파이썬을 활용한 머신러닝 분석

엑셈 | 김정수 연구원

목 차

1. 파이썬이란
2. 실습환경 구성
3. Data Handling
4. 머신러닝을 위한 라이브러리
5. 머신러닝 실습 및 심화

1. 파이썬?

역사

1990년 네덜란드의 수학 및 컴퓨터 과학 국립연구소에서 귀도 반 로섬(Guido van rossum)에 의해 만들어진 인터프리터 언어

사전적 의미

영어사전 단어·숙어 1-5 / 31건

python 미국식 [ˈpaɪθɑːn] 영국식 [ˈpaɪθən]
비단뱀

Python 영국식
1. 피톤(Apollo가 Delphi에서 퇴치한 거대한 뱀).
2. (p-) 비단뱀류, 거대한 구렁이.

사랑받는 언어

TIOBE INDEX

Apr 2017	Apr 2016	Change	Programming Language	Ratings	Change
1	1		Java	15.568%	-5.28%
2	2		C	6.966%	-6.94%
3	3		C++	4.554%	-1.36%
4	4		C#	3.579%	-0.22%
5	5		Python	3.457%	+0.13%
6	6		PHP	3.376%	+0.38%

PYPL INDEX

Worldwide, Apr 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.6 %	-1.4 %
2		Python	15.1 %	+3.1 %
3		PHP	9.4 %	-1.0 %
4		C#	8.3 %	-0.5 %
5	↑	Javascript	7.8 %	+0.4 %
6	↓	C++	7.0 %	-0.3 %

REDMONK INDEX

1 JavaScript
2 Java
3 Python
4 PHP
5 C#
6 C++
7 CSS

IEEE SPECTRUM

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7

파이썬?

특성

- 인간다운 언어
- 강력한 언어
- 간결한 언어
- 속도가 빠른언어

```
1 import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

특성

인간다운 언어

- 프로그래밍이란 인간이 생각하는 것을 컴퓨터에 지시하는 행위
- 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어
- 프로그래머는 컴퓨터 사고 체계에 맞추어서 프로그래밍을 하려고 힘쓸 필요가 없다.

```
if 4 in [1,2,3,4]: print("4 가 있습니다")
```

- 위의 예제를 다음처럼 읽을 수 있다.
- "만약 4가 1,2,3,4, 중에 있으면 4가 있습니다를 출력한다."
- 프로그램을 모르더라도 직관적으로 무엇을 뜻하는지 알 수 있다.
- 마치 영어 문장을 읽는 듯하다.

특성

강력한 언어

- 오픈소스 이며 만들고자 하는 프로그램 대부분을 만들 수 있다.
- 시스템 프로그래밍이나 하드웨어 제어와 같은 매우 복잡하여 연산이 많은 프로그램은 파이썬과 어울리지 않는다
- 이런 약점을 극복할 수 있게끔 다른 언어로 만든 프로그램을 파이썬 프로그램에 포함 시킬 수 있다.
- 다른 언어와 찰떡 궁합
  - 프로그램의 전반적인 뼈대는 파이썬으로 만들고, 빠른 실행 속도를 필요로 하는 부분은 c로 만들어서 파이썬 프로그램에 포함 시킬 수 있다.

특성

강력한 언어

```c
//2개의 수를 더하는 C 샘플 코드 - int and floats

#include <stdio.h>

int add_int(int, int);
float add_float(float, float);

int add_int(int num1, int num2){
    return num1 + num2;
}

float add_float(float num1, float num2){
    return num1 + num2;
}
```

```
#리눅스용
$  gcc -shared -Wl,-soname,adder -o adder.so -fPIC add.c
```

파이썬 코드

```python
from ctypes import *

#load the shared object file
adder = CDLL('./adder.so')

#Find sum of integers
res_int = adder.add_int(4,5)
print "Sum of 4 and 5 = " + str(res_int)

#Find sum of floats
a = c_float(5.5)
b = c_float(4.1)

add_float = adder.add_float
add_float.restype = c_float
print "Sum of 5.5 and 4.1 = ", str(add_float(a, b))
```

특성

간결한 언어

- 간결함이란 철학을 소스코드에 그대로 담아내고 있다.
- 다른 사람들의 소스코드가 한눈에 들어오기 때문에 공동 작업에 좋은 영향을 준다.
- 줄을 맞추지 않으면 실행이 안됨. 이는 가독성에 큰 도움을 준다.

파이썬

```python
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

들여쓰기가 잘 된 C

```c
int factorial(int x) {
    if(x == 0)     {
        return 1;
    }     else     {
        return x * factorial(x - 1);
    }
}
```

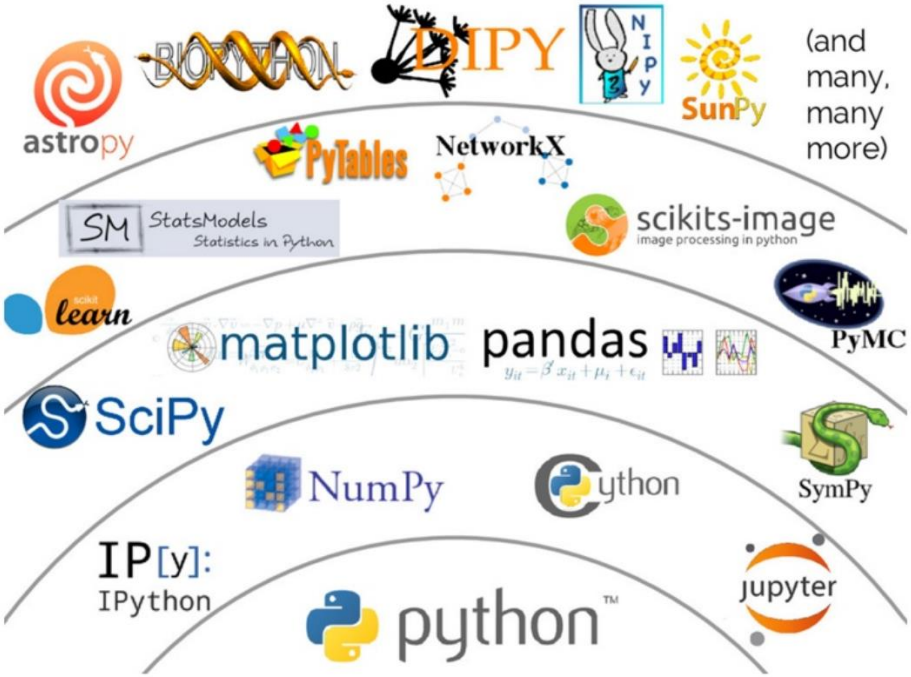
읽기 어렵게 쓰인 C

```c
int factorial(int x) {
 if(x == 0) {return 1;} else
 {return x * factorial(x - 1); } }
```

특성

빠른 개발속도

- 제공되는 라이브러리들이 많아 구현 할 필요 없이 바로 가져다 쓸 수 있다.
- 대부분의 라이브러리들이 충분히 검증 되어있어 직접 만드는 것보다 성능이 좋음.



2. 실습환경 구성

아나콘다 - 파이썬 백과사전

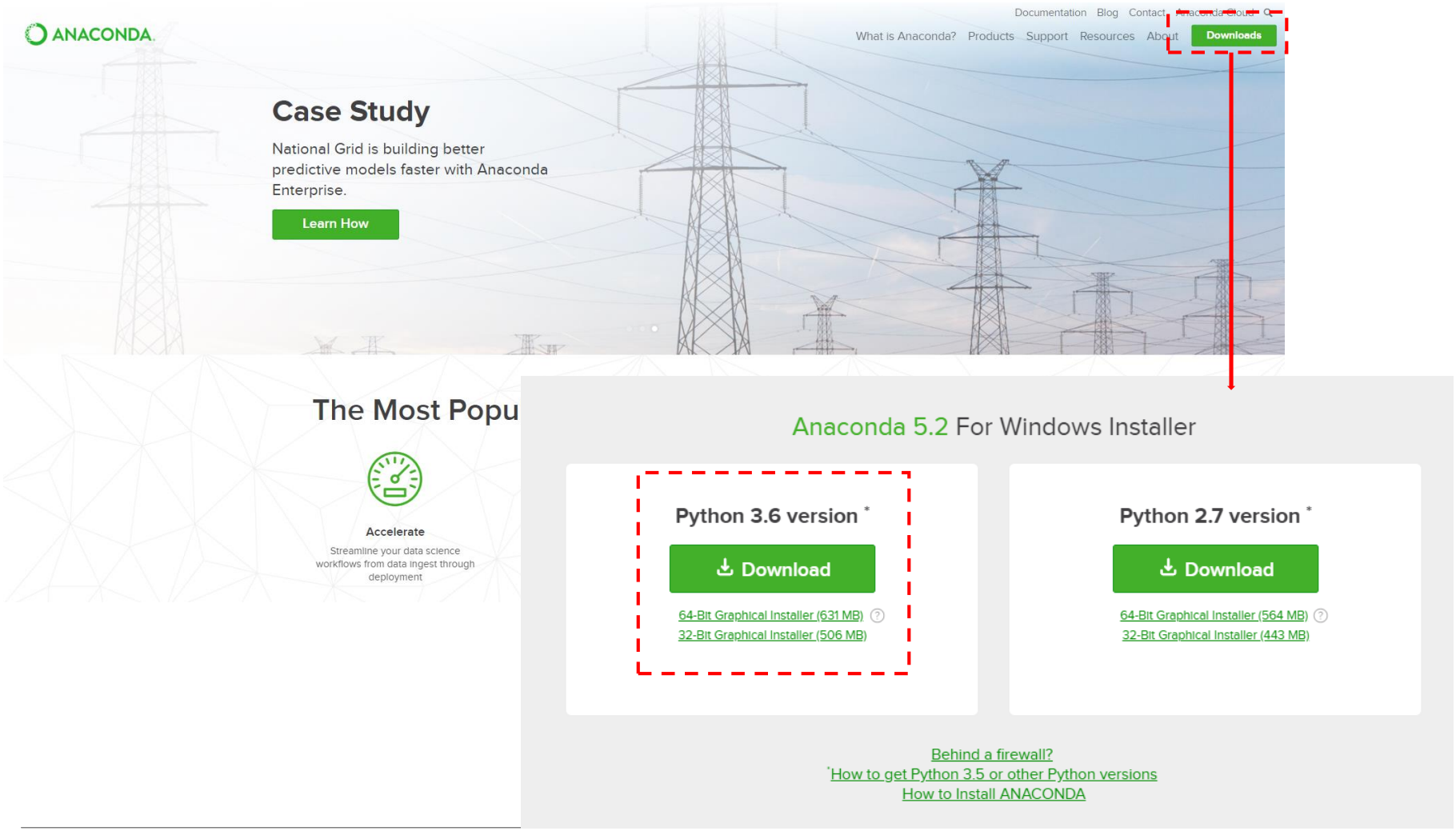
- Anaconda는 패키지 관리 및 배포를 단순화하는 것을 목표로 하는 Python 과 R 프로그래밍 언어의 배포판으로서, 데이터 사이언스와 대용량 데이터 처리, 예측분석 등의 머신러닝에 적합한 오픈소스이다.

- Anaconda를 설치하면 데이터 분석 및 처리하는 패키지가 기본 내장되어 있어 사용자가 개별적으로 설치할 필요가 없어진다.
  - 데이터 과학을 위한 250이상의 유명한 패키지들이 있다.

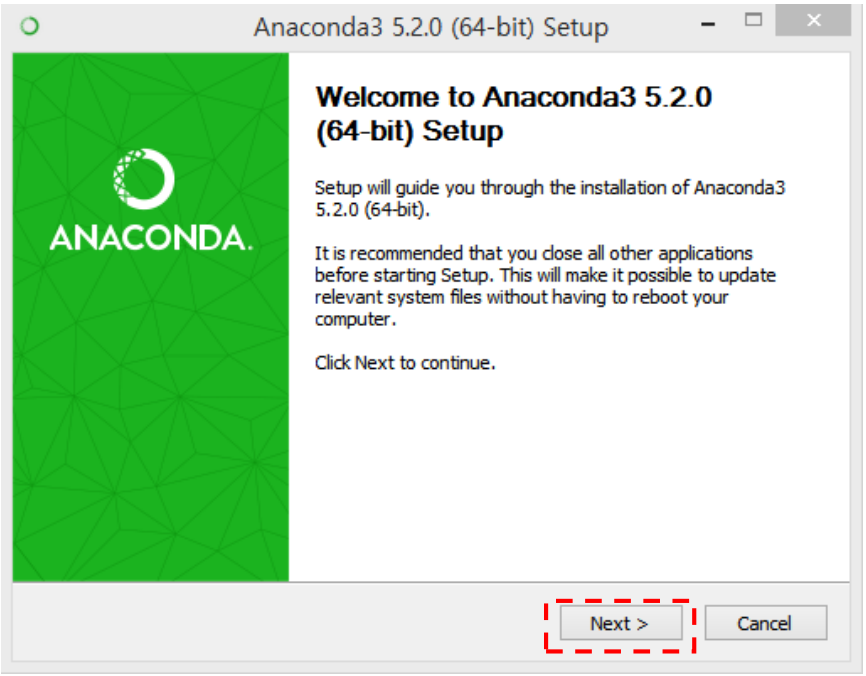
- Python의 용도가 분석 및 기계학습이라면 Anaconda 설치를 권장한다.

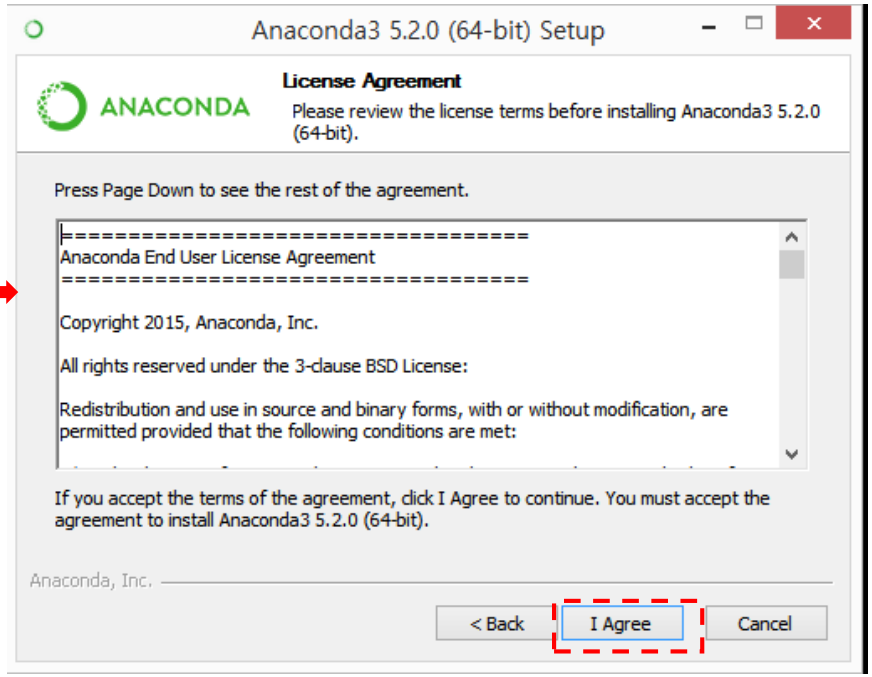
아나콘다 설치

https://www.anaconda.com/

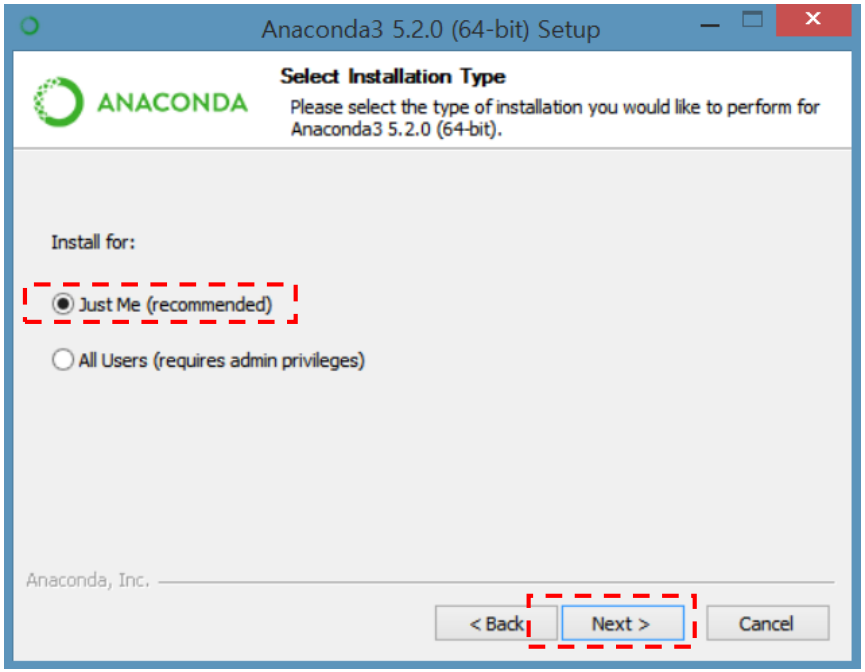


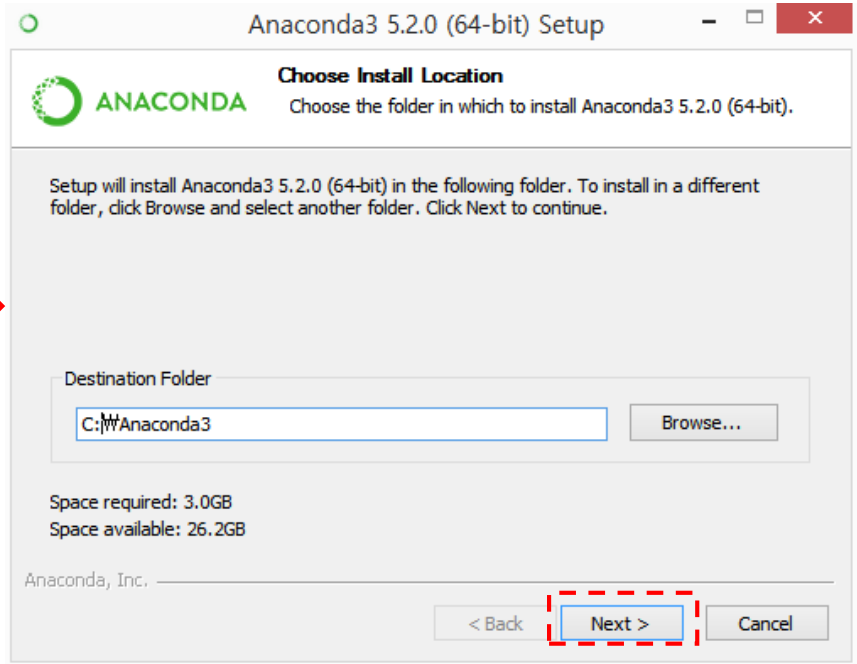
아나콘다 설치


Anaconda3 5.2.0 (64-bit) Setup
ANACONDA.
Welcome to Anaconda3 5.2.0 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 5.2.0 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel


Anaconda3 5.2.0 (64-bit) Setup
ANACONDA
License Agreement
Please review the license terms before installing Anaconda3 5.2.0 (64-bit).
Press Page Down to see the rest of the agreement.
Anaconda End User License Agreement
Copyright 2015, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 5.2.0 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel

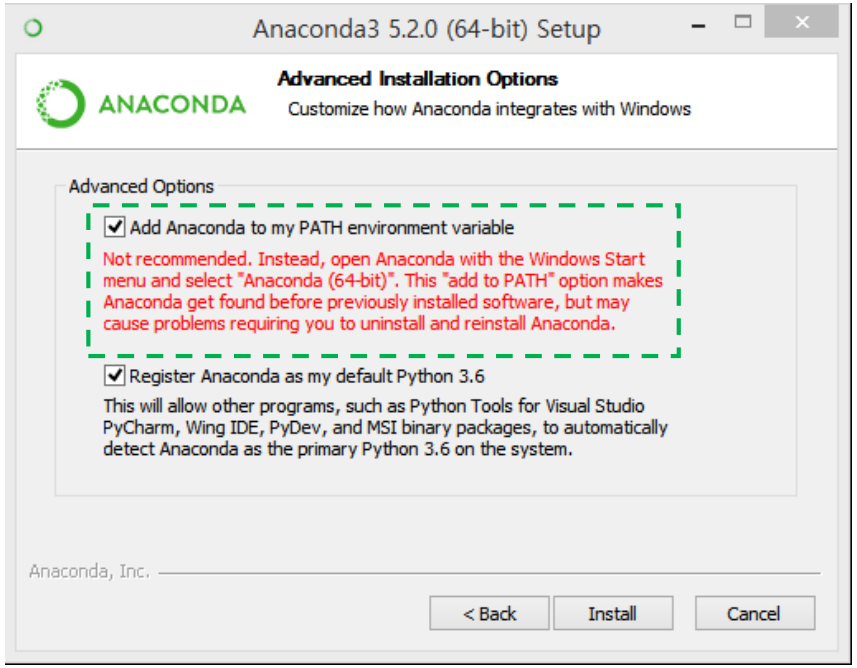
아나콘다 설치

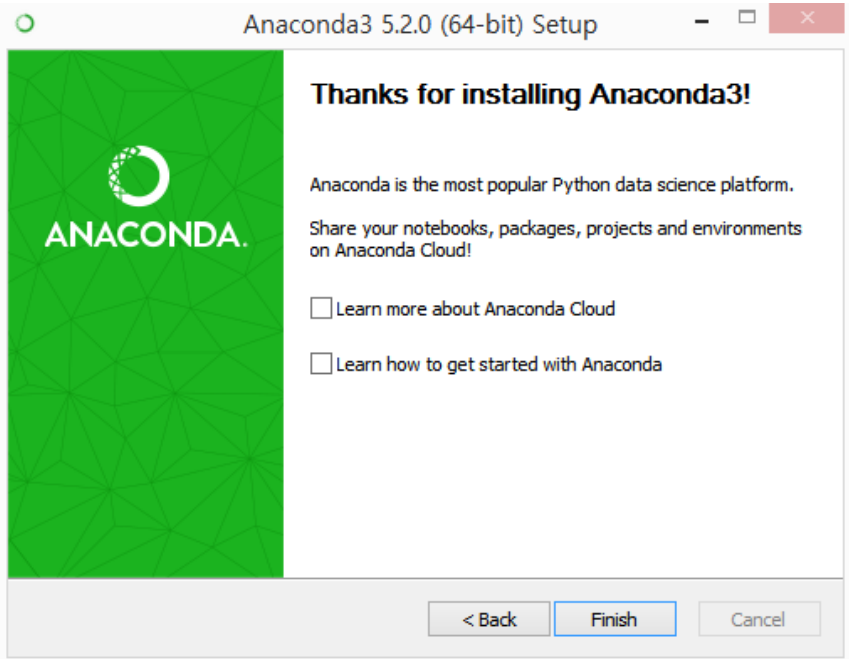

Anaconda3 5.2.0 (64-bit) Setup
ANACONDA
Select Installation Type
Please select the type of installation you would like to perform for Anaconda3 5.2.0 (64-bit).
Install for:
Just Me (recommended)
All Users (requires admin privileges)
Anaconda, Inc.
< Back
Next >
Cancel


Anaconda3 5.2.0 (64-bit) Setup
ANACONDA
Choose Install Location
Choose the folder in which to install Anaconda3 5.2.0 (64-bit).
Setup will install Anaconda3 5.2.0 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.
Destination Folder
C:₩Anaconda3
Browse...
Space required: 3.0GB
Space available: 26.2GB
Anaconda, Inc.
< Back
Next >
Cancel

아나콘다 설치

- 아래 path 체크박스를 체크하지 않으면 사용자가 직접 경로 설정을 해야한다.





Conda 를 활용한 패키지 관리

- 윈도우, 맥OS, 리눅스에서 쓸 수 있는 오픈소스 패키지/환경 관리 시스템
  - conda 명령어를 통해 실행한다.

Anaconda Prompt

```
(base) C:₩Users₩kjsclub12>conda list
# packages in environment at C:₩Anaconda3:
#
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0            py36he6757f0_0
alabaster                 0.7.10           py36hcd07829_0
anaconda                  5.2.0                    py36_3
anaconda-client           1.6.14                   py36_0
anaconda-navigator        1.8.7                    py36_0
anaconda-project          0.8.2            py36hfad2e28_0
apriori                   1.0.0                     <pip>
apyori                    1.1.1                     <pip>
asn1crypto                0.24.0                   py36_0
astroid                   1.6.3                    py36_0
astropy                   3.0.2            py36h452e1ab_1
attrs                     18.1.0                   py36_0
babel                     2.5.3                    py36_0
backcall                  0.1.0                    py36_0
backports                 1.0              py36h81696a8_1
backports.shutil_get_terminal_size 1.0.0            py36h79ab834_2
beautifulsoup4            4.6.0            py36hd4cc5e8_1
bitarray                  0.8.1            py36hfa6e2cd_1
bkcharts                  0.2              py36h7e685f7_0
blas                      1.0                         mkl
```

아나콘다 설치

자주 사용하는 Conda 명령어

기능	명령어
패키지를 설치	$ conda install numpy # numpy 설치
동시에 여러 패키지를 설치	$ conda install numpy scipy pandas # numpy, scipy, pandas 동시 설치
특정 버전 설치	$ conda install numpy=1.10 # 특정 버전 설치
패키지 제거	$ conda remove package_name
패키지 업데이트	$ conda update package_name
모든 패키지 업데이트	$ conda update –all
설치된 목록 출력	$ conda list
설치하려는 패키지 검색	$ conda search search_term

가상환경 설정

Conda 활용

- 파이썬에서는 한 라이브러리에 대해 하나의 버전만 설치가능
- 여러 개의 프로젝트를 진행하게 되면 문제가 될 수 있기때문에, 작업을 바꿀 때마다 다른 버전의 라이브러리를 설치해야 한다.
- 이를 방지하기 위한 격리된 독립적인 가상환경이 제공된다.
- 일반적으로 프로젝트마다 다른 하나의 가상환경을 생성한 후 작업을 시작한다.
- 가상환경의 대표적인 모듈은 3가지가 있다. (venv, virtualenv, conda)
- Anconda python을 설치하였을 때 기본적으로 conda 모듈이 같이 설치 되기 때문에 여기서도 conda를 사용한다.

가상환경 설정

- Anconda prompt 창에서 아래의 명령어 실행
  > conda create --name 환경이름 python = version입력(ex. 2.7,3.6)

- Anaconda3 폴더 아래 envs라는 폴더 안에 가상환경이 생성되며, 파이썬 2.7을 포함한 패키지들이 인스톨 된다.

- y를 눌러 진행하도록 하자.

```
(base) C:WUsersWEXEM>conda create --name py27 python=2.7
Solving environment: done

## Package Plan ##

  environment location: C:WAnaconda3WenvsWpy27

  added / updated specs:
    - python=2.7


The following NEW packages will be INSTALLED:

    certifi:        2018.4.16-py27_0
    pip:            10.0.1-py27_0
    python:         2.7.15-he216670_0
    setuptools:     39.2.0-py27_0
    vc:             9-h7299396_1
    vs2008_runtime: 9.00.30729.1-hfaea7d5_1
    wheel:          0.31.1-py27_0
    wincertstore:   0.2-py27hf04cefb_0

Proceed ([y]/n)?
```

가상환경 설정

- conda activate py27 명령어로 새로 만들어진 가상환경을 실행 할 수 있다.

- conda deactivate 명령어는 가상환경을 종료하고 기존 환경으로 돌아오게 하는 명령어 이다.

- 3.6 버전의 새로운 가상환경을 만들고 2.7 과 번갈아 가며 작업을 수행하도록 해보자.
  conda info --envs 명령어로 환경정보를 볼 수 있다.

```
Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate py27
#
# To deactivate an active environment, use
#
#     $ conda deactivate


(base) C:₩Users₩EXEM>python --version
Python 3.6.5 :: Anaconda, Inc.

(base) C:₩Users₩EXEM>conda activate py27

(py27) C:₩Users₩EXEM>python --version
Python 2.7.15 :: Anaconda, Inc.

(py27) C:₩Users₩EXEM>conda deactivate

(base) C:₩Users₩EXEM>python --version
Python 3.6.5 :: Anaconda, Inc.

(base) C:₩Users₩EXEM>
```

3. Data Handling

쥬피터 노트북과 친해지기

- 기존의 Ipython Notebook에서 시작하여 코드, 이미지, 분석결과, 코멘트, 수식 등을 분석자가 "노트북" 처럼 keep 할 수 있게 하여 다른 사람들이 분석의 흐름을 읽기 쉽게 만들 수 있는 유연하고 강력 한 툴이다.

< 주피터 노트북 초기 화면>

쥬피터 노트북과 친해지기

- 윈도우환경 아나콘다에서 python 2.x, python3.x 다중커널 설정
  - ▶ python 3.x 가상환경에 커널 인스톨

```
(py36) > conda install ipykernel
```

  - ▶ python 3.x에 커널 설정

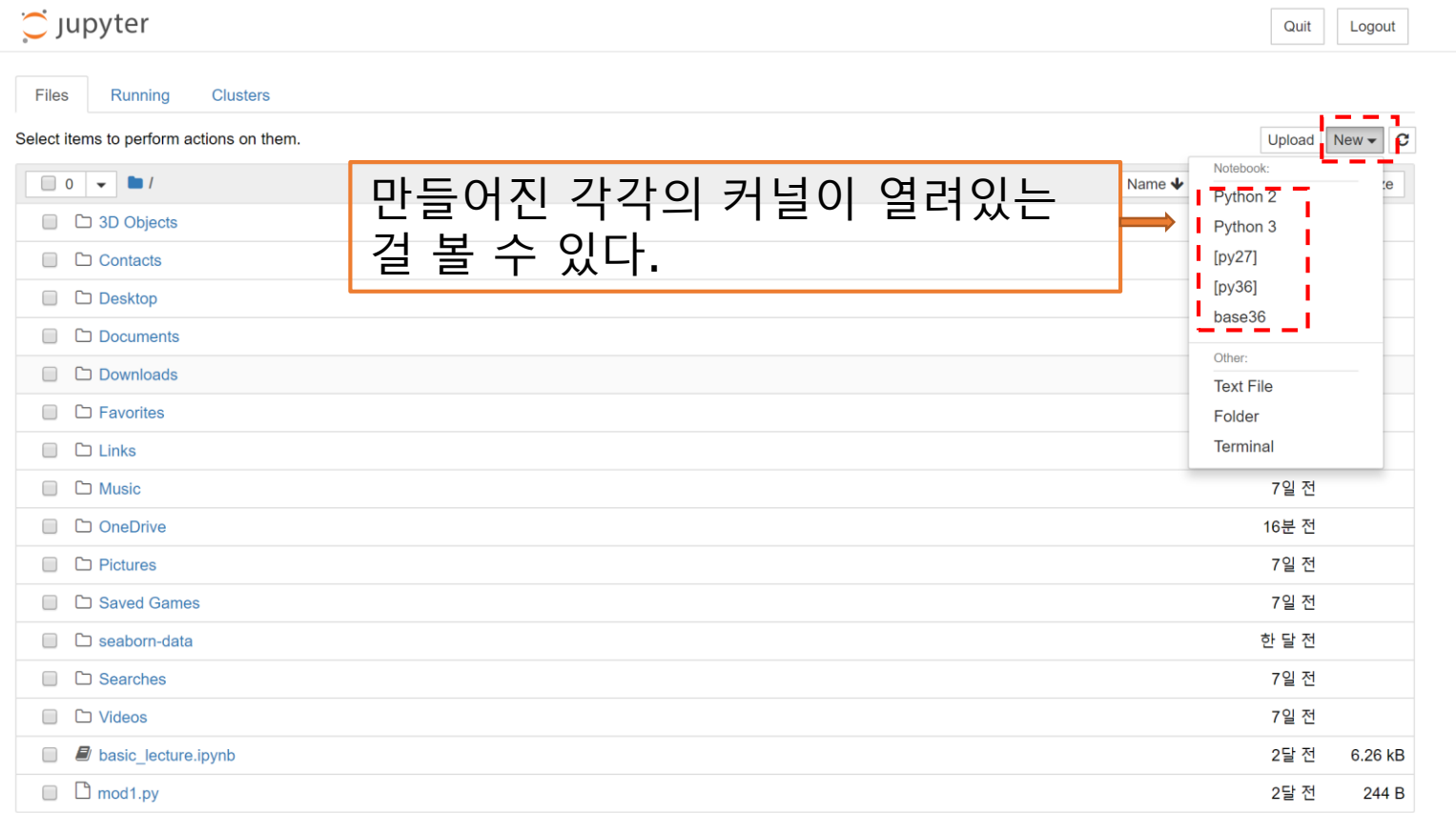
```
(py36) > python -m ipykernel install --user --name [virtualEnv] --display-
         name "[displayKenrelName]"
```

  - ▶ 기존 python 3버전 아나콘다를 설치하였을 때 그 버전에 맞는 커널 하나는 열려있다.

쥬피터 노트북과 친해지기

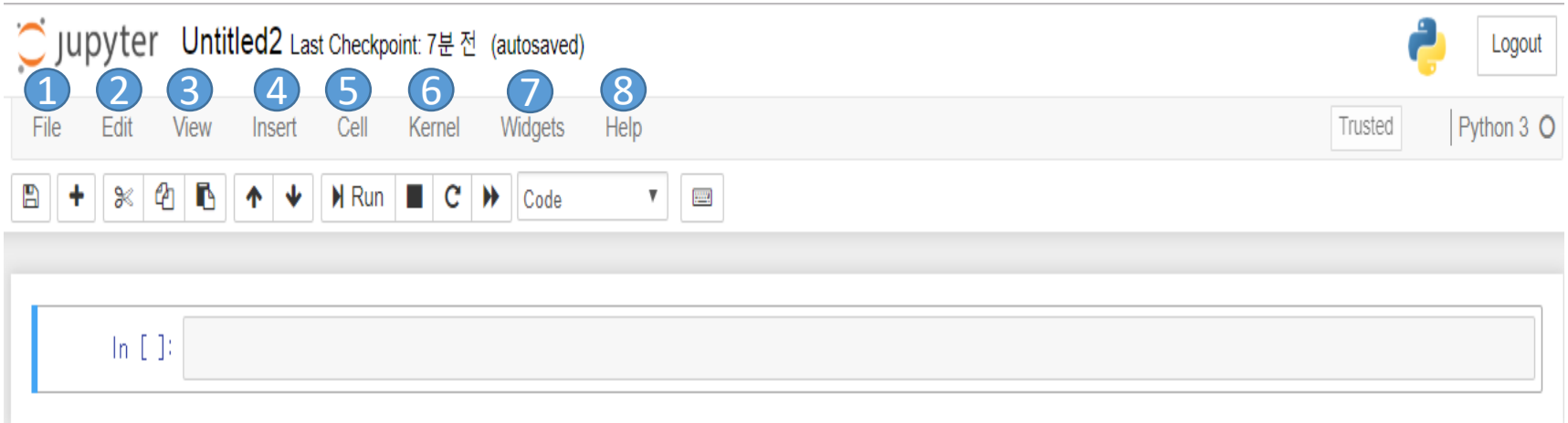
주피터 노트북에서 파이썬 실행

```
(py36) > jupyter notebook
```



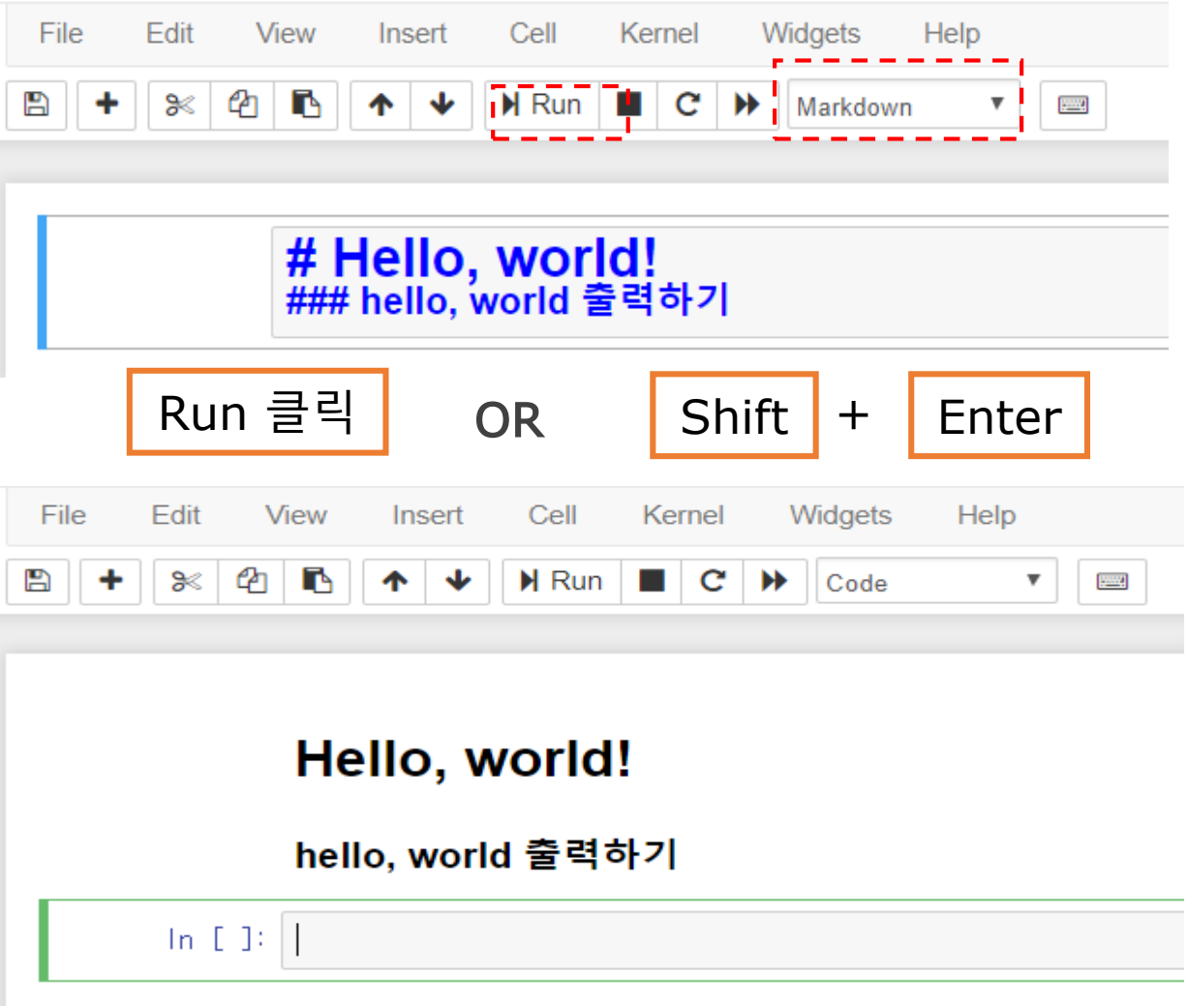
만들어진 각각의 커널이 열려있는 걸 볼 수 있다.

쥬피터 노트북과 친해지기



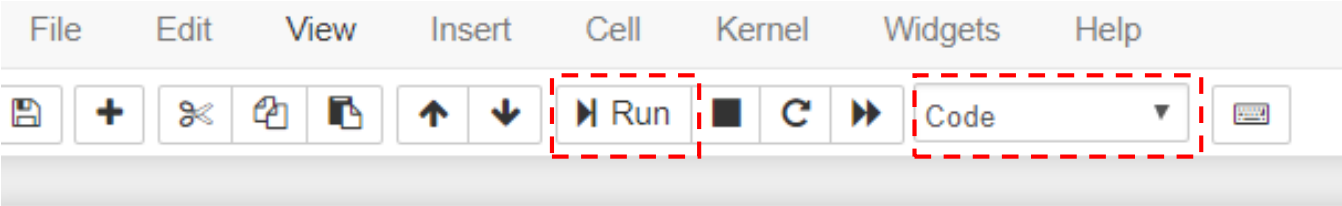
1. 워킹 디렉토리를 보여줍니다
2. 편집 작업 메뉴 입니다.
3. 화면 설정 메뉴 입니다.
4. Cell 을 insert 하는 메뉴입니다.
5. Cell 관리 메뉴입니다.
6. 커널을 제어하는 메뉴입니다.
7. 위젯 관리 메뉴입니다.
8. 단축키 관리, 쥬피터노트북 정보를 볼 수 있는 메뉴입니다.

쥬피터 노트북과 친해지기



쥬피터 노트북과 친해지기

Run 클릭 OR Shift + Enter



Hello, world!

hello, world 출력하기

```
In [1]: 3 + 4
Out[1]: 7

In [2]: 3 * 5
Out[2]: 15
```

파이썬 기초

- 점프 투 파이썬
- http://pythonstudy.xyz/python/article/1-Python-%EC%86%8C%EA%B0%9C
- https://opentutorials.org/course/2991/17970
- https://zetawiki.com/wiki/%ED%8C%8C%EC%9D%B4%EC%8D%AC_%EC%A0%9C%EC%96%B4%EB%AC%B8
- https://www.codecademy.com/ko/learn/learn-python?composer_curriculum_redirect=python-ko
- https://www.inflearn.com/course/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EA%B8%B0%EC%B4%88-%EA%B0%95%EC%A2%8C/#course-curriculum

4. 라이브러리

numpy?

- “Numerical Python” 의 약자로 대규모 다차원 배열과 행렬 연산에 필요한 다양한 함수 제공
- 데이터분석을 할 때 사용되는 라이브러리 pandas, matplotlib의 기반
- 기본적으로 array 라는 단위로 데이터 관리
- 행렬이라는 개념으로 생각하면 됨

https://www.youtube.com/watch?v=aHthqCgsSFs&list=PLBHVuYlKEkULZLnKLzRq1CnNBOBlBTkqp

numpy 특징

- 일반 list에 비해 빠르고, 메모리에 효율적
- 반복문(for,while) 없이 데이터 배열에 대한 처리를 지원함
- 선형대수와 관련된 다양한 기능을 제공
- C, C++, 포트란으로 작성된 함수를 파이썬으로 연동하여 쓸 수 있도록 구성

numpy import

```
import numpy as np
```

- numpy의 호출 방법
- 일반적으로 numpy는 np라는 alias(별칭) 이용해서 호출
- 세계적인 약속 같은 것으로 특별한 이유는 없음

Array creation

```
import numpy as np
test_array = np.array([1, 3, 5, 7], float)
test_array
```

array([1., 3., 5., 7.])

```
type(test_array)
```

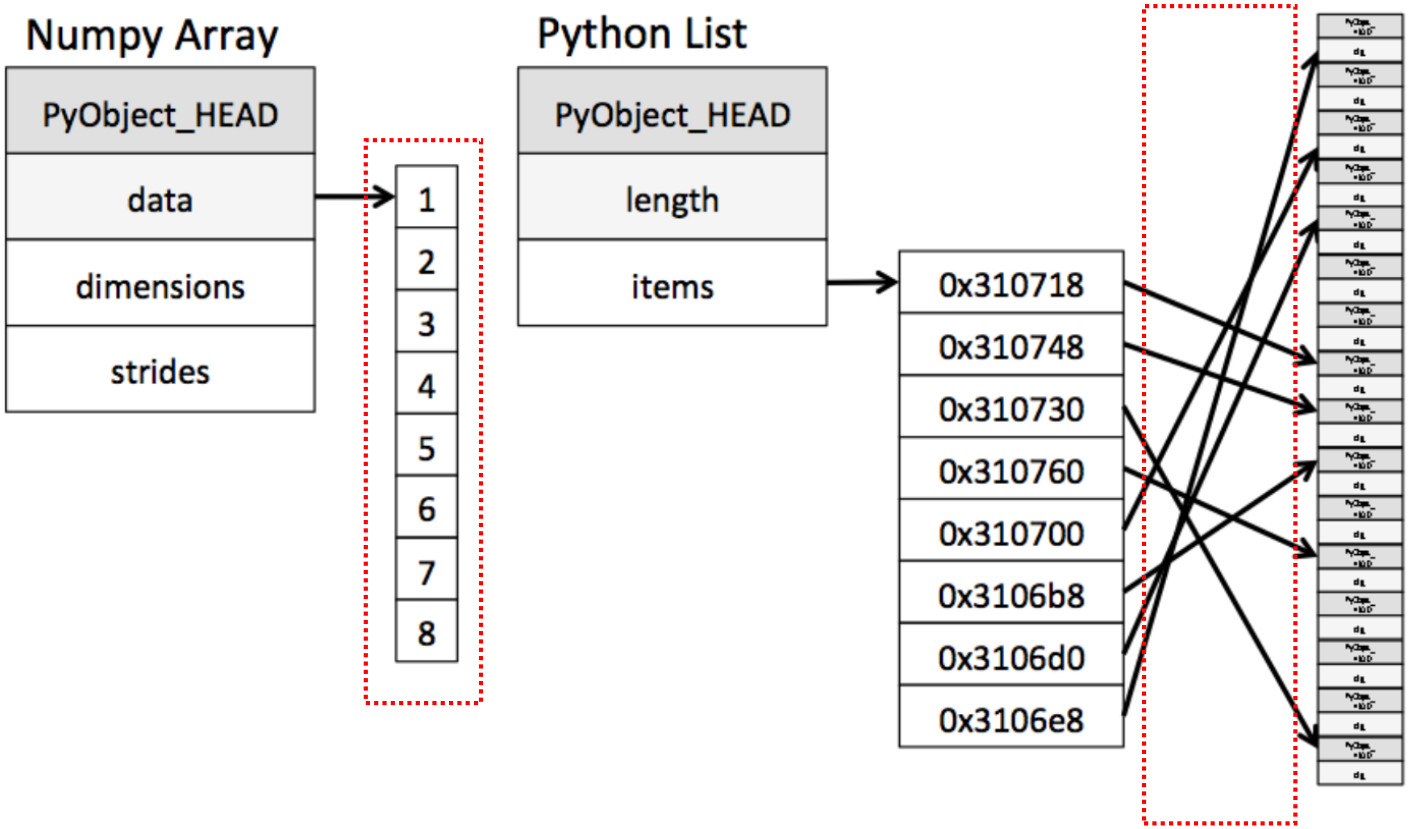
numpy.ndarray

```
type(test_array[3])
```

numpy.float64

- numpy는 np.array 함수를 활용하여 배열을 생성 -> ndarray
- numpy는 하나의 데이터 type만 배열에 넣을 수 있음
- list와 가장 큰 차이점, Dynimic typing not supported
- C의 array를 사용하여 배열 생성

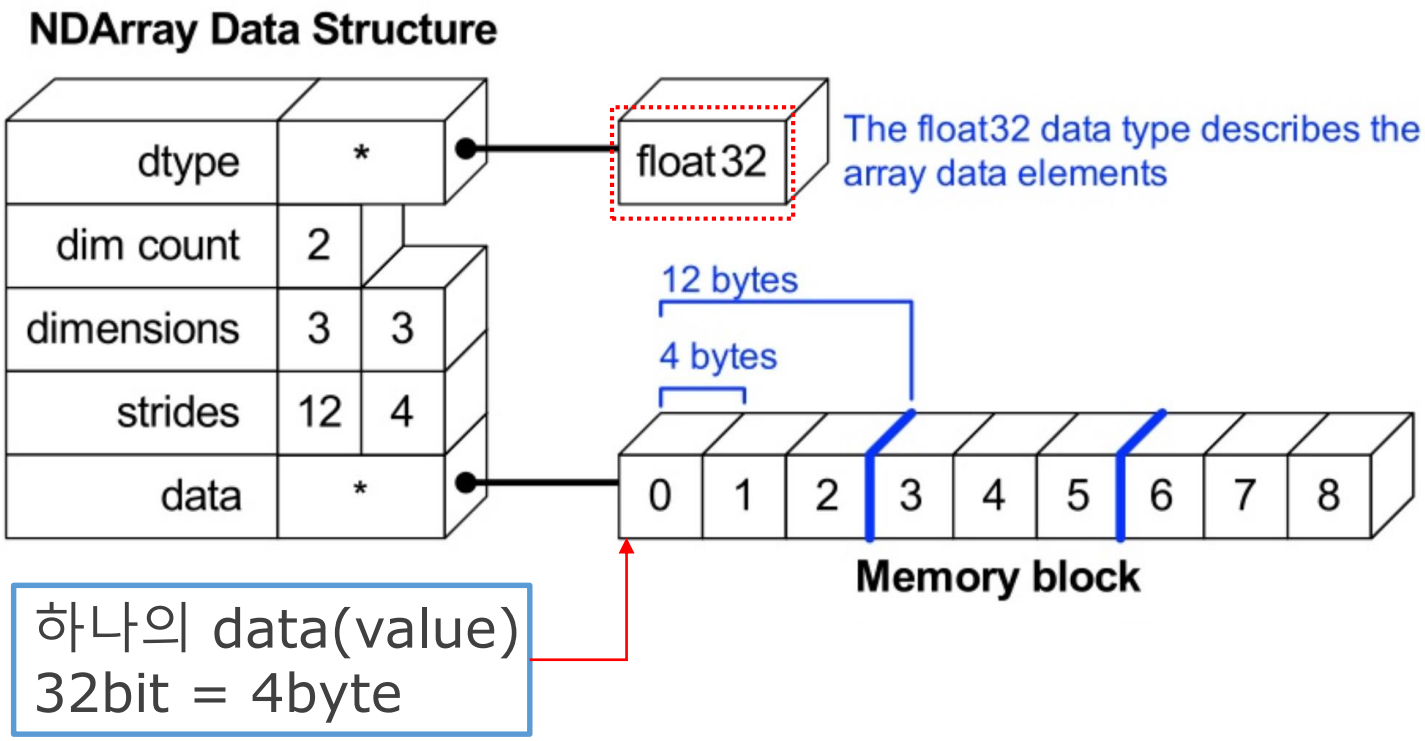
Array creation



- Numpy : data가 연속되어 있다. 같은 크기의 메모리에 할당
- List : 값의 주소값을 저장하여 하나씩 찾아와야 한다

http://jakevdp.github.io/blog/2014/05/09/why-python-is-slow/

Array creation



4byte씩 붙어서 memory block이 구성된다

Array shape(vector)

- Array (vector, matrix, tensor)의 크기, 형태 등에 대한 정보

```
test_array = np.array([1, 3, 5, "7"], float)
test_array
```

```
array([ 1., 3., 5., 7. ])
```

1	3	5	7

→ (4,)

ndarray의 구성

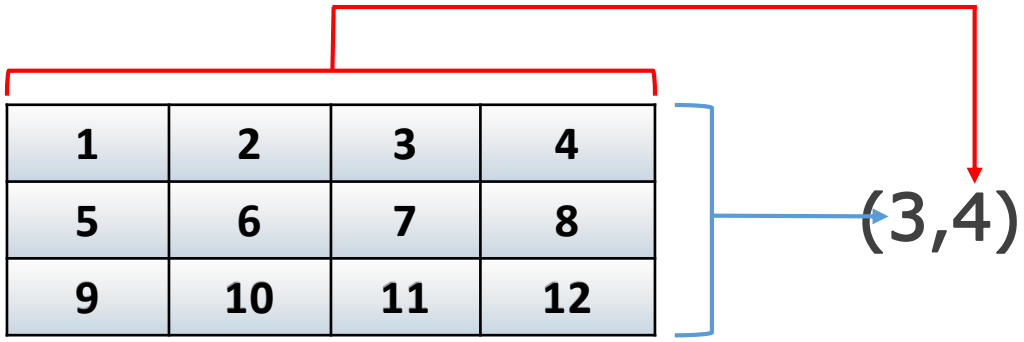
ndarray의 shape
(type : tuple)

- 1차원에 4개의 element가 있는 벡터

Array shape(matrix)

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

(3,4)



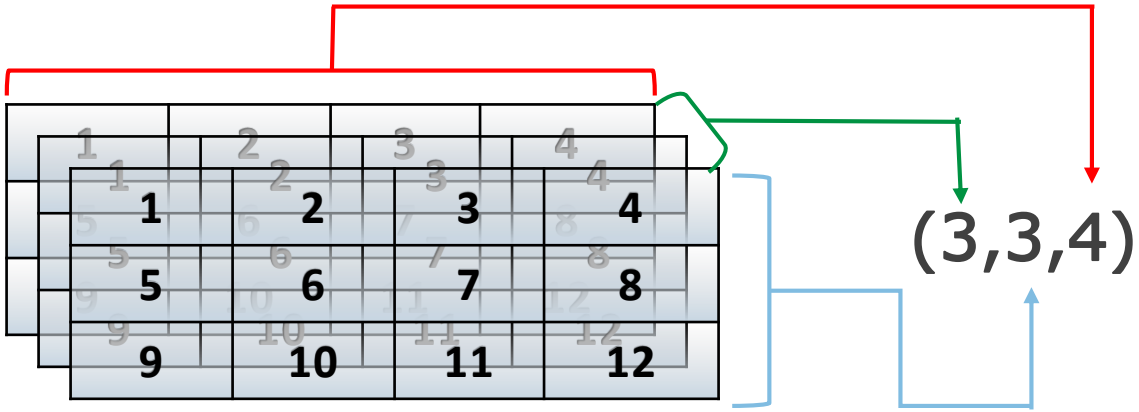
ndarray의 구성
2차원 matrix

ndarray의 shape
(type : tuple)

Array shape(3rd order tensor)

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

(3, 3, 4)



ndarray의 구성
3차원 matrix

ndarray의 shape
(type : tuple)

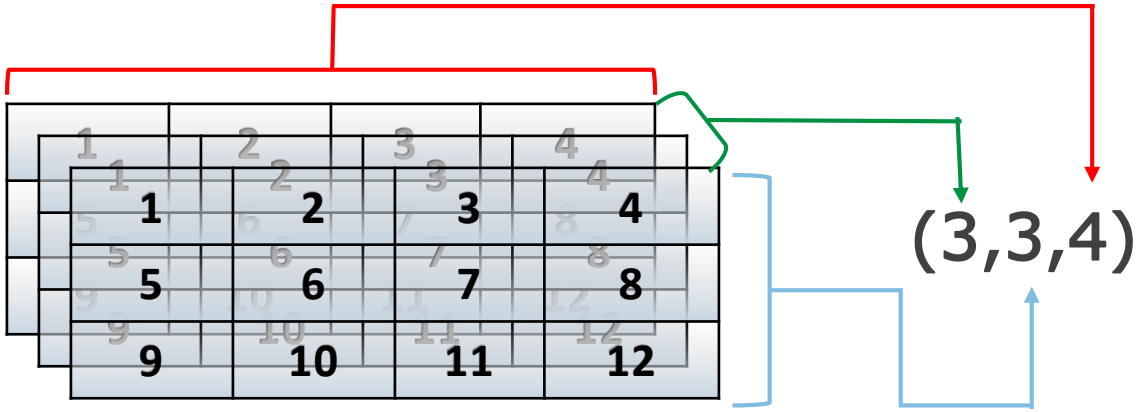
Array shape – ndim & size

```
tensor = [ [[1,2,3,4],[5,6,7,8],[9,10,11,12]],
           [[1,2,3,4],[5,6,7,8],[9,10,11,12]],
           [[1,2,3,4],[5,6,7,8],[9,10,11,12]] ]
np.array(tensor, int).ndim  # number of dimension
```

3

```
np.array(tensor, int).size  # data의 개수
```

36



ndarray의 구성
3차원 matrix

ndarray의 shape
(type : tuple)

Array dtype

- Ndarray의 single element가 가지는 data type
- 각 element가 차지하는 memory의 크기가 결정됨

```
np.array( [[1, 2.6, 3.2], [4, 5.1, 6]], dtype = int )
```

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

```
np.array( [[1, 2.6, 3.2], [4, "5", 6]], dtype = np.float32 )
```

```
array([[1. , 2.6 , 3.2],
       [4. , 5. , 6. ]]. dtype=float32 )
```

Array dtype

- nbyte : ndarray object의 메모리 크기 리턴

```
np.array( [[1, 2.6, 3.2], [4, "5", 6]], dtype = np.float32 ).nbytes
```

24 ➔ 32bits = 4bytes → 6 * 4bytes

```
np.array( [[1, 2.6, 3.2], [4, "5", 6]], dtype = np. float64 ).nbytes
```

48 ➔ 64bits = 8bytes → 6 * 8bytes

```
np.array( [[1, 2.6, 3.2], [4, "5", 6]], dtype = np.int8 ).nbytes
```

6 ➔ 8bits = 1bytes → 6 * 1bytes

reshape

- Array의 shape의 크기를 변경함 (element의 개수는 동일)

```
t_matrix = [[1, 2, 3, 4], [5, 6, 7, 8]]
np.array(t_matrix).shape
```

(2, 4)

```
np.array(t_matrix).reshape(8, )
```

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

```
np.array(t_matrix).reshape(8, ).shape
```

(8,)

1	2	3	4
5	6	7	8

(2,4)

reshape →

1	2	3	4	5	6	7	8

(8,)

reshape

- Array의 size만 같다면 다차원으로 자유로이 변형 가능

```
np.array(t_matrix).reshape(2, 4).shape
```

(2, 4)

```
np.array(t_matrix).reshape(-1, 2).shape
```

(4, 2) ➔ -1은 size를 기반으로 row개수 산정한다

```
np.array(t_matrix).reshape(2, 2, 2)
```

```
array([[[1, 2],
        [3, 4],

       [[5, 6],
        [7, 8]]])
```

```
np.array(t_matrix).reshape(2, 2, 2).shape
```

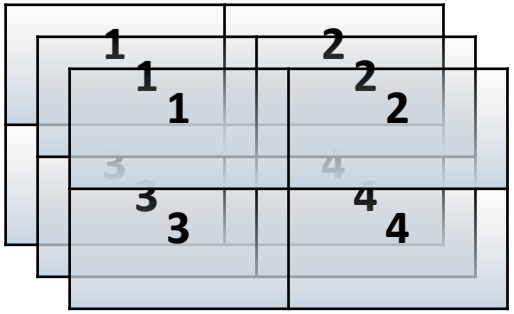
(2, 2, 2)

flatten

- 다차원 array를 1차원 array로 변환

```
t_matrix = [ [[1, 2], [3, 4]], [[1, 2], [3, 4]], [[1, 2], [3, 4]] ]
np.array(t_matrix).flatten()
```

array([1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4])



(3, 2, 2)

1	2	3	4	1	2	3	4	1	2	3	4

(16,)

indexing

- list와 달리 이차원 배열에서 [0,0] 과 같은 표기법 제공
- matrix 일경우 앞은 row 뒤는 column

```
a = np.array([[1, 2.2, 3], [4, 5, 6.3], int)
print(a)
print(a[0,0])  # row 0자리, column 0 자리 값 출력
              ‼
print(a[0][0]) # row 0자리, column 0 자리 값 출력

a[0,0] = 7   # matrix 0,0 에 7 할당
a[0][0] = 8  # matrix 0,0 에 8 할당
```

indexing

```
a = np.array([[1, 2.2, 3], [4, 5, 6.3], int)
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

```
print(a[0,0])
```

```
1
```

```
print(a[0][0])
```

```
1
```

```
a[0, 0] = 7
print(a)
```

```
[[7 2 3]
 [4 5 6]]
```

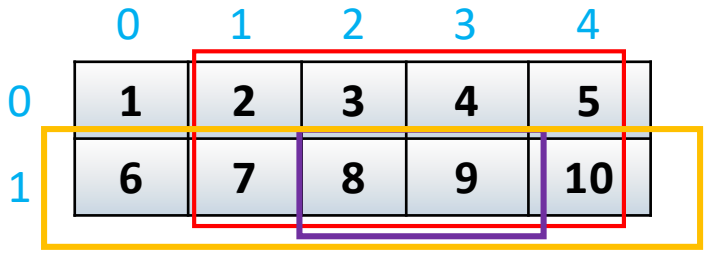
```
a[0][0] = 8
print(a)
```

```
[[8 2 3]
 [4 5 6]]
```

slicing

- list와 달리 행과 열 부분을 나눠서 slicing이 가능함
- matrix 부분 집합 추출할 때 유용

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

a[:, 1:]  # 전체 row의 1 열 이상
a[1, 2:4]  # 1 row의 2 열~3 열
a[1:3]  # 1 row ~ 로우 전체
```



slicing

```
a = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]], int)
print(a)
```

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

```
a[:, 1:]
```

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

```
a[1, 2:4]
```

```
array([8, 9])
```

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

```
array([[ 6,  7,  8,  9, 10]])
```

slicing

```
a = np.array([[0,1,2,3,4], [5,6,7,8,9], [10,11,12,13,14]], int)
print(a)
```

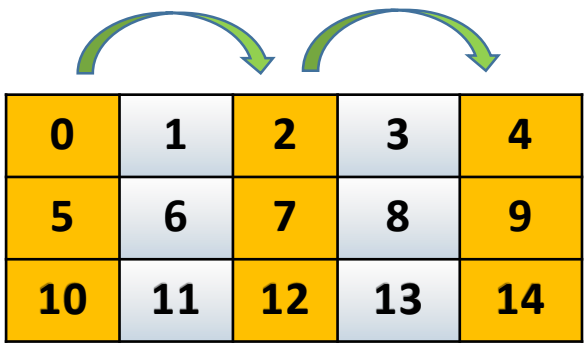
```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
```

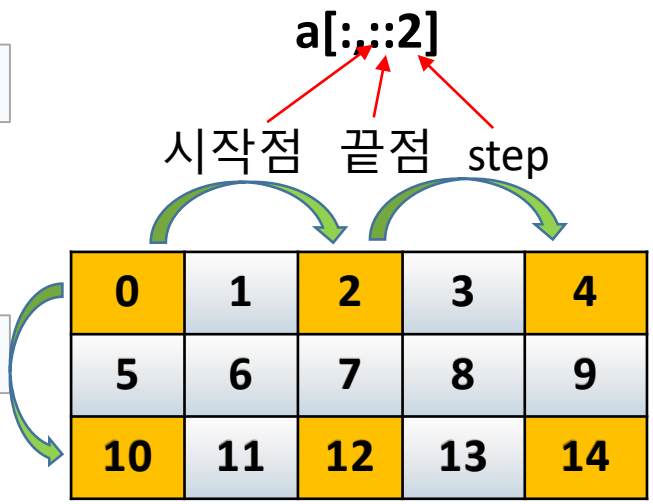
```
a[:,::2]
```

```
array([[ 0,  2,  4],
       [ 5,  7,  9],
       [10, 12, 14]])
```

```
a[::2,::2]
```

```
array([[ 0,  2,  4],
       [10, 12, 14]])
```





arange

- array의 범위를 지정하여, 값의 list를 생성하는 명령어

```
np.arange(20)  # list의 range와 같은 역할, intege로 0부터 19까지 배열추출
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

```
np.arange(0, 1, 0.2) # float 가능
```

```
array([0. , 0.2, 0.4, 0.6, 0.8])
```

```
np.arange(20).reshape(4,5)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```

zeros, ones, empty

- zeros : 0으로 가득찬 ndarray 생성

np.zeros(shape, dtype, order)

```
np.zeros(shape = (5,2), dtype = np.int8)  # 5 by 2 zero matrix 생성, int8
```

```
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]], dtype=int8)
```

```
np.zeros((5,2)) # 5 by 2 zero matrix 생성, float
```

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

zeros, ones, empty

- ones : 1로 가득찬 ndarray 생성

  np.ones(shape, dtype, order)

```
np.ones(shape = (5,2), dtype = np.int8)  # 5 by 2 one matrix 생성, int8
```

```
array([[1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1]], dtype=int8)
```

```
np.ones((5,2)) # 5 by 2 one matrix 생성, float
```

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

zeros, ones, empty

- empty : shape만 주어지고 비어있는 ndarray 생성
  (memory initialization 이 되지 않음)

np.empty(shape, dtype, order)

```
np.empty(shape = (3,2), dtype = np.int8)
```

```
array([[64,  0],
       [ 0,  0],
       [ 0,  0]], dtype=int8)
```

```
np.empty((3,2))
```

```
array([[4.24399158e-314, 8.48798317e-314],
       [4.24399158e-314, 8.48798317e-314],
       [4.24399158e-314, 8.48798317e-314]])
```

something_like

- 기존 ndarray의 shape 크기만큼 1 or 0 or empty array 반환

```
t_matrix = np.arange(15).reshape(3,5)
np.ones_like(t_matrix)
```

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

```
t_matrix1 = np.arange(15).reshape(3,5)
np.zeros_like(t_matrix1)
```

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

```
t_matrix2 = np.arange(15).reshape(3,5)
np.empty_like(t_matrix2)
```

```
array([[     0,      0,      0,      0,      0],
       [     0,      0,      0,      0,      0],
       [  1224,      0,      0, 6357104,      0]])
```

identity

- 단위 행렬(i 행렬)을 생성
  n ➔ number of rows

```
np.identity(n = 3, dtype = np.int8)
```

```
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]], dtype=int8)
```

```
np.identity(n = 5)
```

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

eye

- 대각선이 1인 행렬, k값이 시작 index 변경 가능

```
np.eye(N = 3, M = 4, dtype = np.int)
```

```
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0]], dtype=int8)
```

```
np.eye(4)  # identity 행렬과 같게 출력
```

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

```
np.eye(3, 6, k = 3)  # k → start index
```

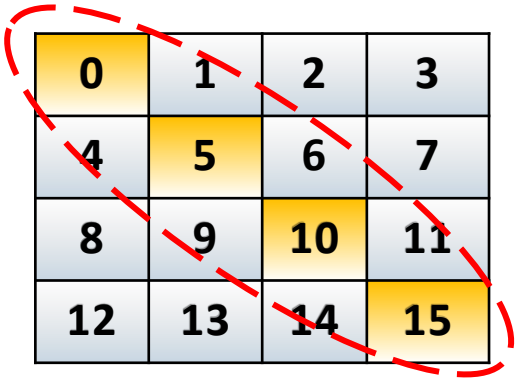
```
array([[0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1.]])
```

diag

- 대각 행렬의 값을 추출

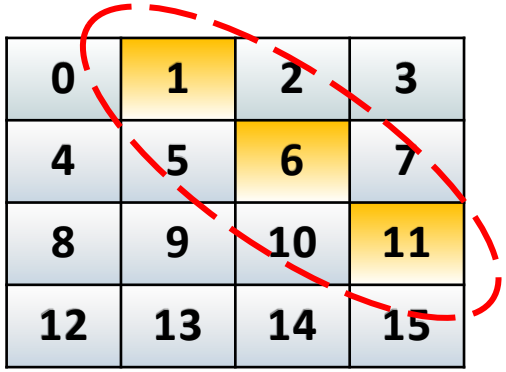
```
t_matrix = np.arange(16).reshape(4, 4)
np.diag(t_matrix)
```

```
array([ 0,  5, 10, 15])
```




```
np.diag(t_matrix, k = 1)
```

```
array([1, 6, 11])
```



random sampling

- 데이터 분포에 따른 sampling으로 array를 생성

```
np.random.uniform(0, 1, 12).reshape(4, 3)  # 균등분포
# np.random.uniform(최소값, 최대값, data 개수)
```

```
array([[0.6471031 , 0.70506654, 0.59970038],
       [0.97119081, 0.87124496, 0.68000872],
       [0.70800846, 0.53908685, 0.57826248],
       [0.54314476, 0.89906667, 0.24140212]])
```

```
np.random.normal(0, 1, 12).reshape(4, 3)  # 정규분포
```

```
array([[-0.29703021, -0.16169222,  0.52078926],
       [ 0.85924274, -0.88387902, -1.43606194],
       [ 1.57495957, -0.18813136, -1.94351926],
       [-0.52597883, -1.30939238, -0.83384486]])
```

sum

- ndarray의 element들의 합, list의 sum과 같은 기능

```
t_array = np.arange(1, 11)
t_array
```

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

```
t_array.sum(dtype = np.int8)
```

```
55
```

```
t_array.sum(dtype = np.float)
```

```
55.0
```

axis

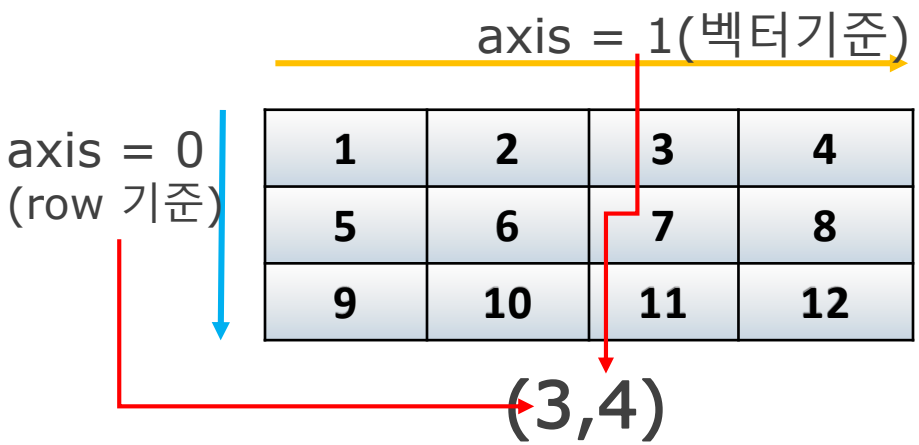
- 모든 operation function을 실행할 때, 기준이 되는 dimension 축

```
t_array = np.arange(1, 13).reshape(3, 4)
t_array
```

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

```
t_array.sum(axis = 0), t_array.sum(axis = 1)
```

```
(array([15, 18, 21, 24]), array([10, 26, 42]))
```

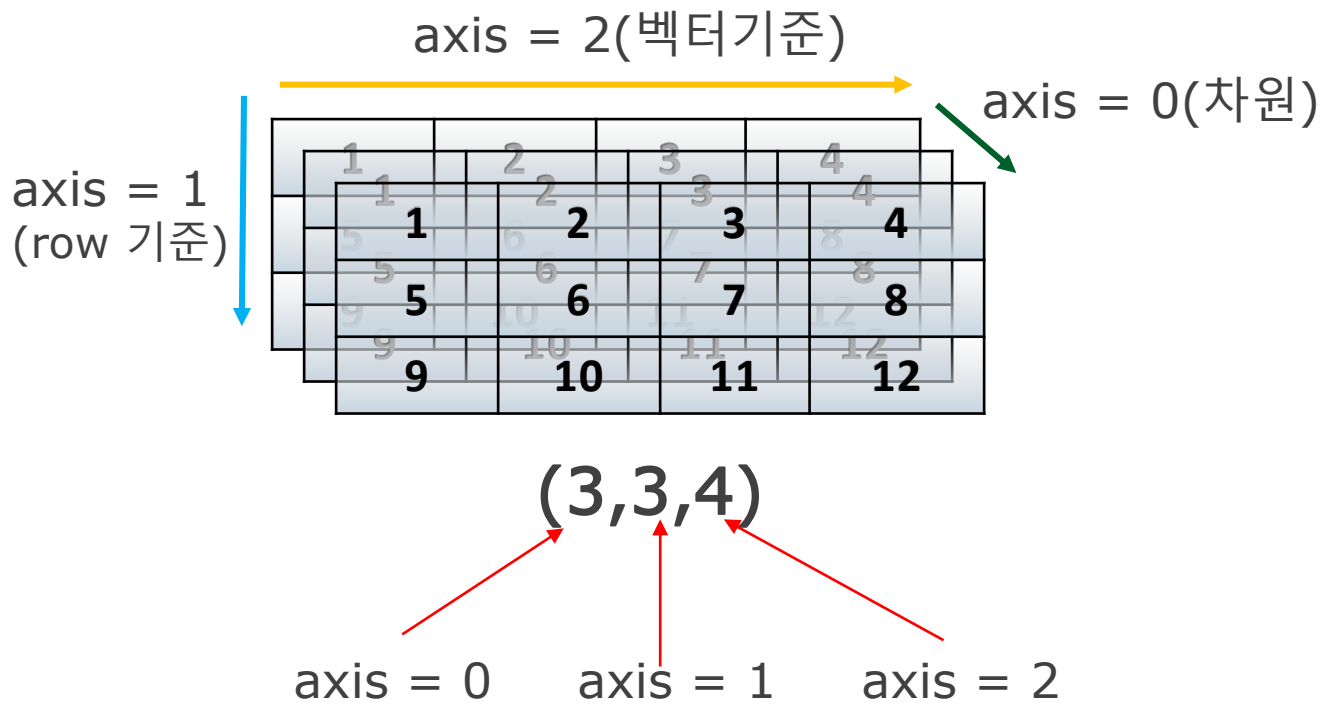


axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축

```
tensor = np.array([t_array, t_array, t_array])
tensor
```

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

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

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



axis

- 모든 operation function을 실행할 때, 기준이 되는 dimension 축

```
tensor.sum(axis = 0)
```

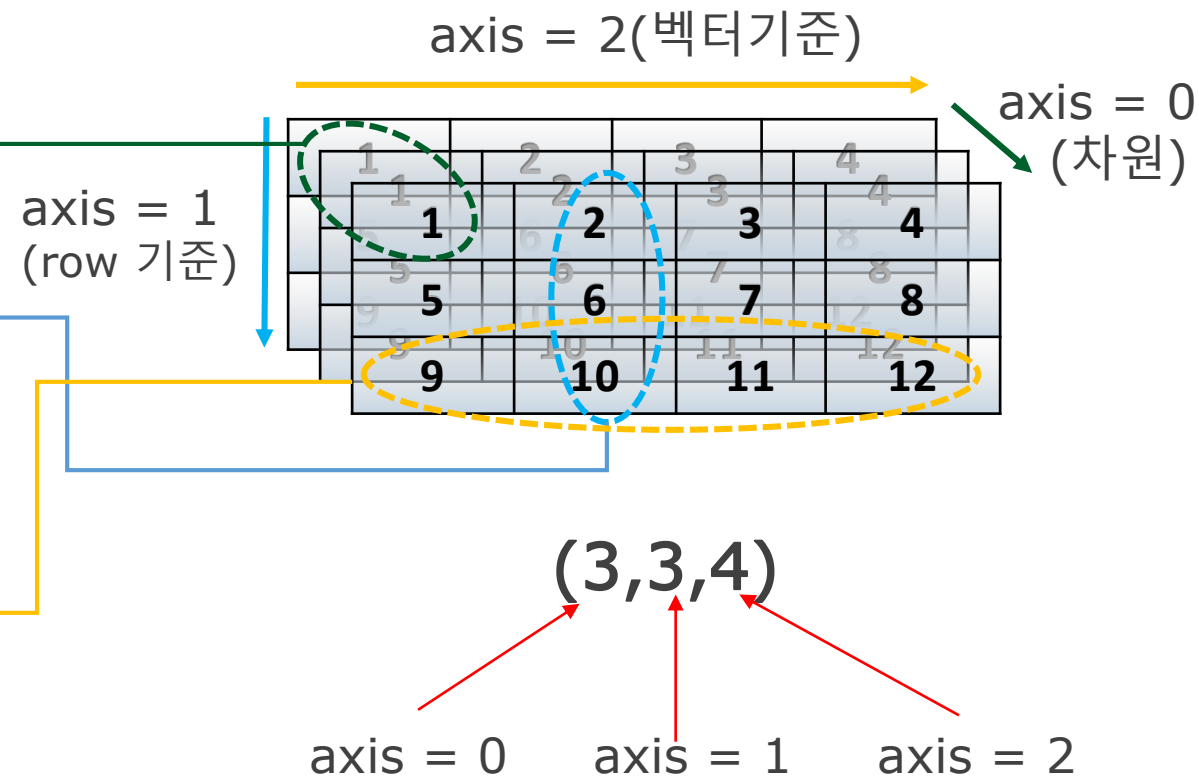
```
array([[ 3,  6,  9, 12],
       [15, 18, 21, 24],
       [27, 30, 33, 36]])
```

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

```
array([[15, 18, 21, 24],
       [15, 18, 21, 24],
       [15, 18, 21, 24]])
```

```
tensor.sum(axis = 2)
```

```
array([[10, 26, 42],
       [10, 26, 42],
       [10, 26, 42]])
```



mean, std

- mean : ndarray의 요소들 간의 평균 반환
- std : ndarray의 요소들 간의 표준편차 반환

```
t_array = np.arange(1, 13).reshape(3, 4)
t_array
```

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

```
t_array.mean(), t_array.mean(axis = 0)
```

```
(6.5, array([5., 6., 7., 8.]))
```

```
t_array.std(), t_array.std(axis = 0)
```

```
(3.452052529534663, array([3.26598632, 3.26598632,
                           3.26598632, 3.26598632]))
```

Mathematical functions

- 다양한 수학 연산자들

지수함수(exponential)

- exp, expm1, exp2, log, log2, log10, log1p, power, sqrt

삼각함수(trigonometric)

- sin, cos, tan, acsin, arccos, atctan

쌍곡선함수(hyperbolic)

- sinh, cosh, tanh, acsinh, arccosh, atctanh

Mathematical functions
- 다양한 수학 연산자들

```
np.exp(t_array)  # 지수함수
```

```
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01],
       [1.48413159e+02, 4.03428793e+02, 1.09663316e+03, 2.98095799e+03],
       [8.10308393e+03, 2.20264658e+04, 5.98741417e+04, 1.62754791e+05]])
```

```
np.sqrt(t_array)  # 루트(√‾)
```

```
array([[1.        , 1.41421356, 1.73205081, 2.        ],
       [2.23606798, 2.44948974, 2.64575131, 2.82842712],
       [3.        , 3.16227766, 3.31662479, 3.46410162]])
```

```
np.sin(t_array)  # sin 함수
```

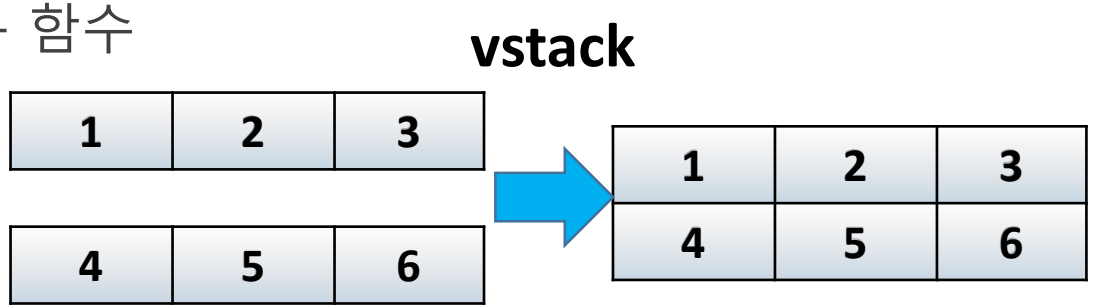
```
array([[ 0.84147098,  0.90929743,  0.14112001, -0.7568025 ],
       [-0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825],
       [ 0.41211849, -0.54402111, -0.99999021, -0.53657292]])
```

concatenate

- Numpy array를 합치는 함수

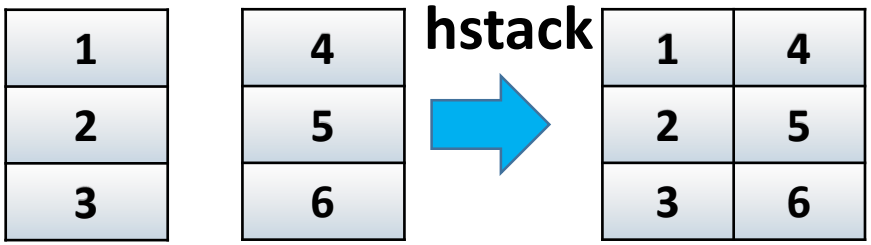
```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.vstack((a, b))
```

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




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

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



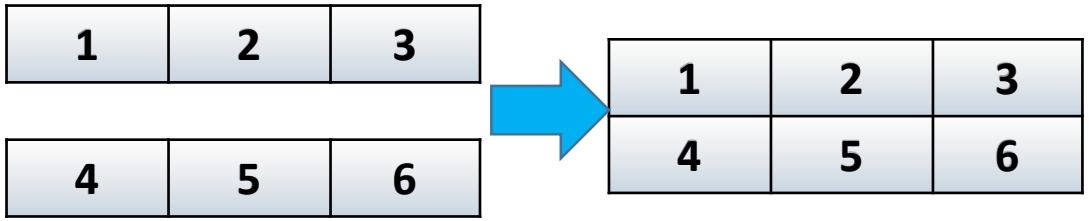
concatenate

- Numpy array를 합치는 함수

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.concatenate((a,b), axis = 0)
```

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

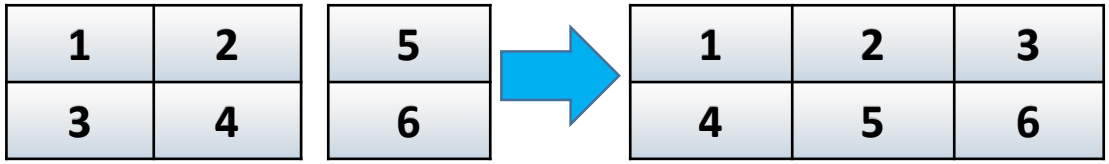
concatenate , axis = 0




```
a = np.array([ [1, 2], [3, 4] ])
b = np.array([ [5, 6] ])
np.concatenate((a, b.T), axis = 1)  # a.T는 a의 역행렬
```

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

concatenate , axis = 1



Operations between arrays

- Numpy는 array간 기본적인 사칙연산 지원

```
a = np.array( [[1, 2, 3], [4, 5, 6]], float )
a + a  # matrix + matrix 연산
```

```
array([[ 2.,  4.,  6.],
       [ 8., 10., 12.]])
```

```
a – a  # matrix – matrix 연산
```

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

```
a * a # matrix 내 요소들간 같은 위치에 있는 값들끼리 연산
```

```
array([[ 1.,  4.,  9.],
       [16., 25., 36.]])
```

Element-wise operations

- Array간 shape이 같을 때 일어나는 연산

1	2	3
4	5	6
7	8	9

×

1	2	3
4	5	6
7	8	9

=

1	4	9
16	25	36
49	64	81

```
t_matrix = np.arange(1, 10).reshape(3, 3)
t_matrix * t_matrix
```

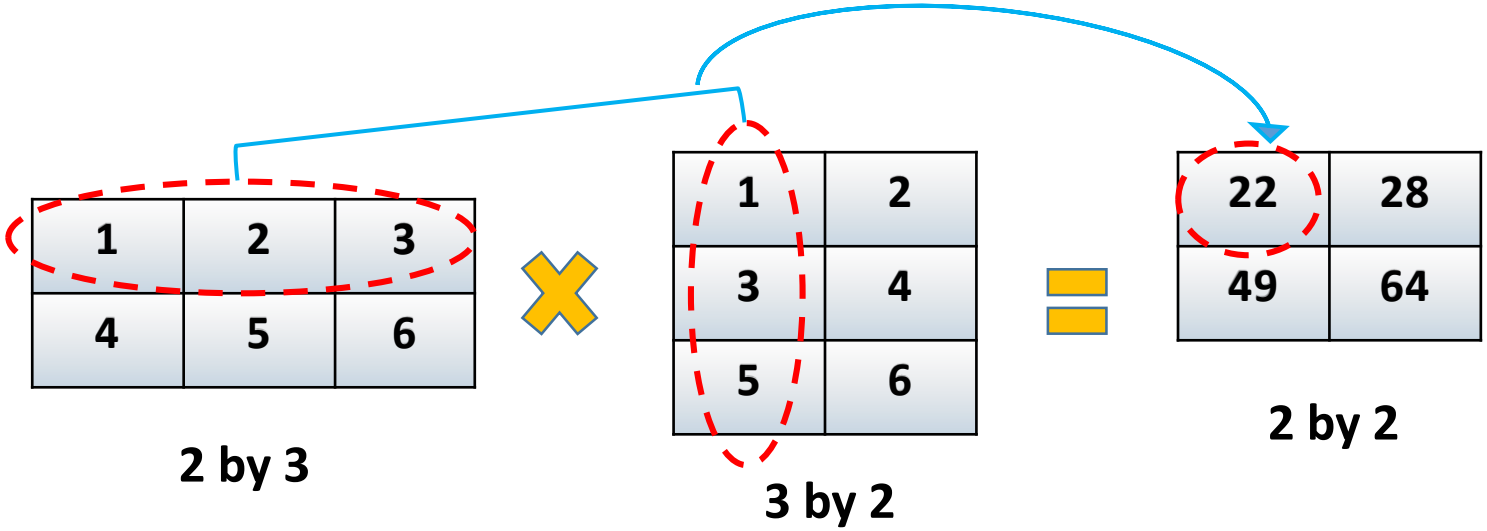
```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

Dot product

- matrix의 기본 연산
- dot 함수 사용

```
dot_a = np.arrange(1, 7).reshape(2, 3)
dot_b = np.arrange(1, 7).reshape(3, 2)
dot_a.dot(dot_b)
```

```
array([[22, 28],
       [49, 64]])
```



transpose

- transpose 또는 T attribute 사용

```
t_a = np.arange(1, 7).reshape(3, 2)
t_a
```

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

```
t_a.transpose()
```

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

```
t_a.T
```

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

```
t_a.T.dot(t_a)
```

```
array([[35, 44],
       [44, 56]])
```

1	2
3	4
5	6

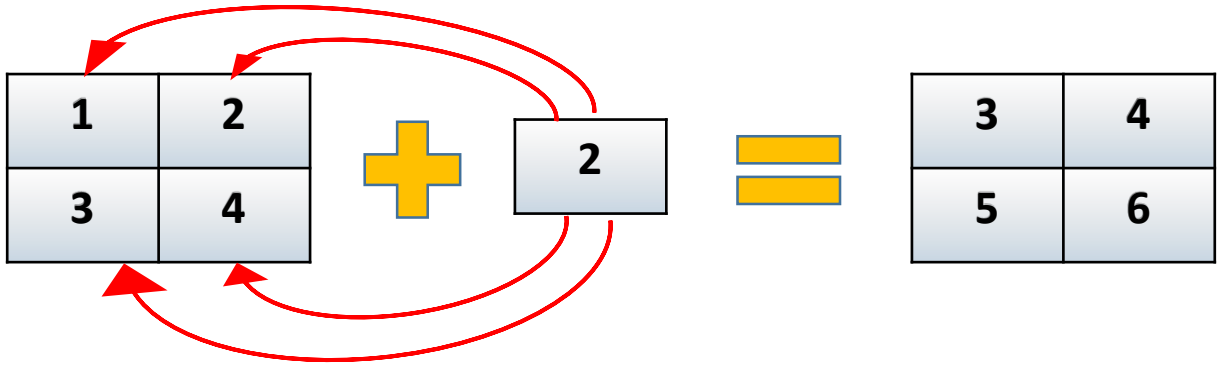
t_a

1	3	5
2	4	6

t_a.transpose() / t_a.T

broadcasting

- shape이 다른 배열 간 연산을 지원하는 기능




```
t_matrix = np.array( [[1, 2], [3, 4]], float )
scalar = 2
t_matrix + scalar  # matrix , scalar 덧셈
```

```
array([[3., 4.],
       [5., 6.]])
```

broadcasting

- shape이 다른 배열 간 연산을 지원하는 기능

```
t_matrix - scalar  # matrix , scalar 뺄셈
```

```
array([[-1.,  0.],
       [ 1.,  2.]])
```

```
t_matrix * 3  # matrix , scalar 곱셈
```

```
array([[ 3.,  6.],
       [ 9., 12.]])
```

```
t_matrix / 3  # matrix , scalar 나눗셈
```

```
array([[0.33333333, 0.66666667],
       [1.        , 1.33333333]])
```

```
t_matrix // 3  # matrix , scalar 몫
```

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

```
t_matrix ** 3  # matrix , scalar 제곱
```

```
array([[ 1.,  8.],
       [27., 64.]])
```

broadcasting

- scalar-matrix 외에도, vector-matrix간의 연산도 지원

```
t_matrix = np.arange(1, 13).reshape(3, 4)
t_vector = np.arange(100, 400, 100)
t_matrix + t_vector
```

```
array([[101, 202, 303],
       [104, 205, 306],
       [107, 208, 309],
       [110, 211, 312]])
```

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

+

100	200	300
100	200	300
100	200	300
100	200	300

=

101	202	303
104	205	306
107	208	309
110	211	312

Numpy performance

- jupyter notebook 환경에서 코드의 퍼포먼스를 체크하는 timeit 이용

```
def sclar_vector_product(scalar, vector):  # scalar와 vector끼리 곱셈
    result = []
    for value in vector:
        result.append(scalar * value)
    return result

i_max = 100000000

vector = list(range(i_max))
scalr = 2

%timeit sclar_vector_product(scalr, vector)  # for loop을 이용한 성능
%timeit[scalar * value for value in range(i_max)] # list comprehension을
                                                  이용한 성능
%timeit np.arange(i_max) * scalar # numpy를 이용한 성능
```

All, Any

- All : Array의 데이터가 전부 조건에 만족하면 True
- Any : Array의 데이터 중 하나라도 조건에 만족하면 True

```
a = np.arange(5)
a
```

```
array([0, 1, 2, 3, 4])
```

```
np.all(a > 3)
```

```
False
```

```
np.all(a < 5)
```

```
True
```

```
np.any(a > 3)
```

```
True
```

```
np.any(a > 5)
```

```
False
```

All, Any

- Numpy는 배열의 크기가 동일 할 때 element간 비교의 결과를 Boolean type으로 리턴

```
a = np.array( [1, 5, 3], float)
b = np.array( [4, 7, 2], float)
a > b
```

array([False, False, True])

```
a == b
```

array([False, False, False])

```
(a > b).any()
```

True

```
(a > b).all()
```

False

logical_and, logical_not, logical_or

```
a = np.array( [2, 3, 1], float )
np.logical_and( a > 0 , a <3 )   # and 조건의 비교
```

```
array([ True, False,  True])
```

```
b = np.array( [ False, True, True ], bool )
np.logical_not(b)  # not 조건의 비교
```

```
array([ True, False, False])
```

```
c = np.array( [ False, False, False ], bool)
np.logical_or( b, c )  # or 조건의 비교
```

```
array([False,  True,  True])
```

np.where

- where(조건, True, False)

```
a = np.array( [2, 3, 1], float )
np.where( a > 1 , 0, 3 )
```

array([0, 0, 3])

```
a = np.arange( 3, 10)
np.where( a > 6 )  # True 값의 index 값 반환
```

(array([4, 5, 6], dtype=int64),)

```
a = np.array( [ 2, np.NaN, np.Inf ], float )
np.isnan( a )  # null 인 경우 True
```

array([False, True, False])

```
np.isfinite( a )  # 한정된 수인 경우 True
```

array([True, False, False])

argmax, argmin

- array내 최대값 또는 최소값의 index 반환

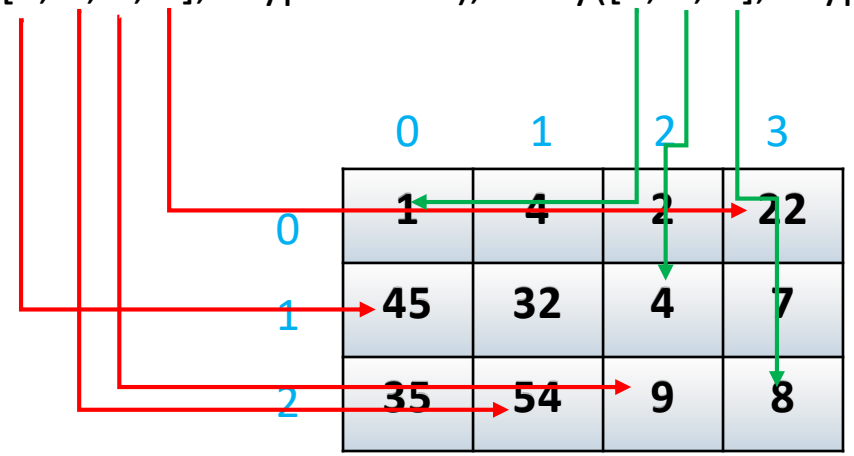
```
a = np.array( [2, 3, 1, 5, 6, 22, 11 ] )
np.argmax(a), np.argmin(a)
```

(5, 2)

- axis 기반의 반환

```
a = np.array( [ [1, 4, 2, 22], [45, 32, 4, 7], [34, 54, 9, 8] ] )
np. argmax(a, axis = 0), np.argmin(a, axis = 1)
```

(array([1, 2, 2, 0], dtype=int64), array([0, 2, 3], dtype=int64))



boolean index

- numpy의 배열은 특정 조건에 따른 값을 배열 형태로 추출 가능
- comparison operation 함수들도 모두 사용 가능

```
t_a = np.array( [3, 5, 8, 0, 7, 4], float )
t_a > 4
```

```
array([False,  True,  True, False,  True, False])
```

```
t_a[t_a > 4]  # 조건이 True 인 index의 요소값만 추출
```

```
array([5., 8., 7.])
```

```
t_c = t_a < 4
t_c
```

```
array([ True, False, False,  True, False, False])
```

```
t_a[t_c]
```

```
array([3., 0.])
```

fancy index

- numpy의 array를 index value로 사용하여 값을 추출

```
f_a = np.array( [1, 2, 3, 4, 5, 6], float )
f_b = np.array( [1, 0, 2, 0, 1, 4], int )  # 반드시 integer로 선언
f_a[f_b]  # bracket index, b 배열의 값을 index로 하여 a의 값 추출
```

```
array([2., 1., 3., 1., 2., 5.])
```

```
f_a.take(f_b)  # take : bracket index와 같은 효과
```

```
array([2., 1., 3., 1., 2., 5.])
```

0	1	2	3	4	5
1	**2**	**3**	**4**	**5**	**6**

fancy index

- matrix 형태 데이터

```
f_a = np.array( [ [1, 3, 7], [3, 4, 5]  ], float )
f_b = np.array( [1, 0, 1, 0, 1], int )
f_c = np.array( [1, 1, 0, 0, 2], int )
f_a[f_b, f_c]        # [axis = 0, axis = 1 ]
```

array([4., 3., 3., 1., 5.])

	0	1	2
0	1	3	7
1	3	4	5

loadtxt, savetxt

- text type의 데이터를 일고, 저장

```
a = np.loadtxt( "./price.txt“ )
a[:5] # python 기본타입float 형태로 보임
```

```
array([[1.00e+00, 2.30e+04, 3.40e+03, 1.96e+04],
       [2.00e+00, 4.20e+04, 6.90e+03, 3.51e+04],
       [3.00e+00, 6.10e+04, 2.70e+03, 5.83e+04],
       [4.00e+00, 1.90e+04, 8.60e+03, 1.04e+04],
       [5.00e+00, 6.90e+04, 3.70e+03, 6.53e+04]])
```

```
a_int = a.astype(int)
a_int[:3]
```

```
array([[    1, 23000,  3400, 19600],
       [    2, 42000,  6900, 35100],
       [    3, 61000,  2700, 58300]])
```

loadtxt, savetxt

- text type의 데이터를 일고, 저장

```
np.savetxt( "int_price.csv" , a_int, delimiter = "," )
```

jupyter int_price.csv a minute ago

File Edit View Language

```
1.000000000000000000e+00,2.300000000000000000e+04,3.400000000000000000e+03,1.960000000000000000e+04
2.000000000000000000e+00,4.200000000000000000e+04,6.900000000000000000e+03,3.510000000000000000e+04
3.000000000000000000e+00,6.100000000000000000e+04,2.700000000000000000e+03,5.830000000000000000e+04
4.000000000000000000e+00,1.900000000000000000e+04,8.600000000000000000e+03,1.040000000000000000e+04
5.000000000000000000e+00,6.900000000000000000e+04,3.700000000000000000e+03,6.530000000000000000e+04
6.000000000000000000e+00,9.000000000000000000e+04,6.400000000000000000e+03,8.360000000000000000e+04
7.000000000000000000e+00,5.200000000000000000e+04,3.900000000000000000e+03,4.810000000000000000e+04
8.000000000000000000e+00,2.800000000000000000e+04,6.800000000000000000e+03,2.120000000000000000e+04
9.000000000000000000e+00,8.100000000000000000e+04,1.000000000000000000e+04,7.100000000000000000e+04
1.000000000000000000e+01,2.600000000000000000e+04,5.400000000000000000e+03,2.060000000000000000e+04
```

```
np.savetxt( "int_price_2.csv" , a_int, fmt = '%d', delimiter = "," )
```

jupyter int_price_2.csv 2분 전

File Edit View Language

```
1,23000,3400,19600
2,42000,6900,35100
3,61000,2700,58300
4,19000,8600,10400
5,69000,3700,65300
6,90000,6400,83600
7,52000,3900,48100
8,28000,6800,21200
9,81000,10000,71000
10,26000,5400,20600
11,16000,4600,11400
```

Numpy

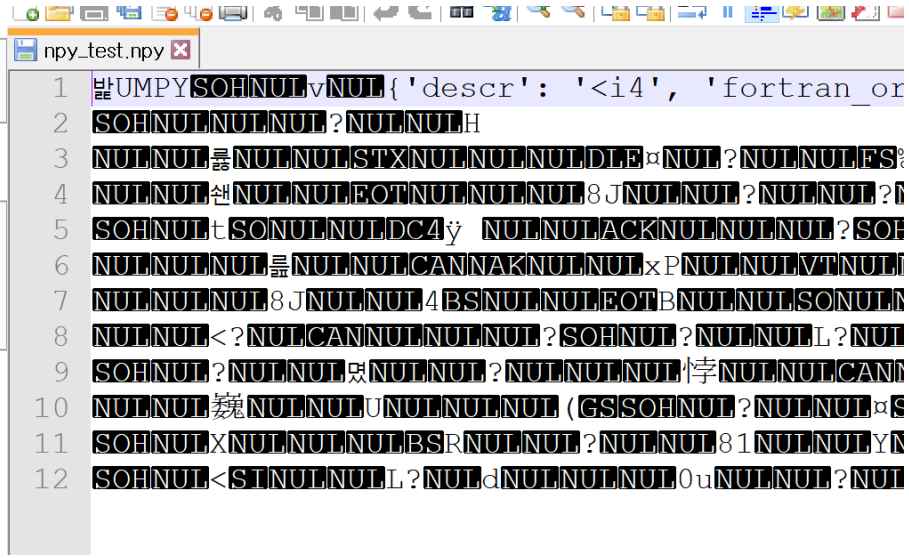
numpy object : npy

- Numpy object (pickle) 형태로 데이터를 저장하고 불러옴
- Binary 형태로 파일 저장

```
np.save( "npy_test.npy " , arr = a_int )
```

```
npy_array = np.load( file = "npy_test.npy")
npy_array[:5]
```

```
array([[   1, 23000,  3400, 19600],
       [   2, 42000,  6900, 35100],
       [   3, 61000,  2700, 58300],
       [   4, 19000,  8600, 10400],
       [   5, 69000,  3700, 65300]])
```



4. 라이브러리
(2) pandas

Pandas - overview

- 구조화된 데이터의 처리를 지원하는 python 라이브러리
- python계의 엑셀
- 고성능 array 계산 라이브러리인 numpy와 통합하여, 강력한 “스프레드시트” 처리 기능을 제공
- 인덱싱, 연산용 함수, 전처리 함수 등을 제공함

Pandas

http://tripkendall.com/2018/01/pandas-for-python/

데이터로딩

```
import pandas as pd
```

```
data_url = 'http://archive.ics.uci.edu/ml/machine-learning-
                databases/housing/housing.data'
df_data = pd.read_csv(data_url, sep = '\s+', header = None)
# csv 타입 데이터 로드, separate는 빈공간으로 저장, Column은 없음
```

```
df_data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

pandas 구성

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

serise

DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

Series

- column vector를 표현하는 object
- index가 포함된 numpy

```
from pandas import Series, DataFrame
import pandas as pd
ex_obj = Series( )
```

shift + tab

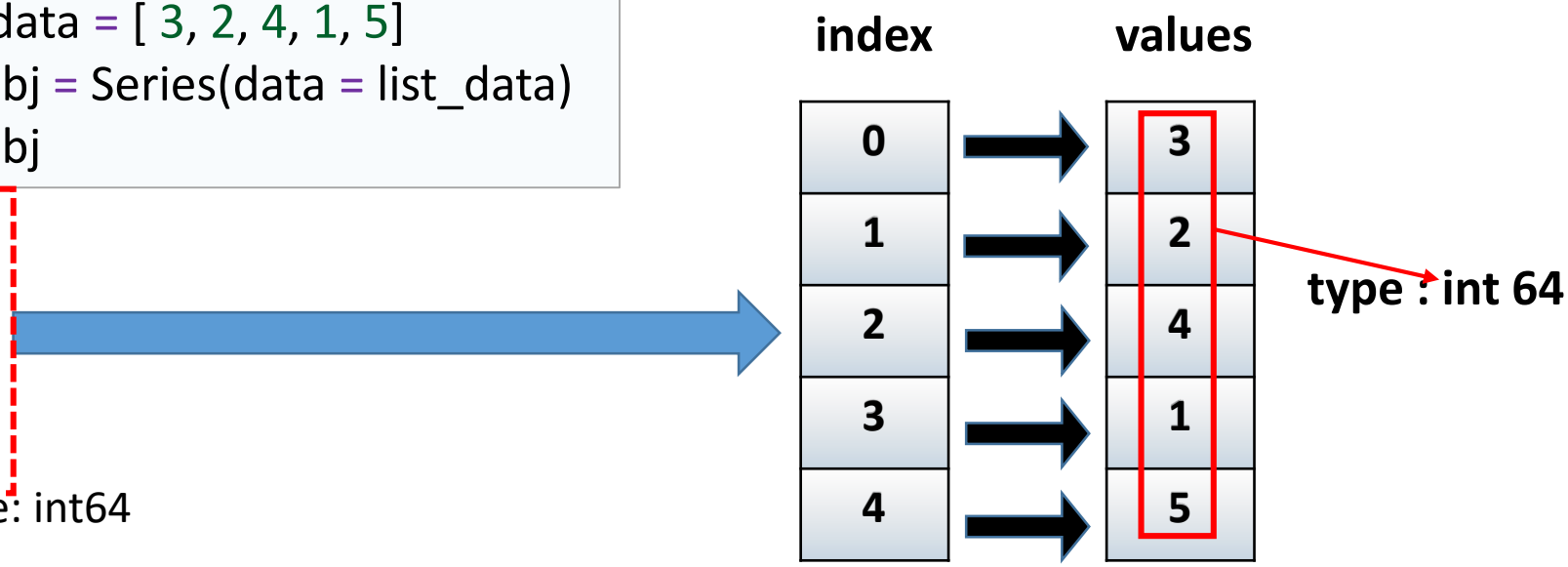
```
Init signature: Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)
Docstring:
One-dimensional ndarray with axis labels (including time series).
```

Series

- column vector를 표현하는 object
- numpy.ndarray 의 subclass
- 어떤 type의 data라도 다룰 수 있다.
- index labels은 필요하나 순서대로 정렬 될 필요는 없다
- duplicates 가능하다 (성능 안좋아짐)

```
list_data = [ 3, 2, 4, 1, 5]
ex_obj = Series(data = list_data)
ex_obj
```

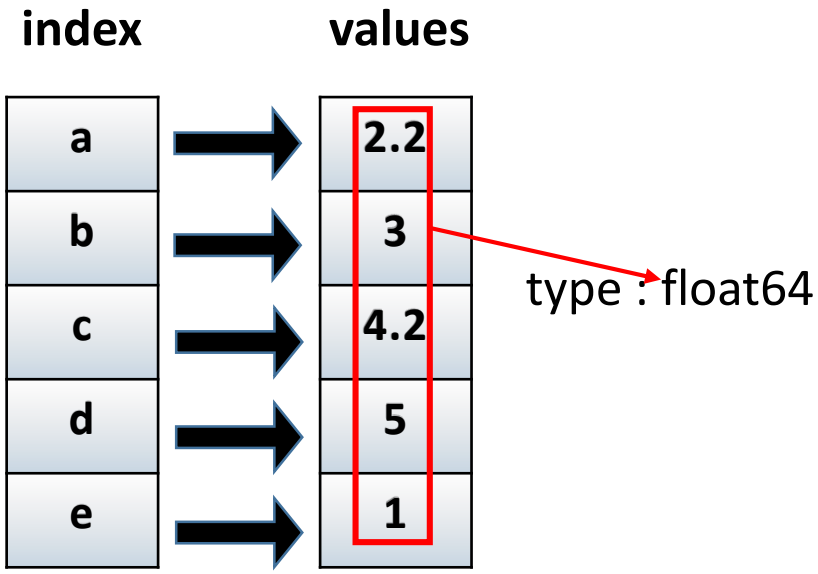
```
0    3
1    2
2    4
3    1
4    5
dtype: int64
```



Series

```
list_data = [ 2.2, 3, 4.2, 5, 1]
list_name = [ "a", "b", "c", "d", "e"]
exp_obj = Series( data = list_data, index = list_name )
exp_obj
```

```
a    2.2
b    3.0
c    4.2
d    5.0
e    1.0
dtype: float64
```

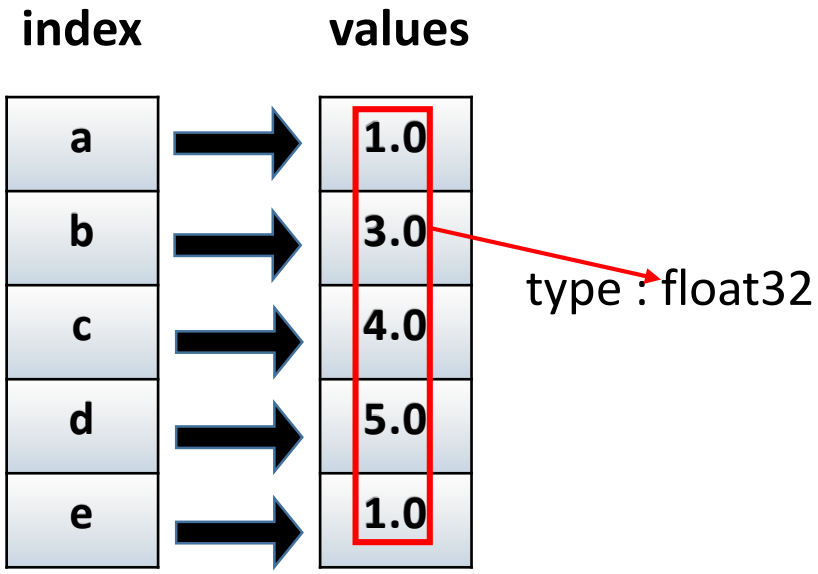


Series

```
dict_data = { "a" : 1, "b" : 3, "c" : 4, "d" : 5, "e" : 1 }
exp_obj = Series( dict_data , dtype = np.float32, name = "dict_series" )
exp_obj
```

data type 설정 **series 이름 설정**

```
a    1.0
b    3.0
c    4.0
d    5.0
e    1.0
Name: dict_series, dtype: float32
```



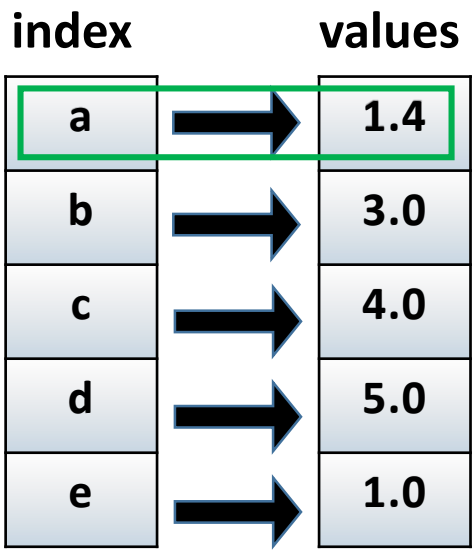
Series

```
exp_obj[ "a" ]  # data의 index에 접근
```

1.0

```
exp_obj[ "a" ] = 1.4  # data index를 통해 값 할당
exp_obj
```

```
a    1.4
b    3.0
c    4.0
d    5.0
e    1.0
Name: dict_series, dtype: float32
```



Series

```
exp_obj.values # 값에 해당하는 리스트 반환. numpy 형태
```

```
array([1.4, 3. , 4. , 5. , 1. ], dtype=float16)
```

```
exp_obj.index # index 리스트 형태로 반환
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
exp_obj.name = "example"
exp_obj.index.name = "index_a"
exp_obj
```

```
index_a
a    1.400391
b    3.000000
c    4.000000
d    5.000000
e    1.000000
Name: example, dtype: float16
```

Series

```
dict_data = { "a" : 1, "b" : 3, "c" : 4, "d" : 5, "e" : 1 }
indexes = [ "a", "b", "c", "D", "e", "f", "g" ]
series_obj_2 = Series(dict_data, index = indexes)
series_obj_2
```

```
a    1.0
b    3.0
c    4.0
D    NaN
e    1.0
f    NaN
g    NaN
dtype: float64
```

index 값 기준으로 series 생성

Series - index가 추가된 numpy, numpy에서 가능한 건 다할 수 있음

```
exp_obj[exp_obj > 1]
```

```
index_a
a    1.400391
b    3.000000
c    4.000000
d    5.000000
Name: example, dtype: float16
```

```
exp_obj * 3
```

```
index_a
a     4.203125
b     9.000000
c    12.000000
d    15.000000
e     3.000000
Name: example, dtype: float16
```

```
np.exp(exp_obj)
```

```
index_a
a      4.058594
b     20.078125
c     54.593750
d    148.375000
e      2.718750
Name: example, dtype: float16
```

```
"x" in ex_obj
```

```
False
```

```
exp_obj.to_dict()
```

```
{'a': 1.400390625, 'b': 3.0, 'c': 4.0,
'd': 5.0, 'e': 1.0}
```

DataFrame

- series를 모아서 만든 data table : 기본 2차원

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222.0	18.7	396.90	5.33	36.2

serise

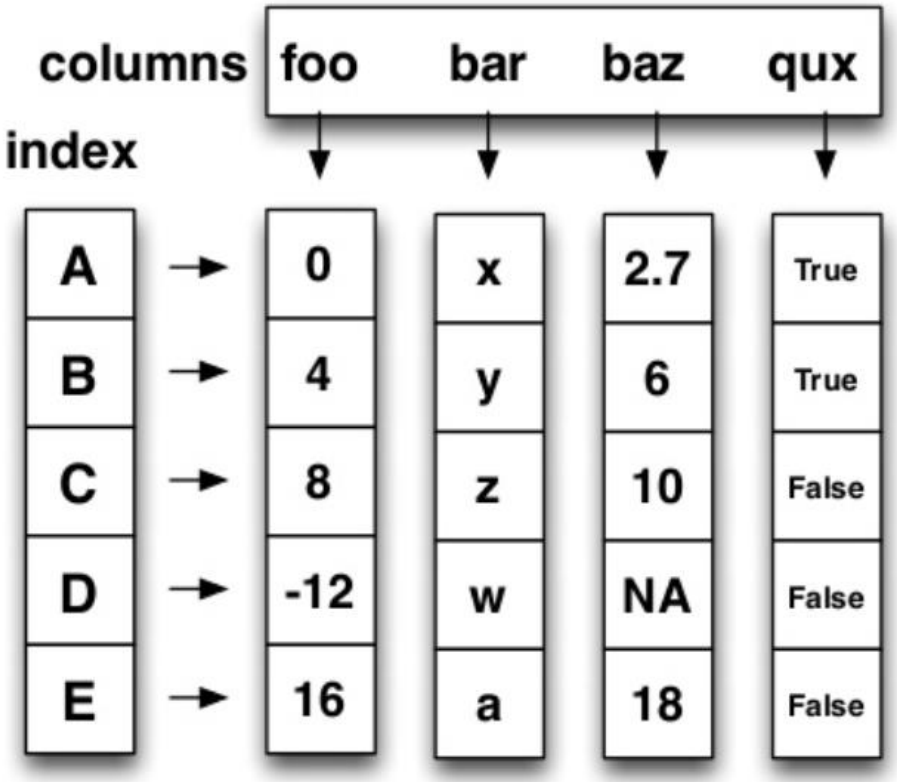
DataFrame 중 하나의 Column에 해당하는 데이터의 모음 Object

DataFrame

Data Table 전체를 포함하는 Object

DataFrame

- 여러 개의 series의 합
- 각각의 컬럼은 다른 type으로 구성될 수 있다.
- row와 column index가 존재
- 컬럼 삽입이나 삭제를 통해 사이즈를 유연하게 조절 할 수 있음



https://www.slideshare.net/wesm/pandas-powerful-data-analysis-tools-for-python

DataFrame

```
r_data = {
   "city" : [
      "Seoul", "Seoul", "Jeonju", "Busan", "Incheon", "Goyang"
   ],
   "name" : [
      "이정수", "문철수", "오영희", "채호야", "문동석", "고나리"
   ],
   "job" : [
      '학생', '정치인', '경찰', '소방관', '영업', 'it'
   ],
   "age" : [
      22, 53, 26, 33, 43, 34
   ]
}
```

DataFrame

```
df = pd.DataFrame( data, columns = [  'name', 'job', 'city', 'age' ] )
df
```

	name	job	city	age
0	이정수	학생	Seoul	22
1	문철수	정치인	Seoul	53
2	오영희	경찰	Jeonju	26
3	채호야	소방관	Busan	33
4	문동석	영업	Incheon	43
5	고나리	it	Goyang	34

DataFrame

```
DataFrame(r_data, columns = [ "age", "city" ] )
```

column 선택

	age	city
0	22	Seoul
1	53	Seoul
2	26	Jeonju
3	33	Busan
4	43	Incheon
5	34	Goyang

```
DataFrame(r_data, columns = [ 'name', 'job', ‘comp‘ ])
```

새로운 column 추가

	name	job	comp
0	이정수	학생	NaN
1	문철수	정치인	NaN
2	오영희	경찰	NaN
3	채호야	소방관	NaN
4	문동석	영업	NaN
5	고나리	it	NaN

DataFrame

```
df = DataFrame(r_data, columns = [ 'name', 'job', 'city', 'age', 'comp' ] )
df.name  # 속성이름으로 추출
```

```
0    이정수
1    문철수
2    오영희
3    채호야
4    문동석
5    고나리
Name: name, dtype: object
```

column 선택 하는건
DataFrame 에서 series data 추출 하는 것

```
df["name"]  # dict type처럼 변수명과 컬럼명으로 추출
```

```
0    이정수
1    문철수
2    오영희
3    채호야
4    문동석
5    고나리
Name: name, dtype: object
```

DataFrame

```
df = DataFrame(r_data, columns = [ 'name', 'job', 'city', 'age', 'comp' ] )
df
```

	name	job	city	age	comp
0	이정수	학생	Seoul	22	NaN
1	문철수	정치인	Seoul	53	NaN
2	오영희	경찰	Jeonju	26	NaN
3	채호야	소방관	Busan	33	NaN
4	문동석	영업	Incheon	43	NaN
5	고나리	it	Goyang	34	NaN

```
df.loc[1] # index location 기준
```

```
name      문철수
job       정치인
city      Seoul
age       53
house     NaN
Name: 1, dtype: object
```

```
df[ "name" ].iloc[1]  # index position 기준
```

'문철수'

DataFrame
- loc, iloc

```
ex_ind = pd.Series(np.nan, index = [ 12, 13, 14, 3, 2, 1 ] )
ex_ind
```

```
12   NaN
13   NaN
14   NaN
3    NaN
2    NaN
1    NaN
dtype: float64
```

```
ex_ind.loc[:2] # index location 기준, 인덱스 name
```

```
12   NaN
13   NaN
14   NaN
3    NaN
2    NaN
dtype: float64
```

```
ex_ind.iloc[:2] # index position 기준
```

```
12   NaN
13   NaN
dtype: float64
```

DataFrame
- column에 새로운 데이터 할당

```
df.comp = df.age > 30
df
```

	name	job	city	age	comp
0	이정수	학생	Seoul	22	False
1	문철수	정치인	Seoul	53	True
2	오영희	경찰	Jeonju	26	False
3	채호야	소방관	Busan	33	True
4	문동석	영업	Incheon	43	True
5	고나리	it	Goyang	34	True

Boolean index 형태

DataFrame
- column에 새로운 데이터 할당

```
df.T  # transpose, numpy 기능 똑같이 쓸 수 있음
```

	0	1	2	3	4	5
name	이정수	문철수	오영희	채호야	문동석	고나리
job	학생	정치인	경찰	소방관	영업	it
city	Seoul	Seoul	Jeonju	Busan	Incheon	Goyang
age	22	53	26	33	43	34
comp	False	True	False	True	True	True

DataFrame
- column에 새로운 데이터 할당

```
df.values # 값 출력
```

```
array([['이정수', '학생', 'Seoul', 22, False],
       ['문철수', '정치인', 'Seoul', 53, True],
       ['오영희', '경찰', 'Jeonju', 26, False],
       ['채호야', '소방관', 'Busan', 33, True],
       ['문동석', '영업', 'Incheon', 43, True],
       ['고나리', 'it', 'Goyang', 34, True]], dtype=object)
```

```
df.to_csv() # csv 형태 출력
```

```
',name,job,city,age,comp\n0,이정수,학
생,Seoul,22,False\n1,문철수,정치인,Seoul,53,True\n2,
오영희,경찰,Jeonju,26,False\n3,채호야,소방
관,Busan,33,True\n4,문동석,영업,Incheon,43,True\n5,
고나리,it,Goyang,34,True\n'
```

DataFrame
- column 삭제

```
del df["comp"]
df
```

	name	job	city	age
0	이정수	학생	Seoul	22
1	문철수	정치인	Seoul	53
2	오영희	경찰	Jeonju	26
3	채호야	소방관	Busan	33
4	문동석	영업	Incheon	43
5	고나리	it	Goyang	34

DataFrame

- dict 안에 dict

```
ex_dict = {'seoul' : {1 : 33, 2 : 22 }, 'busan' : {3 : 35, 4 : 37 } }
ex_dict
```

{'seoul': {1: 33, 2: 22}, 'busan': {3: 35, 4: 37}}

```
DataFrame(ex_dict)
```

	seoul	busan
1	33.0	NaN
2	22.0	NaN
3	NaN	35.0
4	NaN	37.0

column names와 index number를 이용한 data selection

- 엑셀 데이터이용, xlrd 모듈 설치

```
!conda install --y xlrd  # promp상의 conda 명령어 jupyter에서 사용
```

```
df = pd.read_excel("./excel_sample/sampledata.xlsx")
df.head()
```

	OrderDate	Region	Rep	Item	Units	Unit Cost	Total
0	2016-01-06	East	Jones	Pencil	95	1.99	189.05
1	2016-01-23	Central	Kivell	Binder	50	19.99	999.50
2	2016-02-09	Central	Jardine	Pencil	36	4.99	179.64
3	2016-02-26	Central	Gill	Pen	27	19.99	539.73
4	2016-03-15	West	Sorvino	Pencil	56	2.99	167.44

column names와 index number를 이용한 data selection

```
df["Region"].head(3) # 한개 column 선택
```

```
0     East
1  Central
2  Central
Name: Region, dtype: object
```

```
df[ ["Region", "Item", "Unit Cost"] ].head(3)  #1개 이상의 column 선택
```

	Region	Item	Unit Cost
0	East	Pencil	1.99
1	Central	Binder	19.99
2	Central	Pencil	4.99

column names와 index number를 이용한 data selection

```
df[ :3 ] # column 이름 없이 사용하는 index number는 row 기준 표시
```

	OrderDate	Region	Rep	Item	Units	Unit Cost	Total
0	2016-01-06	East	Jones	Pencil	95	1.99	189.05
1	2016-01-23	Central	Kivell	Binder	50	19.99	999.50
2	2016-02-09	Central	Jardine	Pencil	36	4.99	179.64

```
df[ "Region" ][ : 3 ]
# column 이름과 함께 row index 사용시, 해당 column 만 리턴
```

```
0      East
1   Central
2   Central
Name: Region, dtype: object
```

Pandas – selection & drop

Series selection

```
tot_series = df["Total"]
tot_series[:4]
```

```
0    189.05
1    999.50
2    179.64
3    539.73
Name: Total, dtype: float64
```

```
tot_series[[0, 3]]
```

```
0    189.05
3    539.73
Name: Total, dtype: float64
```

```
tot_series[tot_series > 1000]
```

```
12    1619.19
21    1183.26
23    1305.00
34    1005.90
38    1139.43
41    1879.06
Name: Total, dtype: float64
```

Boolean index 형태

```
tot_series > 1000
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12     True
13    False
```

Index 변경

```
df.index = df["Units"]
df.head(3)
```

	OrderDate	Region	Rep	Item	Units	Unit Cost	Total
Units							
95	2016-01-06	East	Jones	Pencil	95	1.99	189.05
50	2016-01-23	Central	Kivell	Binder	50	19.99	999.50
36	2016-02-09	Central	Jardine	Pencil	36	4.99	179.64

```
del df['Units']
df.head(3)
```

	OrderDate	Region	Rep	Item	Unit Cost	Total
Units						
95	2016-01-06	East	Jones	Pencil	1.99	189.05
50	2016-01-23	Central	Kivell	Binder	19.99	999.50
36	2016-02-09	Central	Jardine	Pencil	4.99	179.64

basic, loc, iloc selection

```
df[["OrderDate", "Region"]][:2] # column name과 index number 기준
```

	OrderDate	Region
Units		
95	2016-01-06	East
50	2016-01-23	Central

```
df.loc[[95, 36], ["OrderDate", "Region"]]  # column 과 index name 기준
```

	OrderDate	Region
Units		
95	2016-01-06	East
36	2016-02-09	Central

```
df.iloc[:2, :2]  # column 과 index number 기준
```

	OrderDate	Region
Units		
95	2016-01-06	East
50	2016-01-23	Central

index 재설정

```
df.index = list(range(0,43))
df.head()
```

	OrderDate	Region	Rep	Item	Unit Cost	Total
0	2016-01-06	East	Jones	Pencil	1.99	189.05
1	2016-01-23	Central	Kivell	Binder	19.99	999.50
2	2016-02-09	Central	Jardine	Pencil	4.99	179.64
3	2016-02-26	Central	Gill	Pen	19.99	539.73
4	2016-03-15	West	Sorvino	Pencil	2.99	167.44

data drop

```
df.drop(1).head(3)  # index name로 drop
```

	OrderDate	Region	Rep	Item	Unit Cost	Total
0	2016-01-06	East	Jones	Pencil	1.99	189.05
2	2016-02-09	Central	Jardine	Pencil	4.99	179.64
3	2016-02-26	Central	Gill	Pen	19.99	539.73

```
df.drop([0,1,2,3]).head(3) # 한개 이상의 index name로 drop
```

	OrderDate	Region	Rep	Item	Unit Cost	Total
4	2016-03-15	West	Sorvino	Pencil	2.99	167.44
5	2016-04-01	East	Jones	Binder	4.99	299.40
6	2016-04-18	Central	Andrews	Pencil	1.99	149.25

data drop

- axis 지정한 축을 기준으로 drop

```
df.drop("Rep", axis = 1).head(3)  # df.drop(["Rep", "Item"], axis = 1)
```

	OrderDate	Region	Item	Unit Cost	Total
0	2016-01-06	East	Pencil	1.99	189.05
1	2016-01-23	Central	Binder	19.99	999.50
2	2016-02-09	Central	Pencil	4.99	179.64

- **실제 drop** 되는 것은 아니고, **복사본을 만들**어서 보여주는 것
- drop된 dataframe을 생성하고 싶으면 **다시 할당** 해야함
- **즉시** 반영하고 싶으면 **inplace = True** 사용

```
df.head(3)
```

	OrderDate	Region	Rep	Item	Unit Cost	Total
0	2016-01-06	East	Jones	Pencil	1.99	189.05
1	2016-01-23	Central	Kivell	Binder	19.99	999.50
2	2016-02-09	Central	Jardine	Pencil	4.99	179.64

data drop

```
df_drop = df.drop("Rep", axis = 1)  # drop 반영한 새로운 dataframe 생성
df_drop.head(3)
```

	OrderDate	Region	Item	Unit Cost	Total
0	2016-01-06	East	Pencil	1.99	189.05
1	2016-01-23	Central	Binder	19.99	999.50
2	2016-02-09	Central	Pencil	4.99	179.64

```
df_drop.drop("Item", axis = 1, inplace = True)  # 현재 dataframe 에 바로 반영
df_drop.head(3)
```

	OrderDate	Region	Unit Cost	Total
0	2016-01-06	East	1.99	189.05
1	2016-01-23	Central	19.99	999.50
2	2016-02-09	Central	4.99	179.64

Series operation

- seires는 numpy에 index 추가한것
- index 기준으로 연산 수행
- 겹치는 index가 없을 경우 nan값 반환
- numpy는 사이즈가 맞지 않으면 error

```
s1 = Series( range(1, 6), index = list("abcde"))
s1
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

```
s2 = Series( range(6,11), index = list("cdefg"))
s2
```

```
c    6
d    7
e    8
f    9
g   10
dtype: int64
```

Series operation

```
s1.add(s2)
```

```
a    NaN
b    NaN
c    9.0
d   11.0
e   13.0
f    NaN
g    NaN
dtype: float64
```

```
s1 + s2
```

```
a    NaN
b    NaN
c    9.0
d   11.0
e   13.0
f    NaN
g    NaN
dtype: float64
```

```
s1.add(s2, fill_value = 0)
```

```
a    1.0
b    2.0
c    9.0
d   11.0
e   13.0
f    9.0
g   10.0
dtype: float64
```

DataFrame operation

- DataFrame은 column과 index 모두 고려
- add operation을 쓰면 NaN값 변환시킬 수 있음
- Operation types : add, sub, div, mul

```
df1 = DataFrame(np.arange(4).reshape(2, 2), columns = list("ab"))
df1
```

	a	b
0	0	1
1	2	3

```
df2 = DataFrame(np.arange(9).reshape(3, 3), columns = list("abc"))
df2
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

DataFrame operation

```
df1 + df2
```

	a	b	c
0	0.0	2.0	NaN
1	5.0	7.0	NaN
2	NaN	NaN	NaN

```
df1.add(df2, fill_value = 0)
```

	a	b	c
0	0.0	2.0	2.0
1	5.0	7.0	5.0
2	6.0	7.0	8.0

Series + DataFrame

```
df = DataFrame(np.arange(9).reshape(3, 3),
       columns = list("abc"))
df
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

+

a	b	c
10	**11**	**12**
10	11	12
10	11	12

```
s = Series(np.arange(10,13),
      index = list("abc"))
s
```

```
a    10
b    11
c    12
dtype: int32
```

```
df + s
```

	a	b	c
0	10	12	14
1	13	15	17
2	16	18	20

column 기준으로 broadcasting 발생

Series + DataFrame

```
df = DataFrame(np.arange(9).reshape(3, 3),
       columns = list("abc"))
df
```

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

0	1	2
10	**11**	**12**
10	11	12
10	11	12

```
s2 = Series(np.arange(10, 13))
s2
```

```
0    10
1    11
2    12
dtype: int32
```

column 기준으로 broadcasting 발생한다. 서로 같은 index가 존재하지 않으므로 NaN이 리턴

```
df + s2
```

	a	b	c	0	1	2
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN

Series + DataFrame

```
df.add(s2, axis = 0)
```

	a	b	c
0	10	11	12
1	14	15	16
2	18	19	20

axis 기준으로
row broadcasting 실행

	a	b	c
0	0	1	2
1	3	4	5
2	6	7	8

+

0	**10**	10	10
1	**11**	11	11
2	**12**	12	12

Lambda

- 한 줄로 함수를 표현하는 익명 함수 기법
- Lisp 언어에서 시작된 기법으로 오늘날 현대언어에 많이 사용

lambda argument : expression

```
def f(x):
    return x + 5

f(3)
```

8

```
f = lambda x : x + 5
f(3)
```

8

Pandas – lambda, map, apply

Lambda

- 한 줄로 함수를 표현하는 익명 함수 기법
- Lisp 언어에서 시작된 기법으로 오늘날 현대언어에 많이 사용

```
f = lambda x: x / 3  # 하나의 인수만 처리
f(6)
```

2.0

```
f = lambda x: x ** 2  # 하나의 인수만 처리
f(4)
```

16

```
(lambda x: x + 4)(2)  # 이름 할당하지 않는 lambda
```

6

map 함수

- 함수와 sequence형 데이터를 인자로 받아 각 요소마다 입력받은 함수를 적용하여 list로 반환
- 일반적으로 함수를 lambda 형태로 표현

map(function, sequence)

```
seq = [1, 2, 3, 4, 5]
f = lambda x: x * 3
list(map(f, seq))
```

[3, 6, 9, 12, 15]

```
seq = [3, 2, 6, 4, 5]
f = lambda x: x - 3
list(map(f, seq))
```

[0, -1, 3, 1, 2]

map 함수

- 함수와 sequence형 데이터를 인자로 받아 각 요소마다 입력받은 함수를 적용하여 list로 반환
- 일반적으로 함수를 lambda 형태로 표현

map(function, sequence)

```
f = lambda x, y: x + y
list(map(f, seq, seq))  # 두개이상의 인수가 있을 때는
                        # 두개의 sequence 형을 써야함
```

[2, 4, 6, 8, 10]

```
list(map(lambda x: x * x, seq))  # 익명함수 그대로 사용
# python3 에서 list 함수를 쓰지않으면 memory 주소가 나옴
```

[1, 4, 9, 16, 25]

map for series

- Pandas의 series type의 데이터에도 map 함수 사용가능
- function 대신 dict, sequence형 자료등으로 대체 가능

```
s = Series(np.arange(10))
s.head(5)
```

```
0    0
1    1
2    2
3    3
4    4
dtype: int32
```

```
s.map(lambda x: x + 3).head(5)
```

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

map for series

```
dic = {1: 'apple', 2: 'banana', 3: 'carrot'}
s.map(dic).head(5)  # dict type으로 데이터 교체, 값이 없으면 NaN
```

```
0      NaN
1    apple
2   banana
3   carrot
4      NaN
dtype: object
```

```
s2 = Series(np.arange(100, 500, 100))
s.map(s2).head(5)  # 같은 위치의 데이터를 s2로 전환
```

```
0    100.0
1    200.0
2    300.0
3    400.0
4      NaN
dtype: float64
```

map for series – ex)

```
df = pd.read_csv("./excel_sample/map_1.csv")
df.head()
```

	city	age	sex	job	marriage	height
0	seoul	23	male	student	n	67.43
1	busan	42	male	salesman	y	76.21
2	jeonju	52	female	ceo	y	54.33
3	incheon	13	male	student	n	35.22
4	daejeon	43	female	professor	n	48.99

map for series – ex)

```
df.sex.unique()
```

array(['male', 'female'], dtype = object)

```
df["sex_code"] = df.sex.map({"male": 0, "female": 1})
# 성별 string type -> int type
df.head()
```

	city	age	sex	job	marriage	height	sex_code
0	seoul	23	male	student	n	67.43	0
1	busan	42	male	salesman	y	76.21	0
2	jeonju	52	female	ceo	y	54.33	1
3	incheon	13	male	student	n	35.22	0
4	daejeon	43	female	professor	n	48.99	1

```
df.sex_code.unique()
```

array([0, 1], dtype=int64)

replace function

- Map함수의 기능 중 데이터 변환 기능만 담당
- 데이터 변환시 많이 사용하는 함수

```
df.marriage.replace({"y": 1, "n": 0}).head()
```

```
0    0
1    1
2    1
3    0
4    0
Name: marriage, dtype: int64
```

```
df.marriage.replace(["y", "n"], [1, 0]).head()
```

```
0    0
1    1
2    1
3    0
4    0
Name: marriage, dtype: int64
```

replace function

```
df.head()
```

	city	age	sex	job	marriage	height	sex_code
0	seoul	23	male	student	n	67.43	0
1	busan	42	male	salesman	y	76.21	0
2	jeonju	52	female	ceo	y	54.33	1
3	incheon	13	male	student	n	35.22	0
4	daejeon	43	female	professor	n	48.99	1

```
df.marriage.replace({"y": 1, "n": 0}, inplace = True)
df.head()
```

	city	age	sex	job	marriage	height	sex_code
0	seoul	23	male	student	0	67.43	0
1	busan	42	male	salesman	1	76.21	0
2	jeonju	52	female	ceo	1	54.33	1
3	incheon	13	male	student	0	35.22	0
4	daejeon	43	female	professor	0	48.99	1

apply for dataframe

- map과 달리, series 전체(column)에 해당 함수를 적용
- series 데이터로 입력받아 handling 가능

```
df_2 = df[["age", "height"]]
df_2.head()
```

	age	height
0	23	67.43
1	42	76.21
2	52	54.33
3	13	35.22
4	43	48.99

apply for dataframe

```
f = lambda x: x.max() - x.min()
df_2.apply(f)  # 각 column 별로 결과값 반환
```

```
age       39.00
height    60.44
dtype: float64
```

- **내장 연산 함수를 사용할 때도 똑같은 효과를 거둘 수 있음**
- **mean, std 등 사용가능**

```
df_2.sum()
```

```
age       299.00
height    531.54
dtype: float64
```

```
df_2.apply(sum)
```

```
age       299.00
height    531.54
dtype: float64
```

apply for dataframe

- scalar 값 이외에 series 값의 반환도 가능함

```
def f(x):
    return Series([x.min(), x.max(), x.mean()], index = ["min", "max", "mean"])
df_2.apply(f)
```

	age	height
min	13.000000	35.22
max	52.000000	95.66
mean	33.222222	59.06

applymap for dataframe

- series 단위가 아닌 element(요소)단위로 함수 적용
- series 단위에 apply를 적용시킬 때와 같은 효과

```
f = lambda x: -x
df_2.applymap(f).head(5)
```

	age	height
0	-23	-67.43
1	-42	-76.21
2	-52	-54.33
3	-13	-35.22
4	-43	-48.99

```
f = lambda x: -x
df_2["height"].apply(f).head(5)
```

```
0   -67.43
1   -76.21
2   -54.33
3   -35.22
4   -48.99
Name: height, dtype: float64
```

Pandas 내장함수(Built-in functions)

- Numeric type 데이터의 요약 정보를 보여줌

```
df = pd.read_csv("./excel_sample/map_1.csv")
df.head()
```

	city	age	sex	job	marriage	height
0	seoul	23	male	student	n	67.43
1	busan	42	male	salesman	y	76.21
2	jeonju	52	female	ceo	y	54.33
3	incheon	13	male	student	n	35.22
4	daejeon	43	female	professor	n	48.99

Pandas 내장함수(Built-in functions)

- Numeric type 데이터의 요약 정보를 보여줌

```
df.describe()
```

	age	height
count	9.000000	9.000000
mean	33.222222	59.060000
std	12.978615	18.133007
min	13.000000	35.220000
25%	24.000000	48.990000
50%	33.000000	54.330000
75%	43.000000	67.430000
max	52.000000	95.660000

unique

- series data의 유일한 값을 list로 리턴

```
df.job.unique() # 유일한 직업의 값 list
```

```
array(['student', 'salesman', 'ceo', 'professor', 'clerk', 'researcher',
       'police', 'artist'], dtype=object)
```

```
np.array(dict(enumerate(df["job"].unique()))) # dict type으로 index
```

```
array({0: 'student', 1: 'salesman', 2: 'ceo', 3: 'professor', 4: 'clerk',
       5: 'researcher', 6: 'police', 7: 'artist'},    dtype=object)
```

unique

- series data의 유일한 값을 list로 리턴

```
key = np.array(list(enumerate(df["job"].unique())), dtype=str)[:, 1].tolist()
key  # label index 값 추출
```

```
['student',
 'salesman',
 'ceo',
 'professor',
 'clerk',
 'researcher',
 'police',
 'artist']
```

```
value =list(map(int, np.array(list(enumerate(df["job"].unique())))[:, 0].tolist()))
value  # label 값 추출
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

unique

- series data의 유일한 값을 list로 리턴

```
df["job"].replace(to_replace = key, value = value, inplace = True)
df.head()  # label str → index 값으로 변환
```

	city	age	sex	job	marriage	height
0	seoul	23	male	0	n	67.43
1	busan	42	male	1	y	76.21
2	jeonju	52	female	2	y	54.33
3	incheon	13	male	0	n	35.22
4	daejeon	43	female	3	n	48.99

sum

- 기본적인 column 또는 row 값의 연산을 지원
- sub, mean, min, max, count, median, mad, var등

```
df.sum(axis = 0)  # column 합산
```

```
city          36.00
age          299.00
sex            5.00
job           28.00
marriage       4.00
height       531.54
dtype: float64
```

```
df.sum(axis = 1).head()  # row 합산
```

```
0    90.43
1   121.21
2   112.33
3    51.22
4    99.99
dtype: float64
```

isnull

- column 또는 row 값의 NaN(null)값의 index 반환

```
df.isnull().head(3)
```

	city	age	sex	job	marriage	height
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False

```
df.isnull().sum().head(3)
```

```
city    0
age     0
sex     0
dtype: int64
```

sort_values

- column 값을 기준으로 데이터를 sorting

```
df.sort_values(["age", "height"], ascending = True).head(5)
# ascending = True는 오름차순 정렬
```

	city	age	sex	job	marriage	height
3	3	13	0	0	0	35.22
0	0	23	0	0	0	67.43
8	8	24	1	7	0	46.15
5	5	24	1	4	1	51.32
6	6	33	0	5	0	95.66

cumsum

- column 값을 기준으로 데이터를 sorting

```
df.cumsum().head(5) # 누적합
```

	city	age	sex	job	marriage	height
0	0.0	23.0	0.0	0.0	0.0	67.43
1	1.0	65.0	0.0	1.0	1.0	143.64
2	3.0	117.0	1.0	3.0	2.0	197.97
3	6.0	130.0	1.0	3.0	2.0	233.19
4	10.0	173.0	2.0	6.0	2.0	282.18

sort_values

- column 갑을 기준으로 데이터를 sorting

```
df.cummax().head(5) # 내려가면서 가장 큰값만 보여줌
```

	city	age	sex	job	marriage	height
0	0.0	23.0	0.0	0.0	0.0	67.43
1	1.0	42.0	0.0	1.0	1.0	76.21
2	2.0	52.0	1.0	2.0	1.0	76.21
3	3.0	52.0	1.0	2.0	1.0	76.21
4	4.0	52.0	1.0	3.0	1.0	76.21

Correlation & Covariance

- 상관계수와 공분산을 구하는 함수
- corr, cov, corrwith

```
df.age.corr(df.height)  # df의 col1, col2 상관관계 계산
```

0.27022986518660663

```
df.age.cov(df.height)  # 변수간 공분산 계산
```

63.596249999999976

```
df.corrwith(df.height)  #  다른시리즈나 데이터프레인과 상관관계 계산
                        #  하나의 변수와 나머지 변수 간의 상관관계를 볼 때 좋음
```

Correlation & Covariance

- 상관계수와 공분산을 구하는 함수
- corr, cov, corrwith

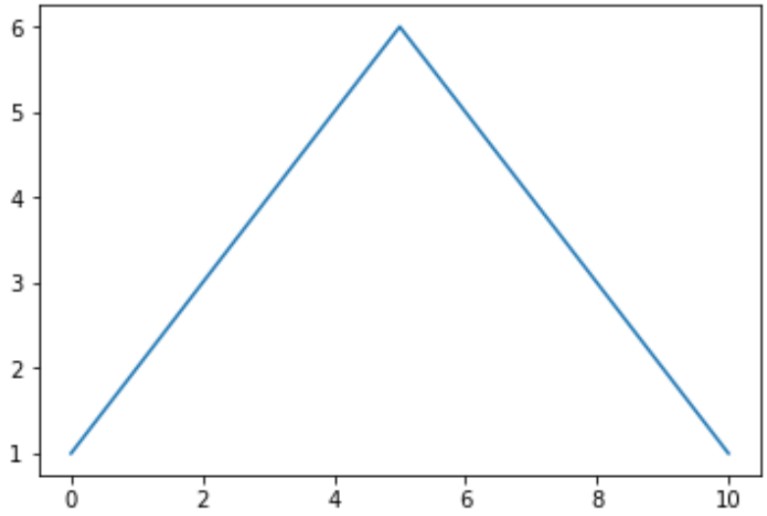
```
df.corr() # 전체 상관관계
```

	city	age	sex	job	marriage	height
city	1.000000	-0.049236	0.519615	0.940845	-0.086603	-0.116544
age	-0.049236	1.000000	0.399997	0.163983	0.550250	0.270230
sex	0.519615	0.399997	1.000000	0.594442	0.350000	-0.500684
job	0.940845	0.163983	0.594442	1.000000	0.051245	0.059439
marriage	-0.086603	0.550250	0.350000	0.051245	1.000000	0.024197
height	-0.116544	0.270230	-0.500684	0.059439	0.024197	1.000000

4. 라이브러리
(2) matplotlib

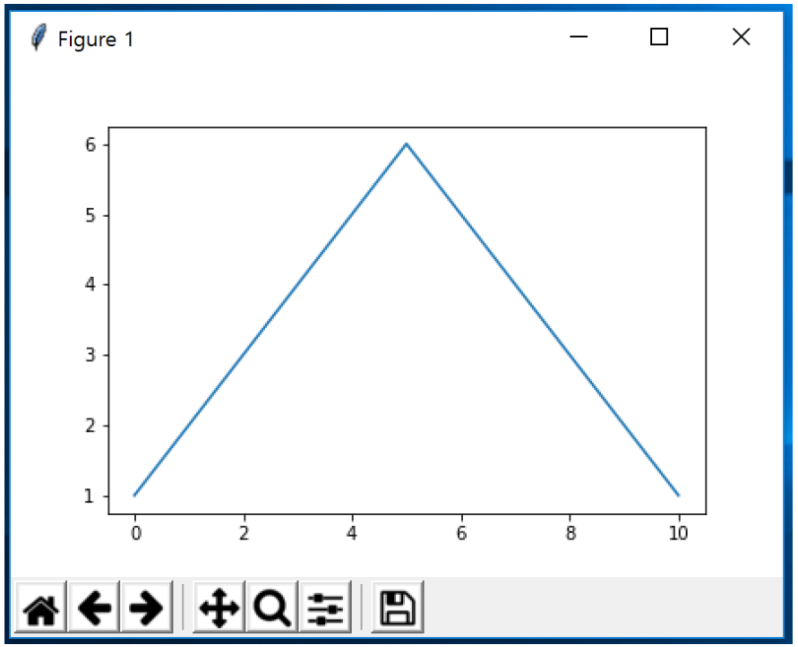
기초다지기

```
import matplotlib.pyplot as plt
%matplotlib inline  # jupyter notebook상에서 그래프 출력

plt.figure()  # 객체 선언
plt.plot([1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1])
plt.show()  # 보여주기
```



기초다지기

```
%matplotlib tk  # 주피터노트북 외부에서 출력

plt.figure()  # 객체 선언
plt.plot([1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1])
plt.show()  # 보여주기
```



Numpy 활용

```
import numpy as np
x = np.arange(0, 12, 0.01)
plt.figure
plt.plot(x)
plt.show()
```

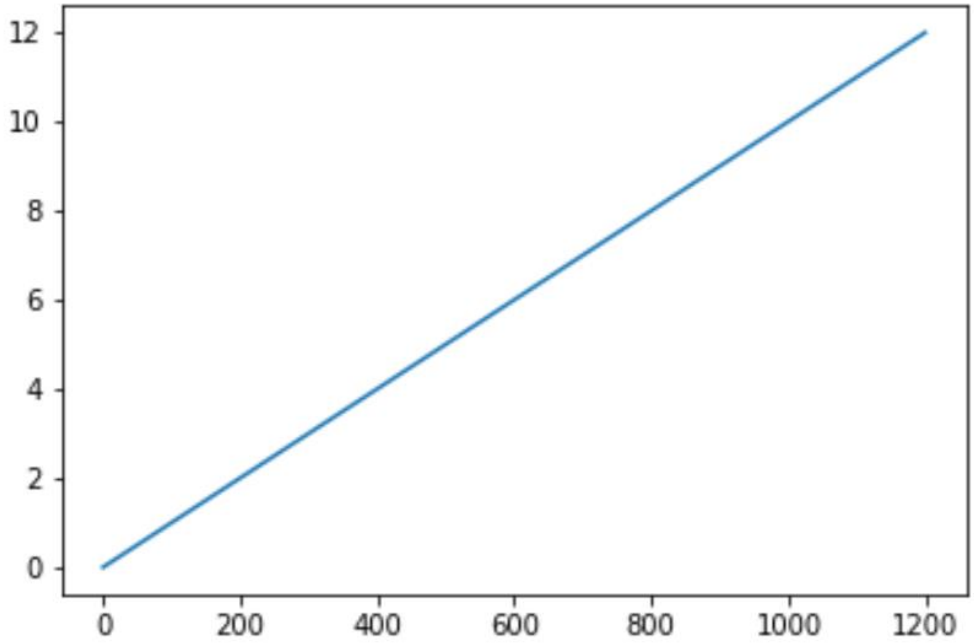
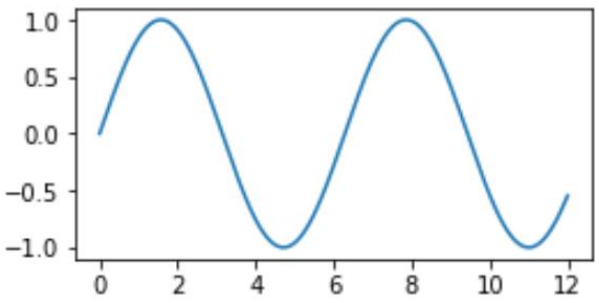
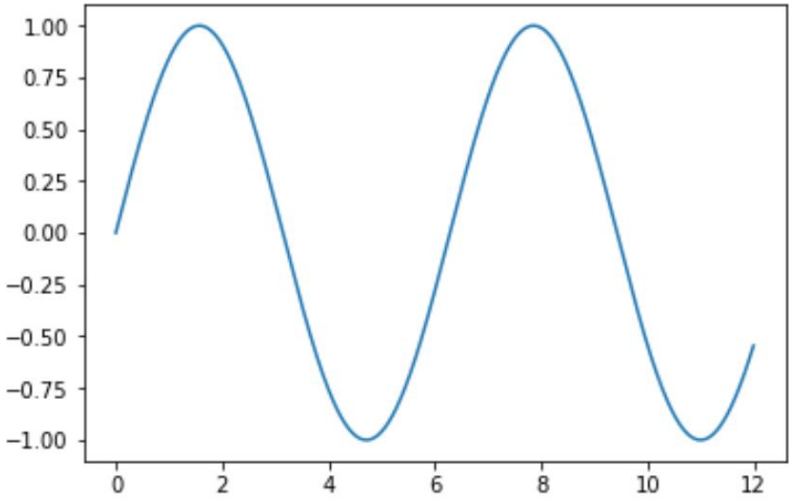


figure 크기조정

```
plt.figure(figsize = (4, 2))
plt.plot(x, y)
plt.show()
```

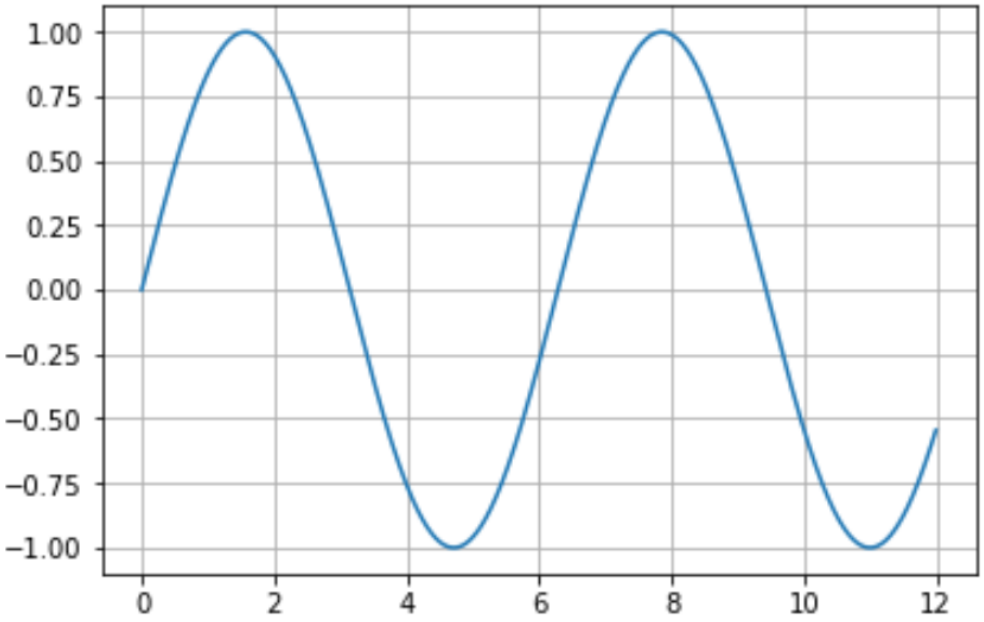



```
plt.figure(figsize = (6, 4))
plt.plot(x, y)
plt.show()
```



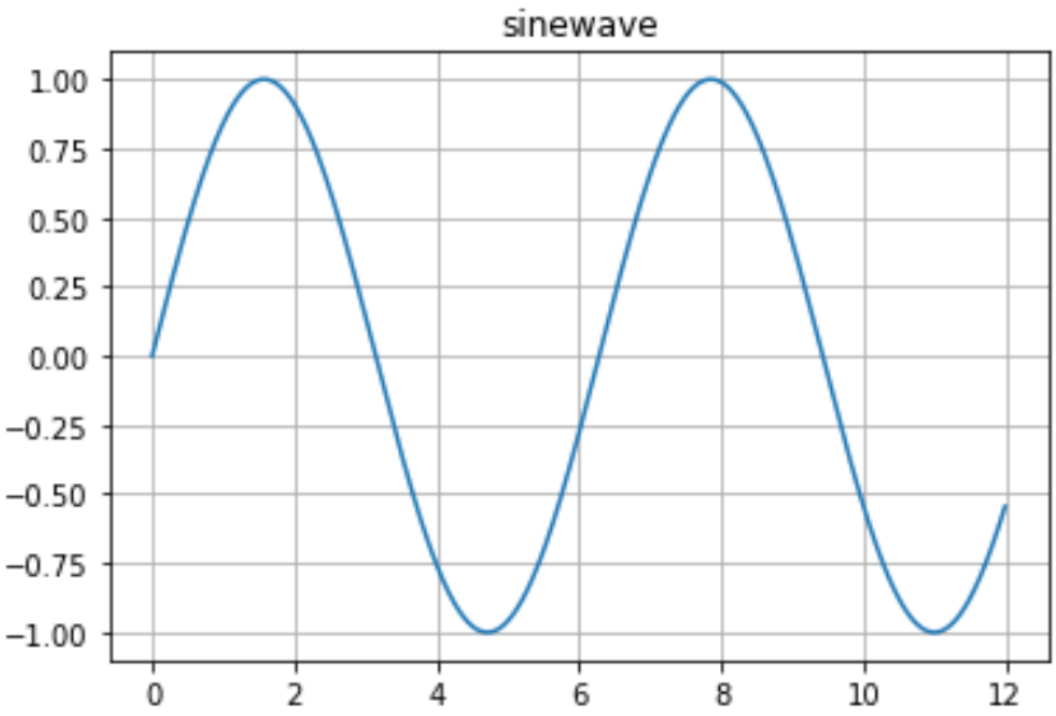
눈금(grid)

```
plt.figure(figsize = (6, 4))
plt.plot(x, y)
plt.grid() # grid(color='r', linestyle='-', linewidth=2)
plt.show()
```



타이틀(title)

```
plt.figure(figsize = (6, 4))
plt.plot(x, y)
plt.grid()
plt.title("sinewave")
plt.show()
```



라벨(label)

```
dy = np.diff(y) # 차분함수 적용
```

```
len(x), len(dy)
```

```
(1200, 1199)
```

```
%whos
```

```
Variable   Type       Data/Info
--------------------------------
dy         ndarray    1199: 1199 elems, type `float64`, 9592 bytes
np         module     <module 'numpy' from 'C:\<...>ges\\numpy\\__init__.py'>
plt        module     <module 'matplotlib.pyplo<...>\\matplotlib\\pyplot.py'>
x          ndarray    1200: 1200 elems, type `float64`, 9600 bytes
y          ndarray    1200: 1200 elems, type `float64`, 9600 bytes
```

라벨(label)

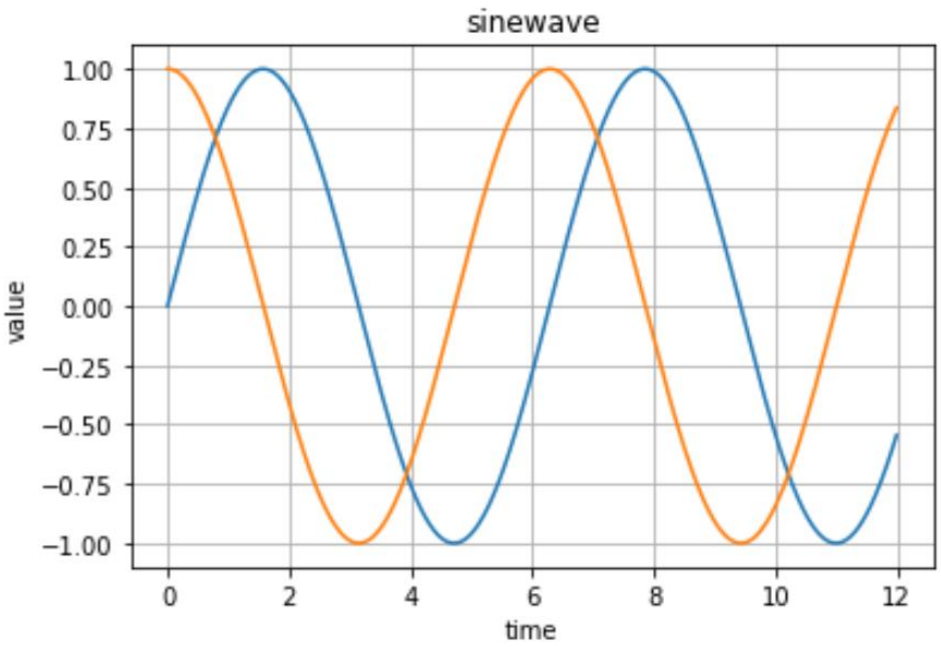
```
plt.figure(figsize = (6, 4))
plt.plot(x, y)
plt.plot(x, dy)
plt.grid()
plt.title("sinewave")
plt.xlabel('time') # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.show()
```

```
    241             raise ValueError("x and y must have same first dimension, but "
--> 242                              "have shapes {} and {}".format(x.shape, y.shape))
    243         if x.ndim > 2 or y.ndim > 2:
    244             raise ValueError("x and y can be no greater than 2-D, but have "

ValueError: x and y must have same first dimension, but have shapes (1200,) and (1199,)
```

라벨(label)

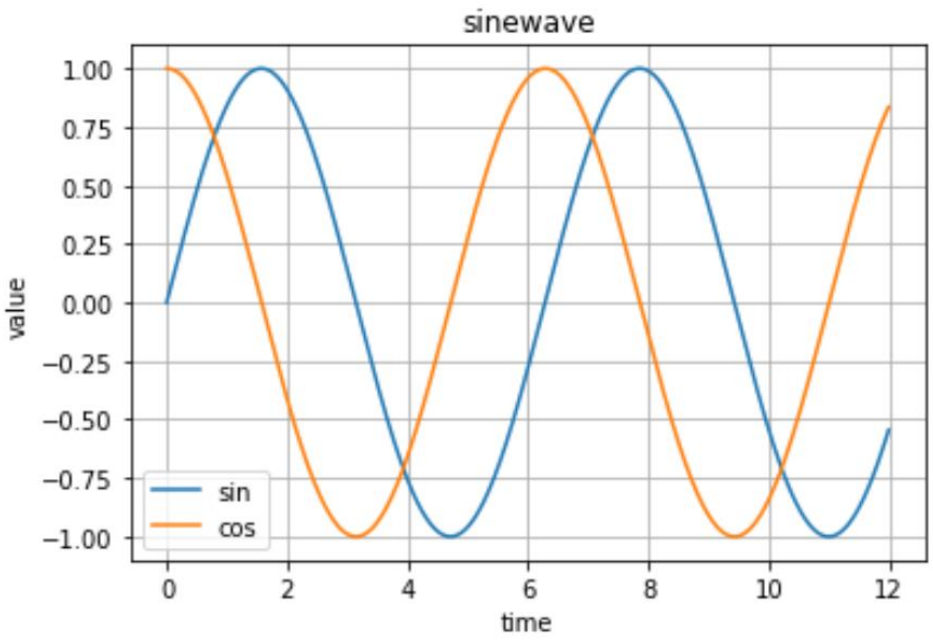
```
dy= np.insert(dy, 0, 0.01)/0.01
```

```
plt.figure(figsize = (6, 4))
plt.plot(x, y)
plt.plot(x, dy)
plt.grid()
plt.title("sinewave")
plt.xlabel('time')  # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.show()
```



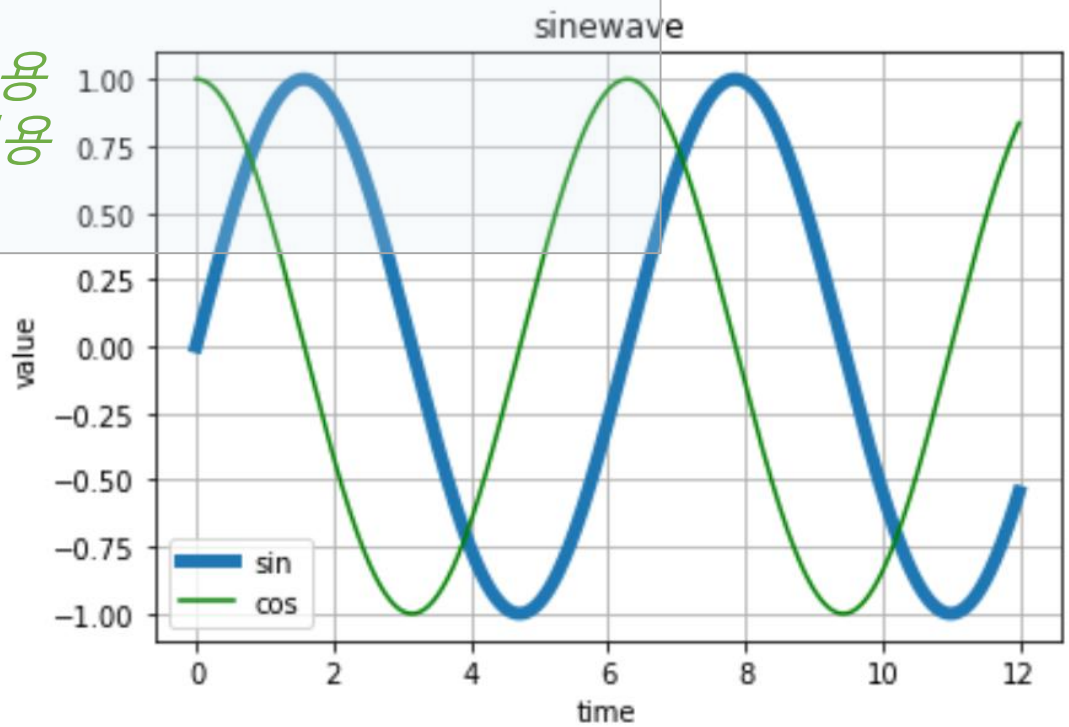
범례(legend)

```
plt.figure(figsize = (6, 4))
plt.plot(x, y, label = 'sin')
plt.plot(x, dy, label = 'cos')
plt.grid()
plt.legend()
plt.title("sinewave")
plt.xlabel('time')  # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.show()
```



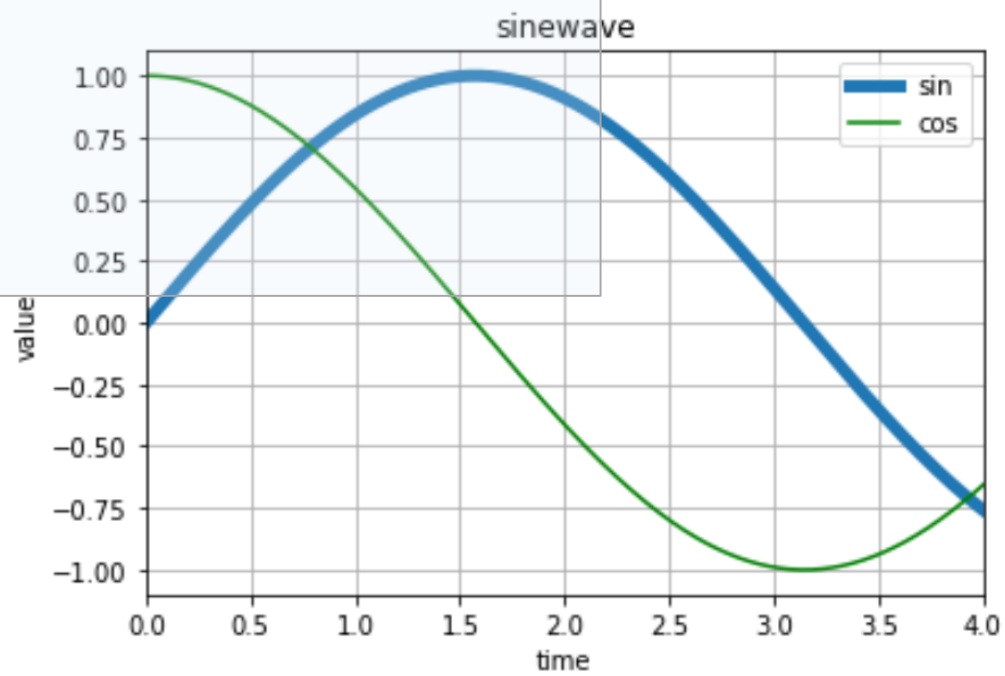
색 및 라인의 크기

```
plt.figure(figsize = (6, 4))
plt.plot(x, y, lw = 5, label = 'sin')   # lw옵션 : 라인크기조정
plt.plot(x, dy, 'g', label = 'cos')
plt.grid()  # 'g' 색 옵션
plt.legend()
plt.title("sinewave")
plt.xlabel('time')  # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.show()
```



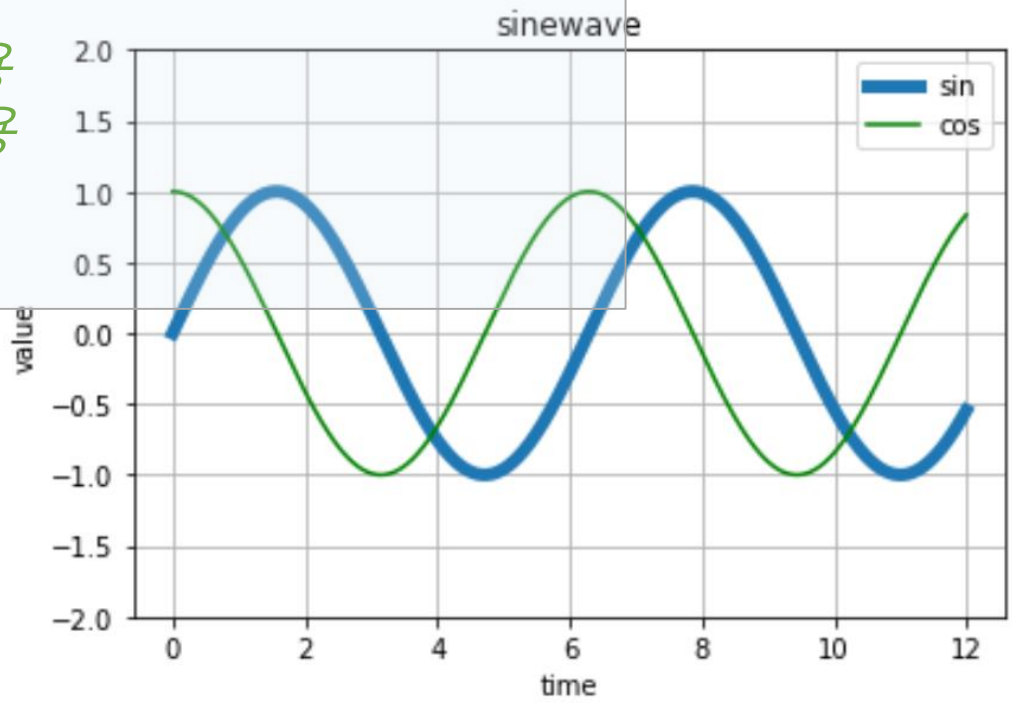
xlim – x, y 축 범위지정

```
plt.figure(figsize = (6, 4))
plt.plot(x, y, lw = 5, label = 'sin')   # lw옵션 : 라인크기조정
plt.plot(x, dy, 'g', label = 'cos')
 plt.grid()  # 'g' 색 옵션
plt.legend()
plt.title("sinewave")
plt.xlabel('time')  # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.xlim(0, 4)
plt.show()
```



xlim – x, y 축 범위지정

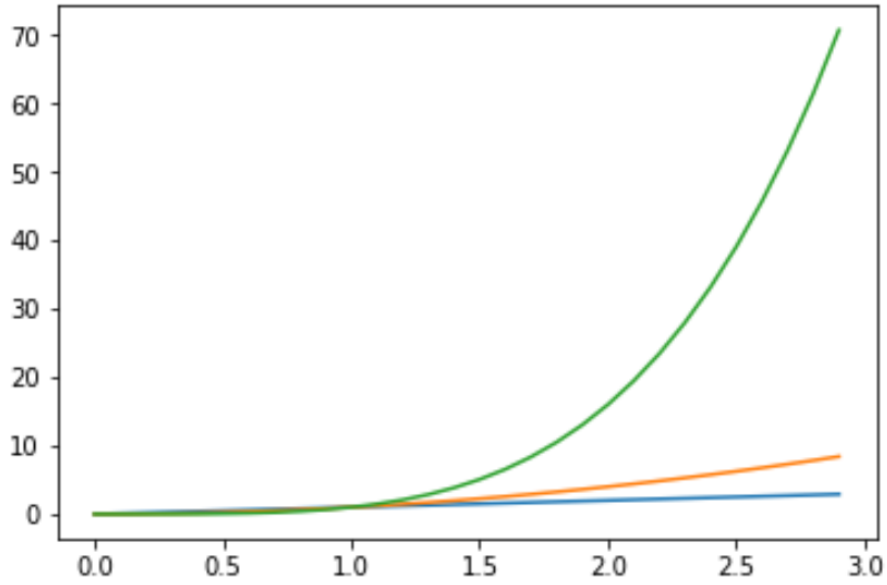
```
plt.figure(figsize = (6, 4))
plt.plot(x, y, lw = 5, label = 'sin')   # lw옵션 : 라인크기조정
plt.plot(x, dy, 'g', label = 'cos')
plt.grid()  # 'g' 색 옵션
plt.legend()
plt.title("sinewave")
plt.xlabel('time')  # x축 라벨 적용
plt.ylabel('value') # y축 라벨 적용
plt.ylim(-2, 2)
plt.show()
```



마커 활용

- https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

```
a = np.arange(0, 3, 0.1)

plt.figure()
plt.plot(a, a, a, a**2, a, a**4)
# plot 안에 여러개의 x,y 데이터를 가지면, 다수의 그래프 가능
plt.show()
```

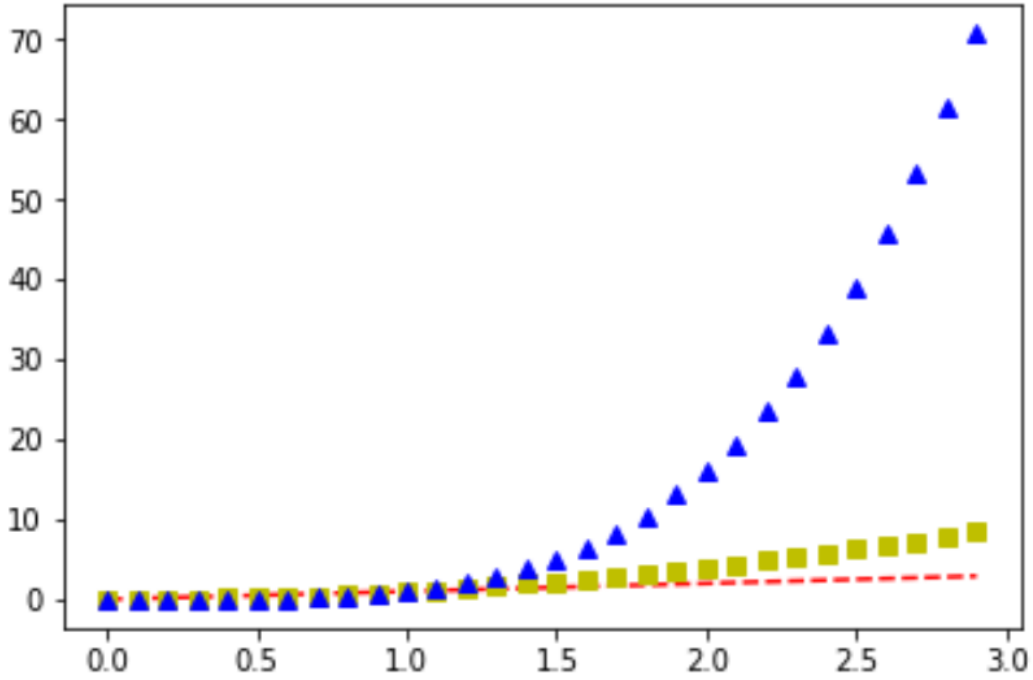


마커

```
a = np.arange(0, 3, 0.1)

plt.figure()
plt.plot(a, a, 'r--', a, a**2, 'ys', a, a**4, 'b^')
plt.show()
```

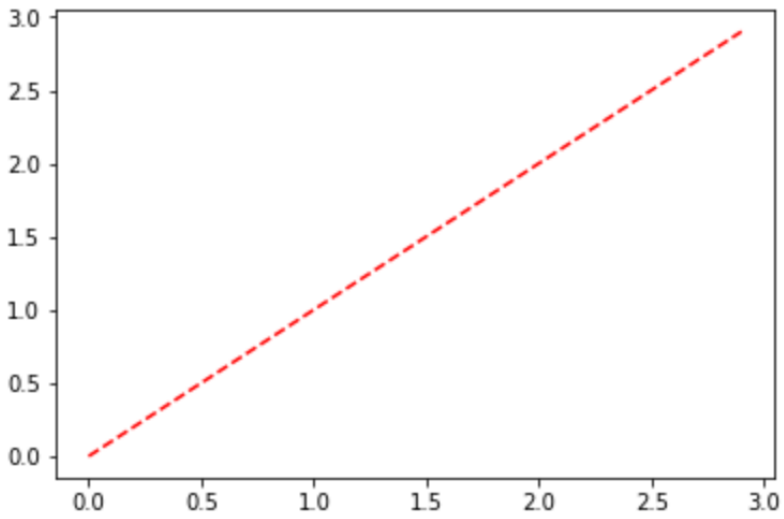
* plot안에 여러개의 x,y 데이터를 가지면, 다수의그래프 가능
* r-- : 빨간색 점선
* 'ys' : 노란색 박스
* 'b^' 위로 향한 세모모양의 파란색

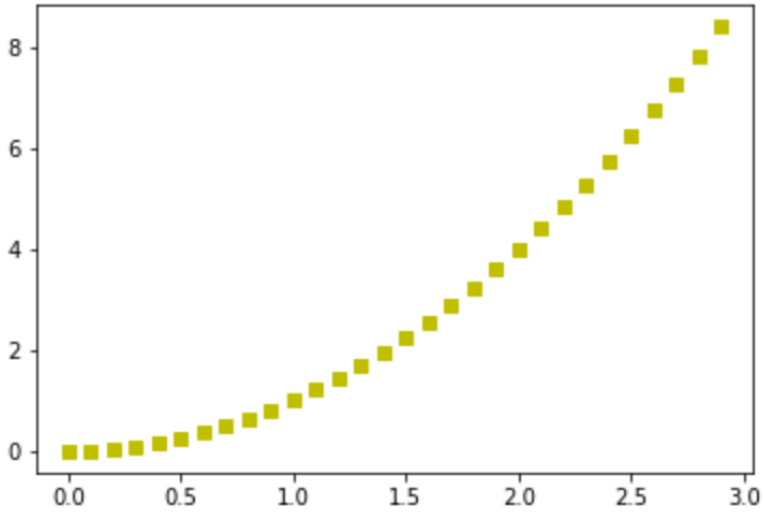
```
a = np.arange(0, 3, 0.1)

plt.figure()
plt.plot(a, a, 'r--‘)
plt.plot(a, a**2, 'ys‘)
plt.plot(a, a**4, 'b^')
plt.show()
```



마커

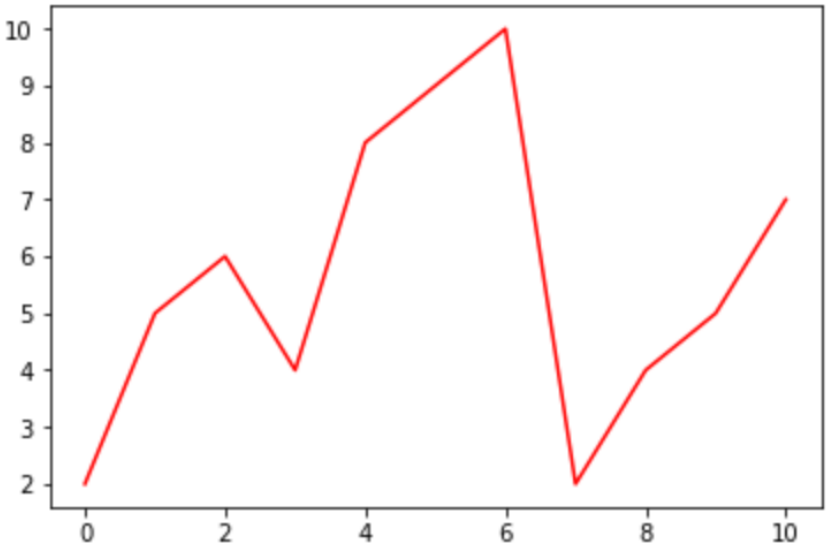
```
a = np.arange(0, 3, 0.1)

plt.figure(1)
plt.plot(a, a, 'r--‘)

plt.figure(1)
plt.plot(a, a**2, 'ys‘)

plt.show()
```



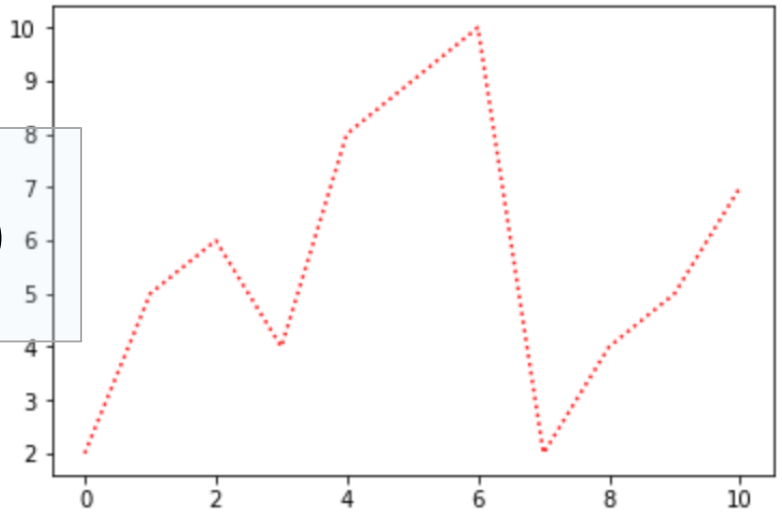


마커

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = [2, 5, 6, 4, 8, 9, 10, 2, 4, 5, 7]

plt.figure()
plt.plot(a, b, color = 'red')
plt.show()
```

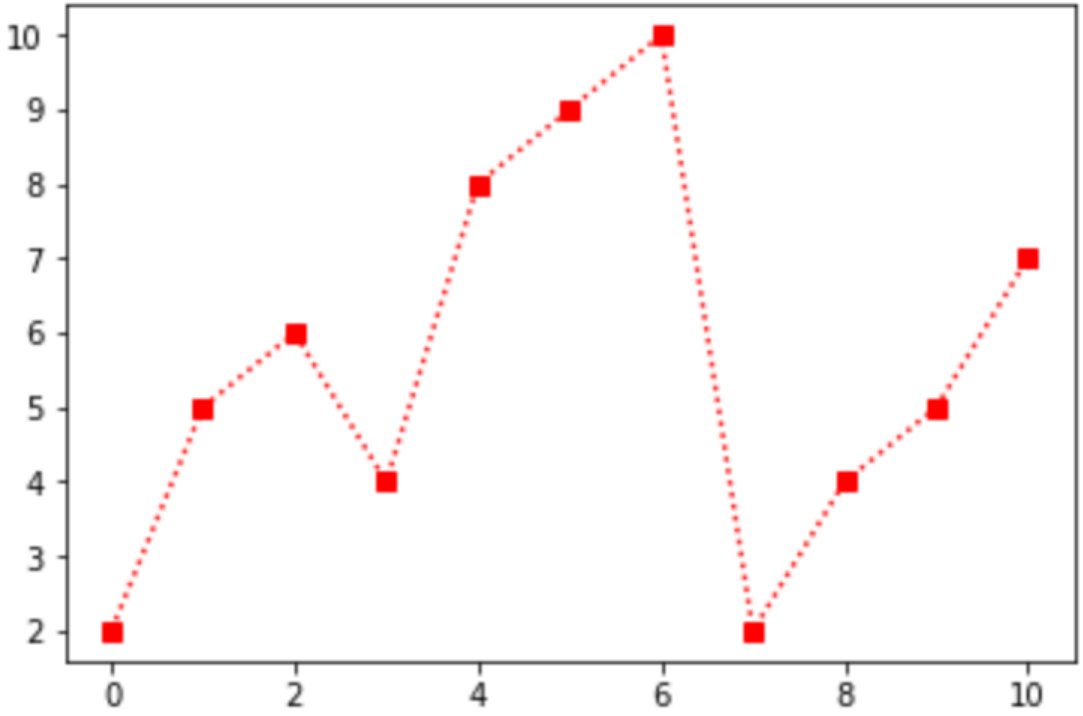



```
plt.figure()
plt.plot(a, b, color = 'red', linestyle = 'dotted')
plt.show()
```



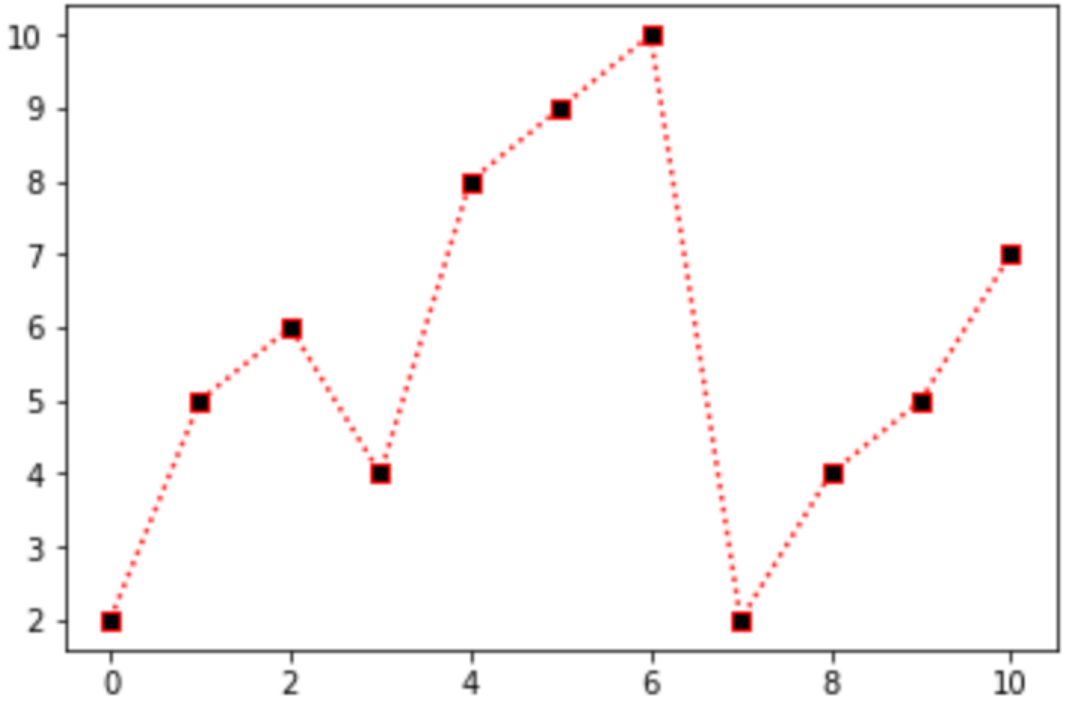
마커

```
plt.figure()
plt.plot(a, b, color = 'red', linestyle = 'dotted‘, marker = 's')
plt.show()
```



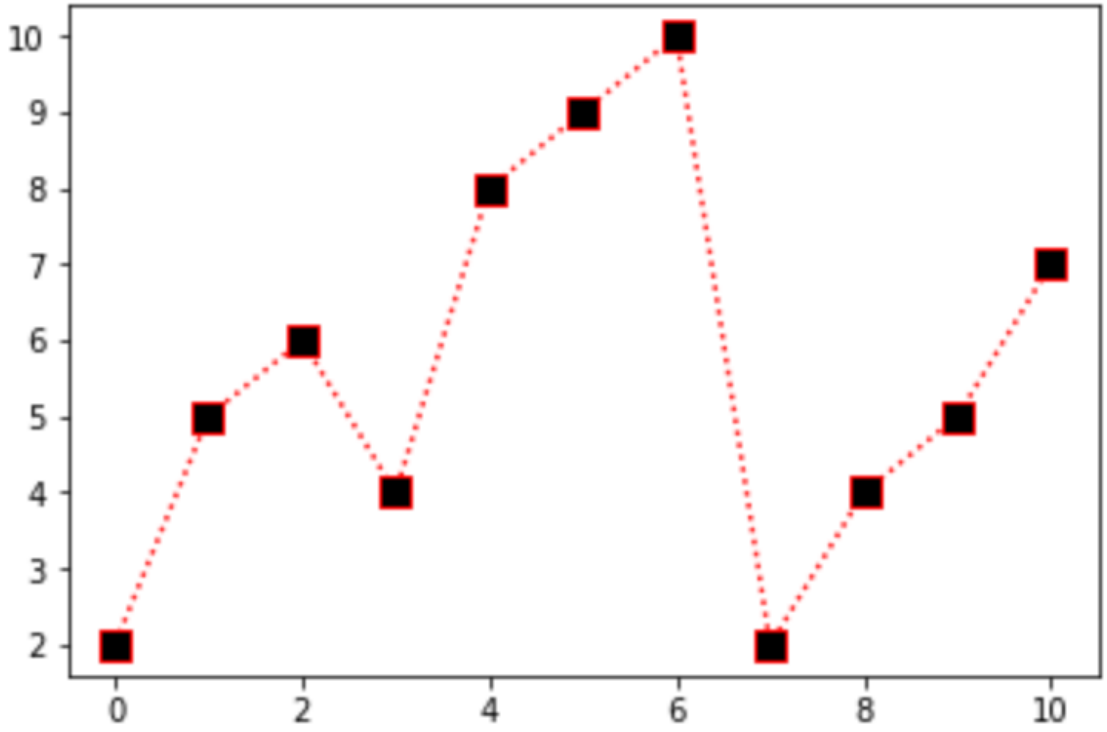
마커

```
plt.figure()
plt.plot(a, b, color = 'red', linestyle = 'dotted', marker = 's', markerfacecolor = 'black')
plt.show()
```



마커

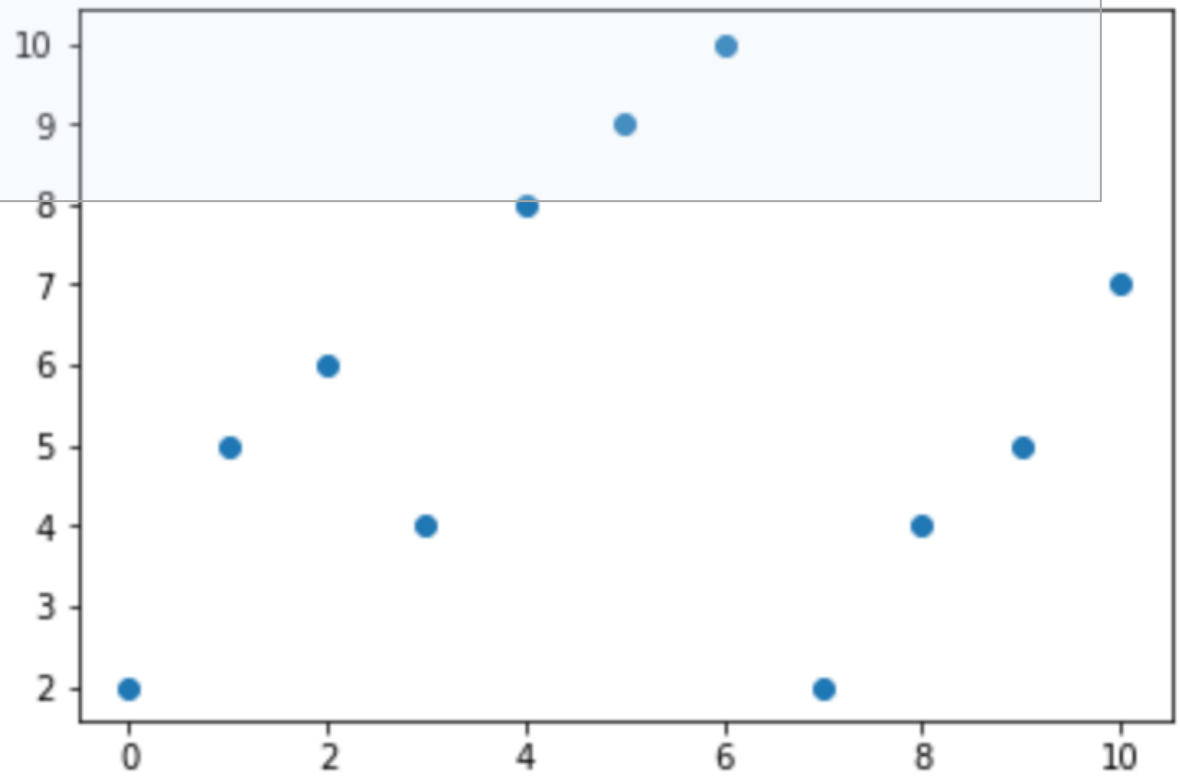
```
plt.figure()
plt.plot(a, b, color = 'red', linestyle = 'dotted‘, marker = 's', markerfacecolor = 'black',
markersize = 10)
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- scatter : 점으로 표현하는 데이터

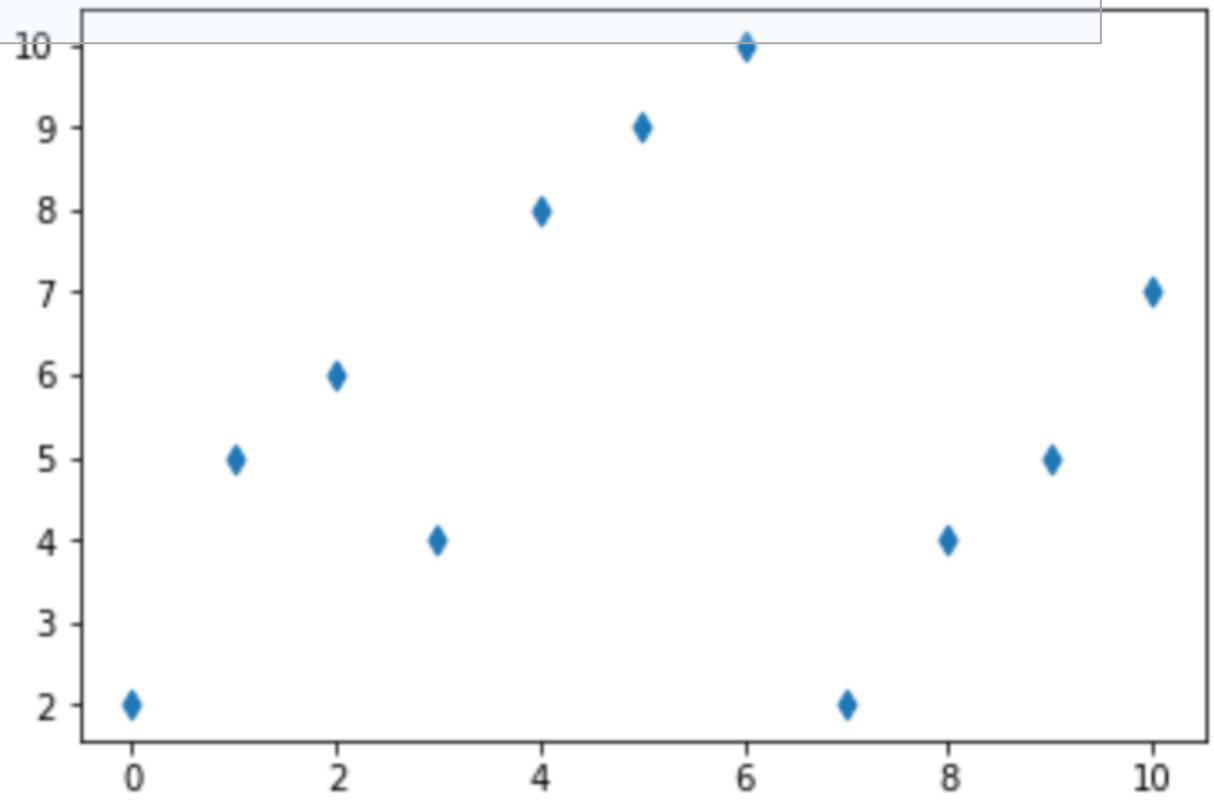
```
a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
b = np.array([2, 5, 6, 4, 8, 9, 10, 2, 4, 5, 7])

plt.figure()
plt.scatter(a,b)
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- scatter : 점으로 표현하는 데이터

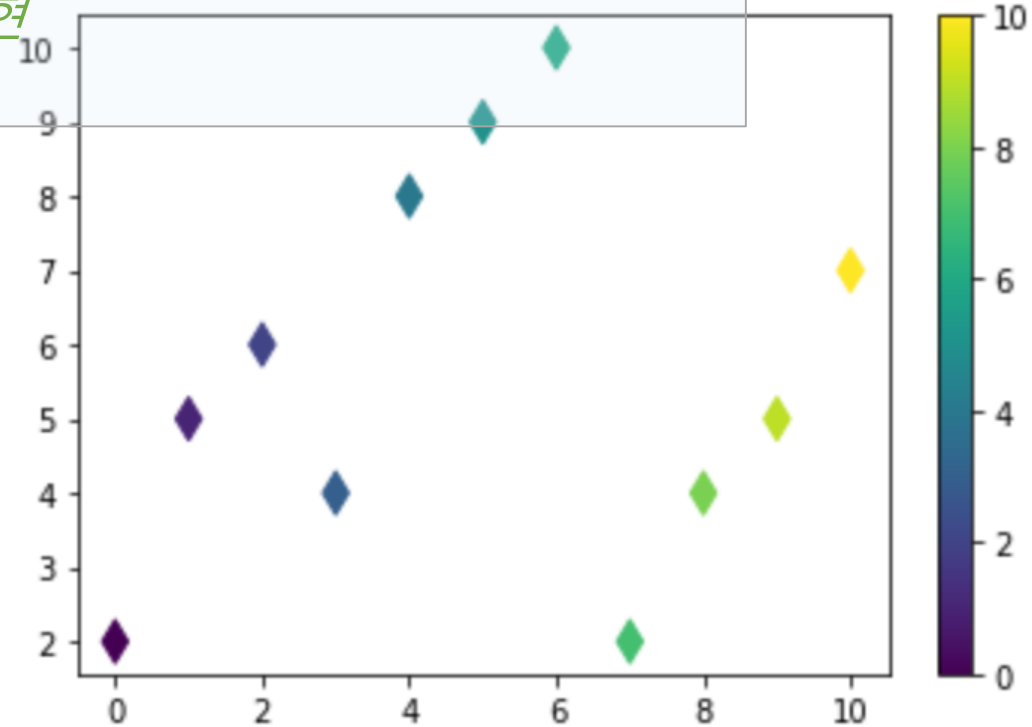
```
plt.figure()
plt.scatter(a, b, marker = 'd')
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- scatter : 점으로 표현하는 데이터

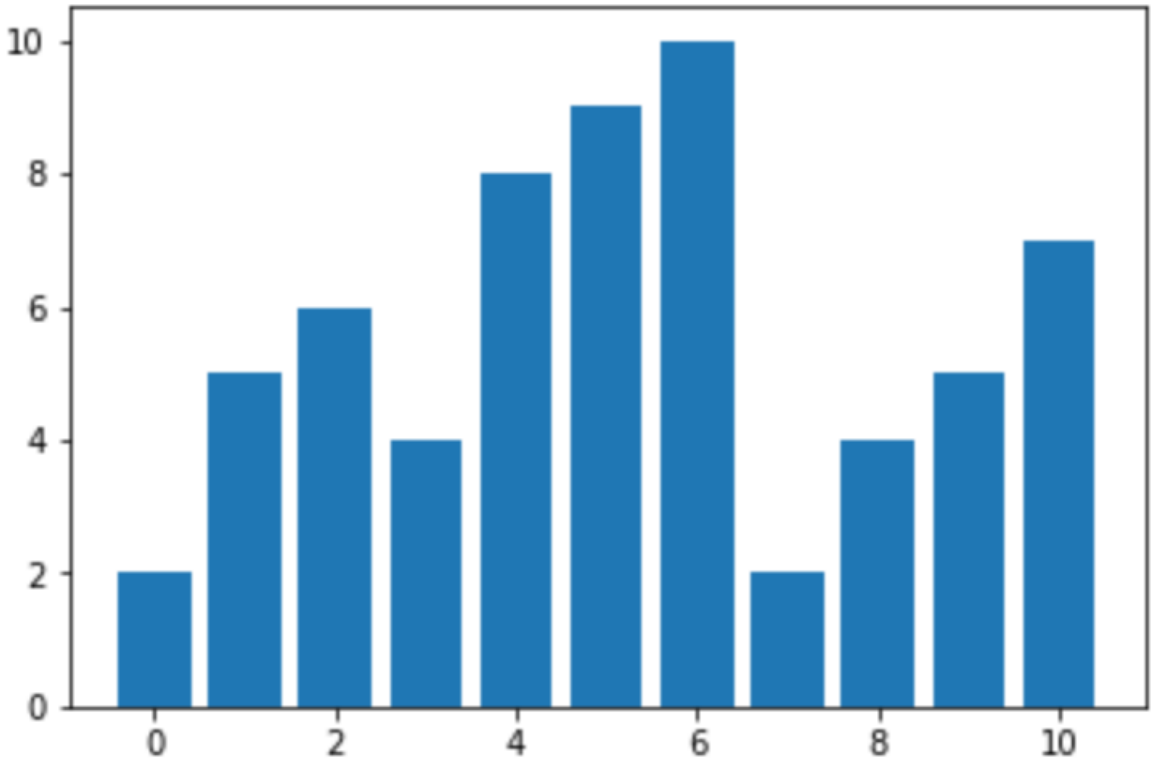
```
colormap = a  # color 지정

plt.figure()
plt.scatter(a, b, s = 100, c = colormap, marker = 'd')
plt.colorbar()  # 사용한 color 표현
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

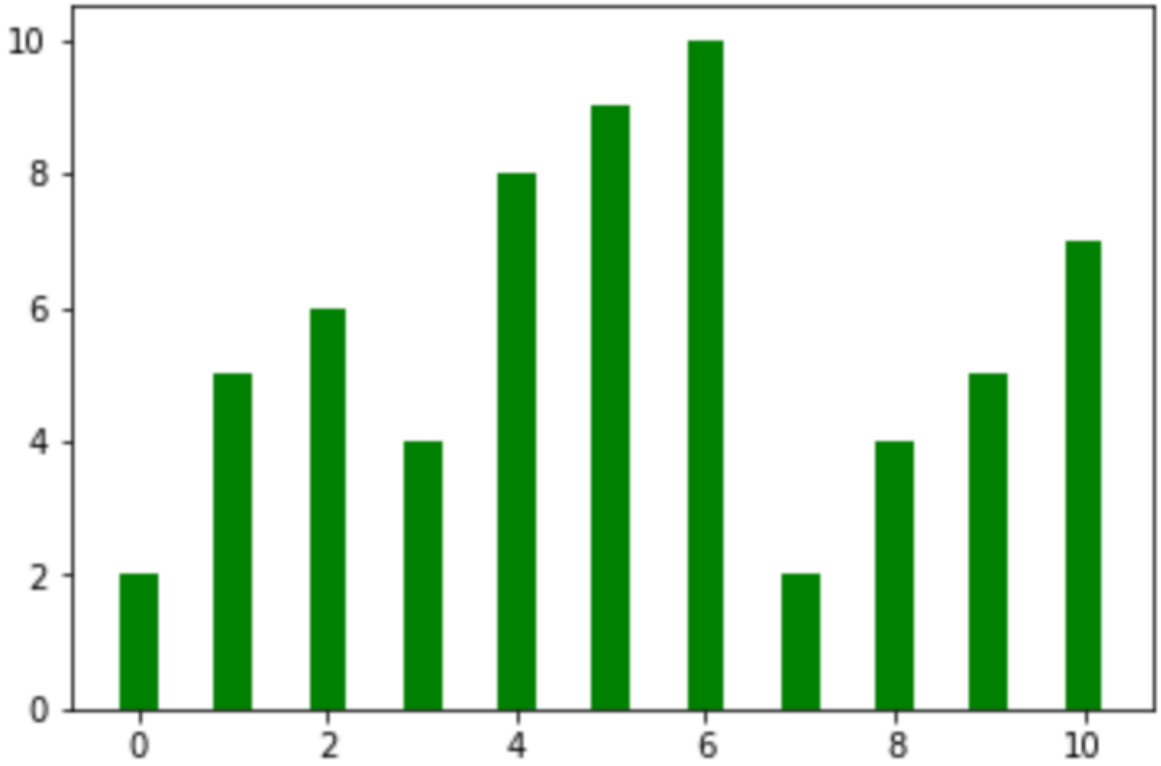
```
plt.figure()
plt.bar(a, b)
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

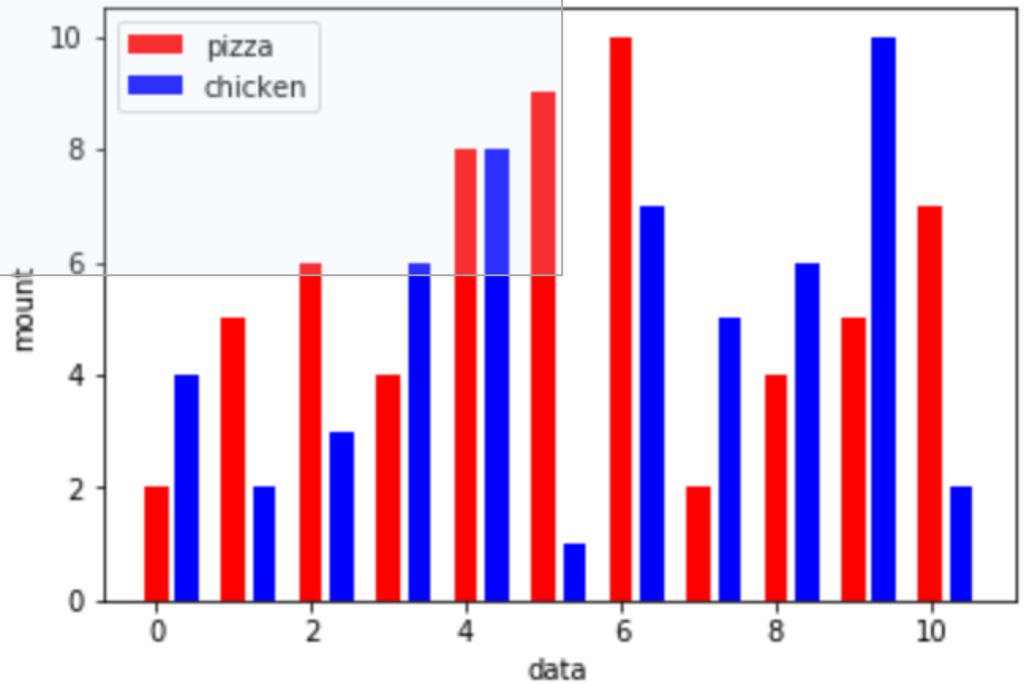
```
plt.figure()
plt.bar(a, b, width = 0.4, color = 'g')  # width : 넓이조정
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

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

plt.figure()
plt.bar(a, b, color = 'r', width = 0.3, label = 'pizza')
plt.bar(a + 0.4, y, color = 'b', width = 0.3, label = 'chicken')
plt.xlabel('data')
plt.ylabel('mount')
plt.legend()
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

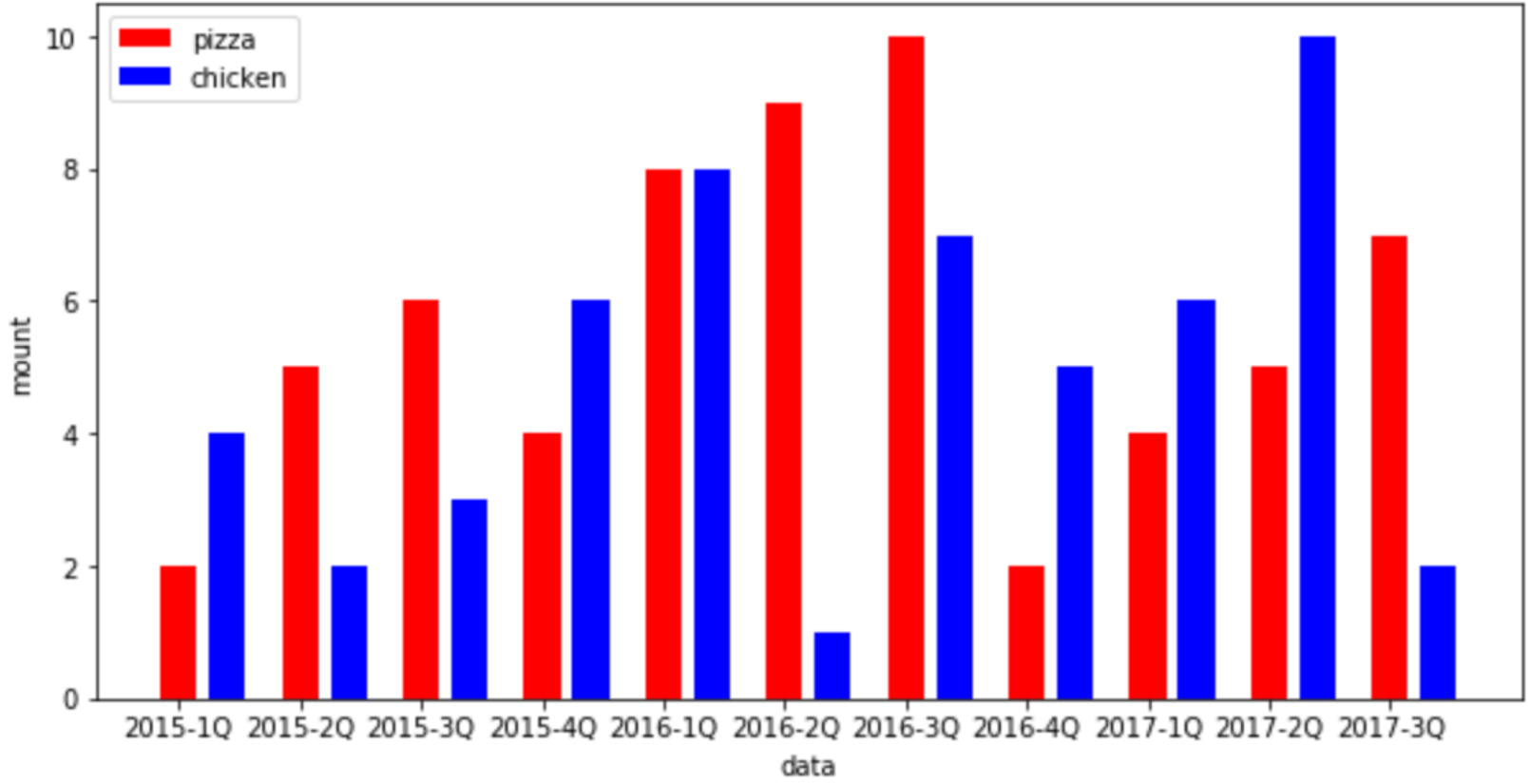
* xticks를 이용한 x축 설정

```
plt.figure(figsize = (20, 10))

plt.bar(a, b, color = 'r', width = 0.3, label = 'pizza')
plt.bar(a + 0.4, y, color = 'b', width = 0.3, label = 'chicken')
plt.xlabel('data')
plt.ylabel('mount')
plt.legend()
plt.xticks(a, ('2015-1Q', '2015-2Q', '2015-3Q', '2015-4Q', '2016-1Q', '2016-2Q', '2016-3Q', '2016-4Q', '2017-1Q', '2017-2Q', '2017-3Q' ))

plt.show()
```

여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

* xticks를 이용한 x축 설정



여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

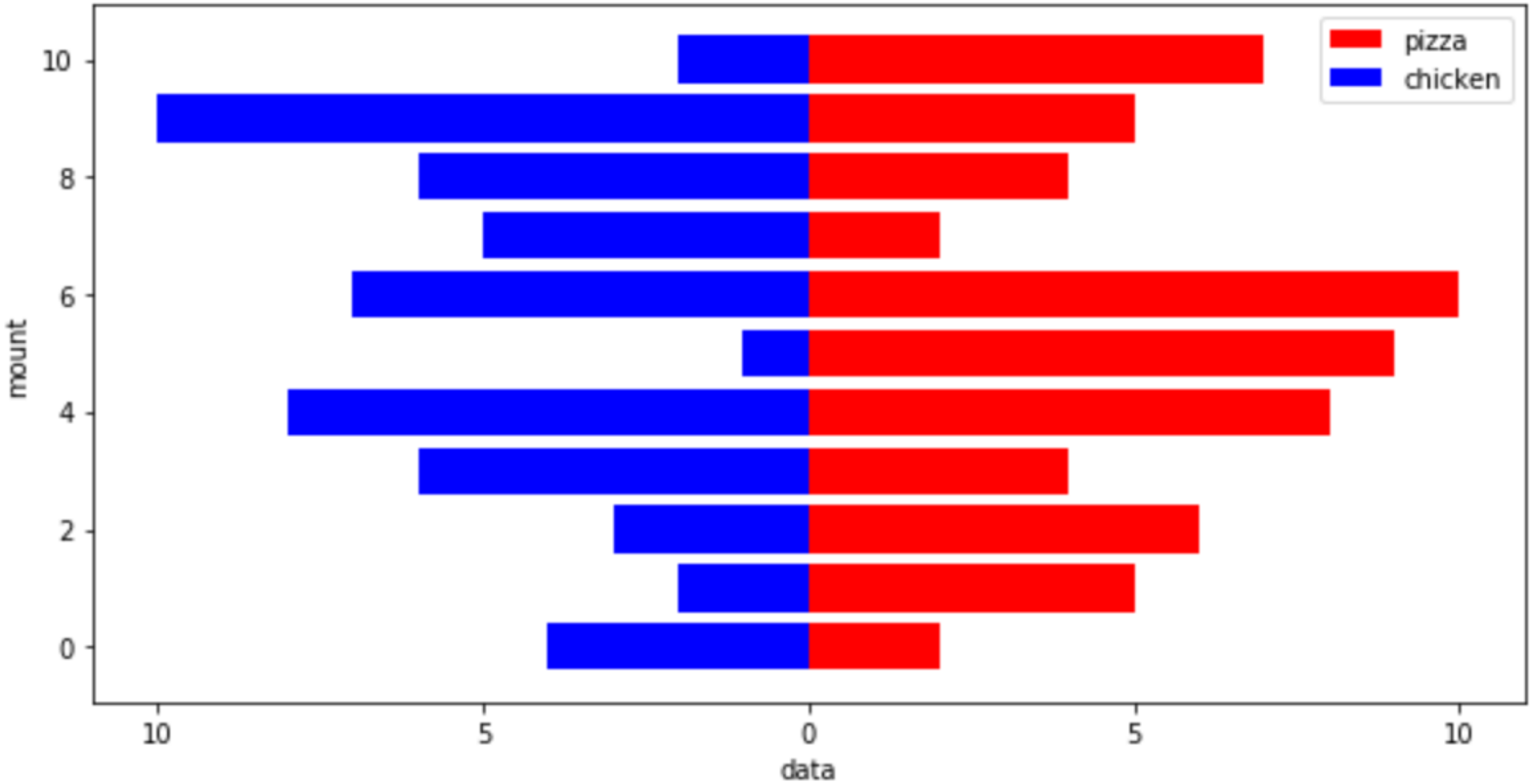
* barh를 이용한 대칭 바그래프

```
plt.figure(figsize = (10, 5))

plt.barh(a, b, color = 'r', width = 0.3, label = 'pizza')
plt.barh(a -y, y, color = 'b', width = 0.3, label = 'chicken')
plt.xlabel('data')
plt.ylabel('mount')
plt.legend()
plt.xticks( [-10, -5, 0, 5, 10], ('10', '5', '0', '5', '10'))

plt.show()
```

여러형태의 그래프(scatter, bar, barh, pie)

- bar : 바형태로 표현하는 데이터

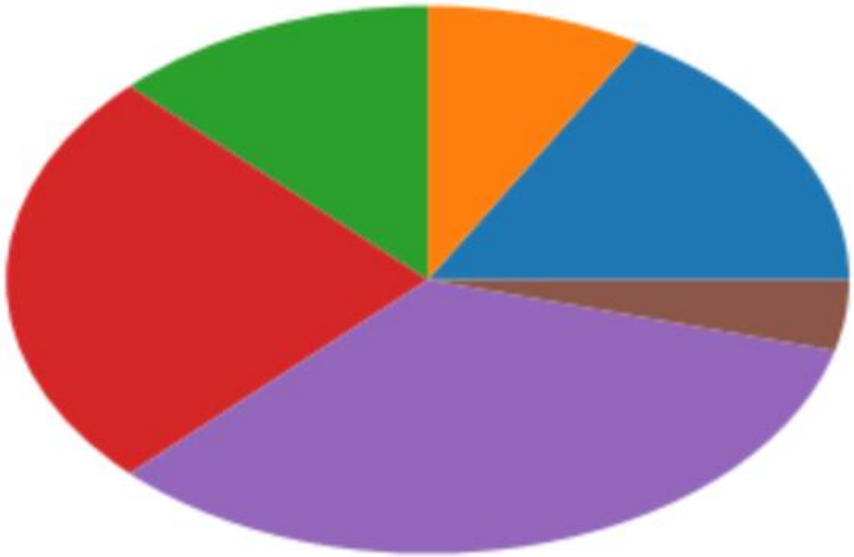
* barh를 이용한 대칭 바그래프



여러형태의 그래프(scatter, bar, barh, pie)

- pie

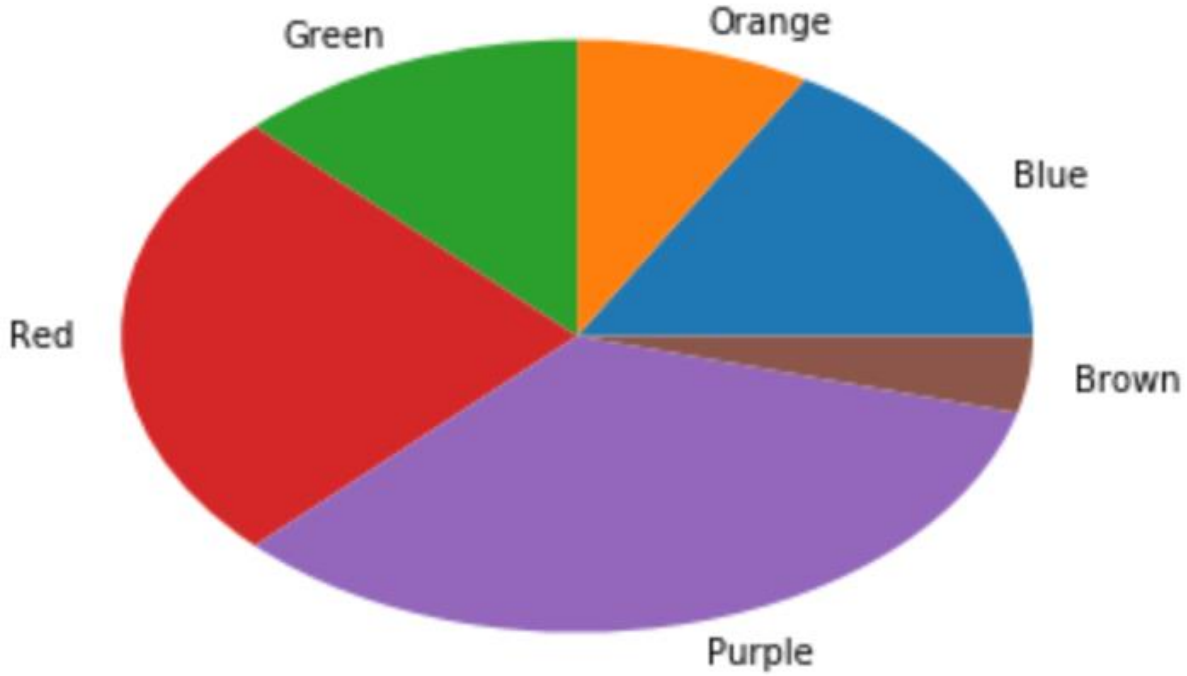
```
y = np.array([4, 2, 3, 6, 8, 1])

plt.figure()
plt.pie(y)
plt.show()
```



여러형태의 그래프(scatter, bar, barh, pie)

- pie

```
label = [ 'Blue', 'Orange', 'Green', 'Red', 'Purple', 'Brown']
plt.figure
plt.pie(y, labels = label)
plt.show()
```



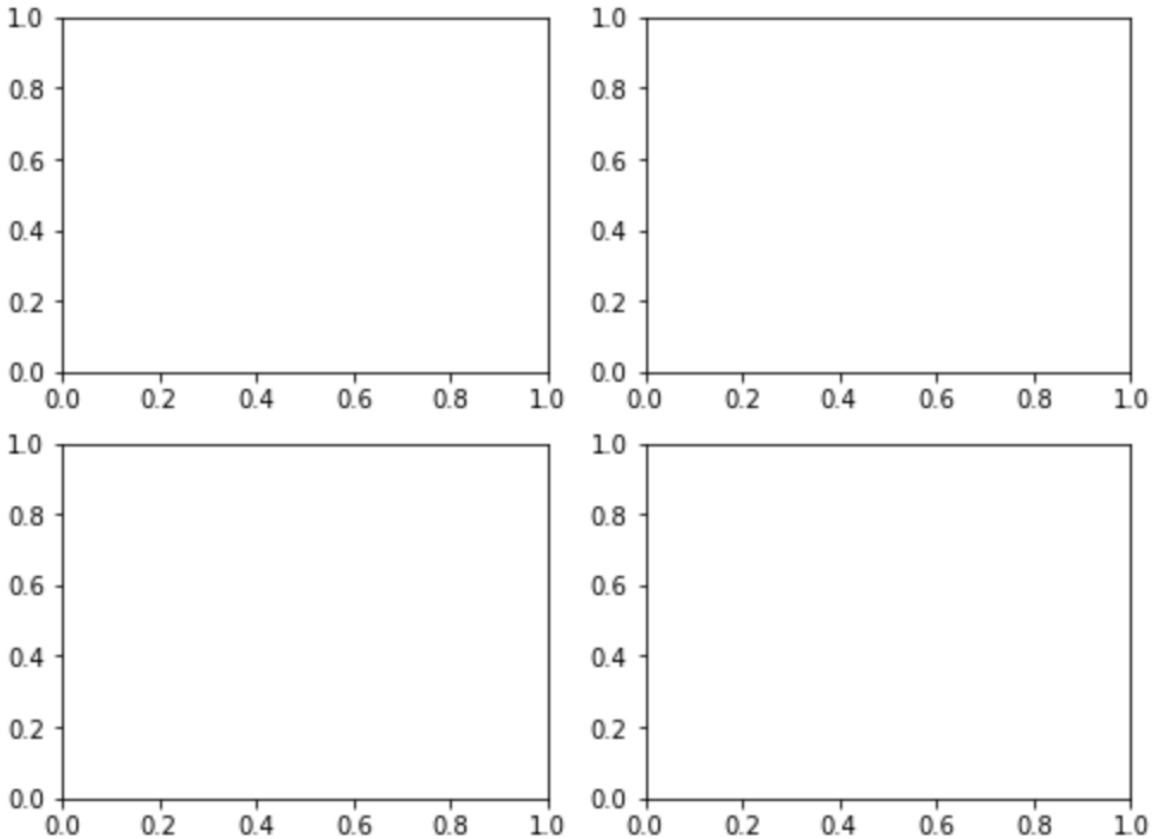
subplot

- 한 화면에 여러 그래프 나타내기

```
plt. figure(figsize = (8, 6))

plt.subplot(221)  # 2 by 2 subplot 중 1 번째
plt.subplot(222)  # 2 by 2 subplot 중 2 번째
plt.subplot(223)  # 2 by 2 subplot 중 3 번째
plt.subplot(224)  # 2 by 2 subplot 중 4 번째

plt.show()
```

subplot

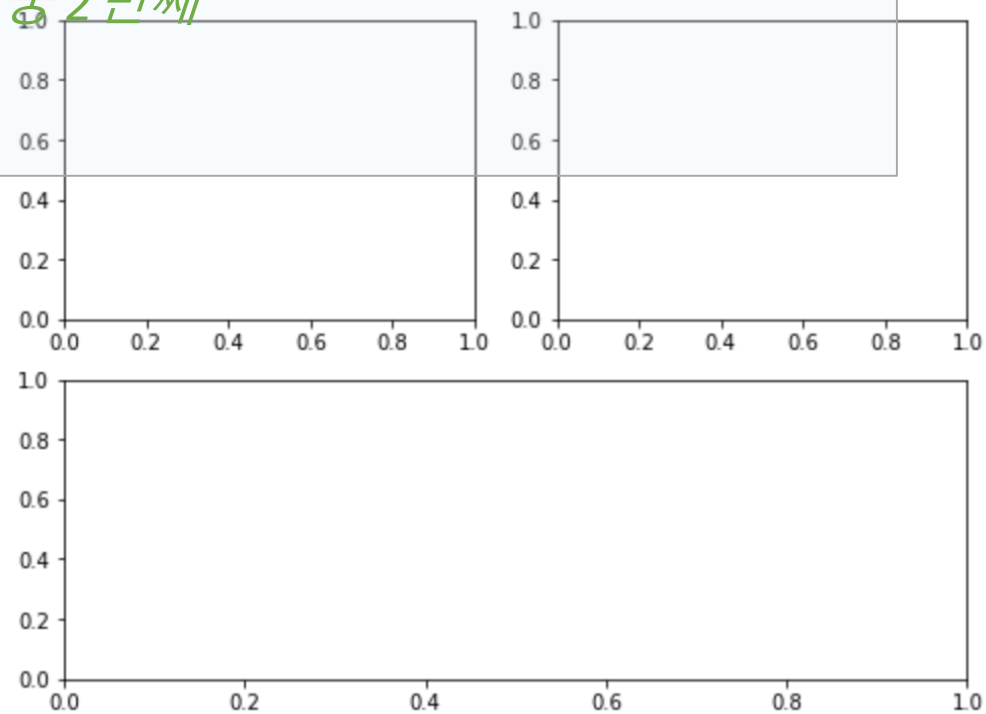
- 한 화면에 여러 그래프 나타내기



subplot

- 한 화면에 여러 그래프 나타내기

```
plt. figure(figsize = (8, 6))

plt.subplot(221)   # 2 by 2 subplot 중 1 번째
plt.subplot(222)   # 2 by 2 subplot 중 2 번째
plt.subplot(212)   # 2 by 1 subplot 중 2 번째

plt.show()
```



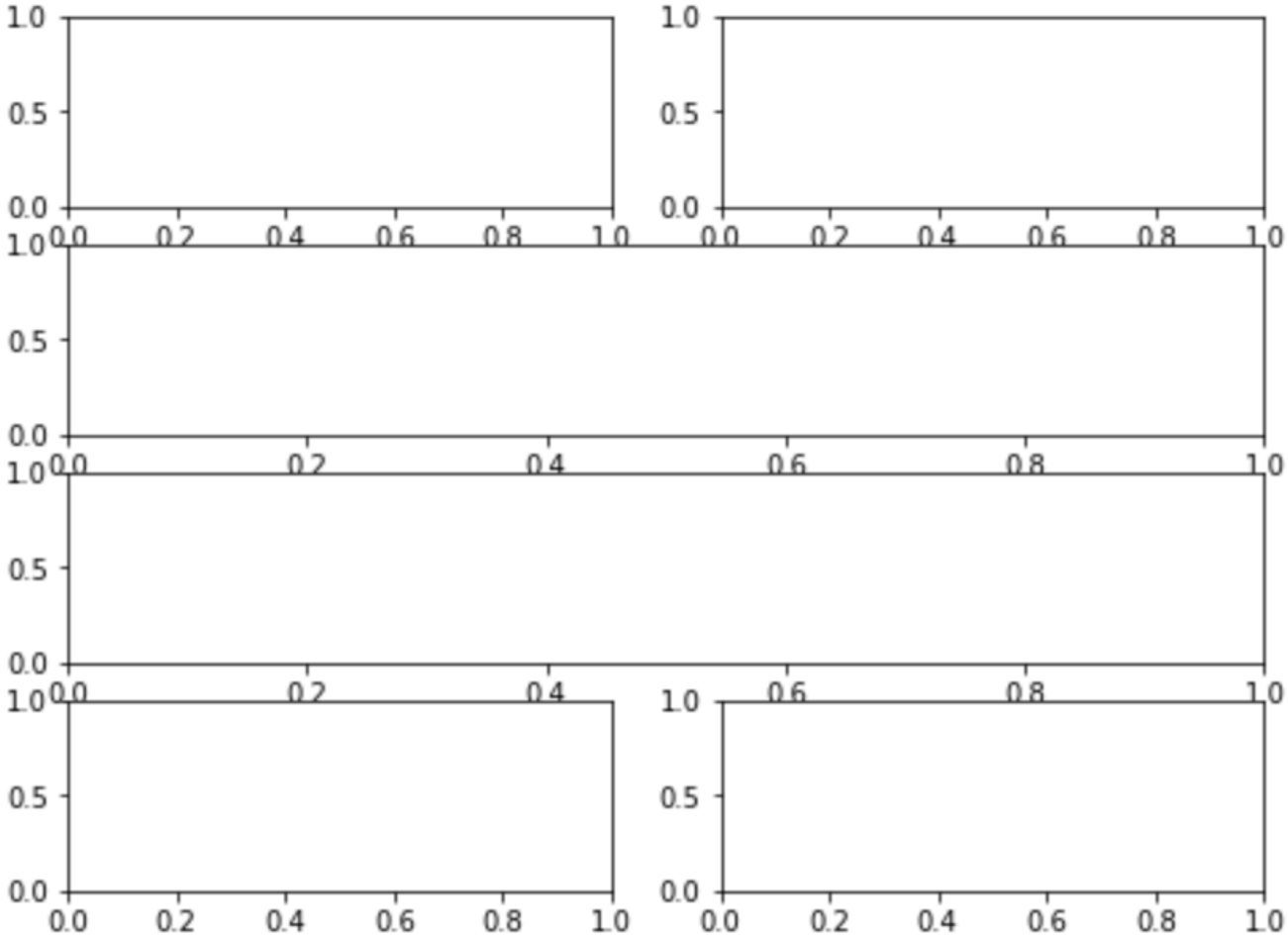
subplot

- 한 화면에 여러 그래프 나타내기

```
plt. figure(figsize = (8, 6))

plt.subplot(421)  # 4 by 2 subplot 중 1 번째
plt.subplot(422)  # 4 by 2 subplot 중 2 번째
plt.subplot(412)  # 4 by 1 subplot 중 2 번째
plt.subplot(413)  # 4 by 1 subplot 중 3 번째
plt.subplot(427)  # 4 by 2 subplot 중 7 번째
plt.subplot(428)  # 4 by 2 subplot 중 8 번째


plt.show()
```

subplot

- 한 화면에 여러 그래프 나타내기



subplot

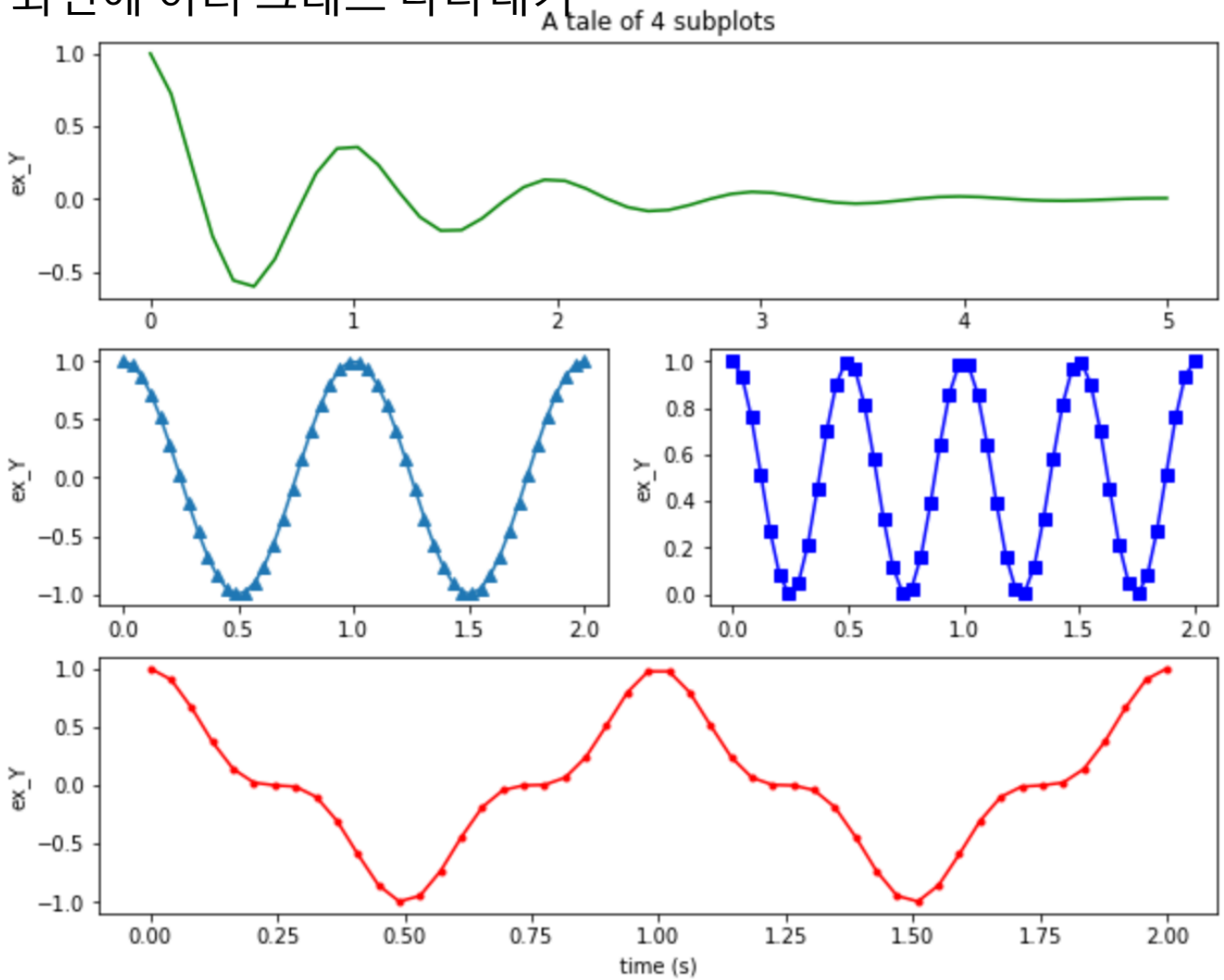
- 한 화면에 여러 그래프 나타내기

```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

plt. figure(figsize = (10, 8))

plt.subplot(3, 1, 1)
plt.plot(x1, y1, 'g-')
plt.title('A tale of 4 subplots')
plt.ylabel('ex_Y')
```

```
plt.subplot(3, 2, 3)
plt.plot(x2, y2, '^-')
plt.ylabel('ex_Y')

plt.subplot(3, 2, 4)
plt.plot(x2, y2**2, '-sb')
plt.ylabel('ex_Y')

plt.subplot(3, 1, 3)
plt.plot(x2, y2**3, 'r.-')
plt.xlabel('time (s)')
plt.ylabel('ex_Y')

plt.show()
```

subplot

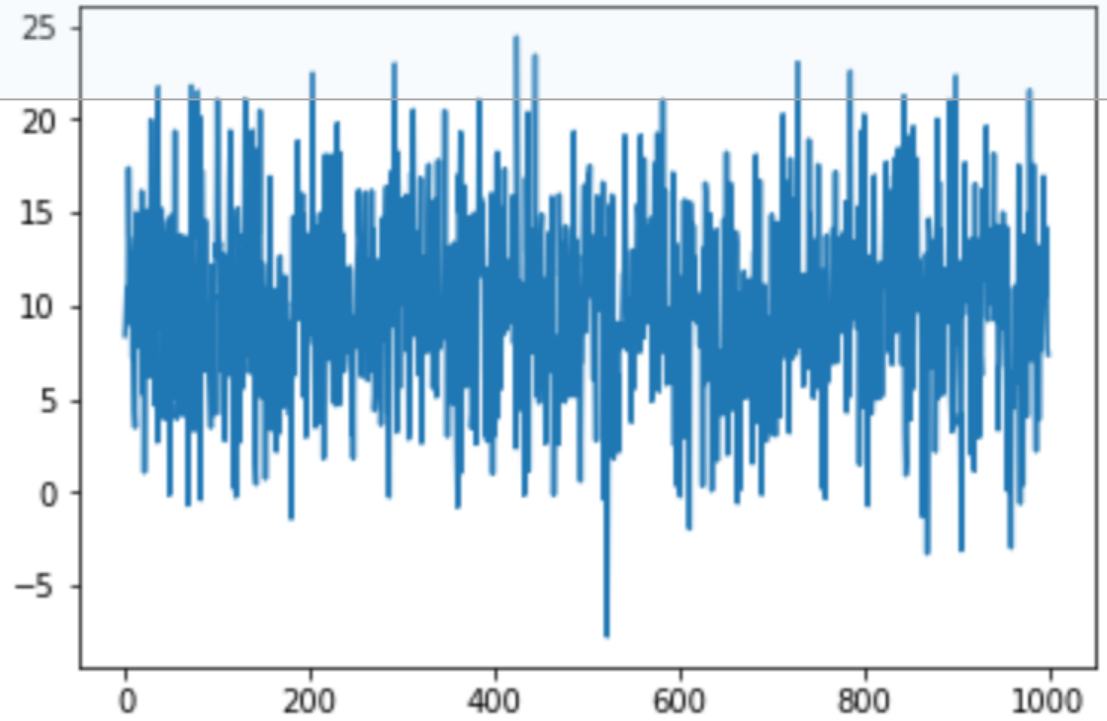
- 한 화면에 여러 그래프 나타내기



히스토그램 & 박스플롯(boxplot)

- 가우스분포를 가지는 랜덤변수 활용

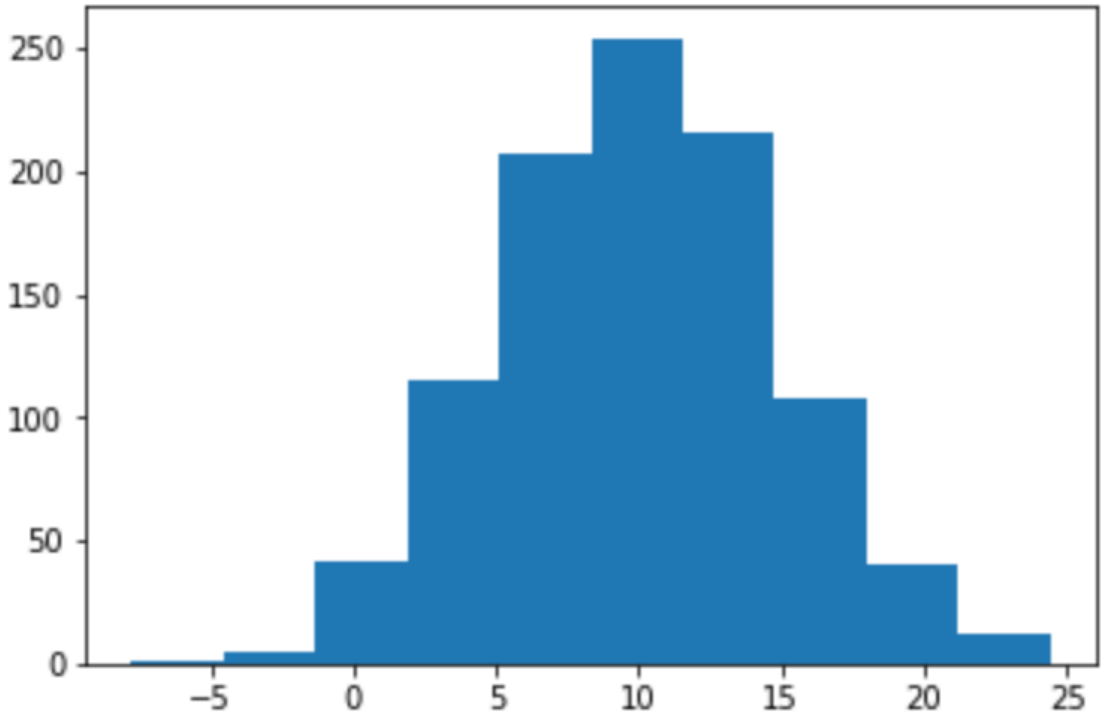
```
data = np.random.normal(10, 5, 1000)
# 중간값 : 10, 크기 : 5, 데이터갯수 : 500개, 가우스 분포이므로 5~15를 넘을 수 있음

plt.figure()
plt.plot(data)
plt.show()
```



히스토그램 & 박스플롯(boxplot)

- 가우스분포를 가지는 랜덤변수 활용

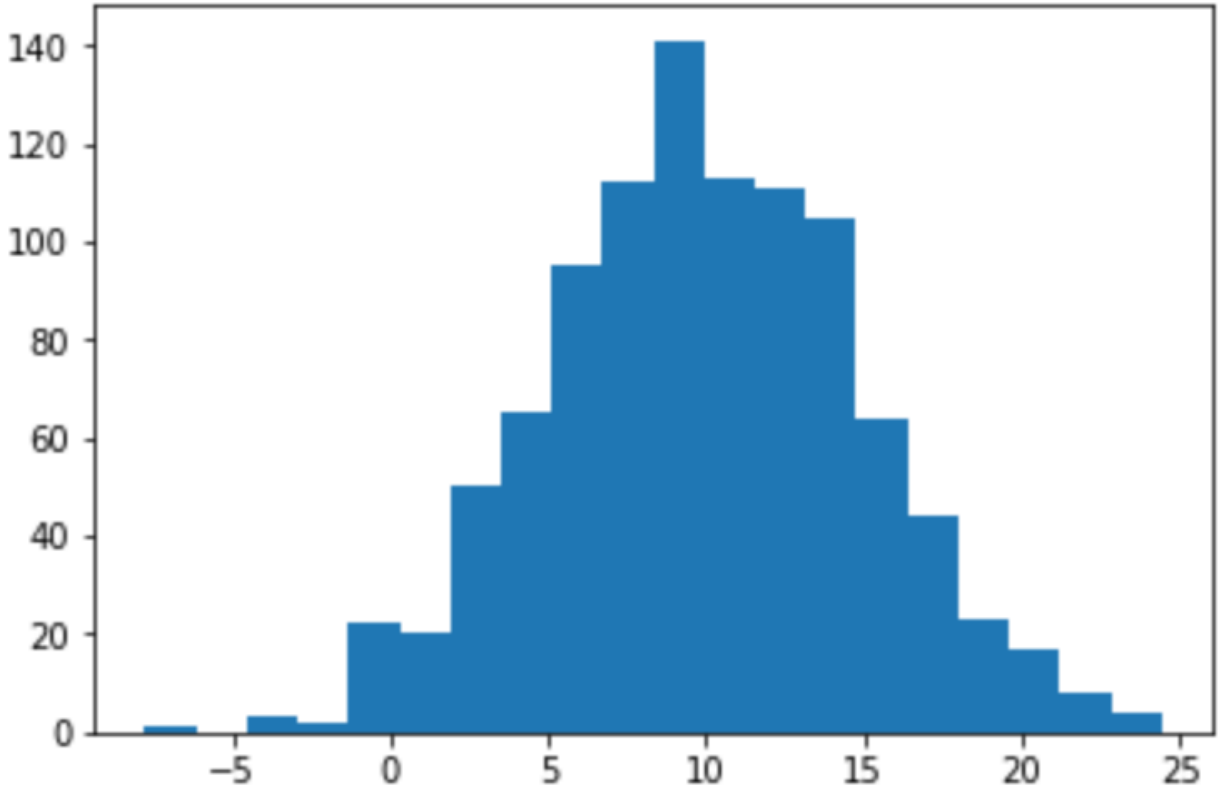
```
plt.figure()
plt.hist(data)
plt.show()
```



히스토그램 & 박스플롯(boxplot)

- 가우스분포를 가지는 랜덤변수 활용

```
plt.figure()
plt.hist(data, bins = 20)
plt.show()
```



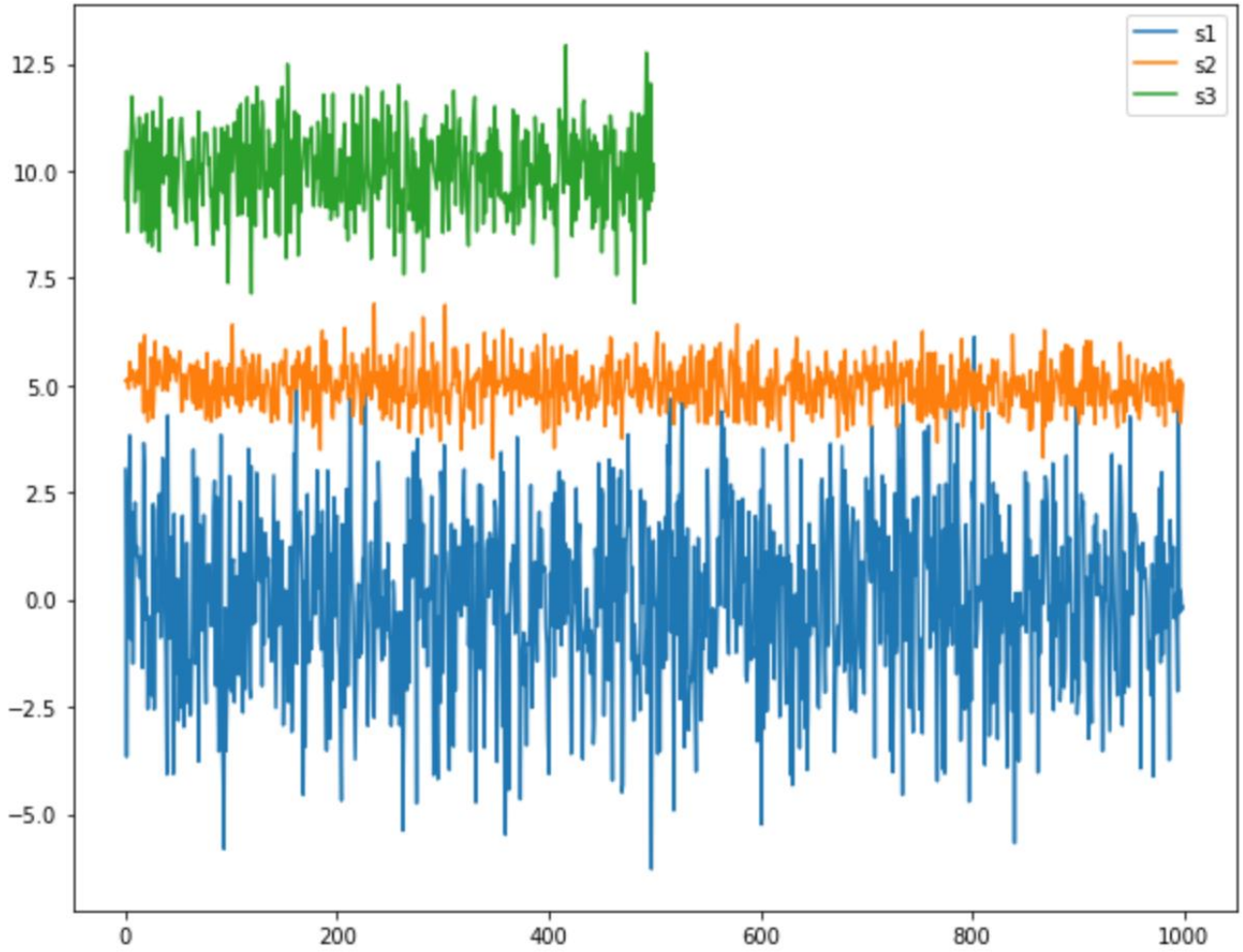
히스토그램 & 박스플롯(boxplot)

- 가우스분포를 가지는 랜덤변수 활용

```
s1 = np.random.normal( loc = 0, scale = 2, size = 1000)
s2 = np.random.normal( loc = 5, scale = 0.5, size = 1000)
s3 = np.random.normal( loc = 10, scale = 1, size = 500)

plt.figure(figsize = (10,8))

plt.plot(s1, label = 's1')
plt.plot(s2, label = 's2')
plt.plot(s3, label = 's3')
plt.legend()

plt.show()
```

히스토그램 & 박스플롯(boxplot)

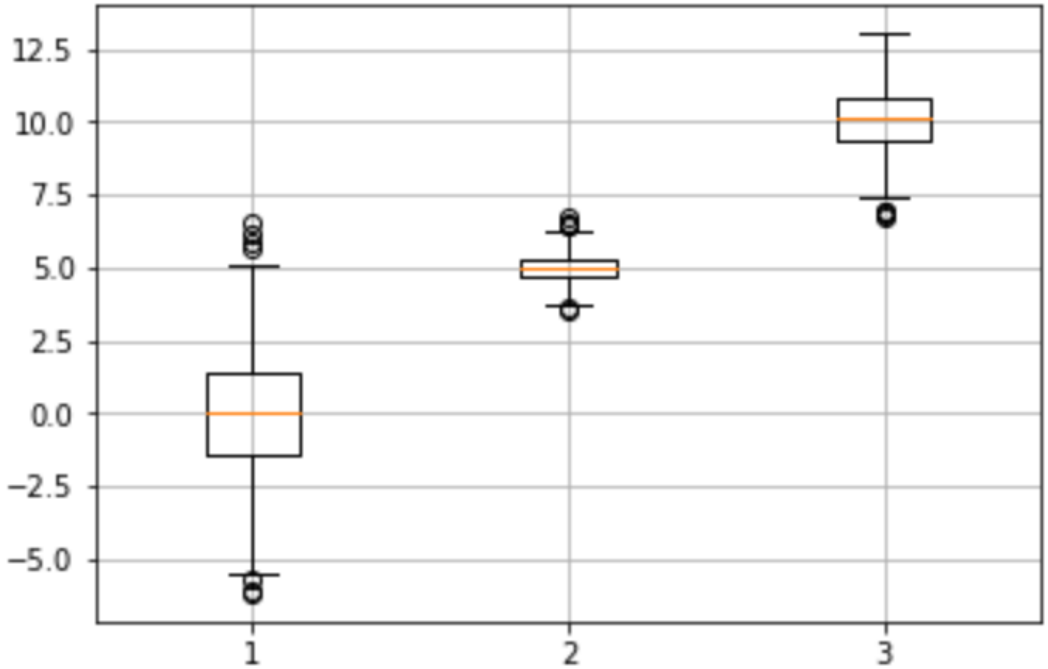
- 가우스분포를 가지는 랜덤변수 활용



히스토그램 & 박스플롯(boxplot)

- 가우스분포를 가지는 랜덤변수 활용

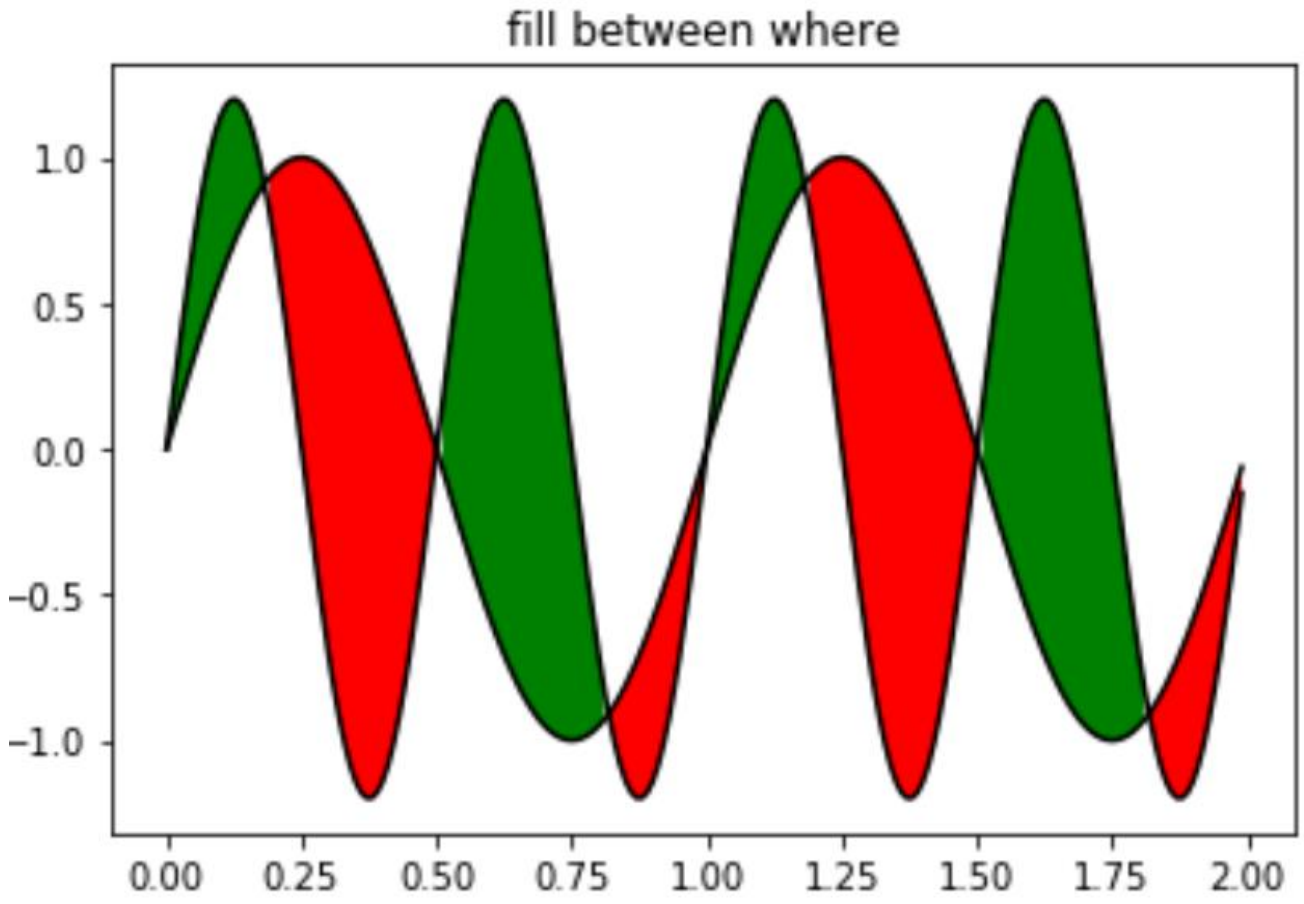
```
plt.figure()
plt.boxplot((s1, s2, s3))
plt.grid()
plt.show()
```



example_1

```
# https://matplotlib.org/gallery/pyplots/whats_new_98_4_fill_between.html#sphx-glr-gallery-pyplots-whats-new-98-4-fill-between-py
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(0.0, 2, 0.01)
y1 = np.sin(2*np.pi*x)
y2 = 1.2*np.sin(4*np.pi*x)

fig = plt.figure()
ax = fig.subplots()
ax.plot(x, y1, x, y2, color = 'black')
ax.fill_between(x, y1, y2, where = y2 > y1, facecolor = 'green')
ax.fill_between(x, y1, y2, where = y2 <= y1, facecolor = 'red')
ax.set_title('fill between where')

plt.show()
```

example_1


fill between where
1.0
0.5
0.0
−0.5
−1.0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

example_1

```
x[:10]
```

```
array([0.  , 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09])
```

```
y1[:10]
```

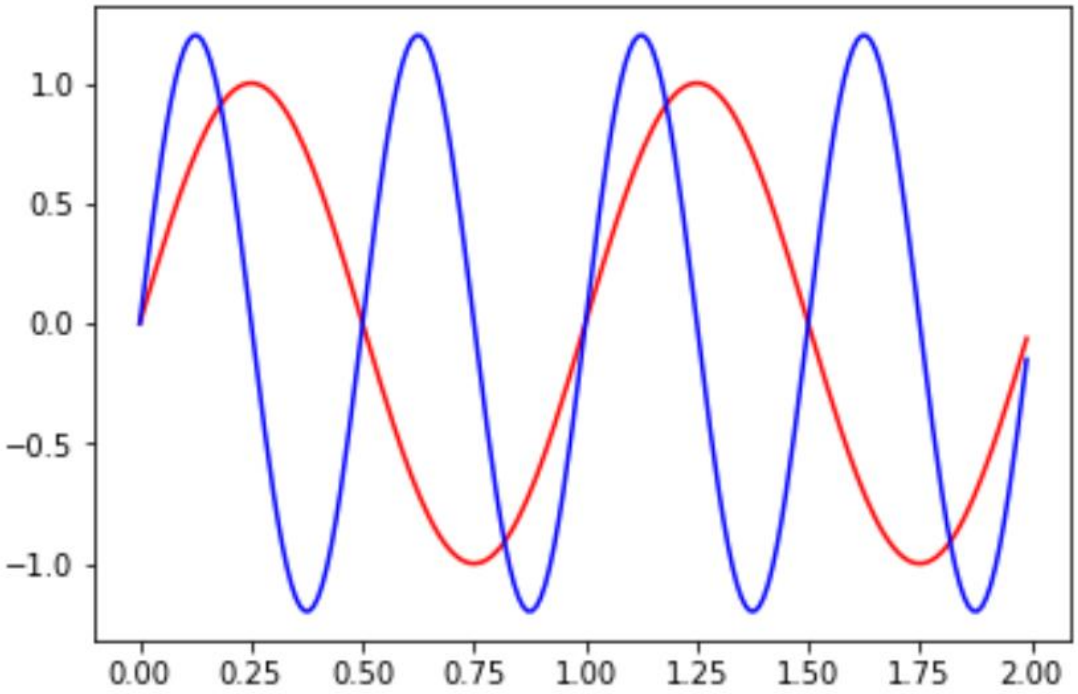
```
array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929, 0.48175367,
       0.53582679])
```

```
y2[:10]
```

```
array([0.        , 0.06279052, 0.12533323, 0.18738131, 0.24868989,
       0.30901699, 0.36812455, 0.42577929, 0.48175367,
       0.53582679])
```

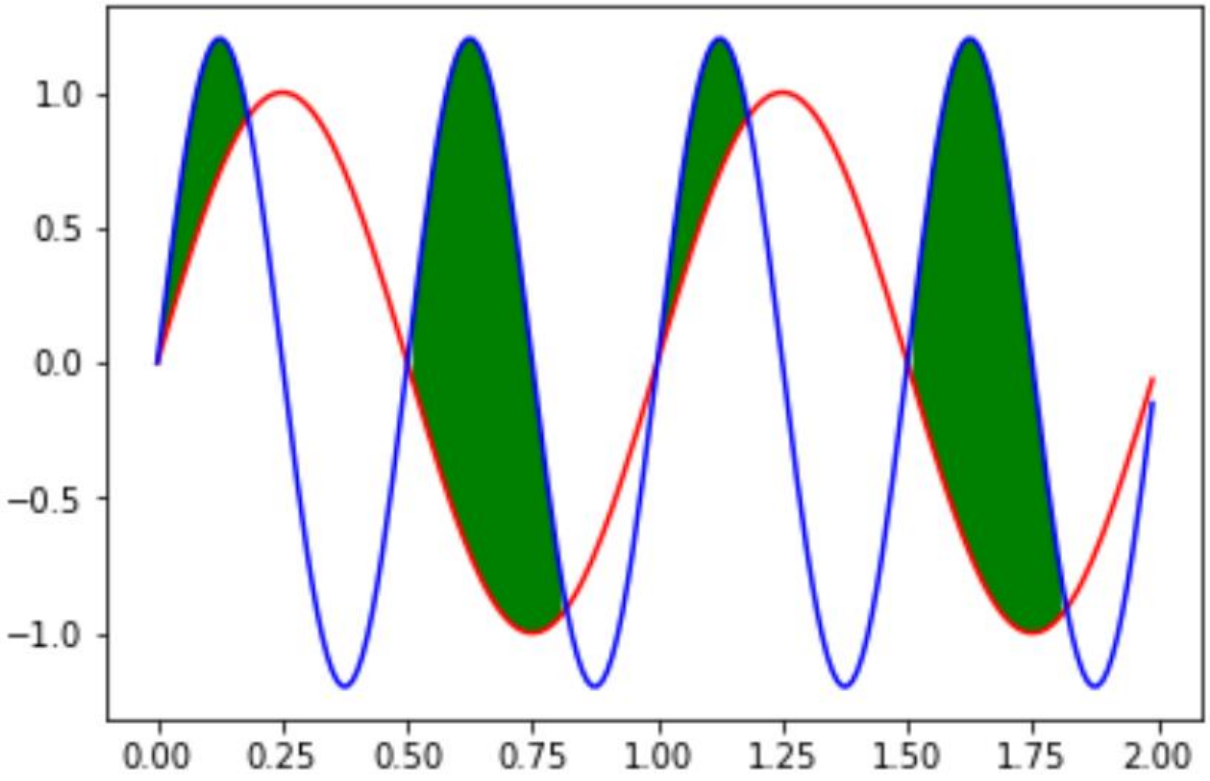
example_1

```
plt.figure()
plt.plot(x, y1, 'r', x, y2, 'b')
plt.show()
```



example_1

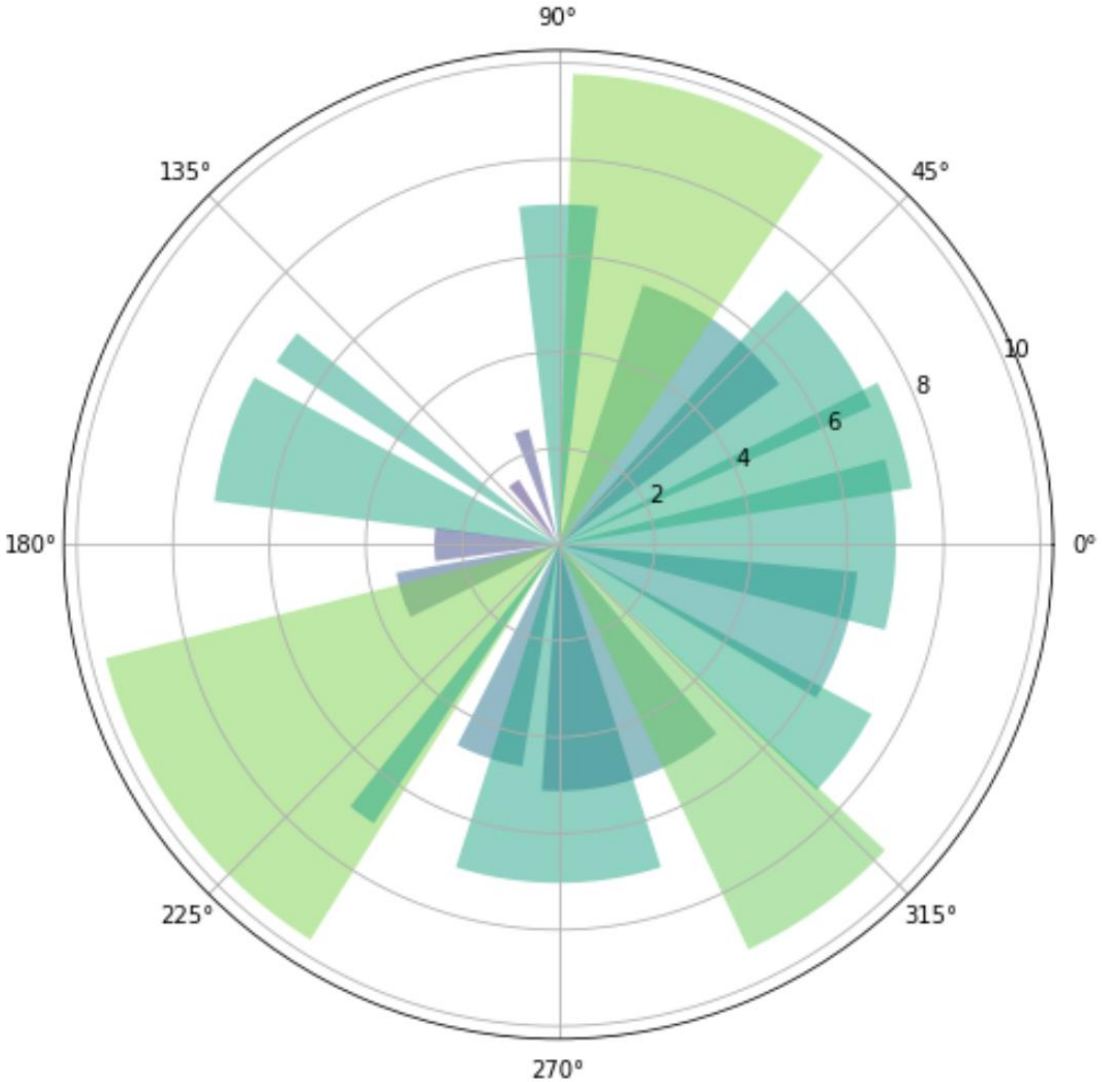
```
fig
plt.plot(x, y1, 'r', x, y2, 'b')
plt.fill_between(x, y1, y2, where = y2 > y1, facecolor = 'green')
plt.show()
```



example_2

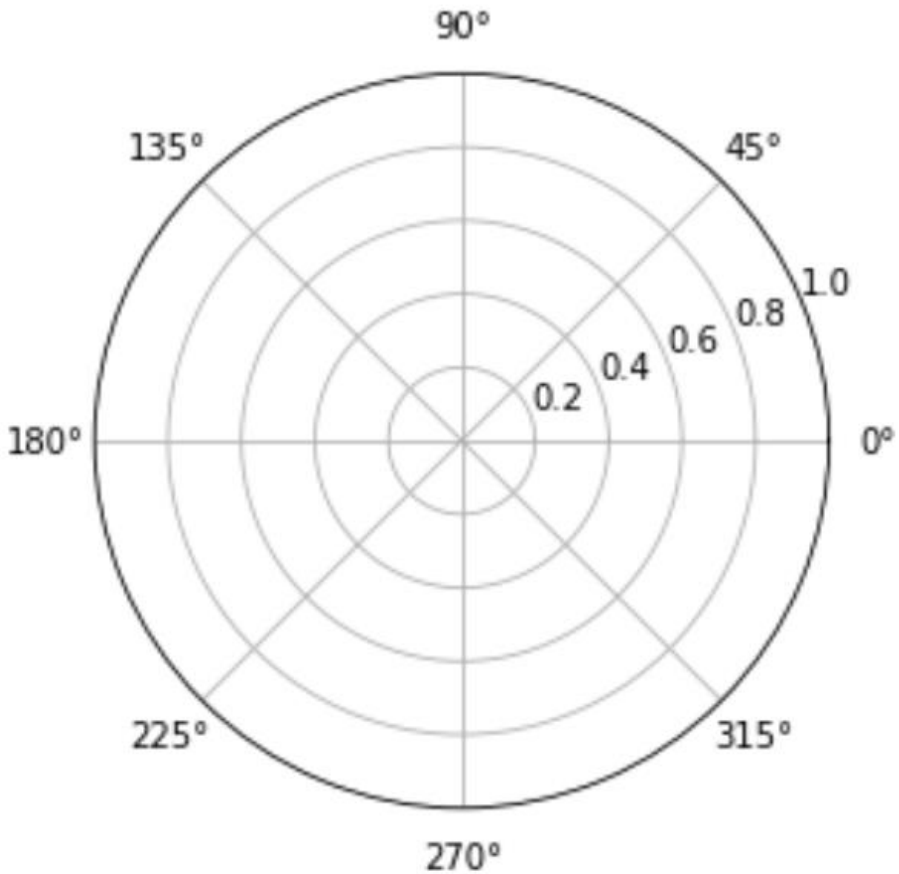
```
# Fixing random state for reproducibility
np.random.seed(19680801)

# Compute pie slices
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint = False)  # 0~20 사이를 2*pi로 균등하게 나눈 값
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)

plt. figure(figsize = (10, 8))

ax = plt.subplot(111, projection = 'polar')
bars = ax.bar(theta, radii, width = width, bottom = 0.0)

# Use custom colors and opacity
for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.viridis(r / 12 ))
    bar.set_alpha(0.5)

plt.show()
```

example_2



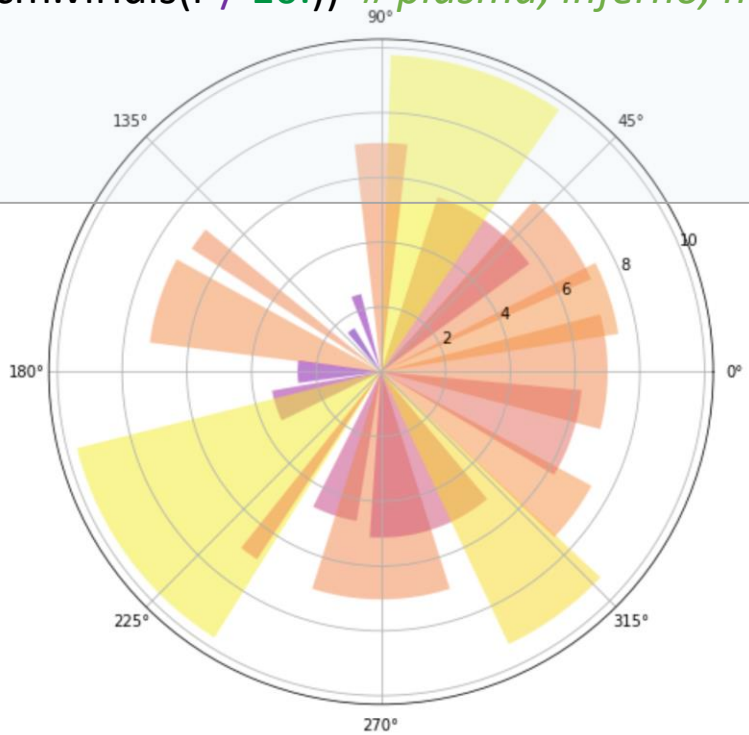
example_2

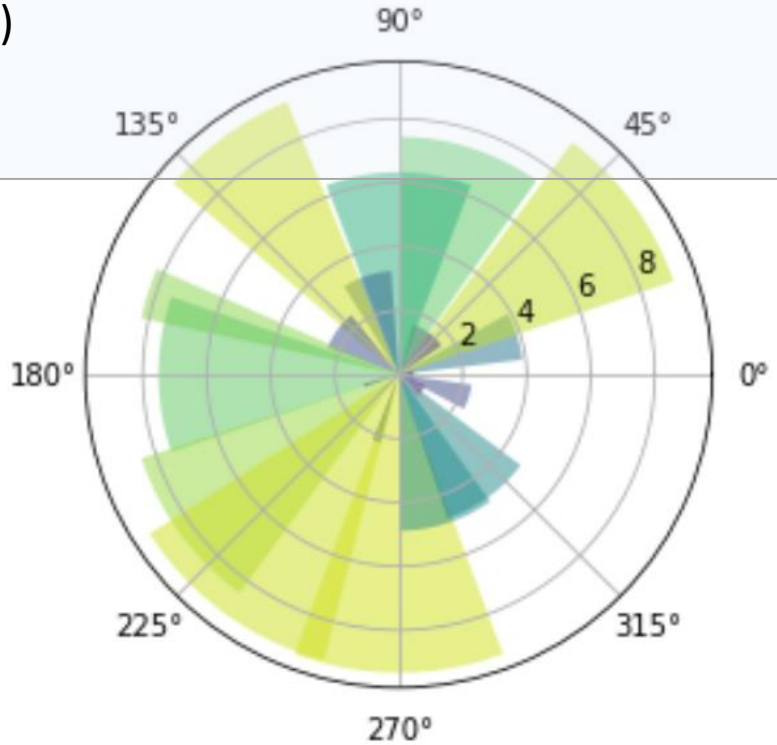
```
plt.subplot(111, projection = 'polar')
plt.show()
```



example_2

```
plt.figure(figsize = (10, 8))
ax = plt.subplot(111, projection = 'polar')
bars = plt.subplot(111, projection = 'polar').bar(theta, radii, width = width, bottom = 0.0)

for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.viridis(r / 10.))  # plasma, inferno, magma
    bar.set_alpha(0.5)

plt.show()
```



example_2

```
ax = plt.subplot(111, projection = 'polar')
bars = ax.bar(theta, radii, width = width, bottom = 0.0)

for r, bar in zip(radii, bars):
    bar.set_facecolor(plt.cm.plasma(r / 10.))  # plasma, inferno, magma
    bar.set_alpha(0.5)

plt.show()
```