가천대학교
Gachon University

04

Basic Probability & Statistics

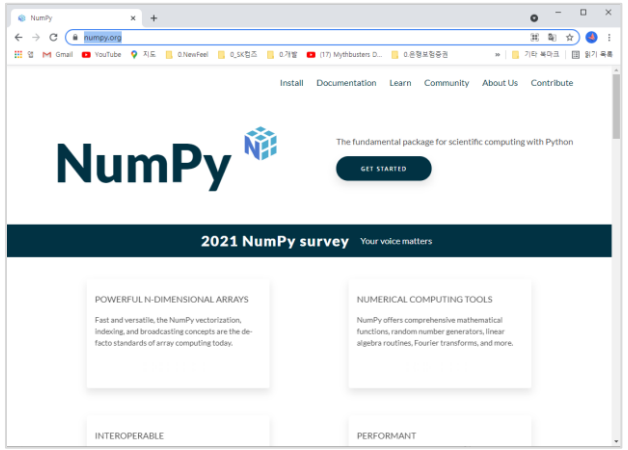
확률과 통계
(numpy, pandas)

조윤실

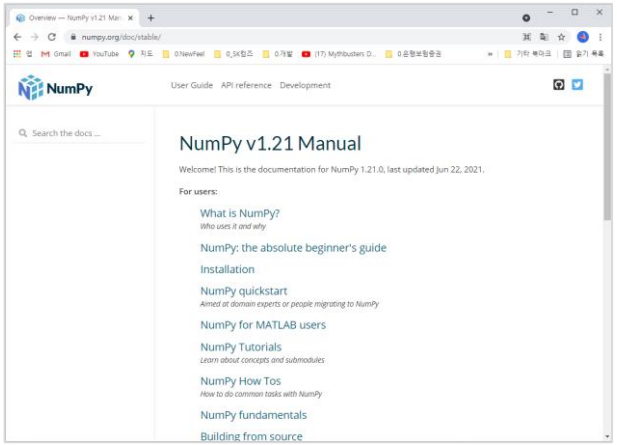
데이터 분석 기초

CONTENTS

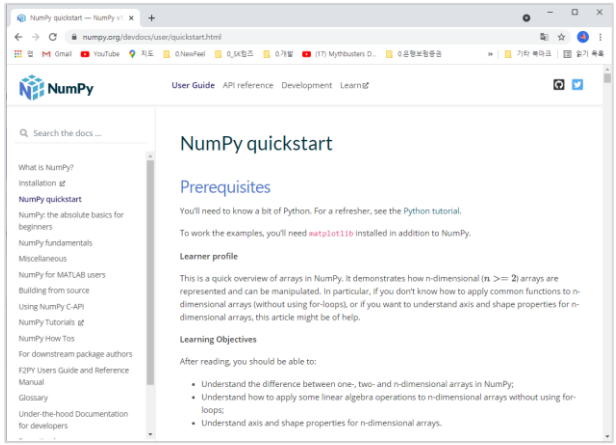
Numpy 관련 사이트



https://numpy.org/



https://numpy.org/doc/stable/



https://numpy.org/devdocs/user/quickstart.html

01 numpy

01. numpy 배열
02. 배열의 생성과 변형
03. 배열의 연산
04. 기술 통계
05. 난수 발생

01. numpy 배열

numpy

- **numpy**

https://numpy.org/

  - Numerical Python의 줄임말
  - 고성능의 수치계산을 위해 C언어로 구현된 수치해석용 파이썬 패키지
  - 벡터 및 행렬 연산을 하는 선형대수에 있어서 매우 편리한 기능을 제공
  - 파이썬은 자체적으로 배열 자료형을 제공하지 않음
  - 배열을 사용하는 패키지인 Numpy를 import해서 사용함
  - Numpy의 배열(array) 구조 → ndarray: n dimension array

배열(ndarray)	리스트(list)
연속된 메모리 구조 동작 속도가 빠르다 comma를 사용하지 않고 공백으로 요소 구별 오직 한 가지 타입만 가질 수 있음	Linked List구조(불연속) 동작 속도가 느리다 comma로 요소구별 서로 다른 자료형 사용가능

numpy 사용하기

- **numpy 라이브러리 설치**

  pip install numpy

- **numpy 사용**

```
1 import numpy as np
```

numpy 배열 만들기

- **1차원 배열 만들기**

  array라는 함수에 리스트를 넣으면 배열로 변환해 준다.

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

```
print(arr)
```

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

comma(,)가 없다

```
print(type(arr))
```

```
<class 'numpy.ndarray'>
```

2차원 배열 만들기

- numpy를 사용하여 아래와 같은 2차원 배열을 만들어 보세요.

numpy를 이용해 2차원 array를 만들어 보세요.

```
arr = np.array([[1,2,3,4],[5,6,7,8]])
print(arr*10)
```

numpy 배열 만들기

- **2차원 배열 만들기**

  2차원 배열은 **행렬(matrix)**이라고 하는데 행렬에서는 가로줄을 **행**(row)이라고 하고 세로줄을 **열**(column)이라고 부른다

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

```
[[1 2 3]
 [3 4 5]]
```

```
# 행의 개수
len(arr)
```

```
2
```

```
# 열의 개수
len(arr[0])
```

```
3
```

numpy 배열 만들기

- **3차원 배열 만들기**

```
d = np.array([[[1, 2, 3, 4],
               [5, 6, 7, 8],
               [9, 10, 11, 12]],
              [[11, 12, 13, 14],
               [15, 16, 17, 18],
               [19, 20, 21, 22]]])   # 2 x 3 x 4 array
print(d)
```

괄호의 개수를 확인해라!!!

```
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[11 12 13 14]
  [15 16 17 18]
  [19 20 21 22]]]
```

```
len(d), len(d[0]), len(d[0][0])
```

```
(2, 3, 4)
```

numpy 배열의 차원과 크기

- **배열의 차원과 크기 알아내기**

  배열의 차원: **ndim** 속성

  배열의 크기: **shape** 속성

```
arr = np.array([1, 2, 3])
print(arr.ndim)
print(arr.shape)
```

```
1
(3,)
```

1차원

- **numpy shape**
  - Array의 크기
  - 데이터의 개수, 몇 차원 데이터인지 확인
  - 1차원 Shape : (x,)
  - 2차원 Shape : (x,y) , 행렬(matrix)
  - 3차원 Shape : (x,y,z) , (면,행,열)
  - n차원 Shape : (x,y,z,....)

```
arr = np.array([[0, 1, 2], [3, 4, 5]])
print(arr.ndim)
print(arr.shape)
```

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

2차원

```
print(d.ndim)
print(d.shape)
```

```
3
(2, 3, 4)
```

3차원

numpy 배열의 크기 변경

- **배열의 크기 변경**

내부 데이터는 보존한 채로 형태만 변경: **reshape**

```
1 na = np.arange(12)
2 print(na)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
1 nb = na.reshape(3,4)
2 print(nb)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
1 nb = na.reshape(3,-1)  # -1 : 자동으로 값이 계산
2 print(nb)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
1 nb = na.reshape(2,2,-1)
2 print(nb)
```

```
[[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
```

```
1 nb = na.reshape(2,-1,2)
2 print(nb)
```

```
[[[ 0  1]
  [ 2  3]
  [ 4  5]]

 [[ 6  7]
  [ 8  9]
  [10 11]]]
```

numpy 배열의 크기 변경

- **1차원 배열로 변경**

다차원 배열을 무조건 1차원 배열로 변경하는 메소드: **flatten, ravel**

```
1 nc = nb.flatten()                    # nb.reshape(-1)
2 print(nc)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

```
1 nc = nb.ravel()
2 print(nc)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
```

원본을 변경하면 같이 변경된다.
- ravel(), reshape()

numpy 인덱싱 & 슬라이싱

- **배열의 인덱싱 & 슬라이싱**

```
arr = np.array([0, 1, 2, 3, 4])

print(arr[2])
print(arr[1:3])
print(arr[::-1])
```

```
2
[1 2]
[4 3 2 1 0]
```

```
arr = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])

print(arr[0,0])
print(arr[-1,-1])
print(arr[:,2])
print(arr[1,1:])
print(arr[:2,:2])
```

```
0
7
[2 6]
[5 6 7]
[[0 1]
 [4 5]]
```

numpy 데이터 타입

- **numpy는 한가지 데이터 타입을 가질 수 있다.**

 # Numpy Data types

https://numpy.org/doc/stable/user/basics.types.html

```
1 na = np.array([0,1,2,3.5,4,5,6,7,8,9])
2 print(na)
3 print(na.dtype)
```

```
[0.  1.  2.  3.5 4.  5.  6.  7.  8.  9. ]
float64
```

← 부동소수점 실수

```
1 na = np.array([0,1,2,3.5,'4',5,6,7,8,9])
2 print(na)
3 print(na.dtype)
```

```
['0' '1' '2' '3.5' '4' '5' '6' '7' '8' '9']
<U32
```

← 유니코드 문자열

numpy 데이터 타입

```
a = np.array([1, 2, 3], dtype='int64')
a = np.array([1, 2, 3], dtype='i8')
```

dtype	코드	설명	dtype	코드	설명
int8	i1	부호 있는 8비트 정수형	**float16**	f2	실수형 : 반 정밀도 부동소수점형 (부호 1비트, 지수 5비트, 가수 10비트)
int16	i2	부호 있는 16비트 정수형	**float32**	f4	실수형: 단 정밀도 부동소수점형 (부호 1비트, 지수 8비트, 가수 23비트)
int32	i4	부호 있는 32비트 정수형	**float64**	f8	실수형: 배 정밀도 부동소수점형 (부호 1비트, 지수 11비트, 가수 54비트)
int64	i8	부호 있는 64비트 정수형	**float128**	f16	실수형: 네배 정밀도 부동소수점형 (부호 1비트, 지수 15비트, 가수 112비트)
unit8	u1	부호 없는 8비트 정수형	**complex64**	c8	복소수 (실수부, 허수부 각각 float32)
unit16	u2	부호 없는 16비트 정수형	**complex128**	c16	복소수 (실수부, 허수부 각각 float64)
unit32	u4	부호 없는 32비트 정수형	**complex256**	c32	복소수 (실수부, 허수부 각각 float128)
unit64	u8	부호 없는 64비트 정수형	**bool**	?	불형 (true 또는 false)
unicode	U	unicode문자열	**object**	0	Python 오브젝트형

numpy 데이터 타입

- **numpy의 dtype**

Python 데이터 형	dtype 데이터 형
Int	int 64
float	float 63
string	unicode

- **numpy 숫자형이 취할 수 있는 범위 확인 방법**
  - **numpy.iinfo(*type*)** ex) np.iinfo('int64')

    https://numpy.org/doc/stable/reference/generated/numpy.iinfo.html
  - **numpy.finfo(*type*)**

    https://numpy.org/doc/stable/reference/generated/numpy.finfo.html

numpy 데이터 타입

- **numpy 숫자형이 취할 수 있는 범위(최소값, 최대값) 확인 예**

```
1 print(np.iinfo('int16'))
```

→ 정수 int, unit이 취할 수 있는 범위

```
Machine parameters for int16
---------------------------------------------------------------
min = -32768
max = 32767
---------------------------------------------------------------
```

```
1 print(np.finfo('float'))
```

→ 부동 소수점 float이 취할 수 있는 범위

```
Machine parameters for float64
---------------------------------------------------------------
precision =  15   resolution = 1.000000000000001e-15
machep =    -52   eps =        2.2204460492503131e-16
negep =     -53   epsneg =     1.1102230246251565e-16
minexp =  -1022   tiny =       2.2250738585072014e-308
maxexp =   1024   max =        1.7976931348623157e+308
nexp =       11   min =        -max
---------------------------------------------------------------
```

numpy 데이터 타입

- **numpy 데이터 타입 변환**

```
1  na = np.array([0,1,2,3,4])
2  print(na)
3  print(na.dtype)
4
5  na = na.astype(np.float)
6  print(na)
7  print(na.dtype)
```

```
[0 1 2 3 4]
int32
[0. 1. 2. 3. 4.]
float64
```

 # astype()으로 변환할 때 주의점!

- float에서 int로 변환할 경우 소수점 이하는 잘리고, 마이너스값은 0으로 반올림 됨

- np.round()을 사용하는 경우 짝수에 대한 반올림이 됨, 반올림할 자리의 수가 5이면 반올림 할 때 앞자리의 숫자가 짝수면 내림하고 홀수면 올림함
  >>> round(4.5) *#결과는 4*
  >>> round(3.5) *#결과는 4*

02. 배열의 생성과 변형

다양한 numpy 배열 생성

- **다양한 배열 생성 방법**

  - ✓ **zeros, ones :** 값을 0 , 1로 채워진 배열 생성
  - ✓ **zeros_like, ones_like :** 다른 배열과 같은 크기의 배열 생성 na = np.ones_like(na)
  - ✓ **empty :** 배열만 생성 (값을 초기화 하지 않음)
  - ✓ **arange :** range와 유사한 명령으로 배열 생성
  - ✓ **linspace :** 지정한 구간의 수만큼 분할한 선형구간 배열 생성
  - ✓ **logspace :** 지정한 구간의 수만큼 분할한 로그 구간 배열 생성

배열의 연결

- **배열의 연결**

  행의 수나 열의 수가 같은 두 개 이상의 배열을 연결하여 더 큰 배열을 만드는 명령어

  ✓ **hstack :** 행의 수가 같은 두 개 이상의 배열을 옆으로 연결

  ✓ **vstack :** 열의 수가 같은 두 개 이상의 배열을 위아래로 연결

  ✓ **dstack :** 행이나 열이 아닌 깊이(depth) 방향으로 배열을 합침.
  가장 안쪽 원소의 차원 증가

  ✓ **stack :** axis 인수를 사용하여 사용자가 지정한 차원(축으로) 배열을 연결
  가장 앞쪽에 차원이 생성됨, 배열들의 크기가 모두 같아야 함

  ✓ **r_ :** hstack 명령과 비슷, 대괄호 [ ] 사용하는 메서드

  ✓ **c_ :** 배열의 차원을 증가시킨 후 좌우로 연결하는 메서드

  ✓ **tile :** 동일한 배열을 반복하여 연결

배열 연결하여 만들기

- 아래와 같은 배열을 만들어 보세요.

```
array([[  0.,   0.,   0.,   1.,   1.],
       [  0.,   0.,   0.,   1.,   1.],
       [  0.,   0.,   0.,   1.,   1.],
       [ 10.,  20.,  30.,  40.,  50.],
       [ 60.,  70.,  80.,  90., 100.],
       [110., 120., 130., 140., 150.],
       [  0.,   0.,   0.,   1.,   1.],
       [  0.,   0.,   0.,   1.,   1.],
       [  0.,   0.,   0.,   1.,   1.],
       [ 10.,  20.,  30.,  40.,  50.],
       [ 60.,  70.,  80.,  90., 100.],
       [110., 120., 130., 140., 150.]])
```

Inf & NaN

- **무한대 표현**

  inf : infinity

- **정의할 수 없는 숫자**

  nan : not a number

03. 배열의 연산

배열의 연산

- numpy 배열을 이용하여 아래 z = x + y 를 코딩해 보세요.

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix}, \quad y = \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix},$$

$$z = x + y$$

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ \vdots \\ 10000 \end{bmatrix} + \begin{bmatrix} 10001 \\ 10002 \\ 10003 \\ \vdots \\ 20000 \end{bmatrix} = \begin{bmatrix} 1 + 10001 \\ 2 + 10002 \\ 3 + 10003 \\ \vdots \\ 10000 + 20000 \end{bmatrix} = \begin{bmatrix} 10002 \\ 10004 \\ 10006 \\ \vdots \\ 30000 \end{bmatrix}$$

벡터화 연산

- **벡터화 연산(vectorized operation)**

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

x = np.array(data)

print(2 * x)
```

```
[ 0  2  4  6  8 10 12 14 16 18]
```

```
In [17]:
```

```
a = np.array([1, 2, 3])
b = np.array([10, 20, 30])

print(2 * a + b)
```

```
[12 24 36]
```

```
print(a == 2)
```

```
[False  True False]
```

```
In [20]:
```

```
print(b > 10)
```

```
[False  True  True]
```

```
In [25]:
```

```
print((a == 2) & (b > 10))
```

```
[False  True False]
```

벡터화 연산

- **배열 연산**

```
arr1 = np.array([[1,2,3],[4,5,6]])        # (2,3)
arr2 = np.array([[10,11,12],[13,14,15]]) # (2,3)
arr3 = np.array([10,11,12])                # (3,)
```

```
# array 덧셈
print(arr1 +  arr2)
```

```
[[11 13 15]
 [17 19 21]]
```

```
print(arr1 + arr3)
```

```
[[11 13 15]
 [14 16 18]]
```

```
print(arr1 + 2)
```

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

```
# array 뺄셈
print(arr1 -  arr2)
```

```
[[-9 -9 -9]
 [-9 -9 -9]]
```

```
print(arr3 - arr1)
```

```
[[9 9 9]
 [6 6 6]]
```

```
print(arr2 - 5)
```

```
[[ 5  6  7]
 [ 8  9 10]]
```

```
# array 곱셈
print(arr1 *  arr2)
```

```
[[10 22 36]
 [52 70 90]]
```

```
print(arr1 *  arr3)
```

```
[[10 22 36]
 [40 55 72]]
```

```
print(arr1 *  5)
```

```
[[ 5 10 15]
 [20 25 30]]
```

```
# array 나눗셈
print(arr1 /  arr2)
```

```
[[0.1        0.18181818 0.25      ]
 [0.30769231 0.35714286 0.4       ]]
```

```
print(arr1 /  arr3)
```

```
[[0.1        0.18181818 0.25      ]
 [0.4        0.45454545 0.5       ]]
```

```
print(arr1 /  2)
```

```
[[0.5 1.  1.5]
 [2.  2.5 3. ]]
```

벡터화 연산

- 다음 배열에서 아래 문제를 해결해 보세요.

```
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
              11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
```

1.이 배열에서 3의 배수를 찾아라.

2.이 배열에서 4로 나누면 1이 남는 수를 찾아라.

3.이 배열에서 3으로 나누면 나누어지고 4로 나누면 1이 남는 수를 찾아라.

벡터화 연산

- **브로드캐스팅(broadcasting)**

  벡터(또는 행렬)끼리 덧셈 혹은 뺄셈을 하려면 두 벡터(또는 행렬)의 크기가 같아야 한다

  넘파이에서는 서로 다른 크기를 가진 두 배열의 사칙 연산을 지원함.

  크기가 작은 배열을 자동으로 반복 확장하여 크기가 큰 배열에 맞추는 방법

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

팬시 인덱싱

- **팬시 인덱싱(fancy indexing)**

  인덱스를 통해 배열의 해당 요소에 빠르게 접근하는 방법.

  다른 배열로 배열을 인덱싱 하는 방법

```
# 짝수인 원소만 골라내려면
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
idx = np.array([True, False, True, False, True,
                False, True, False, True, False])
```

```
print(arr[idx])
```

```
[0 2 4 6 8]
```

```
print(arr % 2)
```

```
[0 1 0 1 0 1 0 1 0 1]
```

```
print(arr % 2 == 0)
```

```
[ True False  True False  True False  True False
True False]
```

```
print(arr[arr % 2 == 0])
```

```
[0 2 4 6 8]
```

팬시 인덱싱

① 불린(Boolean) 배열 인덱싱(마스크)

Boolean값을 갖는 마스크를 씌워서 원하는 행렬을 뽑아낼 수 있다

```
names = np.array(['Paul','Paul','Kim','Joan','Lee','Paul','Park'])
data = np.random.rand(7,4)
```

```
print(data)
```

```
[[0.09483565 0.07076099 0.01841305 0.15574146]
 [0.80252536 0.94362766 0.41497645 0.13616134]
 [0.54884556 0.61132454 0.41297004 0.437656  ]
 [0.63100153 0.12035142 0.72316974 0.50573557]
 [0.68255251 0.5446599  0.54084956 0.29680945]
 [0.87752972 0.81739704 0.18685608 0.25091159]
 [0.91739579 0.28068239 0.97233006 0.76582192]]
```

```
# 마스크 만들기
names_mask = (names == 'Paul')
print(names_mask)
```

```
[ True  True False False False  True False]
```

팬시 인덱싱

```
# 마스크 적용하기
# data 중에 names_mask의 값이 True인 행만 출력
print(data[names_mask, :])
```

```
[[0.09483565 0.07076099 0.01841305 0.15574146]
 [0.80252536 0.94362766 0.41497645 0.13616134]
 [0.87752972 0.81739704 0.18685608 0.25091159]]
```

```
# data 중에 요소가 Kim과 같은 행이— 데이터 추출
print(data[names=='Kim', :])
```

```
[[0.54884556 0.61132454 0.41297004 0.437656  ]]
```

```
print(data[(names=='Kim')|(names=='Park'), :] ) # ()괄호를 반드시 쳐야함
```

```
[[0.54884556 0.61132454 0.41297004 0.437656  ]
 [0.91739579 0.28068239 0.97233006 0.76582192]]
```

팬시 인덱싱

- **팬시 인덱싱(fancy indexing)**

② 정수 배열 인덱싱

```
arr = np.array([11, 22, 33, 44, 55, 66, 77, 88, 99])
idx = np.array([0, 2, 4, 6, 8])
```

```
print(arr[idx])
```

```
[11 33 55 77 99]
```

```
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

```
print(arr[:, [True, False, False, True]])
```

```
[[ 1  4]
 [ 5  8]
 [ 9 12]]
```

```
print(arr[[2, 0, 1], :])
```

```
[[ 9 10 11 12]
 [ 1  2  3  4]
 [ 5  6  7  8]]
```

04. 기술 통계

numpy 기술 통계 함수

- **기술 통계 함수**

  Numpy는 아래와 같은 데이터 집합에 대해 간단한 통계량을 구할 수 있는 함수를 제공한다

  - ✓ **데이터 개수(count) :**
  - ✓ **평균(mean, average) :** 표본 평균 $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x$
  - ✓ **분산(variance) :** 표본 분산 $s^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$
  - ✓ **표준편차(standard deviation) :** 표본 표준편차 $s = \sqrt{s^2}$
  - ✓ **최댓값(maximum) :** 데이터 중에서 가장 큰 값
  - ✓ **최솟값(minimum) :** 데이터 중에서 가장 작은 값
  - ✓ **중앙값(median) :** 데이터를 크기대로 정렬하였을 때 가장 가운데에 있는 수
  - ✓ **사분위수(quartile) :** 데이터를 가장 작은 수부터 가장 큰 수까지 크기가 커지는 순서대로 정렬하였을 때 1/4, 2/4, 3/4 위치에 있는 수, 각각 1사분위수, 2사분위수, 3사분위수라고 함

기술 통계 함수 실습

- 아래 표본 집합 x를 가지고 기술 통계 함수를 실행해 h세요.

```
x = {18,5,10,23,19,-8,10,0,0,5,2,15,8,2,5,4,15,-1,4,-7,-24,7,9,-6,23,-13}
```

- ✓ **데이터 개수(count)** : len(x)
- ✓ **평균(mean, average) :** np.mean(x)
- ✓ **분산(variance) :** np.var(x)
- ✓ **표준편차(standard deviation) :** np.std(x)
- ✓ **최댓값(maximum) :** np.max(x), cf) np.argmax(x)
- ✓ **최솟값(minimum) :** np.min(x) cf) np.argmin(x)
- ✓ **중앙값(median) :** np.median(x)
- ✓ **사분위수(quartile) :** np.percentile(x, 25) # 1사분위 수, np.percentile(x, 50) # 2사분위 수
  np.percentile(x, 0) # 최소값, np.percentile(x, 100) # 최대값

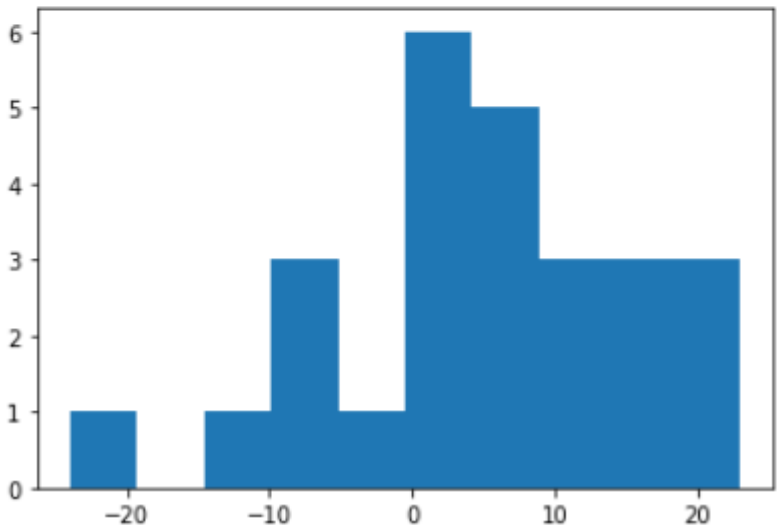
기술 통계 함수 실습

- 히스토그램 그리기

```
1 bins = np.arange(-25,25,5)  # 도수분포구간
2 hist, bins = np.histogram(x, bins)
3 print (hist)
4 print (bins)
```

```
[1 0 1 3 1 6 6 2 4]
[-25 -20 -15 -10  -5   0   5  10  15  20]
```

```
1 import matplotlib.pyplot as plt
2 plt.hist(x)
3 plt.show()
```



05. 난수 발생

난수 발생

- **난수 관련 함수**

  데이터를 무작위로 섞거나 임의의 수 즉, 난수(random number)를 발생시키는 방법은 numpy의 서브 패키지인 random에서 제공한다

  - ✓ **시드 설정:** np.random.seed(0)
  - ✓ **난수 발생:**

    rand 0부터 1사이의 균일 분포: ex) np.random.rand(5) (0과 1사이의 난수 5개 발생)

    randn 기댓값이 0이고 표준편차가 1인 표준 정규 분포: ex) np.random.randn(5)

    randint 균일 분포의 정수 난수: ex) np.random.randint(10, 20, size=5)
  - ✓ **데이터 순서 임의로 바꾸기:** ex) np.random.shuffle(x)
  - ✓ **데이터 샘플링 :** ex) np.random.choice(x, size=None, replace=True, p=None)

카운트, 정렬

- **카운트: bincount**

  배열 요소의 개수를 세는 메소드 ex) np.bincount(x)

- **유니크한 요소만 카운트: unique**

  배열 요소의 개수를 세는 메소드 ex) np.unique(x)

- **요소 정렬: sort**

  배열 요소의 개수를 세는 메소드 ex) np.sort(x)

난수 실습

- 주사위를 100번 던져서 나오는 숫자를 각 숫자별로 카운트하고 평균을 구하라.

- 가격이 10,000원인 주식이 있다. 이 주식의 일간 수익률(%)은 기댓값이 0%이고 표준편차가 1%인 표준 정규 분포를 따른다고 하자. 250일 동안의 주가를 무작위로 생성하라

02 pandas

01. pandas 사용하기
02. 배열의 생성과 변형
03. 배열의 연산
04. 기술 통계
05. 난수 발생

01. pandas 사용하기

Pandas

- **Pandas** https://pandas.pydata.org/
  - 파이썬 데이터 분석 라이브러리
  - 대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있음
  - 판다스는 이런 데이터를 다루기 위한 시리즈(series) 클래스와 데이터프레임(DataFrame)클래스를 제공하여 대용량의 데이터 처리에 편리한 패키지 라이브러리

- **Pandas 라이브러리 설치**

  pip install pandas

- **Pandas 사용**

```
import pandas as pd
```

Pandas 데이터 구조

- **Pandas에서 사용하는 데이터 구조**
  - Pandas에서 사용하는 데이터 구조는 **Series**와 **DataFrame**
  - 이 데이터 구조는 빅데이터 분석에 있어서 높은 수준의 성능을 보여준다.

1. **Series 객체:** 1차원 배열, 1차원 ndarray와 호환

   시리즈 = 인덱스(index) + 값(value)

2. **DataFrame 객체:** 2차원 배열, 서로 다른 자료형을 사용할 수 있다

   행(인스턴스), 열(컬럼, 피쳐(feature))

Series

- **Series 객체 정의하기**

```
# Series 정의하기
data = pd.Series([4, 7, -5, 3])

print(data)
print(type(data))
```

```
0    4
1    7
2   -5
3    3
dtype: int64
<class 'pandas.core.series.Series'>
```

```
print(data.values)

print(data.index)

print(data.dtype)
```

```
[ 4  7 -5  3]
RangeIndex(start=0, stop=4, step=1)
int64
```

Series

- **Series index 변경하기**

```
# 인덱스 바꾸기
data = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])
print(data)

data.index = ['A', 'B', 'C', 'D']
print(data)
```

```
d    4
b    7
a   -5
c    3
dtype: int64
A    4
B    7
C   -5
D    3
dtype: int64
```

Series

- **Python Dictionary 자료형을 Series data로 만들 수 있다**

```
# dictionary의 key가 Series의 index가 된다
dicData = {'국어': 90, '영어': 85, '수학': 95, '과학': 75}
data = pd.Series(dicData)

print(data)
```

```
국어    90
영어    85
수학    95
과학    75
dtype: int64
```

```
data.name = 'Grade Card'
data.index.name = 'Subject'
print(data)
```

```
Subject
국어    90
영어    85
수학    95
과학    75
Name: Grade Card, dtype: int64
```

Series

- **시리즈 인덱싱**

```
data = pd.Series({'국어': 90, '영어': 85, '수학': 95, '과학': 75})

data[1:2]
```

```
영어    85
dtype: int64
```

```
data['영어':'과학']
```

```
영어    85
수학    95
과학    75
dtype: int64
```

```
data[(data > 90)]
```

```
수학    95
dtype: int64
```

Series

▪ 시리즈 연산

- 넘파이 배열처럼 시리즈도 벡터화 연산을 할 수 있다.
- 인덱스(index) 값은 변하지 않고, 값(value) 에만 적용된다.

```
data1 = pd.Series({'국어': 90, '영어': 85, '수학': 95, '과학': 75})
data2 = pd.Series({'국어': 80, '영어': 95, '수학': 85, '과학': 75})

data1 - data2
```

```
국어    10
영어   -10
수학    10
과학     0
dtype: int64
```

Series

- 요소 추가, 삭제

```
data = pd.Series({'국어': 90, '영어': 85, '수학': 95, '과학': 75})

# 요소 추가
data['미술'] = 100

# 요소 삭제
del data['영어']

data
```

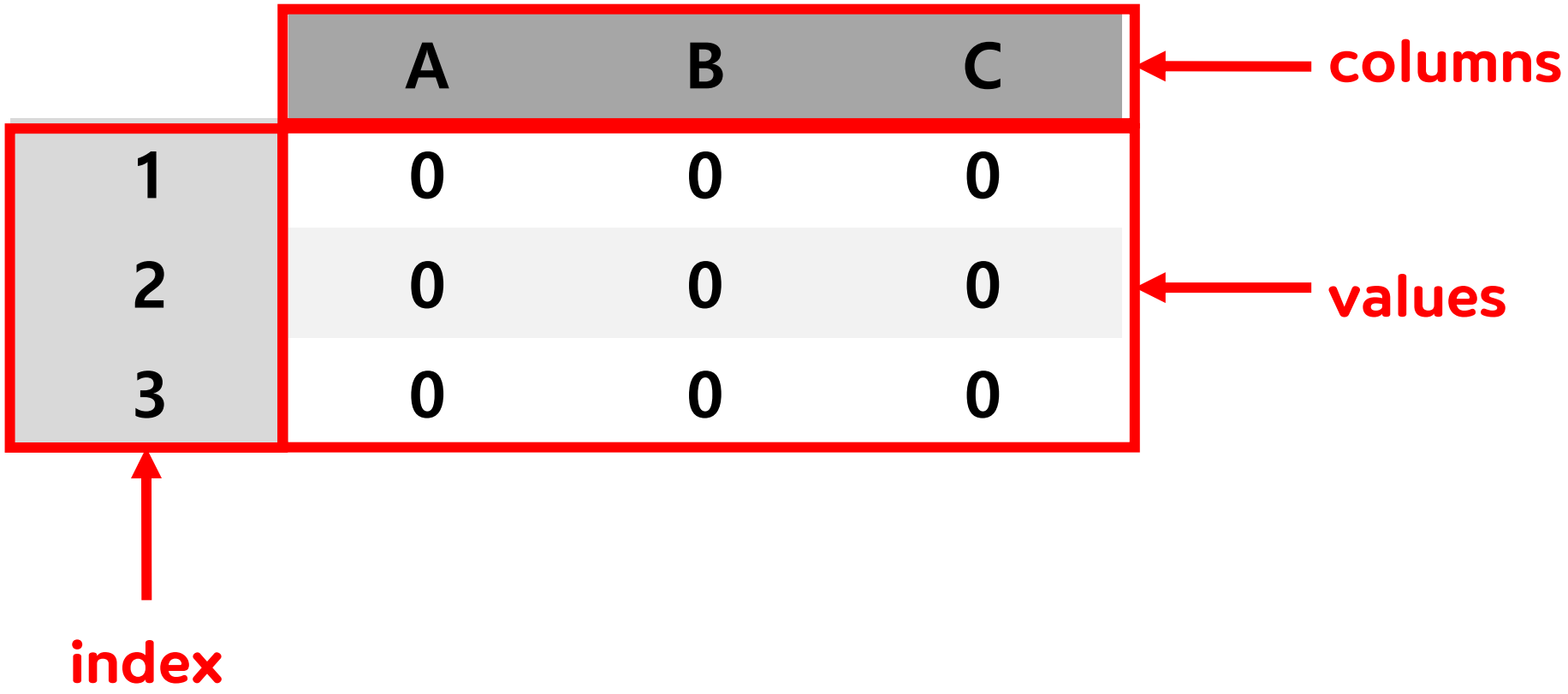
```
국어     90
수학     95
과학     75
미술    100
dtype: int64
```

Series 객체 만들기

- 시간 순서의 의미 또는 연속성을 담고 있는 예로 시리즈 객체를 2개 만들어 보세요.

DataFrame

- **DataFrame 구조**



DataFrame

2. DataFrame 객체 사용하기

A. 2차원 리스트, 배열을 사용하여 생성

```
# 2차원 리스트, 배열을 사용하여 생성
df = pd.DataFrame([[1,2,3],
                   [4,5,6],
                   [7,8,9]])
print(df)
print(type(df))
print(df.values)
print(df.columns)
print(df.index)
```

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
<class 'pandas.core.frame.DataFrame'>
[[1 2 3]
 [4 5 6]
 [7 8 9]]
RangeIndex(start=0, stop=3, step=1)
RangeIndex(start=0, stop=3, step=1)
```

```
df.name = 'test'
df.index.name = 'index'
df.columns.name = 'num'
print(df)
```

```
num    0  1  2
index
0      1  2  3
1      4  5  6
2      7  8  9
```

DataFrame

B. Python Dictionary 자료형을 사용하여 생성

```
# Python Dictionary 자료형을 사용하여 생성
score_table = {'성명':['홍길동','강동원','원빈','서강준','나'],
               '영어':[50,60,70,80,90],
               '수학':[60,50,70,70,90]}

df = pd.DataFrame(score_table)

print(df)
print(type(df))
print(df.values)
print(df.columns)
print(df.index)
```

value 개수 일치시켜서

```
   성명  영어  수학
0  홍길동  50  60
1  강동원  60  50
2   원빈  70  70
3  서강준  80  70
4    나  90  90
<class 'pandas.core.frame.DataFrame'>
[['홍길동' 50 60]
 ['강동원' 60 50]
 ['원빈' 70 70]
 ['서강준' 80 70]
 ['나' 90 90]]
Index(['성명', '영어', '수학'], dtype='object')
RangeIndex(start=0, stop=5, step=1)
```

DataFrame

```
score_table = {'성명':['홍길동','강동원','원빈','서강준','나'],
               '영어':[50,60,70,80,90],
               '수학':[60,50,70,70,90]}

df = pd.DataFrame(score_table, columns=["영어", "수학", "성명"],
                          index=["one", "two", "three", "four", "five"])
print(df)
print(type(df))
print(df.values)
print(df.columns)
print(df.index)
```

컬럼명과 인덱스명 변경

```
      영어  수학   성명
one     50  60  홍길동
two     60  50  강동원
three   70  70   원빈
four    80  70  서강준
five    90  90    나
<class 'pandas.core.frame.DataFrame'>
[[50 60 '홍길동']
 [60 50 '강동원']
 [70 70 '원빈']
 [80 70 '서강준']
 [90 90 '나']]
Index(['영어', '수학', '성명'], dtype='object')
Index(['one', 'two', 'three', 'four', 'five'], dtype='object')
```

```
df.name = 'Grade Card'
df.index.name = 'name'
df.columns.name = 'subject'
print(df)
```

```
subject  영어  수학   성명
name
one      50  60  홍길동
two      60  50  강동원
three    70  70   원빈
four     80  70  서강준
five     90  90    나
```

DataFrame

```
df = pd.DataFrame(df, columns=["성명","영어", "수학"],
                                index=["one", "two", "three", "four", "five"])
df
```

컬럼 순서 변경

	성명	영어	수학
one	홍길동	70.0	60.0
two	강동원	70.0	50.0
three	원빈	70.0	70.0
four	서강준	70.0	70.0
five	나	70.0	90.0

DataFrame

- **DataFrame 복사하기**

```
df2 = df                # view, 메모리 공유
df3 = df.copy()         # copy, 별도의 메모리 사용
df4 = df.iloc[:,0:2] # copy, 별도의 메모리 사용
df
```

subject name	성명	영어	수학	국어	과학
one	홍길동	70.0	60	87.0	NaN
two	강동원	70.0	50	87.0	80.0
three	원빈	70.0	70	87.0	NaN
four	서강준	70.0	70	87.0	70.0
five	나	70.0	90	87.0	90.0
six	90	90.0	유재석	100.0	100.0

DataFrame

- **DataFrame의 Column(열)에 접근하기**

```
print(df['성명'])

print(df[['성명', '영어']])
```

```
one      홍길동
two      강동원
three     원빈
four     서강준
five       나
Name: 성명, dtype: object
       성명  영어
one    홍길동  50
two    강동원  60
three   원빈  70
four   서강준  80
five     나  90
```

- **DataFrame의 열 값 변경하기**

```
df['영어'] = np.mean(df['영어'])

print(df['영어'])
```

```
0    70.0
1    70.0
2    70.0
3    70.0
4    70.0
Name: 영어, dtype: float64
```

DataFrame

- **DataFrame의 Row(행)에 접근하기**

```
# 특정 개수만큼 행 출력
df.head(3)    # df.head(n=3)
```

	성명	영어	수학
one	BTS	50	60
two	아이유	60	50
three	원빈	70	70

```
# 특정 조건으로 행 검색
df.query('성명=="BTS"')
```

	성명	영어	수학
one	BTS	50	60

```
df.query('영어 > 70')['성명']
```

```
four    블랙핑크
five       나
Name: 성명, dtype: object
```

DataFrame

- **DataFrame에 새로운 열 추가하기**

```
# 새로운 열을 추가하기
import numpy as np
df['국어'] = np.random.randint(100)
df
```

subject	성명	영어	수학	국어	과학
name					
one	홍길동	70.0	60.0	79	NaN
two	강동원	70.0	50.0	79	80.0
three	원빈	70.0	70.0	79	NaN
four	서강준	70.0	70.0	79	70.0
five	나	70.0	90.0	79	90.0

```
# Series를 추가할 수도 있다
val = pd.Series([80, 70, 90], index=['two','four','five'])
#print(val)
df['과학'] = val

df
```

subject	성명	영어	수학	국어	과학
name					
one	홍길동	70.0	60.0	36	NaN
two	강동원	70.0	50.0	36	80.0
three	원빈	70.0	70.0	36	NaN
four	서강준	70.0	70.0	36	70.0
five	나	70.0	90.0	36	90.0

DataFrame

- **DataFrame에 새로운 행 추가하기**

```
# 열에 값 지정해서 새로운 행 추가하기
df.loc['six',:] = [90,90,'유재석',100,100]
df
```

subject name	성명	영어	수학	국어	과학
one	홍길동	70.0	60	79.0	NaN
two	강동원	70.0	50	79.0	80.0
three	원빈	70.0	70	79.0	NaN
four	서강준	70.0	70	79.0	70.0
five	나	70.0	90	79.0	90.0
six	90	90.0	유재석	100.0	100.0

DataFrame

- **DataFrame의 열 삭제하기**

```
del df['과학']
df
```

subject name	성명	영어	수학	국어
one	홍길동	70.0	60	79.0
two	강동원	70.0	50	79.0
three	원빈	70.0	70	79.0
four	서강준	70.0	70	79.0
five	나	70.0	90	79.0
six	90	90.0	유재석	100.0

- **DataFrame의 특정 행 삭제하기**

```
# 특정 행 drop하기
df2 = df.drop(['six'])
df2
```

subject name	성명	영어	수학	국어	과학
one	홍길동	70.0	60	87.0	NaN
two	강동원	70.0	50	87.0	80.0
three	원빈	70.0	70	87.0	NaN
four	서강준	70.0	70	87.0	70.0
five	나	70.0	90	87.0	90.0

DataFrame

- **csv 파일 가져오기**

```
pd.read_csv(파일명)
```

- **한글 파일 깨짐 현상 해결책**

한글 인코딩: Microsoft사에서 만든 cp949/ms949 인코딩, euc-kr인코딩, utf-8 인코딩 등등

- 해결책 (1) - engine='python' ex) pd.read_csv(파일명, engine='python')
- 해결책 (2) - encoding='utf-8' ex) pd.read_csv(파일명, encoding='utf-8')
- 해결책 (3) - Excel에서 인코딩 옵션 변경
  (다른 이름으로 저장에서 - CSV UTF-8 (쉼표로 분리) 로 변경하여 저장)

CSV 파일 읽어오기

- 외부 데이터를 읽어서 DafaFrame 객체로 만들어 보세요.

1) 인구밀도(인구주택총조사기준)

자료갱신일: 2016-12-14 / 수록기간: 5년 1966 ~ 2015 / 자료문의처 : 042-481-3756 / 기능문의: KC

일괄설정 + | 항목[1/1] | 행정구역별[18/18] | 시점[3/11]

(단위 : 명/㎢) 새창보기 | 주석정보 | 주소정보 | 행렬전환

행정구역별	2015	2010	2005
전국	509.2	485.6	474.5
서울특별시	16,364.0	16,188.9	16,221.0
부산광역시	4,479.9	4,452.3	4,609.4
대구광역시	2,791.0	2,767.4	2,786.5
인천광역시	2,755.5	2,587.5	2,546.3
광주광역시	2,998.8	2,945.6	2,827.5
대전광역시	2,852.3	2,781.2	2,673.0
울산광역시	1,099.6	1,022.3	992.5
세종특별자치시	439.0	-	-
경기도	1,226.4	1,119.3	1,028.1
강원도	90.2	88.2	88.2
충청북도	214.6	203.4	196.5
충청남도	256.6	235.0	219.7
전라북도	227.4	220.3	221.5
전라남도	146.1	142.2	150.7
경상북도	140.8	136.6	137.1
경상남도	316.4	300.0	290.5
제주도	327.5	287.7	287.8

(데이터 예제)

https://kosis.kr/statisticsList/statisticsListIndex.do?parentId=A.1&vwcd=MT_ZTITLE&menuId=M_01_01#content-group

CSV 파일 읽어오기

- 읽어 온 데이터를 가지고 DataFrame에서 다룬 내용 행 & 열 조회/추가/삭제를 실습해 보세요.

	행정구역별	2005	2010	2015
0	전국	474.5	485.6	509.2
1	서울특별시	16221.0	16188.9	16364.0
2	부산광역시	4609.4	4452.3	4479.9
3	대구광역시	2786.5	2767.4	2791.0
4	인천광역시	2546.3	2587.5	2755.5
5	광주광역시	2827.5	2945.6	2998.8
6	대전광역시	2673.0	2781.2	2852.3
7	울산광역시	992.5	1022.3	1099.6

03 matplotlib

01. matplotlib 사용하기

Matplotlib

- Matplotlib https://matplotlib.org
  - **데이터 시각화(data visualization)**는
    데이터 분석 결과를 쉽게 이해할 수 있도록 시각적으로 표현하고 전달되는 과정을 말한다.
  - **데이터 시각화의 목적**은
    도표(graph)라는 수단을 통해 정보를 명확하고 효과적으로 전달하는 것이다

matplotlib

- **다양한 그래프 사용하기**

https://matplotlib.org/gallery/index.html

Matplotlib

- 라이브러리 설치

```
pip install matplotlib
```

- matplotlib 사용

```
import matplotlib
```

```
import matplotlib
import matplotlib.pyplot as plt
```

Matplotlib

- Matplotlib 사용 예제 : 선 그래프

```
import matplotlib
import matplotlib.pyplot as plt

# 한글출력 설정
matplotlib.rcParams['font.family'] = 'Malgun Gothic'  # '맑은 고딕'으로 설정

# 그래프 크기 설정
plt.rcParams['figure.figsize'] = (10, 5) # (가로,세로) 인치 단위

# X축, Y축 데이터
X = list(range(2005,2015,1))
Y = [15,7,2,10,8,5,14,9,18,8]

plt.plot(X, Y, c='r') #선 그래프
plt.grid(True)
plt.title('년도별 그래프')
plt.show()
```

Matplotlib

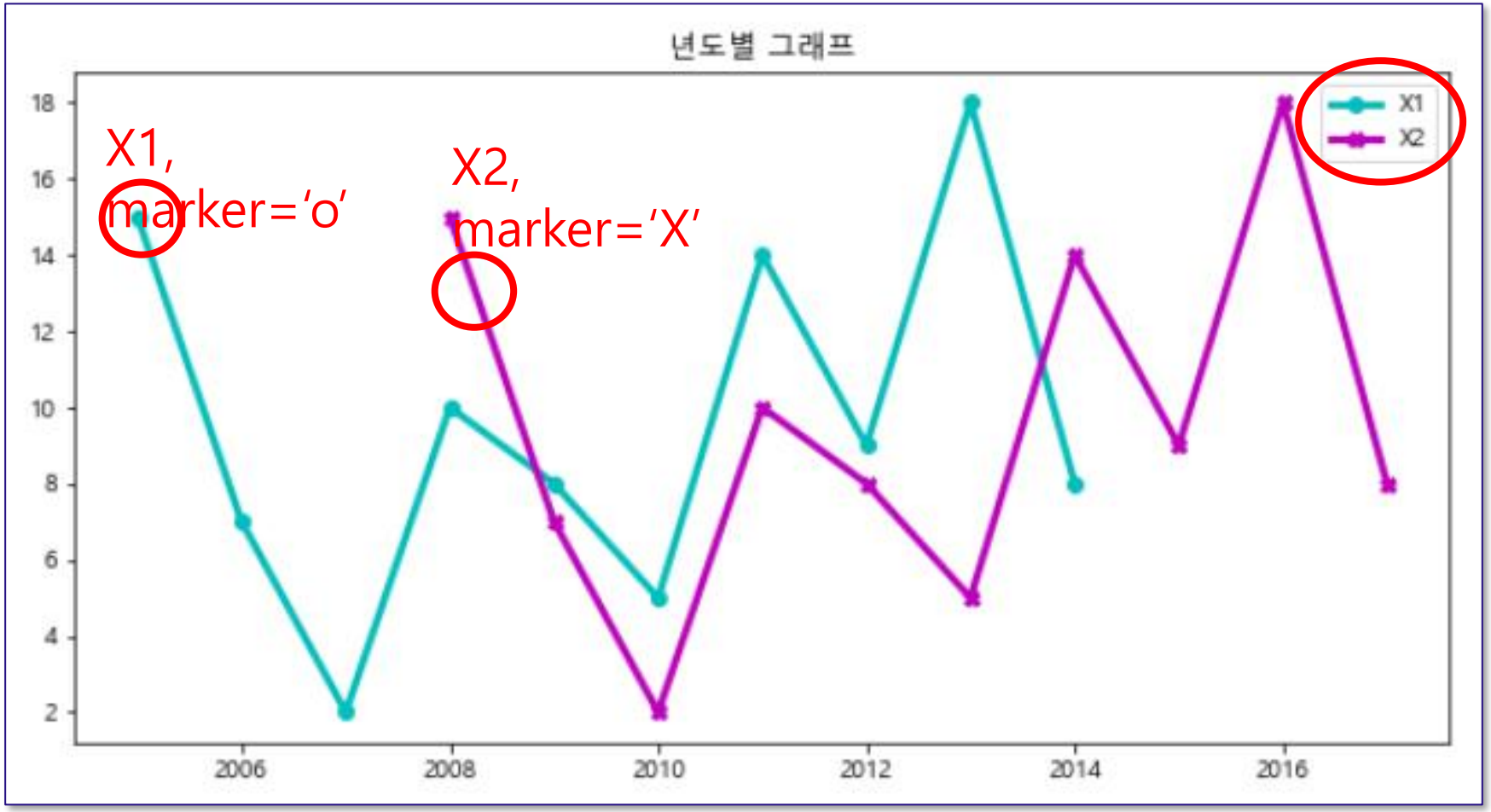
```
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd

# 한글출력 설정
matplotlib.rcParams['font.family'] = 'Malgun Gothic’

# 그래프 크기 설정
plt.rcParams['figure.figsize'] = (10, 5)
plt.rcParams['lines.linewidth'] = 3  # 선 두께
plt.rcParams['axes.grid'] = True     # 차트내 격자선

data    = [[15,7,2,10,8,5,14,9,18,8]]
columns = list(range(2005,2015,1))
index   = ['횟수']
df      = pd.DataFrame(data,columns=columns,index=index)
x, y = df.columns, df.values[0]

plt.plot(x, y, 'o', linestyle=‘dashed', c='b')
plt.show()
```

데이터 분석에 필요한 라이브러리

- **Matplotlib 참고 사이트**
  - 예시 플롯 (이미지와 소스 코드 포함)과 튜토리얼(파이썬으로 데이터 시각화하기)
    https://matplotlib.org/stable/gallery/index.html
    https://wikidocs.net/book/5011
  - Linestyles(선 스타일과 소스 코드 포함)
    https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html?highlight=linestyle
  - Colors(튜토리얼 및 예시와 소스 코드 포함)
    https://matplotlib.org/stable/gallery/color/named_colors.html#sphx-glr-gallery-color-named-colors-py
  - matplotlib.markers(마커와 소스 코드 포함)
    https://matplotlib.org/stable/api/markers_api.html?highlight=marker#module-matplotlib.markers

실습문제

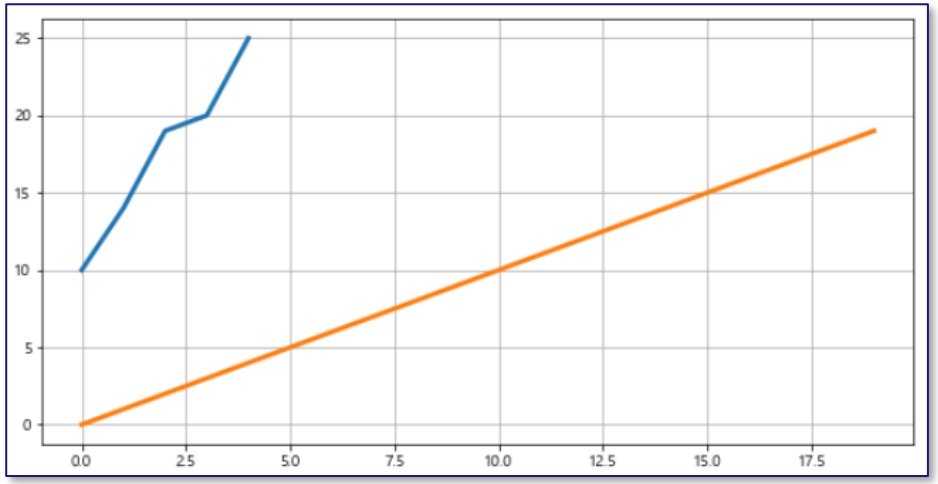
- 앞에서 작성한 ‘년도별 그래프’ 예제를 이용하여 한 개의 그래프 포맷 안에 2개의 선 그래프를 그려보세요.



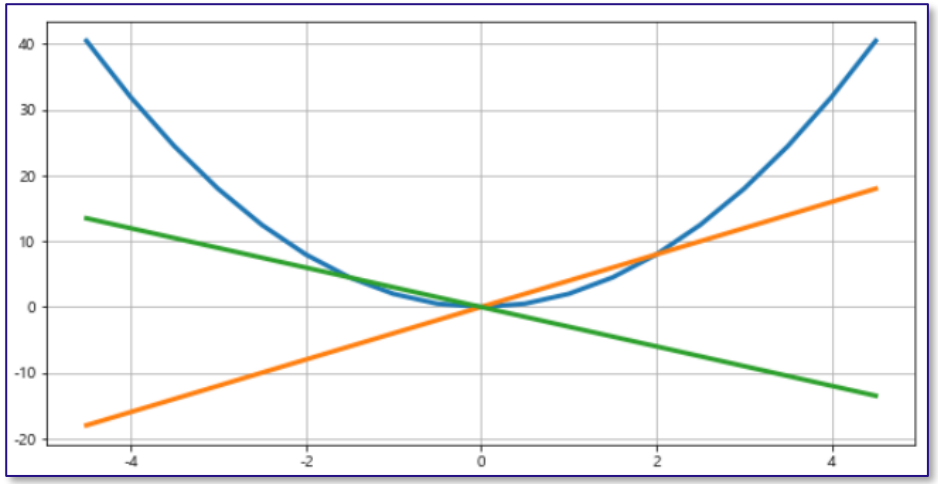
Matplotlib

- 선 그래프 : numpy 데이터

```
import matplotlib.pyplot as plt
import numpy as np

a = np.array([10,14,19,20,25])
plt.plot(a)
a2 = np.arange(20)
plt.plot(a2)
plt.show()
```



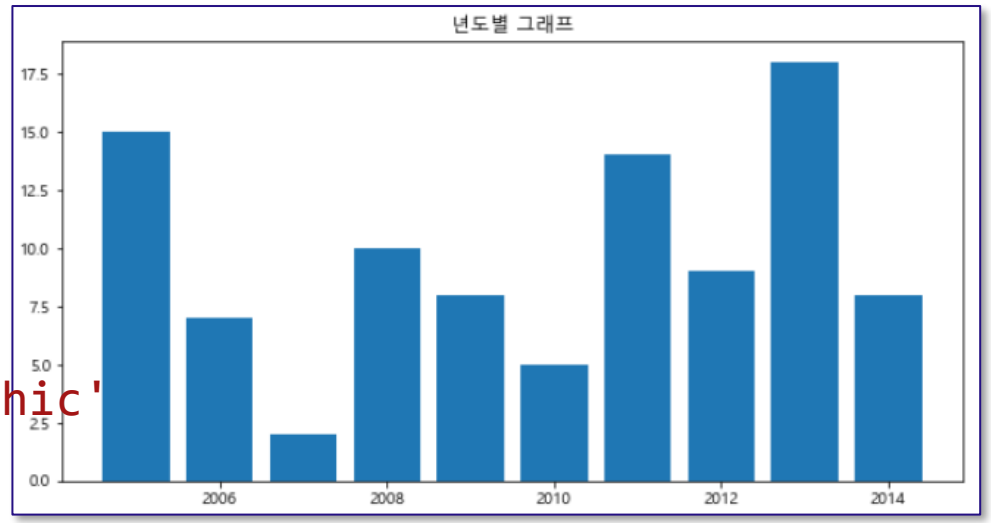

```
# 2차 방정식의 그래프 : f(x) = a*x^2 + b
x = np.arange(-4.5,5,0.5)
y = 2*x**2
plt.plot(x, y)

plt.plot(x, 4*x)
plt.plot(x,-3*x)
plt.show()
```



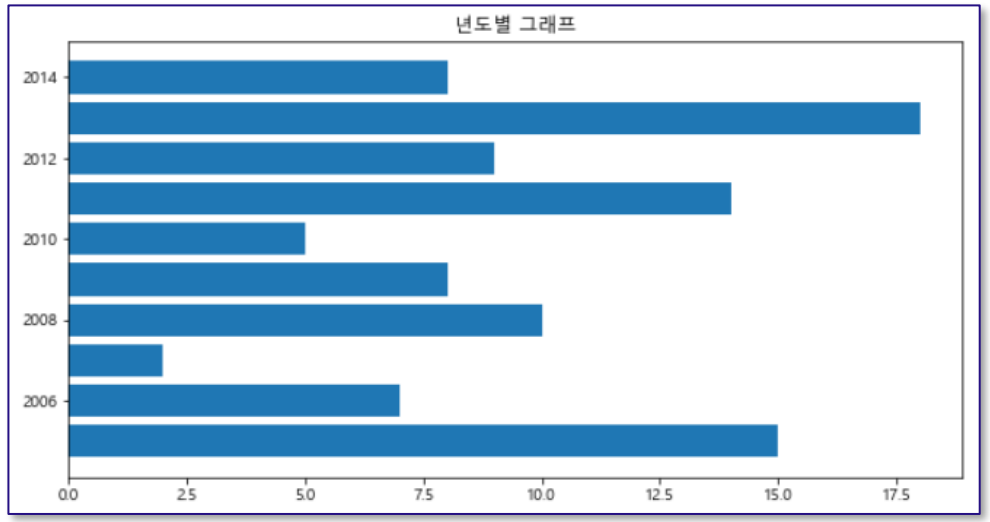
Matplotlib

- 막대 그래프

```
import matplotlib
import matplotlib.pyplot as plt

# 한글출력 설정
matplotlib.rcParams['font.family'] = 'Malgun Gothic'

# 그래프 크기 설정
plt.rcParams['figure.figsize'] = (10, 5)

# X축, Y축 데이터
X = list(range(2005,2015,1))
Y = [15,7,2,10,8,5,14,9,18,8]

plt.bar(X, Y) #막대 그래프
plt.grid(False)
plt.title('년도별 그래프')
plt.show()
```



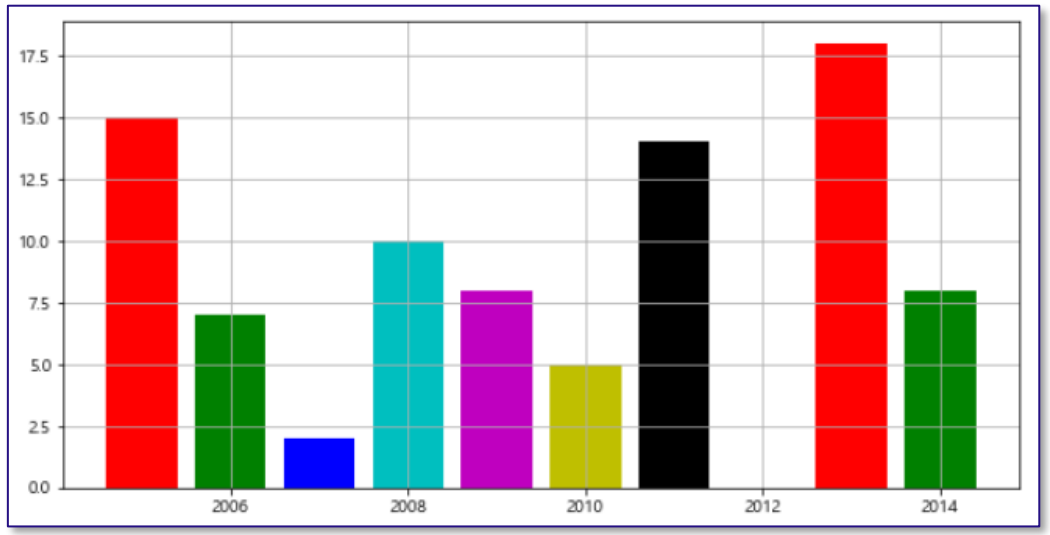

```
plt.barh(X, Y) #막대 그래프
```



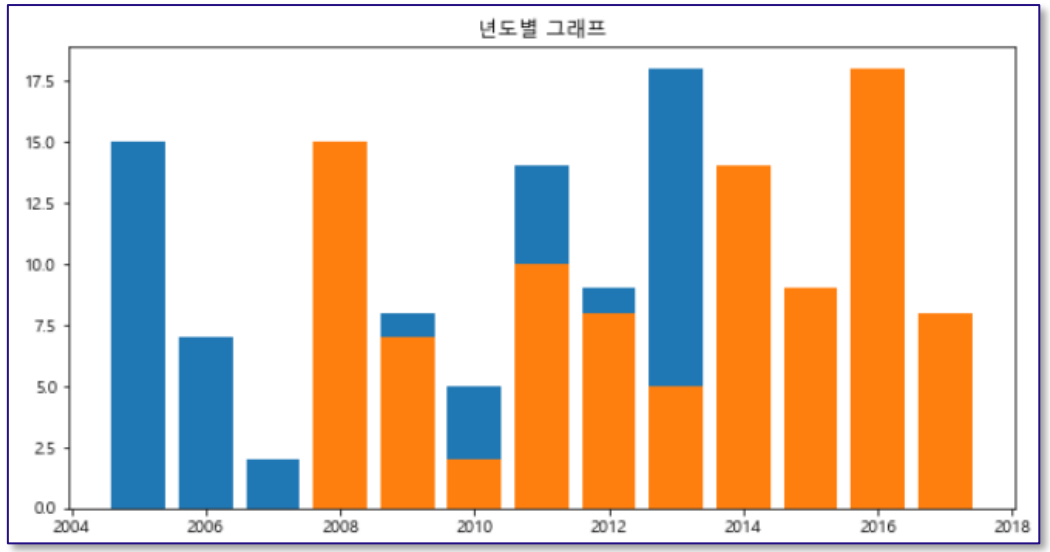
Matplotlib

- 막대 그래프

```
#그래프 색상 지정
colors = ['r','g','b','c','m','y','k','w']
plt.bar(X, Y,color=colors)
plt.show()
```



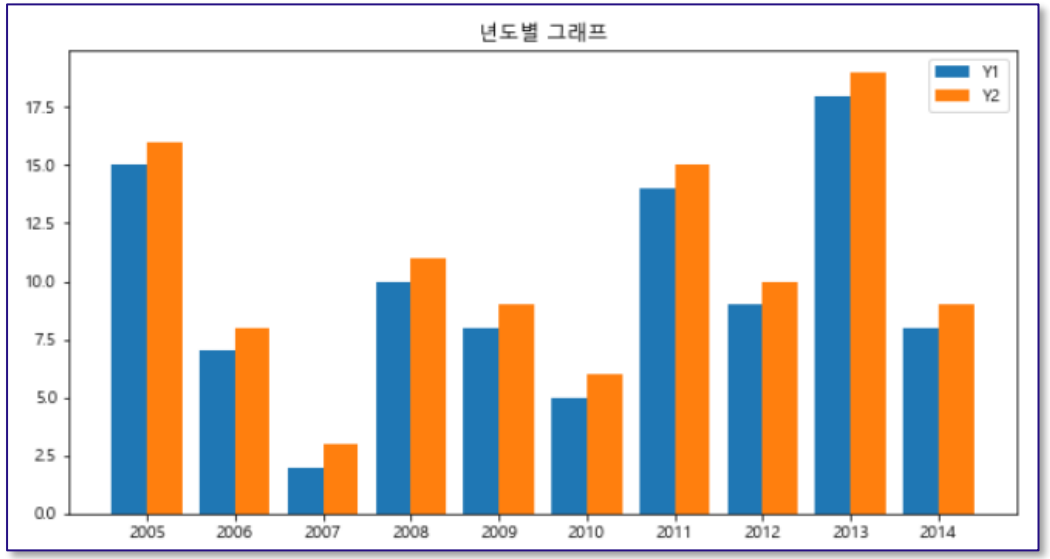

```
# X축, Y축 데이터
X1 = list(range(2005,2015,1))
X2 = [x+3 for x in X1]
Y = [15,7,2,10,8,5,14,9,18,8]

plt.bar(X1, Y)      # 막대 그래프
plt.bar(X2, Y)      # 막대 그래프
plt.title('년도별 그래프')
plt.grid(False)
plt.show()
```



Matplotlib

- 막대 그래프

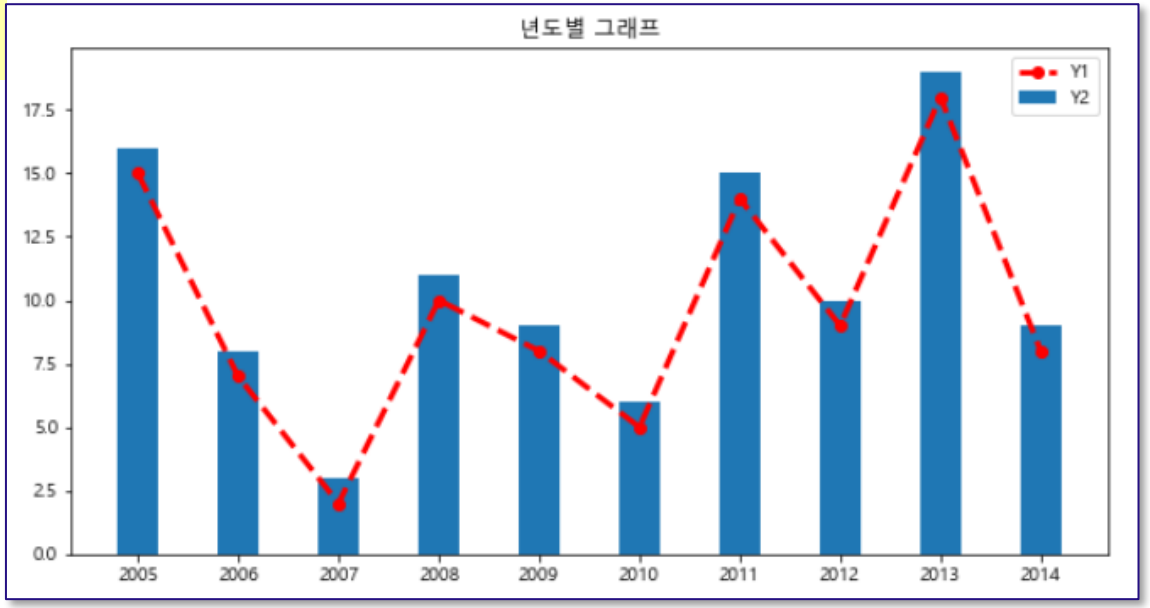
```
import numpy as np

# X축, Y축 데이터
X = list(range(2005,2015,1))
Y1= [15,7,2,10,8,5,14,9,18,8]
Y2= [y+1 for y in Y1]
lable = X
X = np.arange(len(X))

plt.bar(X-0.2, Y1, width=0.4, label='Y1')
plt.bar(X+0.2, Y2, width=0.4, label='Y2')
plt.xticks(X, lable)
plt.title('년도별 그래프')
plt.grid(False)
plt.legend()
plt.show()
```



Matplotlib

- 혼합 그래프 : 막대 + 선 그래프

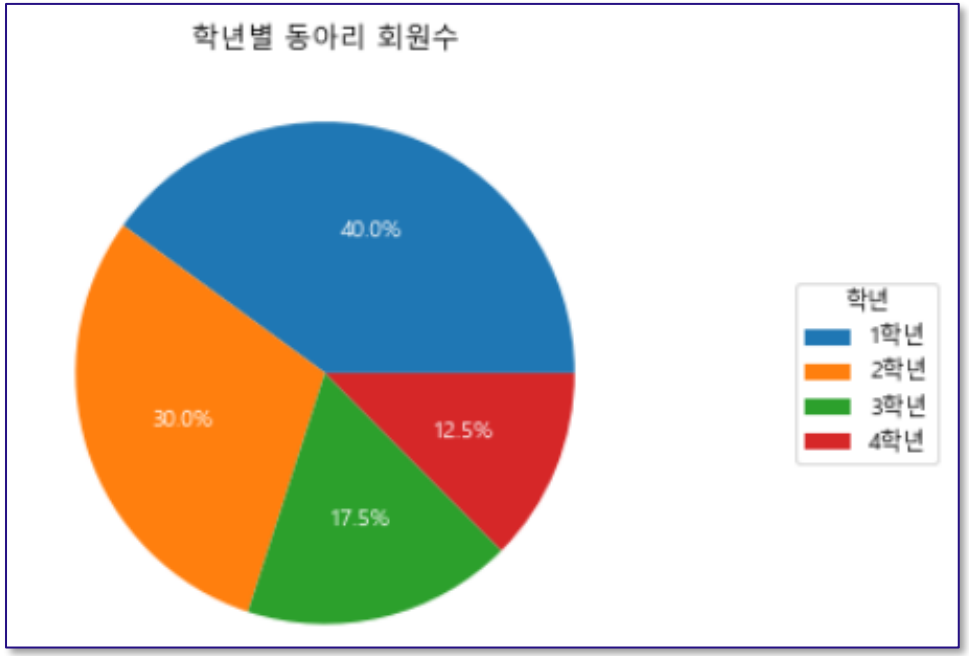
```
# X축, Y축 데이터
X = list(range(2005,2015,1))
Y1= [15,7,2,10,8,5,14,9,18,8]
Y2= [y+1 for y in Y1]

plt.plot(X, Y1, 'o', linestyle='dashed', c='r', label='Y1')      # 선 그래프
plt.bar(X, Y2, width=0.4, label='Y2')      # 막대 그래프
plt.xticks(X, lable)

plt.title('년도별 그래프')
plt.grid(False)
plt.legend()
plt.show()
```



Matplotlib

- 원 그래프

```
import matplotlib.pyplot as plt
import pandas as pd

idx = ['1학년','2학년','3학년','4학년']
val = [16,12,7,5]
per = [val[x]/sum(val) *100 for x in range(len(val))]

plt.pie(per, autopct='%1.1f%%',
            textprops=dict(color="w"))
plt.title("학년별 동아리 회원수")
plt.legend(idx, title='학년', loc="center right",
           bbox_to_anchor=(1, 0, 0.5, 1))
plt.show()
```



04

실습: 공공데이터 분석

공공 데이터(사이트)선택

- 기상청 [기상자료개방포털]
  https://data.kma.go.kr/cmmn/main.do

공공 데이터 자료 검색

- 기상관측 데이터

※주의!!!
기상청 자료를 다운로드 하기 위해서는 사이트에 회원가입이 되어 있어야 합니다.

#--체크항목

평균현지기압

월합강수량

평균풍속

최심적설

평균기온

최고기온

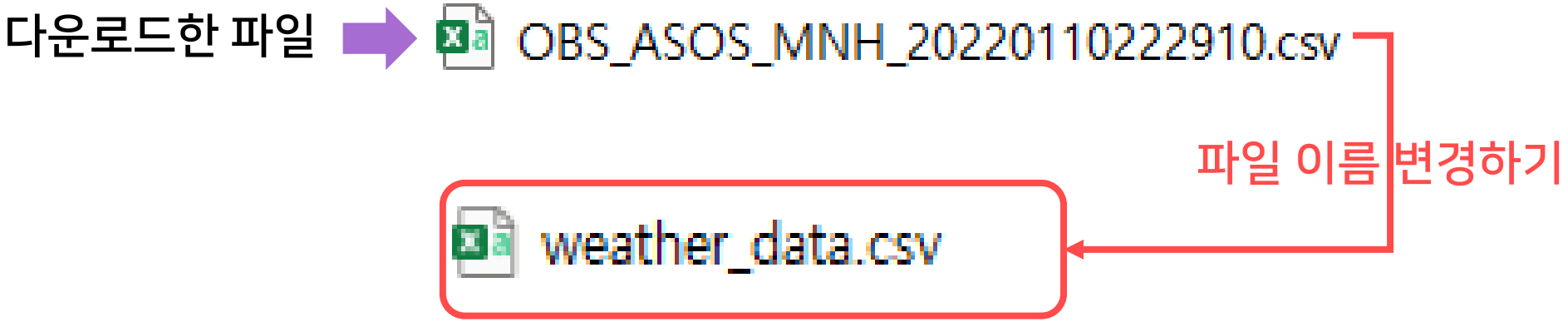
최저기온

평균상태습도

합계 일조시간

데이터 수집

- 수집된 파일


다운로드한 파일
OBS_ASOS_MNH_202201102229l0.csv
파일 이름 변경하기
weather_data.csv

데이터 가공하기

- 데이터 가공하기 순서

1. (다운로드한) 데이터 불러오기
2. 컬럼명 변경하기
3. 컬럼 추가하기
4. 컬럼 삭제하기
5. 특징 데이터 검색하기
6. 가공된 데이터 저장하기

데이터 가공하기

- 다운로드한 파일 불러오기

```
import pandas as pd

# OBS_ASOS_MNH_~.csv 파일명 데이터를 'weather_data.csv'로 변경한 후
  사용합니다.
File =  'c:/python/master/data/weather_data.csv'
df = pd.read_csv(file, encoding='CP949', engine='python’)

print(type(df))
df
```

데이터 가공하기

	지점	지점명	일시	평균기온(°C)	최고기온(°C)	최저기온(°C)	평균현지기압(hPa)	평균상대습도(%)	월합강수량(00~24h만)(mm)	평균풍속(m/s)	합계 일조시간(hr)	최심적설(cm)
0	90	속초	1968-01	-0.7	11.4	-11.4	1018.2	47.0	0.0	2.1	218.6	NaN
1	90	속초	1968-02	-2.1	9.7	-9.9	1022.8	51.0	3.3	2.2	236.5	1.8
2	90	속초	1968-03	5.9	20.5	-3.2	1016.1	57.0	8.8	2.1	223.7	0.0
3	90	속초	1968-04	10.5	21.4	4.0	1019.4	63.0	6.9	2.0	255.7	NaN
4	90	속초	1968-05	14.6	24.5	9.3	1011.7	74.0	49.1	2.2	187.8	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
54240	295	남해	2021-06	22.1	29.7	14.5	1002.9	77.0	140.3	1.2	209.9	NaN
54241	295	남해	2021-07	26.1	33.0	19.3	1003.3	83.0	507.5	1.3	198.4	NaN
54242	295	남해	2021-08	26.1	35.5	20.7	1003.2	82.0	473.2	1.3	174.5	NaN
54243	295	남해	2021-09	22.2	30.1	16.3	1008.1	82.0	164.6	1.5	95.2	NaN
54244	295	남해	2021-10	17.2	28.9	4.7	1015.3	74.0	43.6	1.3	203.2	NaN

54245 rows × 12 columns

데이터 가공하기

- 컬럼명 변경하기

```
df.columns
df.columns = ['지점', '지점명', '년월', '평균기온', '최고기온',
              '최저기온', '평균기압', '평균상대습도', '월합강수량',
              '평균풍속','일조시간', '최심적설']
df
```

지점	평균기온(°C)	최고기온(°C)	최저기온(°C)	평균현지기압 (hPa)	평균상대습도 (%)	월합강수량 (00~24h만) (mm)	평균풍속 (m/s)	합계 일조시간 (hr)	최심적설(cm)

지점	지점명	년월	평균기온	최고기온	최저기온	평균기압	평균상대습도	월합강수량	평균풍속	일조시간	최심적설

데이터 가공하기

- 컬럼 추가하기

```
df.insert(1,'신규',df['지점'])
```

	지점	신규	지점명	년월
0	90	90	속초	1968-01
1	90	90	속초	1968-02
2	90	90	속초	1968-03

- 컬럼 삭제하기

```
df.drop('신규', axis=1, inplace=True)
```

	지점	지점명	년월
0	90	속초	1968-01
1	90	속초	1968-02
2	90	속초	1968-03

데이터 가공하기

- '년도' 컬럼 추가하기
  - 년월의 컬럼 중 (4자리)년도를 2번째 인덱스에 '년도'란 이름으로 추가하기

```
df.insert(2,'년도',df['년월'].str[0:4])
df
```

	지점	지점명	년도	년월
0	90	속초	1968	1968-01
1	90	속초	1968	1968-02
2	90	속초	1968	1968-03

데이터 가공하기

- 특징 데이터 검색하기
  - 지점명 '제주'인 데이터 추출하기

```
df.query(" 지점명 == '제주' ")

df[ df['지점명'] == '제주' ]

df.iloc[ df.index[ df['지점명'] == '제주' ] ]
```

모두 동일한 결과

데이터 가공하기

- 특징 데이터 검색하기
  - 최초 기상관측 년월

```
min(df['일시'])

df['일시'].min()
```

모두 동일한 결과

  - 최근(마지막) 기상관측 년월

```
max(df['일시'])

df['일시'].max()
```

모두 동일한 결과

실습문제

- 기상관측 데이터를 통해 가장 더웠던 년, 월과 그 때의 최고기온은?

```
df.query(f" 최고기온 == {df['최고기온'].max()} ")

df[df['최고기온'] == df['최고기온'].max()]
```

모두 동일한 결과

	지점	지점명	년도	년월	평균기온	최고기온	최저기온	평균기압	평균상대습도	월합강수량	평균풍속	일조시간	최심적설
34662	212	홍천	2018	2018-08	27.2	41.0	15.8	991.7	73.0	300.6	1.4	234.4	NaN

- 기상관측 데이터를 통해 가장 추웠던 년, 월과 그 때의 최저기온은?

실습문제

- 기상관측 데이터를 통해 ‘제주’에서 가장 더웠던 년, 월과 그 때의 최고기온은?

- 기상관측 데이터를 통해 ‘제주’에서 가장 추웠던 년, 월과 그 때의 최저기온은?

데이터 가공하기

- 특징 데이터 검색하기
  - 그룹핑: 지점별 평균기온

```
df['평균기온'].groupby(df['지점명']).mean()

df.groupby('지점명')['평균기온'].agg(**{'평균기온':'mean'})
```

모두 동일한 결과

지점명	평균기온
강릉	12.559067
강진군	13.785211
강화	11.149829
거제	14.207338
거창	11.790956
...	...

데이터 가공하기

- 특징 데이터 검색하기
  - **그룹핑**: 지점별 년도별

```
df.groupby(['지점명','년도'])['평균기온'].agg(**{'평균기온':'mean'})
```

		평균기온
지점명	년도	
강릉	1911	NaN
	1912	11.733333
	1913	11.683333
	1914	13.175000
	1915	12.091667
...	...	...
흑산도	2017	13.491667
	2018	13.825000
	2019	14.066667
	2020	14.100000
	2021	16.310000

실습문제

- 기상관측 데이터를 통해 지점별 최고기온은?

```
df.groupby('지점명')['최고기온'].agg(**{'최고기온':'max'}).reset_index()
```

- 기상관측 데이터를 통해 지점별, 년도별 일최다강수량은?

(지점별 최고기온 코드를 활용해서 작성해보세요)

데이터 시각화하기

데이터 시각화

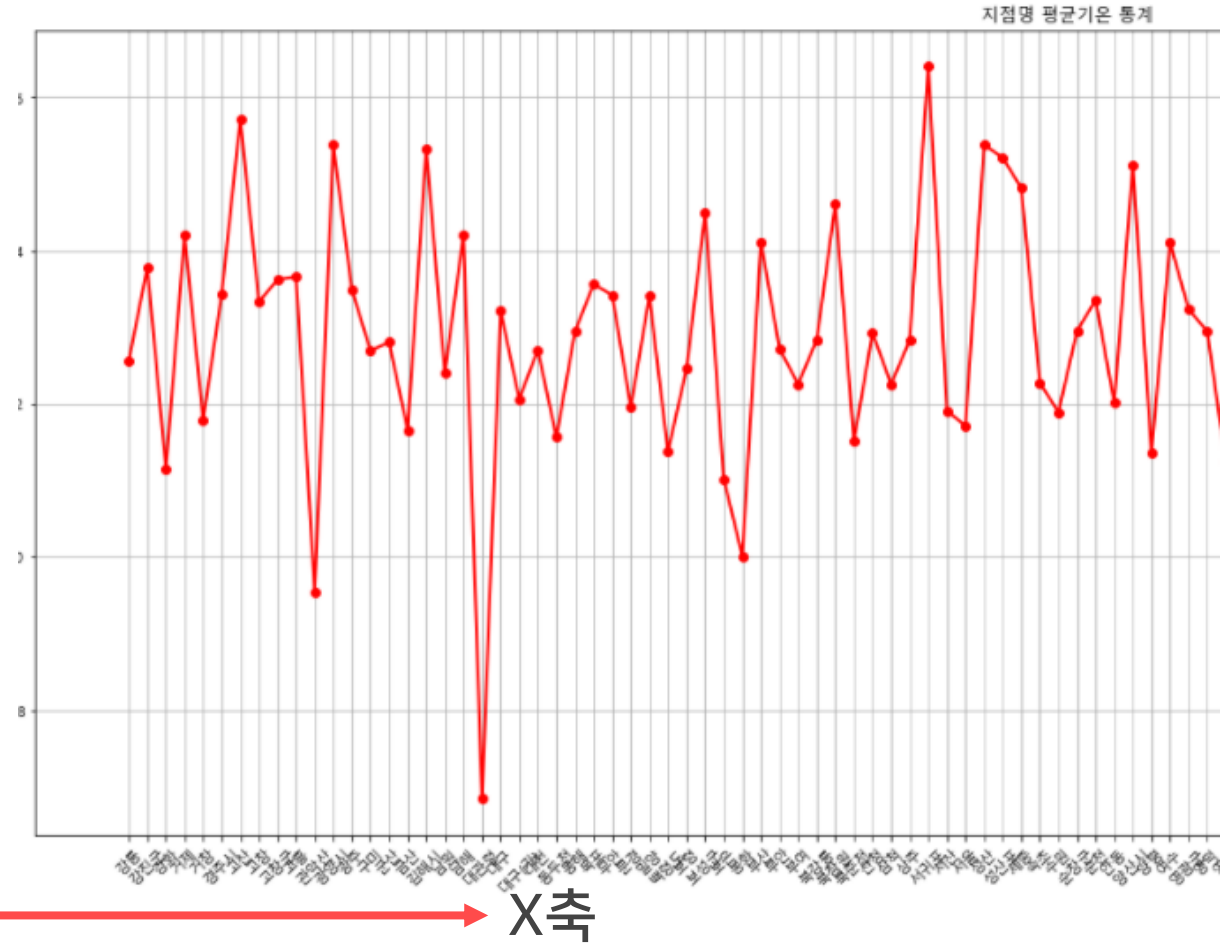
- 2차원 데이터 그래프로 나타내기

  데이터를 그래프로 나타내기 위해서는 X축 Y축에 해당하는 데이터가 준비되어야 한다.

지점명	지점	평균기온
강릉	105.0	12.559067
강진군	259.0	13.785211
강화	201.0	11.149829
거제	294.0	14.207338
거창	284.0	11.790956
...	...	...

Y축

X축



데이터 시각화

- 지점별 평균기온 선그래프로 시각화 하기
  - 그룹핑: 지점별 평균기온

```
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

# 한글출력 설정
matplotlib.rcParams['font.family'] = 'Malgun Gothic'

# 1. 데이터 가져오기
file = 'c:/python/ahnlab_pbl/data/weather_data2.csv'
df = pd.read_csv(file, encoding='CP949', engine='python')

# 2. 지점별 평균기온 데이터 추출하기: reset_index()
data = df.groupby('지점명')['평균기온'].agg(**{'평균기온':'mean'})
# print(data)
```

데이터 시각화

- 지점별 평균기온 선그래프로 시각화 하기

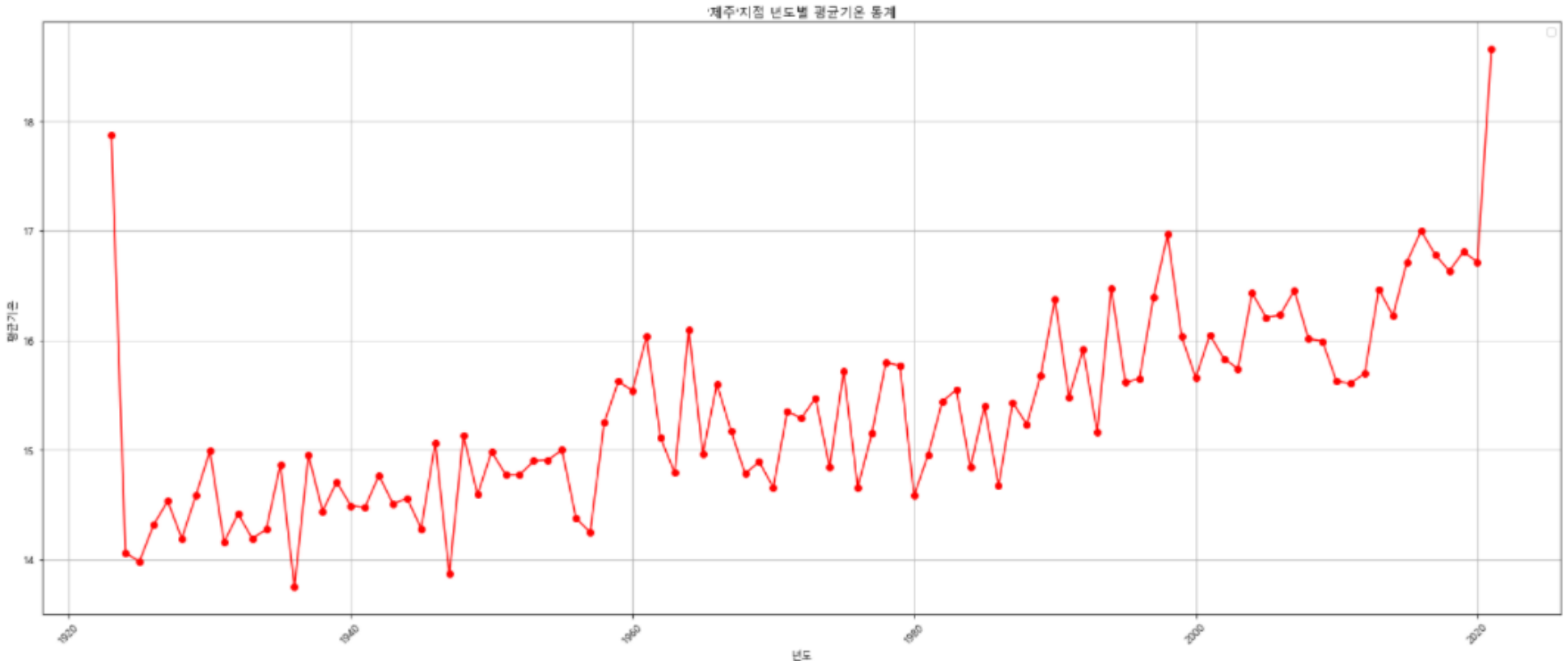
  - 그룹핑: 지점별 평균기온

```
# 3. 그래프로 시각화하기
plt.figure(figsize=(15,5))      # 그래프 크기
plt.title('지점명 평균기온 통계') #제목
plt.legend(['평균기온'])  # 범례,기본 위치 : loc='upper left'
plt.xlabel('지점')         # x축 레이블
plt.ylabel('평균기온')     # y축 레이블
plt.xticks(rotation=45)  # x축 레이블 기울이기
plt.grid()               # 격자 표시
plt.plot(data, 'o', linestyle='solid', c='r')
plt.show()
```

실습문제

- '제주'지점의 년도별 평균기온을 추출하여 선 그래프로 시각화하는 코드를 만들어 보세요.

('제주 ' 지점의 년도별 평균기온 추출 코드를 먼저 작성해보세요)



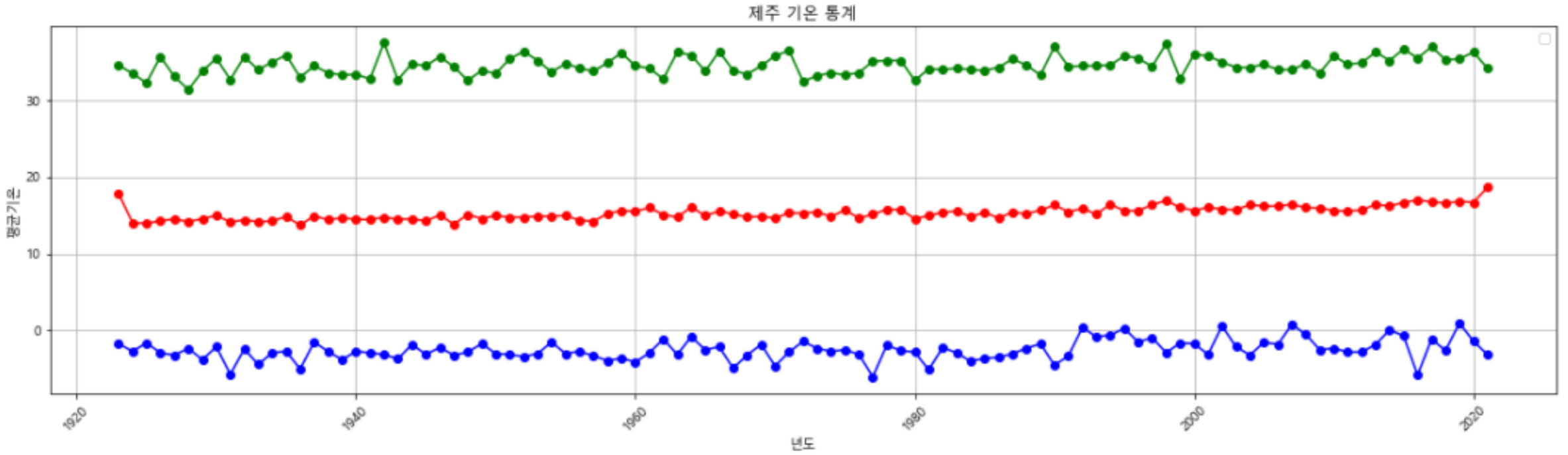
실습문제

- ‘제주'지점의 년도별 평균기온을 추출하여 선 그래프로 시각화하는 코드를 만들어 보세요

코드 작성하기

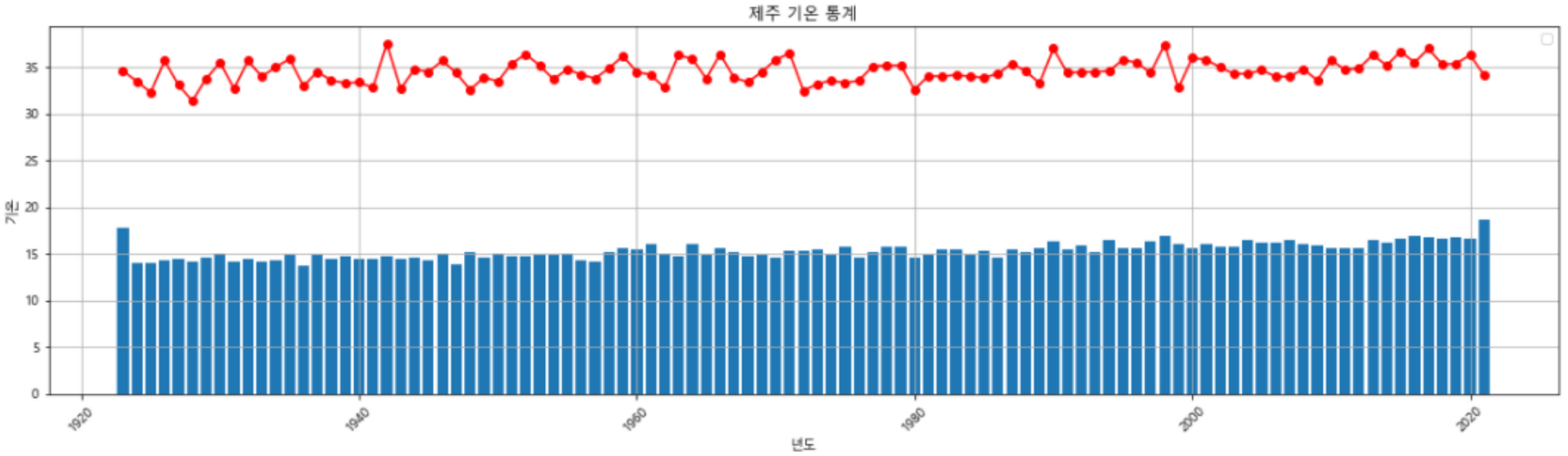
실습문제

- 기상관측 데이터를 통해 '제주'의 평균기온, 최고기온, 최저기온을 하나의 그래프에 동시에 나타내기



실습문제

- 앞에서 만든 코드에서 '제주'의 최고기온(선그래프)과 평균기온(막대그래프)을 혼합 그래프로 동시에 나타내 보세요.



THE END