SUB_PJT_01 (AI)

서울 1반 6팀 이민아

- **사전학습**
  - 사전학습1 인공지능
  - 사전학습2 회귀 및 경사하강법
  - 사전학습3 신경망
  - 사전학습4 파이썬 라이브러리
    - Numpy
    - Matplotlib
    - Tensorflow
    - Keras
    - Argparse
    - Pandas

사전학습4 파이썬 라이브러리

1. Numpy

(1) 정의

- 벡터, 행렬 등 수치 연산을 수행하는 선형대수(Linear algebra) 라이브러리
- 선형대수 관련 수치 연산을 지원하고 내부적으로는 C로 구현되어 있어 연산이 빠른 속도로 수행
- Scientific Computing 을 파이썬에서 가능하게 하기 위해 만들어진 Python 패키지
- 머신 러닝 계산의 기반이 되는 N-Dimensional Array 계산을 쉽고 빠르게 가능하게 하며, Linear Algebra, Fourier Transform, Random Number Capabilities 에 대한 함수를 제공

(2) install

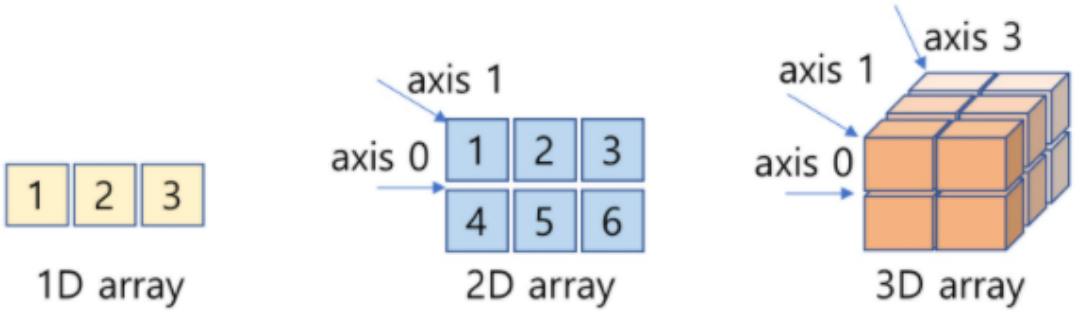
```
pip install numpy
```

(3) import

```
import numpy as np
import numpy.random as npr
```

(4) ndarray

- 배열 : `ndarray`
  - 동차(Homogeneous) 다차원 배열
  - Numpy.array와 Python.array는 다릅니다
- 차원 : `dimension`




```
a = np.arange(6) # 1D
print(a)
# [0 1 2 3 4 5]

b = np.arange(12).reshape(4,3) # 2D
print(b)
# [[ 0  1  2]
#  [ 3  4  5]
#  [ 6  7  8]
#  [ 9 10 11]]

c = np.arange(24).reshape(2,3,4) # 3D 배열은 2차원이 N개 출력되는 형식
print(c)
# [[[ 0  1  2  3]
#   [ 4  5  6  7]
#   [ 8  9 10 11]]
#
#  [[12 13 14 15]
#   [16 17 18 19]
#   [20 21 22 23]]]
```

- 축 : `axis`

```
#1 1개의 축
[1, 2, 1] # 요소(Element) = 길이(Length) = 3

#2 2개의 축
[[ 1, 0, 0], # 1번째 축(행/x축/가로축)은 길이(개수)가 2
 [ 0, 1, 2]] # 2번째 축(열/y축/세로축)은 길이(개수)가 3
```

- 출력 : `pprint`

```
def pprint(arr):
    print("type:{}".format(type(arr)))
    print("shape: {}, dimension: {}, dtype:{}".format(arr.shape, arr.ndim,
arr.dtype))
    print("Array's Data:\n", arr)

arr = np.array([[[1,2,3], [4,5,6]], [[3,2,1], [4,5,6]]], dtype = float)
a= np.array(arr, dtype = float)

pprint(a)

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

```
shape: (2, 2, 3), dimension: 3, dtype:float64
Array's Data:
[[[ 1.  2.  3.]
[ 4.  5.  6.]]

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

(5) 배열 생성

- **np.array()** : 배열 생성 (반드시 `[]` 연속된 배열로 입력)

```
a = np.array([2,3,4])
print(a) # [2 3 4]
print(a.dtype) # int64

b = np.array([1.2, 3.5, 5.1])
print(b.dtype) # float64

c = np.array( [ [1,2], [3,4] ], dtype = complex) # 복소수
print(c)
# [[1.+0.j 2.+0.j]
#  [3.+0.j 4.+0.j]]

a = np.array([1,2,3,4])  # RIGHT
print(a)

a = np.array(1,2,3,4)    # WRONG
print(a)
# ValueError: only 2 non-keyword arguments accepted
```

- np.zeros(shape) : 0으로 구성된 N차원 배열 생성
- np.ones(shape) : 1로 구성된 N차원 배열 생성
- np.eye(N) : (N, N) shape 단위 행렬 생성
- np.empty(shape) : 초기화되지 않은 N차원 배열 생성
- np.full(shape, , fill_value) : fill_value로 구성된 N차원 배열 생성
    -

```
#[3,4] 크기의 배열을 생성하여 0으로 채움
print(np.zeros((3,4)))
# [[0. 0. 0. 0.]
#  [0. 0. 0. 0.]
#  [0. 0. 0. 0.]]

# [2,3,4] 크기의 배열을 생성하여 1로 채움
print(np.ones((2,3,4), dtype=np.int16))
# [[[1 1 1 1]
#   [1 1 1 1]
#   [1 1 1 1]]

#  [[1 1 1 1]
#   [1 1 1 1]
#   [1 1 1 1]]]
```

```
# [2,2] 크기의 배열을 생성하여 7으로 채움
print(np.full((2,2),7))
# [[7 7]
# [7 7]]

np.eye(4)
'''
array([[ 1.,  0.,  0.,  0.],
     [ 0.,  1.,  0.,  0.],
     [ 0.,  0.,  1.,  0.],
     [ 0.,  0.,  0.,  1.]])
'''
```

- np.arange() : N 만큼 차이나는 숫자 생성
- np.linspace() : N 등분한 숫자 생성
- np.logspace() : 로그 스케일로 지정된 범위에서 num개수만큼 균등 간격 숫자 생성

```
# 10이상 30미만 까지 5씩 차이나게 생성
print(np.arange(10, 30, 5, np.float))
# [10. 15. 20. 25.]

# 0이상 2미만 까지 0.3씩 차이나게 생성
print(np.arange(0, 2, 0.3))
# [0.  0.3 0.6 0.9 1.2 1.5 1.8]

# 0~99까지 100등분(100개)
x = np.linspace(0, 99, 100)
print(x)
# [ 0.  1.  2.  3. ... 98. 99.]

# linspace의 데이터 추출 시각화
import matplotlib.pyplot as plt
plt.plot(x, 'o')
plt.show()
```

(7) 배열 속성

- **np.ndarray.reshape()** : 데이터는 그대로 유지한 채 **차원을 쉽게 변경**
- **ndarray.shape** : 배열의 각 **축(axis)의 크기**
- ndarray.ndim : 축의 개수(Dimension)
- ndarray.dtype : 각 요소(Element)의 타입
- ndarray.itemsize : 각 요소(Element)의 타입의 bytes 크기
- **ndarray.size** : 전체 **요소(Element)의 개수**

```
#  (3, 5) 크기의 2D 배열을 생성
a = np.arange(15).reshape(3, 5)

print(a)
# [[ 0  1  2  3  4]
#  [ 5  6  7  8  9]
#  [10 11 12 13 14]]
```

```
print(a.shape) # 각 축의 크기 (행/x축/가로축, 열/y축/세로축)
# (3, 5)

print(a.ndim) # 축의 개수 (배열의 차원 수)
# 2

print(a.dtype) # 각 요소(Element)의 타입
# int64

print(a.itemsize) # 각 요소(Element)의 타입의 bytes 크기
# 8

print(a.size) # 전체 요소의 개수 np.arange(15)
# 15

print(a.dtype.name) # 배열 타입명

print(type(a))
# <class 'numpy.ndarray'>
```

(8) 연산

- element wise 연산 : 숫자가 각각의 요소에 연산이 적용

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
print(b)
# [0 1 2 3]

# a에서 b에 각각의 원소를 -연산
c = a-b
print(c)
# [20 29 38 47]

# b 각각의 원소에 제곱 연산
print(b**2)
# [0 1 4 9]

# a 각각의 원소에 *10 연산
print(10*np.sin(a))
# [ 9.12945251 -9.88031624  7.4511316  -2.62374854]

# a 각각의 원소가 35보다 작은지 Boolean 결과
print(a<35)
# [ True  True False False]
```

- 행렬 연산
  - `*` : 각각의 원소끼리 곱셈 (Elementwise product, Hadamard product)
  - `@` : 행렬 곱셈 (Matrix product)
  - `.dot()` : 행렬 내적 (dot product)

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

```
print(A * B) # * 각각의 원소 곱셈 (Elementwise product, Hadamard product)
# [[2 0]
#  [0 4]]

print(A @ B) # @ 행렬 곱셈 (Matrix product)
# [[5 4]
#  [3 4]]

print(A.dot(B)) # .dot() 행렬 내적 (dot product)
# [[5 4]
#  [3 4]]
```

- 최대 및 최소 : **axis 값을 입력**하면 축을 기준으로도 연산 가능
  - `.sum()` : 모든 요소의 합
  - `.min()` : 모든 요소 중 최소값
  - `.max()` : 모든 요소 중 최대값
  - `.argmax()` : 모든 요소 중 최대값의 인덱스
  - `.cumsum()` : 모든 요소의 누적합

```
a = np.arange(8).reshape(2, 4)**2
print(a)
# [[ 0  1  4  9]
#  [16 25 36 49]]

# 모든 요소의 합
print(a.sum()) # 140

# 모든 요소 중 최소값
print(a.min()) # 0

# 모든 요소 중 최대값
print(a.max()) # 49

# 모든 요소 중 최대값의 인덱스
print(a.argmax()) # 7

# 모든 요소의 누적합
print(a.cumsum()) # [  0   1   5  14  30  55  91 140]

b = np.arange(12).reshape(3,4)
print(b)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

print(b.sum(axis=0))
# [12 15 18 21]

print(b.sum(axis=1))
# [ 6 22 38]
```

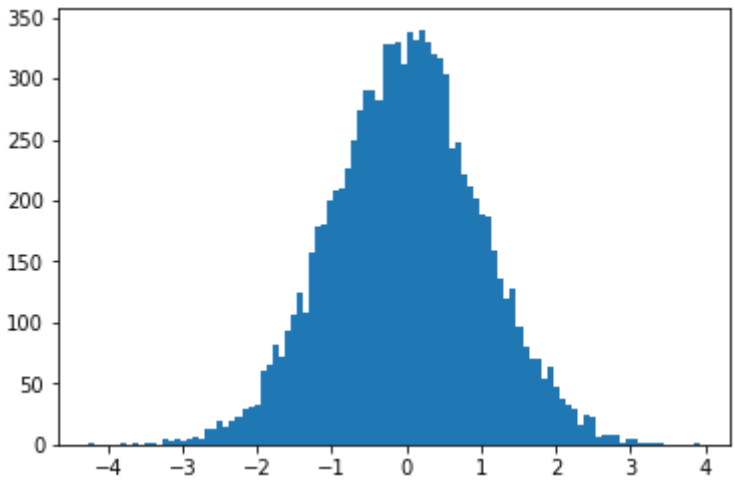
(9) 인덱싱 및 슬라이싱

- np.fromfunction() : 인덱스 번호를 가지고 함수를 정의해 생성

```
def f(x,y):
    return 10*x+y

b = np.fromfunction(f, (5,4), dtype=int)
print(b)
# [[ 0  1  2  3]
#  [10 11 12 13]
#  [20 21 22 23]
#  [30 31 32 33]
#  [40 41 42 43]]

print(b[2,3])
# 23

print(b[0:5, 1])
# [ 1 11 21 31 41]

print(b[ : ,1])
# [ 1 11 21 31 41]

print(b[1:3, : ])
# [[10 11 12 13]
#  [20 21 22 23]]

print(b[-1])
# [40 41 42 43]
```

(10) 난수 생성

- np.random.normal : 정규 분포 확률 밀도에서 표본 추출
    - normal(loc=0.0, scale=1.0, size=None)
    - loc: 정규 분포의 평균
    - scale: 표준편차



```
# 표본 10000개의 배열을 100개 구간(bins)으로 구분할 때, 정규 분포 형태
data = np.random.normal(0, 1, 10000)
import matplotlib.pyplot as plt
plt.hist(data, bins=100)
plt.show()
```

- np.random.rand : 균등 분포
- np.random.randn : 표준 정규 분포
- np.random.randint : 지정된 shape를 배열로 만들고 low부터 high 미만 범위에서 정수 표본 추출 (균등 분포)

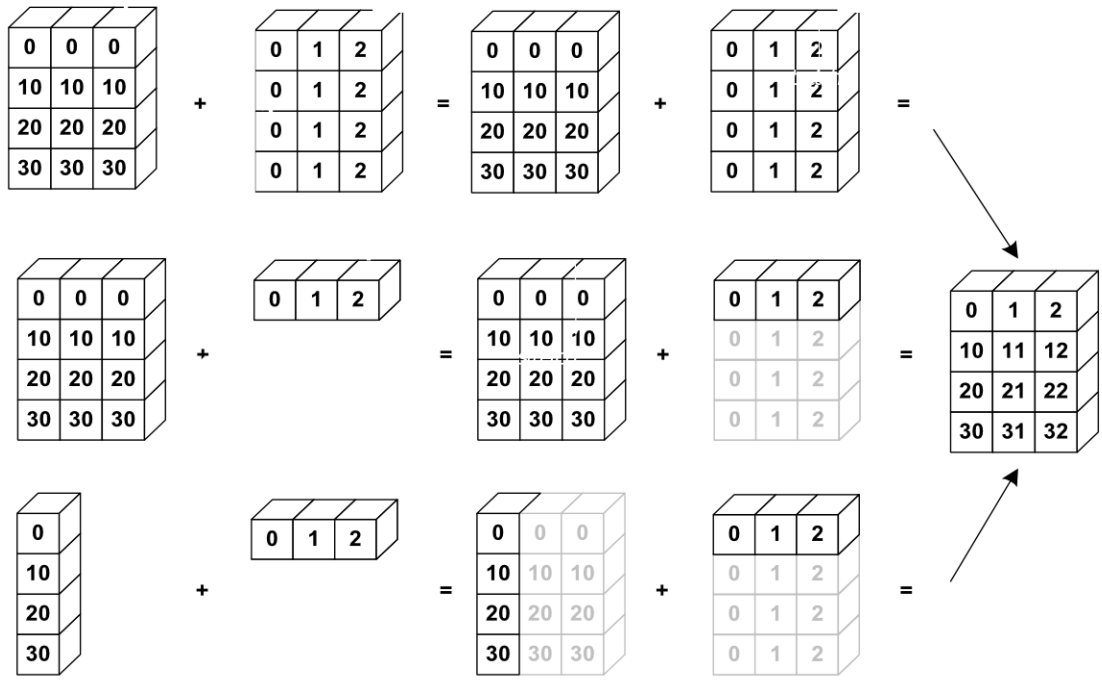
```
np.random.randint(0, 10, (2, 3))
# array([[0, 1, 7],
#       [0, 2, 3]])
```

- np.random.random : 균등 분포

```
np.random.random((2, 2))
# array([[ 0.4573231 ,  0.1649216 ],
#       [ 0.76895461,  0.96333133]])
```

(11) 브로드캐스팅

- 단순하게 편리성을 위해 **Shape가 다른** np.narray 끼리 연산을 지원
- 차원(ndim)이 같고 각 축(axis)의 값이 같거나 1이야 연산이 가능
- **각 축의 값이 다르면** 브로드캐스팅되어 값이 복사



```
np.array([1,2,3,4,5]) * 2

np.array([1,2,3,4,5]) * np.array([2,2,2,2,2])
# Numpy List : Broadcasting
# [2,4,6,8,10]

# Python List : 값이 10개인 배열이 생성
# [1,2,3,4,5,1,2,3,4,5]
```

2. Matplotlib

(1) 정의

- 다양한 형태의 그래프를 Matplotlib 을 사용하여 손쉽게 그리며 **데이터 시각화**를 위해 사용되는 파이썬 라이브러리
- 통계 기능과 색 테마를 지원하는 **Seaborn 라이브러리**를 함께 활용하면 더욱 간편하게 그래프 표현 가능

(2) install

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

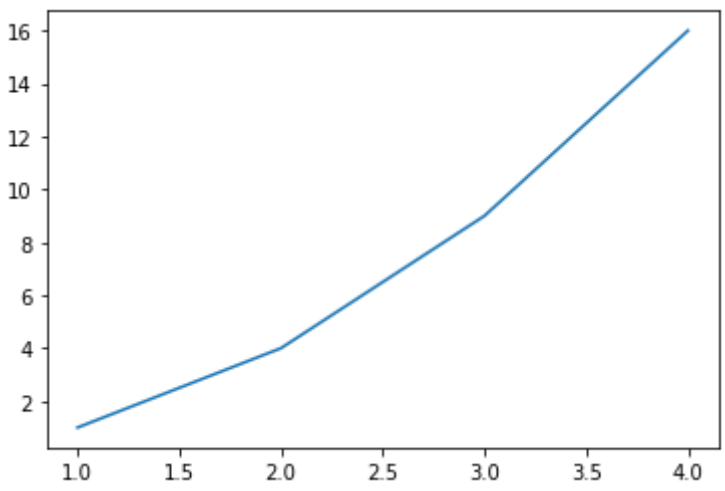
(3) import

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

(4) pyplot

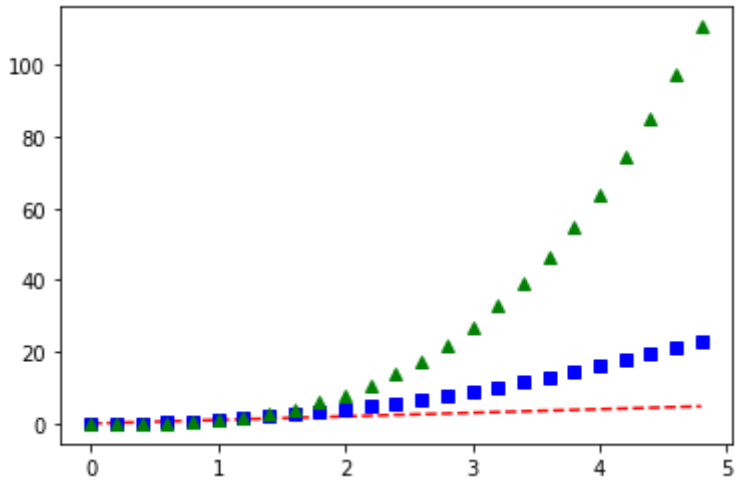
- 그래프 `plt`

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



- 복수 그래프 `plt`

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

# 200ms 간격으로 균일하게 샘플된 시간
t = np.arange(0., 5., 0.2)

# 빨간 대쉬(r--), 파란 사각형(bs), 녹색 삼각형(g^)
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



(5) input

- **y값 입력**

```
import matplotlib.pyplot as plt

plt.plot([2, 3, 5, 10])
#1 리스트 형태로 값들을 입력하면 y 값으로 인식하며 x 값은 기본적으로 [0, 1, 2, 3]
#2 파이썬 튜플 plot((2, 3, 5, 10))
#3 Numpy 어레이 plot(np.array([2, 3, 5, 10])
plt.show()
```

- **x값 및 y값 입력**

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10])
# plt.plot([x_value_list], [y_value_list)
# 점 (1, 2), (2, 3), (3, 5), (4, 10)를 잇는 꺾은선 그래프
plt.show()
```

- dictionary값 입력

```
import matplotlib.pyplot as plt

data_dict = {'data_x': [1, 2, 3, 4, 5], 'data_y': [2, 3, 5, 10, 8]}

plt.plot('data_x', 'data_y', data=data_dict)
plt.show()
```

(6) axis

- **축 라벨 함수**
  - plt.xlabel()
  - plt.ylabel()

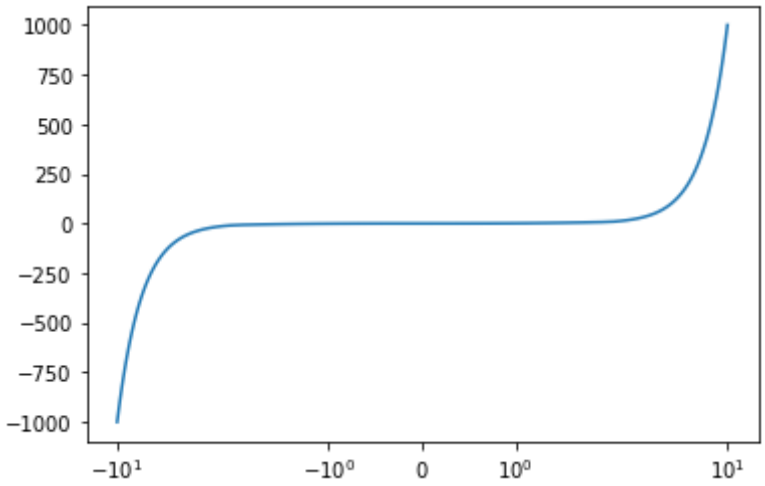
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.show()
```

- **축 라벨 함수 파라미터**
  - 여백 : labelpad
  - 폰트: fontdict
  - 위치 : loc
    - xlabel() : 'left', 'center', 'right'
    - ylabel() : 'bottom', 'center', 'top'

```
import matplotlib.pyplot as plt

font1 = {'family': 'serif',
         'color': 'b',
         'weight': 'bold',
         'size': 14
         }

font2 = {'family': 'fantasy',
         'color': 'deeppink',
         'weight': 'normal',
         'size': 'xx-large'
         }

plt.plot([1, 2, 3, 4], [2, 3, 5, 10])
plt.xlabel('X-Axis', labelpad=15, fontdict=font1, loc='right')
plt.ylabel('Y-Axis', labelpad=20, fontdict=font2, loc='top')
# X축 레이블에 대해서 15pt, Y축 레이블에 대해서 20pt 만큼의 여백
# xlabel() 함수의 loc 파라미터는 X축 레이블의 위치를 지정합니다. ({'left',
'center', 'right'})
# ylabel() 함수의 loc 파라미터는 Y축 레이블의 위치를 지정합니다. ({'bottom',
'center', 'top'})
plt.show()
```

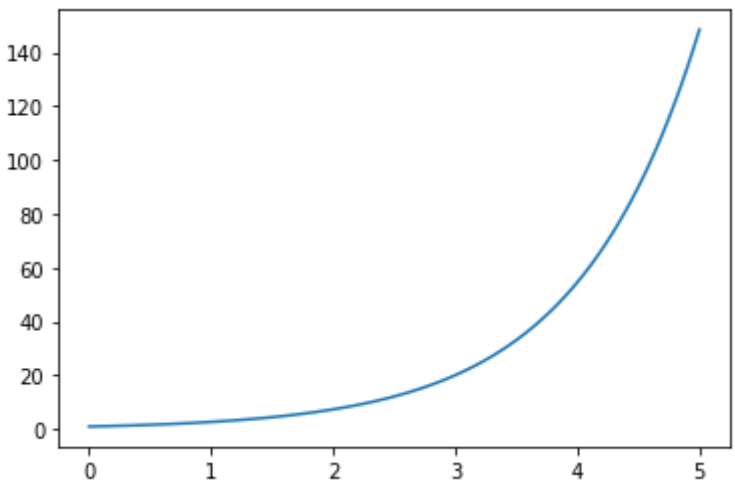
- **축 스케일 함수**
  - plt.xscale()
    - symlog : 양, 음의 방향이 대칭적인 로그 스케일

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

x = np.linspace(-10, 10, 100)
y = x ** 3

plt.plot(x, y)
plt.xscale('symlog')
# Symmetrical Log Scale
# 양, 음의 방향이 대칭적인 로그 스케일

plt.show()
```



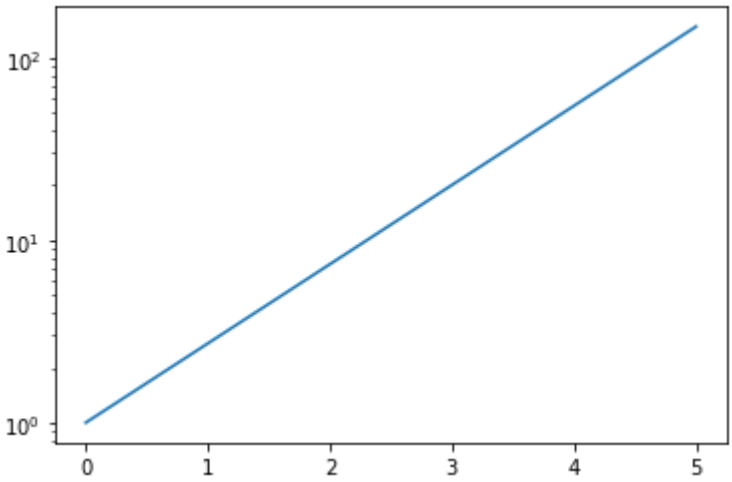
- plt.yscale()
  - linear

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

x = np.linspace(0, 5, 100)
y = np.exp(x) # 지수함수 (Exponential function)

plt.plot(x, y)
plt.yscale('linear')
plt.show()
```



- log : 로그 스케일은 지수함수 (Exponential function)와 같이 기하급수적으로 변화하는 데이터

```
plt.yscale('log')
```



- **축 범위 함수**
  - plt.xlim([xmin, xmax])
  - plt.ylim([ymin, ymax])
  - plt.axis([xmin, xmax, ymin, ymax])

```
plt.xlim([0, 5])       # X축의 범위: [xmin, xmax]
plt.ylim([0, 20])      # Y축의 범위: [ymin, ymax]
plt.axis([0, 5, 0, 20])  # X, Y축의 범위: [xmin, xmax, ymin, ymax]
```

- 축 함수에 따른 그래프 모양
  - 'on' | 'off' | 'equal' | 'scaled' | 'tight' | 'auto' | 'normal' | 'image' | 'square'

```
plt.axis('square') # 정사각형
plt.axis('scaled') # 축 범위 맞춤형
```

(7) style

Colors

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Styles

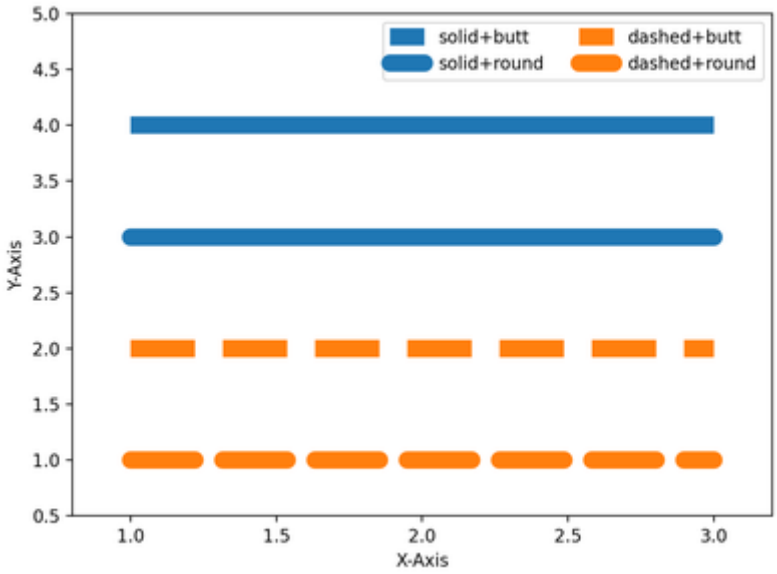
character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

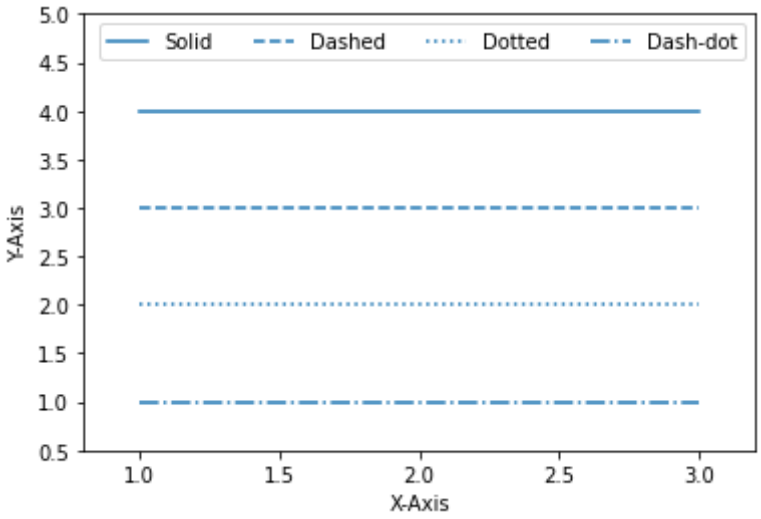
Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

- **선**
  - **선 모양(linestyle)**
    - ' - ' (Solid)
    - ' - - ' (Dashed)
    - ' : ' (Dotted)
    - ' -. ' (Dash-dot)
  - **선 끝 모양(solid_capstyle, dash_capstyle)**
    - 'butt' 뭉뚝한
    - 'round' 둥근 끝 모양

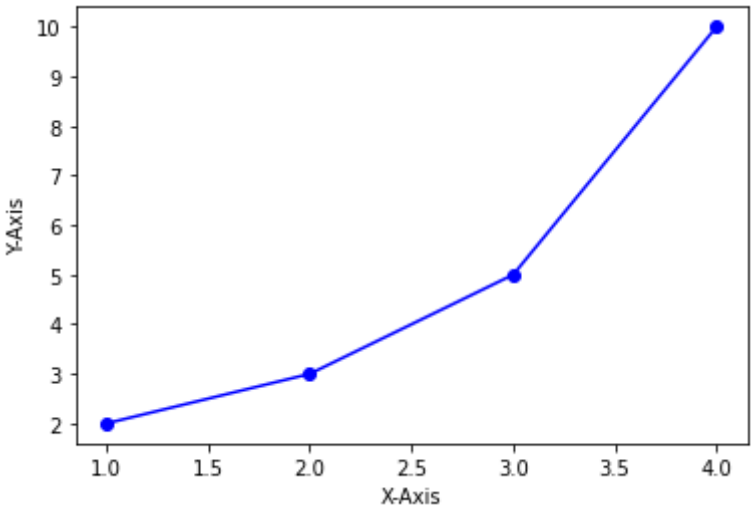
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3], [4, 4, 4], linestyle='solid', linewidth=10,
      solid_capstyle='butt', color='C0', label='solid+butt')
plt.plot([1, 2, 3], [3, 3, 3], linestyle='solid', linewidth=10,
      solid_capstyle='round', color='C0', label='solid+round')

plt.plot([1, 2, 3], [2, 2, 2], linestyle='dashed', linewidth=10,
      dash_capstyle='butt', color='C1', label='dashed+butt')
plt.plot([1, 2, 3], [1, 1, 1], linestyle='dashed', linewidth=10,
      dash_capstyle='round', color='C1', label='dashed+round')


plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.axis([0.8, 3.2, 0.5, 5.0])
plt.legend(loc='upper right', ncol=2)
plt.tight_layout()
plt.show()
```





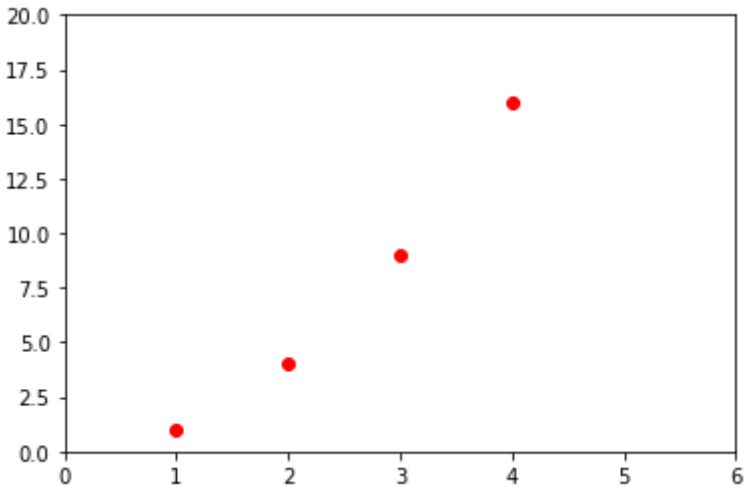
- **마커**

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], 'bo--')      # 파란색 + 마커 + 점선
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```



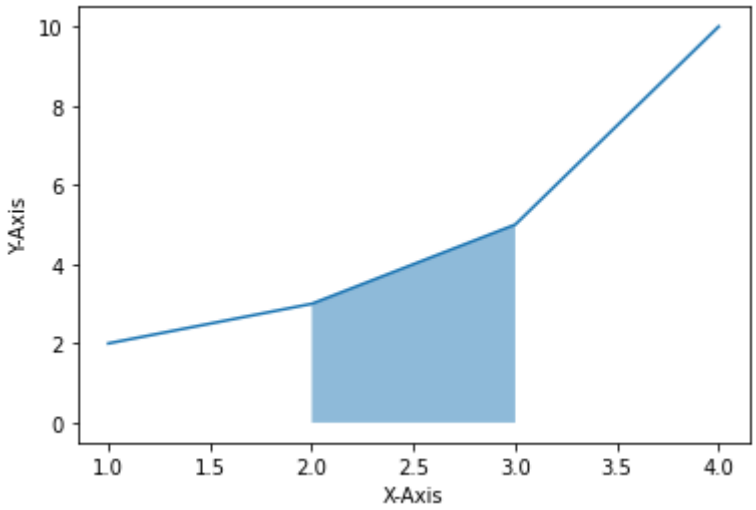
- **색상**

```
import matplotlib.pyplot as plt

# 포맷 문자열 'ro'는 빨간색 ('red')의 원형 ('o')
# 포맷 문자열 'b-'는 파란색 ('blue')의 실선 ('-')
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
# axis() 함수를 이용해서 축의 범위 [xmin, xmax, ymin, ymax]를 지정
plt.axis([0, 6, 0, 20])
plt.show()
```



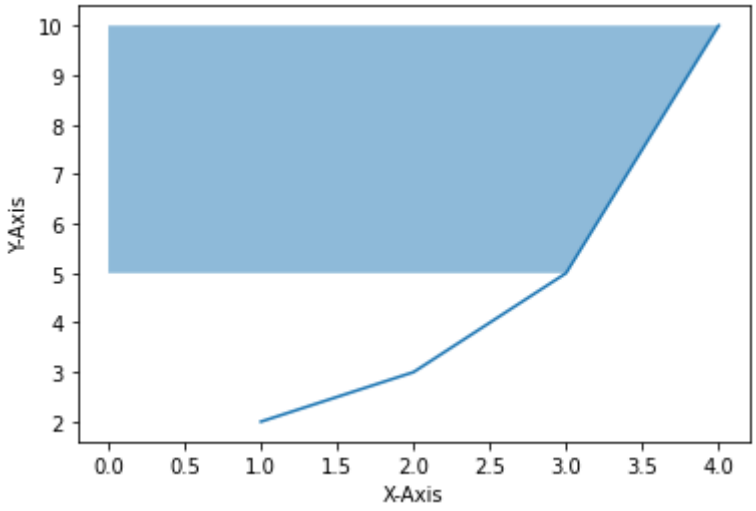
- **채우기**
  - fill_between()
    - 두 수평 방향의 곡선 사이를 채웁니다.
    - 두 개의 그래프 사이 영역을 채우기 위해서 두 개의 y 값의 리스트 y1, y2를 입력

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [2, 3, 5, 10]

plt.plot(x, y)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.fill_between(x[1:3], y[1:3], alpha=0.5)
# fill_between() 함수에 x[1:3], y[1:3]를 순서대로 입력하면
# 네 점 (x[1], y[1]), (x[2], y[2]), (x[1], 0), (x[2], 0)을 잇는 영역이 채워집니다
plt.show()
```



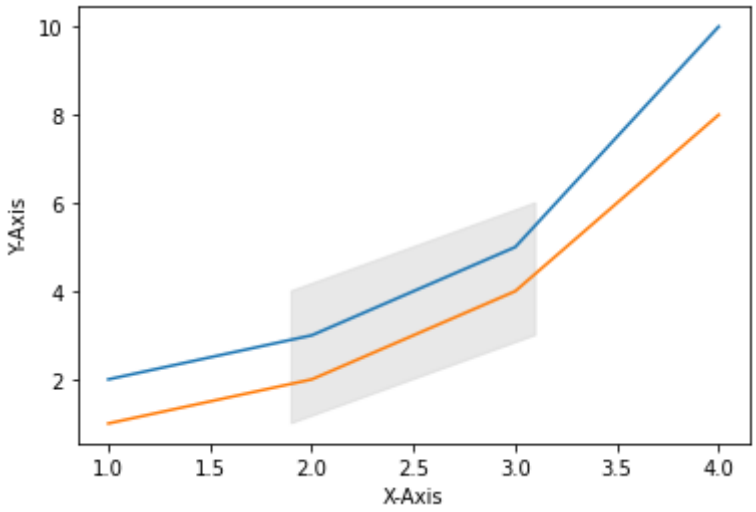
- fill_betweenx()
  - 두 수직 방방의 곡선 사이를 채웁니다.

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [2, 3, 5, 10]

plt.plot(x, y)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.fill_betweenx(y[2:4], x[2:4], alpha=0.5)
# fill_betweenx() 함수에 y[2:4], x[2:4]를 순서대로 입력하면
# 네 점 (x[2], y[2]), (x[3], y[3]), (0, y[2]), (0, y[3])을 잇는 영역이 채워집니다
plt.show()
```



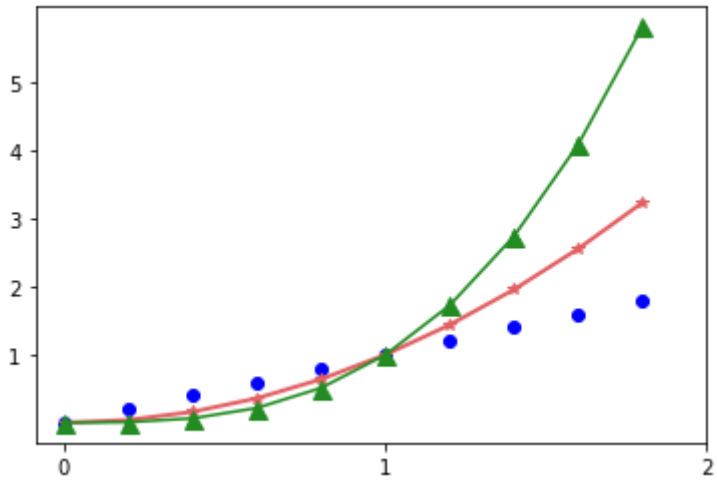
- fill()
  - 각 x, y 점들로 정의되는 다각형 영역을 자유롭게 지정해서 채울 수 있습니다

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y1 = [2, 3, 5, 10]
y2 = [1, 2, 4, 8]

plt.plot(x, y1)
plt.plot(x, y2)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.fill([1.9, 1.9, 3.1, 3.1], [1.0, 4.0, 6.0, 3.0], color='lightgray',
alpha=0.5)

plt.show()
```



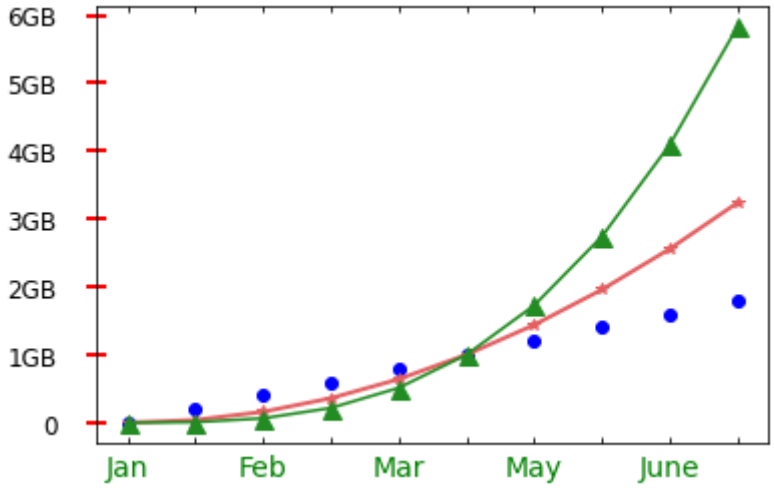
- **눈금**
  - plt.xticks
  - plt.yticks
    - labels 파라미터를 사용하면 눈금 레이블을 문자열의 형태로 지정

```
import numpy as np

x = np.arange(0, 2, 0.2)

plt.plot(x, x, 'bo')
plt.plot(x, x**2, color='#e35f62', marker='*', linewidth=2)
plt.plot(x, x**3, color='forestgreen', marker='^', markersize=9)
plt.xticks([0, 1, 2])
plt.yticks(np.arange(1, 6))
# X축의 [0, 1, 2]의 위치, Y축의 [1, 2, 3, 4, 5]의 위치에 눈금
# plt.xticks(np.arange(0, 2, 0.2), labels=['Jan', '', 'Feb', '', 'Mar',
'', 'May', '', 'June', '', 'July'])
# plt.yticks(np.arange(0, 7), ('0', '1GB', '2GB', '3GB', '4GB', '5GB',
'6GB'))


plt.show()
```



- tick_params()
  - axis : 설정이 적용될 축
  - direction : 'in', 'out', 'inout'으로 설정하면 눈금이 안/밖으로 표시
  - length : 눈금의 길이
  - labelsize : 레이블의 크기
  - labelcolor : 레이블의 색상
  - width : 눈금의 너비
  - color : 눈금의 색상

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

x = np.arange(0, 2, 0.2)

plt.plot(x, x, 'bo')
plt.plot(x, x**2, color='#e35f62', marker='*', linewidth=2)
plt.plot(x, x**3, color='springgreen', marker='^', markersize=9)

# xticks 및 yticks
plt.xticks(np.arange(0, 2, 0.2), labels=['Jan', '', 'Feb', '', 'Mar',
'', 'May', '', 'June', '', 'July'])
plt.yticks(np.arange(0, 7), ('0', '1GB', '2GB', '3GB', '4GB', '5GB',
'6GB'))

# tick_params
plt.tick_params(axis='x', direction='in', length=3, pad=6, labelsize=14,
labelcolor='green', top=True)
plt.tick_params(axis='y', direction='inout', length=10, pad=15,
labelsize=12, width=2, color='r')

plt.show()
```



3. Tensorflow

(1) 정의

- 구글에서 제공하는 머신 러닝을 위한 End-to-End 오픈소스 플랫폼
- 연산과정을 그래프의 형태로 정의하고 그 후에 그래프에 값을 전달하는 방식으로 동작
- Tensorboard 라는 시각화 툴을 제공하여 학습 과정 및 연산 그래프, 결과를 확인하기 쉽게 되어 있다

(2) intsall

```
!pip install -q tensorflow-gpu==2.0.0-rc1
```

(3) import

```
import tensorflow as tf
```

(4) 데이터셋 로드

- 샘플 값을 정수에서 **부동소수로 변환**

```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

(5) Sequential Model

- 훈련에 사용할 **옵티마이저(optimizer)**와 **손실 함수**를 선택

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

(6) Custom Model

- **Model**

  훈련 및 추론 기능을 사용하여 **레이어**를 **객체로 그룹화**

  내장 **훈련, 평가** 및 **예측** 루프(`model.fit()` , `model.evaluate()` , `model.predict()`)를 제공

- **Sequential Model**

  특별한 function, class 정의가 **필요없이** 바로 instance 생성

- **Custom model**

  class로 상속받아 새로 정의한 후 instance를 생성

  `tf.keras.Model` function

  - 정의 (`init`)

    단위 : `units`

    인스턴스화 : `tf.keras.layers.Dense(units=self.units)`

  - 호출 (`call`)

(7) Model Method (compile / fit / predict)

- `model.compile(loss, optimizer, metrics)`
  - **loss** : 손실 함수 (MSE)
    - MeanSquaredError()
    - KLDivergence()
    - CosineSimilarity()
  - **optimizer** : 최적화 함수 (SGD)
    - SGD() (with or without momentum)
    - RMSprop()
    - Adam()
  - metrics : 학습 및 테스트 중에 모델에서 평가할 메트릭 목록 (학습 history에 반환할 값을 설정)
    - AUC()
    - Precision()
    - Recall()
- `model.fit(x, y, epochs, batch_size)`
  - x : 입력 데이터 (x축 데이터)
  - y : 타겟 데이터 (y축 데이터)
  - epochs 데이터 전체에 대한 학습 반복 횟수
  - batch_size : 그라디언트 업데이트 당 샘플 수 (배치의 크기)
- `model.predict()`
  - test_x : x축 데이터 (x값이 주어지면 y값 예측)
  - batch_size : 그라디언트 업데이트 당 샘플 수 (배치의 크기)

```
predict(
    test_x,
    batch_size=None,
    verbose=0,
    steps=None,
    callbacks=None,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False
)
```

- `model.summary()`

(7) 훈련 및 평가

- 훈련된 이미지 분류기는 이 데이터셋에서 약 98%의 정확도를 달성

```
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)

'''
Train on 60000 samples
60000/60000 [==============================] - 4s 68us/sample - loss: 0.2941 -
accuracy: 0.9140
Epoch 2/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.1396 -
accuracy: 0.9587
Epoch 3/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.1046 -
accuracy: 0.9680
Epoch 4/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.0859 -
accuracy: 0.9742
Epoch 5/5
60000/60000 [==============================] - 4s 62us/sample - loss: 0.0724 -
accuracy: 0.9771
10000/1 - 0s - loss: 0.0345 - accuracy: 0.9788
[0.06729823819857557, 0.9788]
'''
```

4. Keras

(1) 정의

- Tensorflow, CNTK, Theano 와 같은 Deep Learning 라이브러리 위에서 실행할 수 있는 High-level Neural Network API
- 빠르게 딥 러닝 연구 및 실험을 가능하게 하는데 중점을 두고 개발
- 개발 시간을 최소화하고 빠르게 결과를 도출할 수 있다는 장점
- 모듈성과 쉬운 확장성을 가지고 있어 현재 널리 사용

(2) intsall

```
pip install keras
```

(3) import

```
import tensorflow as tf
from keras.models import Sequential
from tensorflow import keras
from tensorflow.keras import layers
```

(4) Sequential 모델

- `.add()`를 통해 레이어를 간단하게 쌓을 수 있습니다

```
from keras.models import Sequential
from keras.layers import Dense

#1 모델 생성
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)

#2 모델 생성
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

#3 모델 생성
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))

# 모델 제거
model.pop()
print(len(model.layers))
```

(5) Model Method (compile / fit / evaluate / predict)

- `.compile()`
    - 학습 과정을 조정
    - 옵티마이저를 조정

```
# 학습 과정을 조정
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
# 옵티마이저를 조정
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9,
nesterov=True))
```

- `.fit()` 레이닝 데이터에 대한 반복작업을 수행
- `.evaluate()` 코드 한 줄로 모델의 성능을 평가
- `.predict()` 새로운 데이터에 대해서 예측 결과를 생성

```
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```

5. Argparse

(1) 정의

- 명령창(터미널)에서 실행하는 것을 원칙
- 주피터 노트북의 경우 `args = parser.parse_args()`

(2) import

```
import argparse
import os # output directory를 만드는 등의 역할
```

(3) 생성

```
#1 parser 객체를 생성
parser = argparse.ArgumentParser(description='SUB_PJT_01')

#2 add_argument() method를 통해 원하는 만큼 인자 종류를 추가
#2-1 type 지정
#2-2 default 값 지정
parser.add_argument('--caption_file_path', type=str,
default='./datasets/captions.csv')
# attribute name: -, _ 구분
# --print-number(X)의 경우 args.print_number
# --print_number(O)의 경우 args.print_number

#3 parse_args() method로 명령창에서 주어진 인자를 파싱
args = parser.parse_args()
```

6. pandas

(1) csv

- sep : 구분
- encoding : csv파일을 읽어 올 때 한글이 깨지는 현상은 대게 Encoding오류
  - `encoding='utf-8'` or `encoding='cp949'` 등 파일 인코딩에 맞춰서 해결
  - `engine='python'`
- names : column명 변경

```
import pandas as pd

pd.read_csv('경로/파일명.csv')
# Index 지정
pd.read_csv('경로/파일명.csv', index_col = '인덱스로 지정할 column명')
# header 지정
pd.read_csv('파일명.csv', header = 1)
# column명 바꿔서 불러오기
pd.read_csv('파일명.csv', names = ['Idx', 'col1', 'col2'])
```

(2) list

- `data.columns.tolist()` : 필드명 리스트
- `data.values.tolist()` : 전체 값 리스트(2차원 리스트로 출력)
- `data.칼럼명.tolist()` : 특정 칼럼만 리스트로 출력
- `data.index.tolist()` : 인덱스 -> 리스트로 출력