head()
```

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | fl | category |
|---|--------------|-------|-------|------|-----|------------|-----|-----|-----|----|----------|
| 0 | audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p | compact |
| 1 | audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p | compact |
| 2 | audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p | compact |
| 3 | audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p | compact |
| 4 | audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | compact |

1. pandas 패키지를 임포트 한후 mpg.csv 파일을 불러옴
data의 상위 5개의 데이터를 확인

```
1 data.tail()
```

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | fl | category |
|-----|--------------|--------|-------|------|-----|------------|-----|-----|-----|----|----------|
| 229 | volkswagen | passat | 2.0 | 2008 | 4 | auto(s6) | f | 19 | 28 | p | midsize |
| 230 | volkswagen | passat | 2.0 | 2008 | 4 | manual(m6) | f | 21 | 29 | p | midsize |
| 231 | volkswagen | passat | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | midsize |
| 232 | volkswagen | passat | 2.8 | 1999 | 6 | manual(m5) | f | 18 | 26 | p | midsize |
| 233 | volkswagen | passat | 3.6 | 2008 | 6 | auto(s6) | f | 17 | 26 | p | midsize |

2. tail () 함수를 이용하여 아래 5개의 데이터 확인

DataFrame 예제

```
1 data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 234 entries, 0 to 233
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   manufacturer 234 non-null    object  
 1   model        234 non-null    object  
 2   displ         234 non-null    float64
 3   year          234 non-null    int64  
 4   cyl           234 non-null    int64  
 5   trans         234 non-null    object  
 6   drv           234 non-null    object  
 7   cty           234 non-null    int64  
 8   hwy           234 non-null    int64  
 9   fl            234 non-null    object  
 10  category      234 non-null    object  
dtypes: float64(1), int64(4), object(6)
memory usage: 20.2+ KB
```

3. info() 함수를 이용하여 데이터 탑 확인

```
1 data.describe(include='all')
```

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | f1 | category |
|--------|--------------|-------------|------------|-------------|------------|----------|-----|------------|------------|-----|----------|
| count | 234 | 234 | 234.000000 | 234.000000 | 234.000000 | 234 | 234 | 234.000000 | 234.000000 | 234 | 234 |
| unique | 15 | 38 | NaN | NaN | NaN | 10 | 3 | NaN | NaN | 5 | 7 |
| top | dodge | caravan 2wd | NaN | NaN | NaN | auto(l4) | f | NaN | NaN | r | suv |
| freq | 37 | 11 | NaN | NaN | NaN | 83 | 106 | NaN | NaN | 168 | 62 |
| mean | NaN | NaN | 3.471795 | 2003.500000 | 5.888889 | NaN | NaN | 16.858974 | 23.440171 | NaN | NaN |
| std | NaN | NaN | 1.291959 | 4.509646 | 1.611534 | NaN | NaN | 4.255946 | 5.954643 | NaN | NaN |
| min | NaN | NaN | 1.600000 | 1999.000000 | 4.000000 | NaN | NaN | 9.000000 | 12.000000 | NaN | NaN |
| 25% | NaN | NaN | 2.400000 | 1999.000000 | 4.000000 | NaN | NaN | 14.000000 | 18.000000 | NaN | NaN |
| 50% | NaN | NaN | 3.300000 | 2003.500000 | 6.000000 | NaN | NaN | 17.000000 | 24.000000 | NaN | NaN |
| 75% | NaN | NaN | 4.600000 | 2008.000000 | 8.000000 | NaN | NaN | 19.000000 | 27.000000 | NaN | NaN |
| max | NaN | NaN | 7.000000 | 2008.000000 | 8.000000 | NaN | NaN | 35.000000 | 44.000000 | NaN | NaN |

4. describe()로 요약통계확인

DataFrame 예제

```
1 data[ 'manufacturer' ].describe()
```

```
count      234
unique     15
top       dodge
freq      37
Name: manufacturer, dtype: object
```

5. manufacturer 컬럼의 요약통계 확인

```
1 data[ 'manufacturer' ].unique()
```

```
array(['audi', 'chevrolet', 'dodge', 'ford', 'honda', 'hyundai', 'jeep',
       'land rover', 'lincoln', 'mercury', 'nissan', 'pontiac', 'subaru',
       'toyota', 'volkswagen'], dtype=object)
```

6. unique() 함수로 위와 같은 15개의 회사확인

```
1 data[ 'manufacturer' ].value_counts()
```

```
dodge      37
toyota    34
volkswagen 27
ford      25
chevrolet 19
audi      18
hyundai   14
subaru    14
nissan    13
honda     9
jeep      8
pontiac   5
land rover 4
mercury   4
lincoln   3
Name: manufacturer, dtype: int64
```

7. value_counts() 함수로 회사별 차량수 파악

DataFrame 예제

```
1 data10 = data.sort_values(by = 'total',axis = 0,ascending=False).head(10)
2 data10['manufacturer'].value_counts()
```

```
honda      4
volkswagen 3
toyota     3
Name: manufacturer, dtype: int64
```

8. total연비 상위 10개 회사의 카운트 구함

```
1 import numpy as np
2 data['pass'] = np.where(data['total']>=data['total'].mean() , 'pass' , 'fail' )
3 data
```

| | manufacturer | model | displ | year | cyl | trans | drv | cty | hwy | f1 | category | total | pass |
|---|--------------|-------|-------|------|-----|------------|-----|-----|-----|----|----------|-------|------|
| 0 | audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p | compact | 23.5 | pass |
| 1 | audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p | compact | 25.0 | pass |
| 2 | audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p | compact | 25.5 | pass |
| 3 | audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p | compact | 25.5 | pass |
| 4 | audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p | compact | 21.0 | pass |

9. 평균 이상이면 PASS 이하면 FAIL로 정의

DataFrame 예제

```
1 data[ 'pass' ].value_counts()
```

```
pass      123
fail      111
Name: pass, dtype: int64
```

10. value_counts()를 이용하여 pass와 fail 카운트