

MATPLOTLIB

파이썬 데이터 시각화

소개

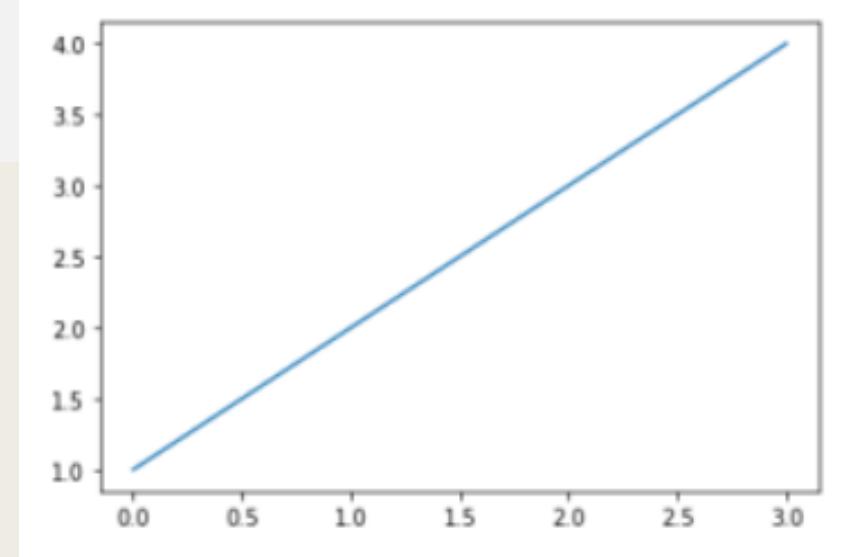
- Matplotlib 데이터 시각화 그래프에 사용 되는 파이썬 라이브러리이다.
- Matplotlib 꺍은선, 막대, 히스토그램 등 다양한 유형의 그래프를 몇 줄의 코드로 간단하게 표현할 수 있다.

설치

- COLAB, Anaconda 는 기본설치가 되어 있어 바로 사용
- 직접설치
- pip install matplotlib

Matplotlib 기본

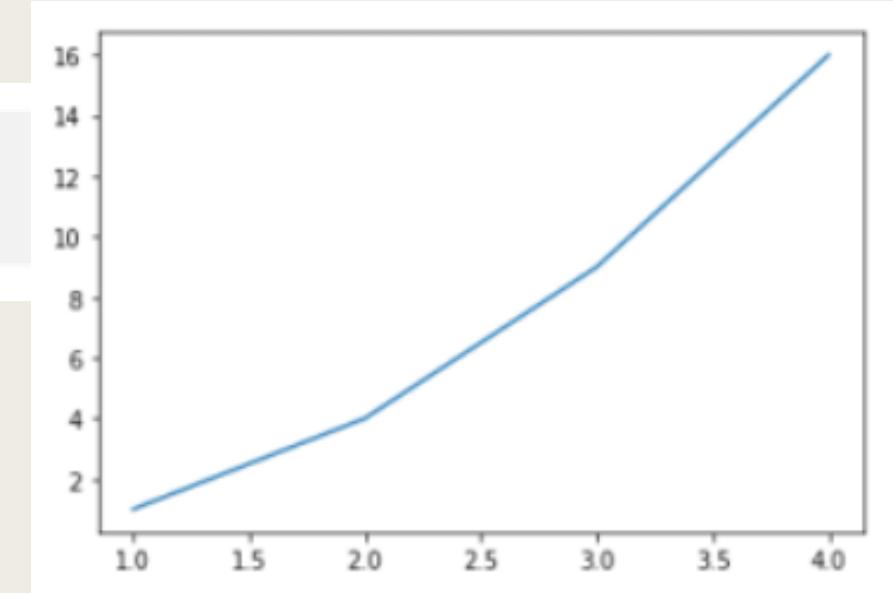
```
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3, 4])  
plt.show()
```



Matplotlib.pyplot 모듈의 plot() 함수에 하나의 숫자 리스트를 입력하면 그래프가 그려진다.

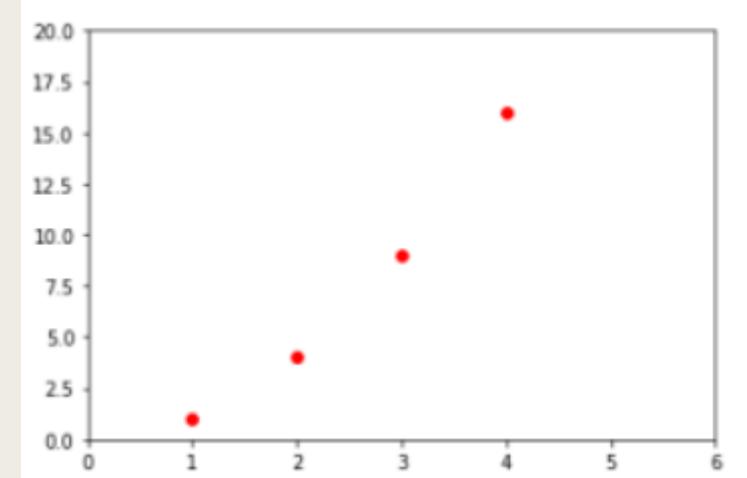
Plot()함수는 리스트의 값들을 y값들이라고 가정한다. X값들을 자동으로 생성한다.

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



- Plot()함수는 다양한 기능을 포함하고 있다. 임의의 개수의 인자를 받을 수 있다.
- 위와 같이 리스트를 입력하면 x와 y값을 그래프로 그릴 수 있다.

```
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')  
plt.axis([0, 6, 0, 20])  
plt.show()
```



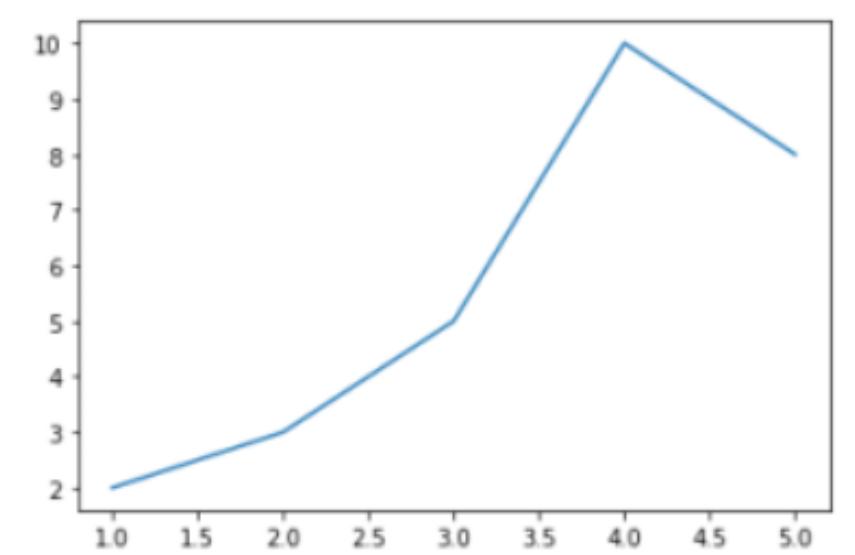
- ro 는 빨간색 red 의 원형 마커 ('o')를 의미한다.
- b- 는 파란색 blue의 실선 ('-')을 의미한다.
- axis() 함수는 축의 범위를 지정한다. [xmin, xmax, ymin, ymax]

Matplotlib 데이터입력

```
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()
```



- `plot()`은 선line 또는 마커 maker 그리기에 사용되는 함수다.
- 파이썬 딕셔너리와 같이 레이블이 있는 데이터를 그래프로 그릴 수 있다.

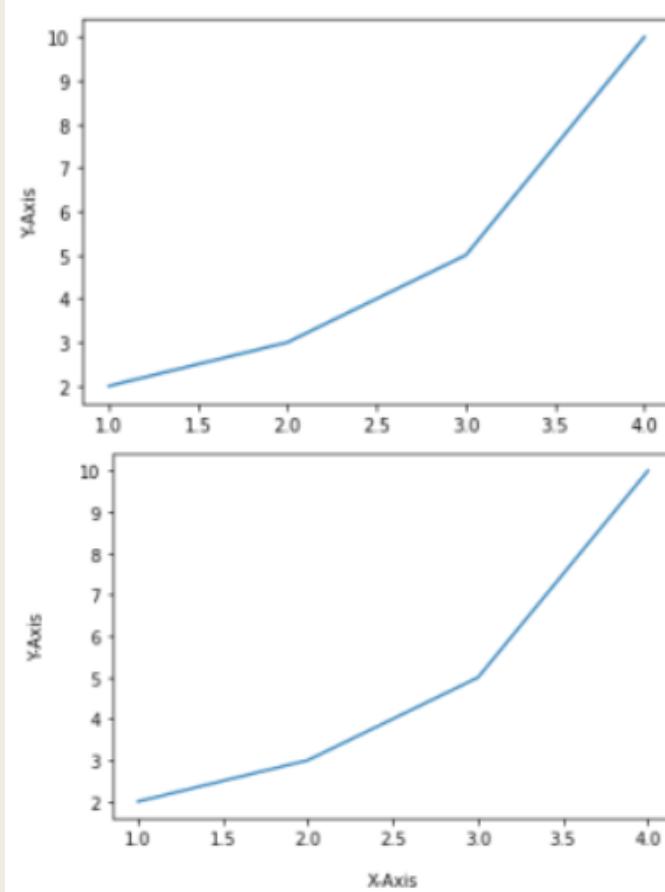
label 설정

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10])
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10])
plt.xlabel('X-Axis', labelpad=15)
plt.ylabel('Y-Axis', labelpad=20)
plt.show()
```



- xlabel(), ylabel() 함수의 labelpad 파라미터는 축 레이블의 여백을 지정할 수 있다.

font 설정

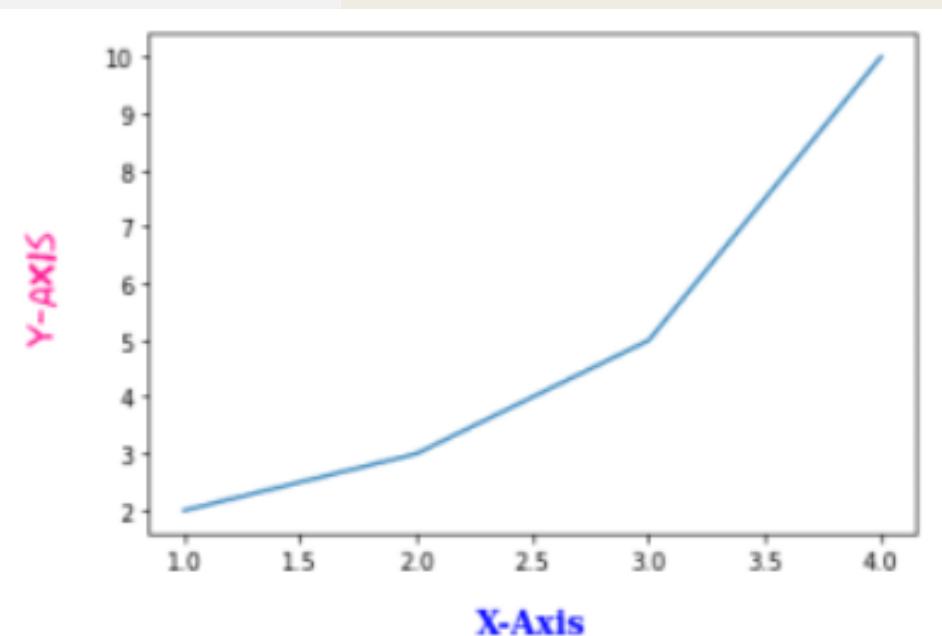
```
import matplotlib.pyplot as plt

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)
plt.ylabel('Y-Axis', labelpad=20, fontdict=font2)
plt.show()
```

xlabel(), ylabel() 함수의 fontdict 파라미터를 사용하면 축 레이블의 폰트 스타일을 설정할 수 있다.



기타 옵션

| Property | Description | Option |
|----------|-------------|---|
| alpha | 텍스트의 투명도 | 0.0 ~ 1.0 (float) |
| color | 텍스트의 색상 | Any Matplotlib color |
| family | 텍스트의 글꼴 | [FONTNAME 'serif' 'sans-serif' 'cursive' 'fantasy' 'monospace'] |
| rotation | 텍스트의 회전각 | [angle in degrees 'vertical' 'horizontal'] |
| size | 텍스트의 크기 | [size in points 'xx-small' 'x-small' 'small' 'medium' 'large' 'x-large' 'xx-large'] |
| weight | 텍스트의 굵기 | [a numeric value in range 0-1000 'ultralight' 'light' 'normal' 'regular' 'book' 'medium' 'roman' 'semibold' 'demibold' 'demi' 'bold' 'heavy' 'extra bold' 'black'] |

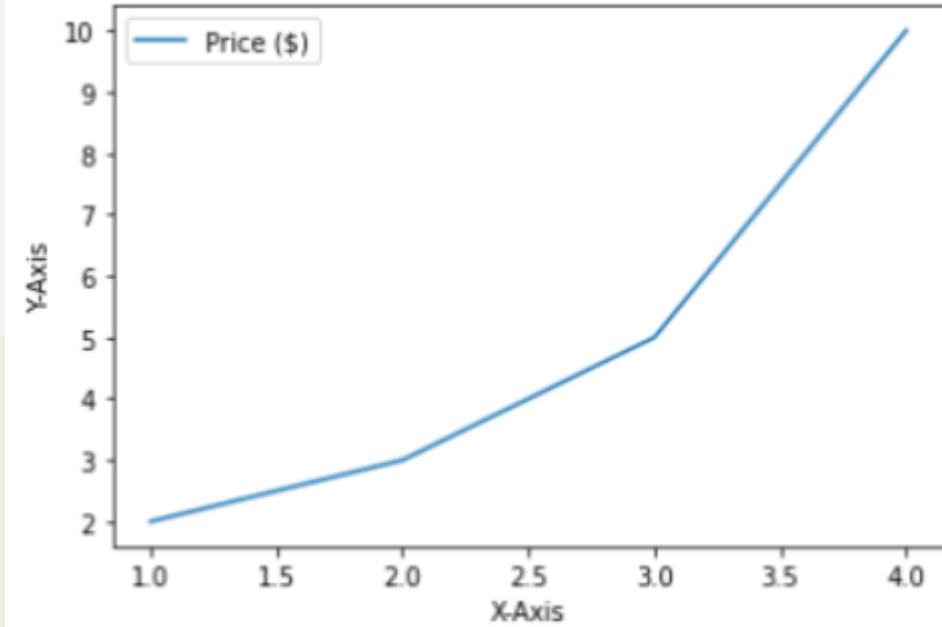
Matplotlib 범례

- 범례 Legend 는 그래프에 데이터의 종류를 표시하기 위한 텍스트이다.
- legend() 함수를 사용해서 그래프에 범례를 표시한다.

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], label='Price ($)')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.legend()

plt.show()
```

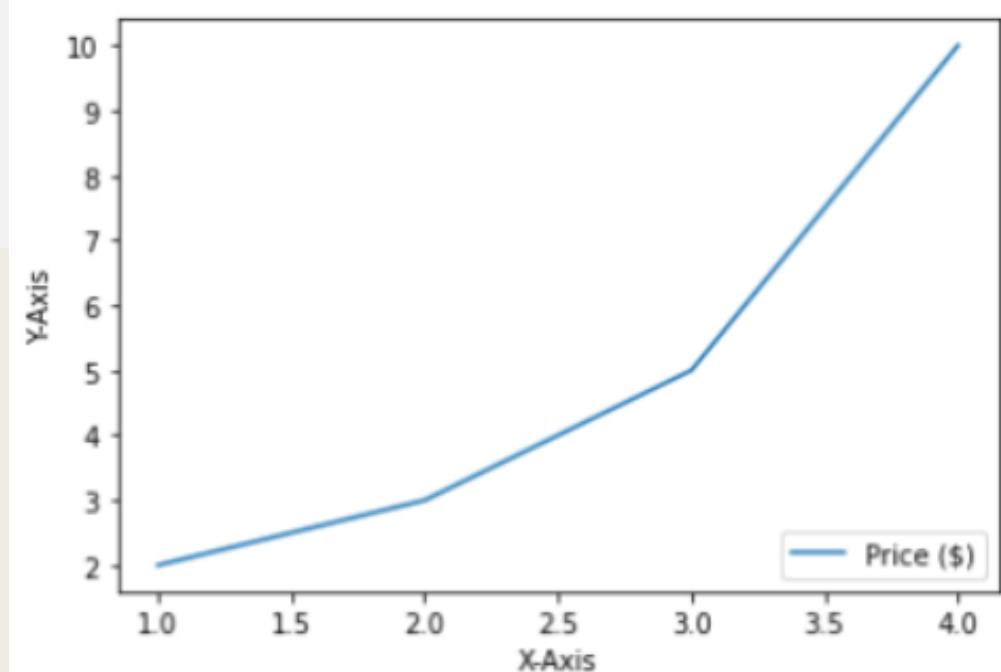


- 그래프 영역에 범례를 나타내기 위해서는 plot()함수에 label문자열을 지정하고 legend() 함수를 호출한다.

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], label='Price ($)')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.legend(loc='lower right')
plt.show()
```

- legend()함수의 loc파라미터를 사용해서
- 범례가 표시될 위치를 설정한다.
- 위치는 문자열로도 지정하고 숫자 코드로도 사용할 수 있다.



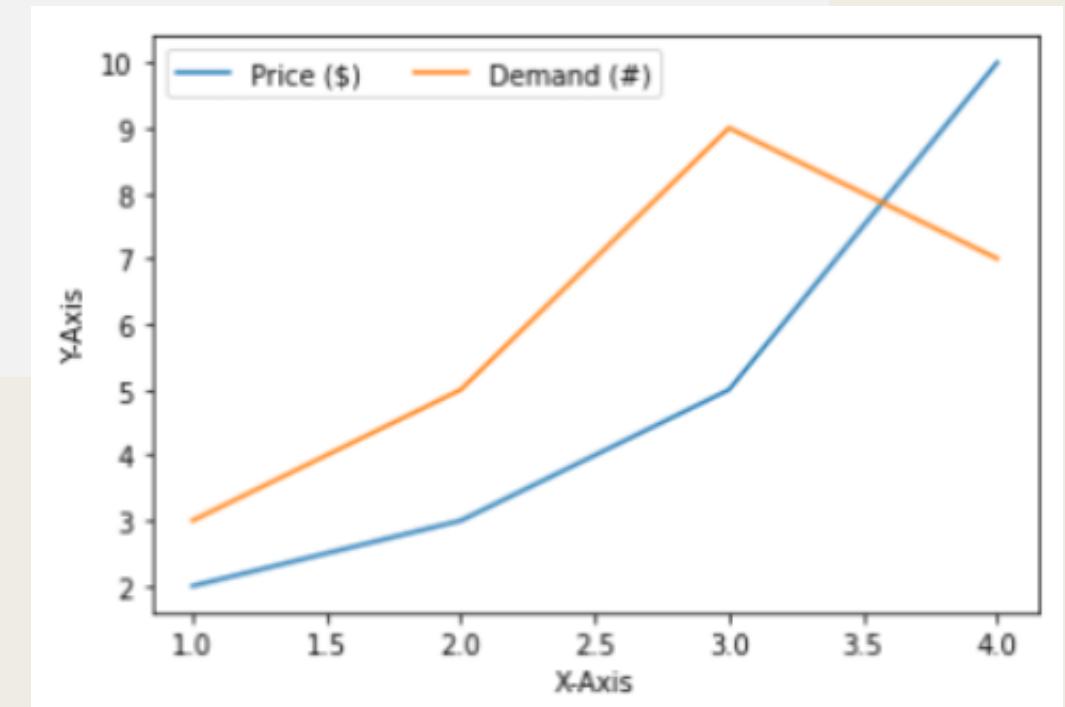
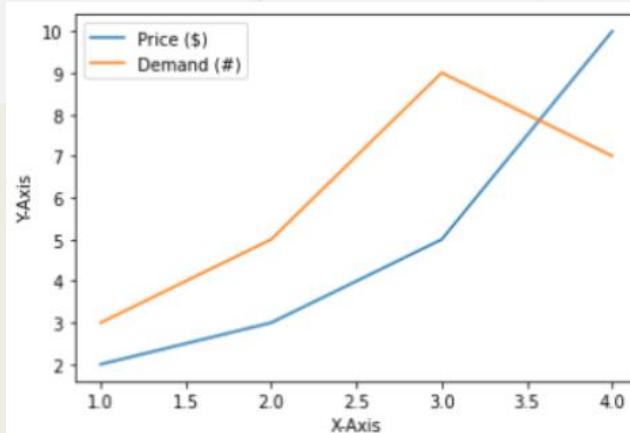
| Location String | Location Code | 설명 |
|-----------------|---------------|--------------------------|
| 'best' | 0 | 그래프의 최적의 위치에 표시합니다.(디플트) |
| 'upper right' | 1 | 그래프의 오른쪽 위에 표시합니다. |
| 'upper left' | 2 | 그래프의 왼쪽 위에 표시합니다. |
| 'lower left' | 3 | 그래프의 왼쪽 아래에 표시합니다. |
| 'lower right' | 4 | 그래프의 오른쪽 아래에 표시합니다. |
| 'right' | 5 | 그래프의 오른쪽에 표시합니다. |
| 'center left' | 6 | 그래프의 왼쪽 가운데에 표시합니다. |
| 'center right' | 7 | 그래프의 오른쪽 가운데에 표시합니다. |
| 'lower center' | 8 | 그래프의 가운데 아래에 표시합니다. |
| 'upper center' | 9 | 그래프의 가운데 위에 표시합니다. |
| 'center' | 10 | 그래프의 가운데에 표시합니다. |

```

import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], label='Price ($)')
plt.plot([1, 2, 3, 4], [3, 5, 9, 7], label='Demand (#)')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
# plt.legend(loc='best')
plt.legend(loc='best', ncol=2)
plt.show()

```

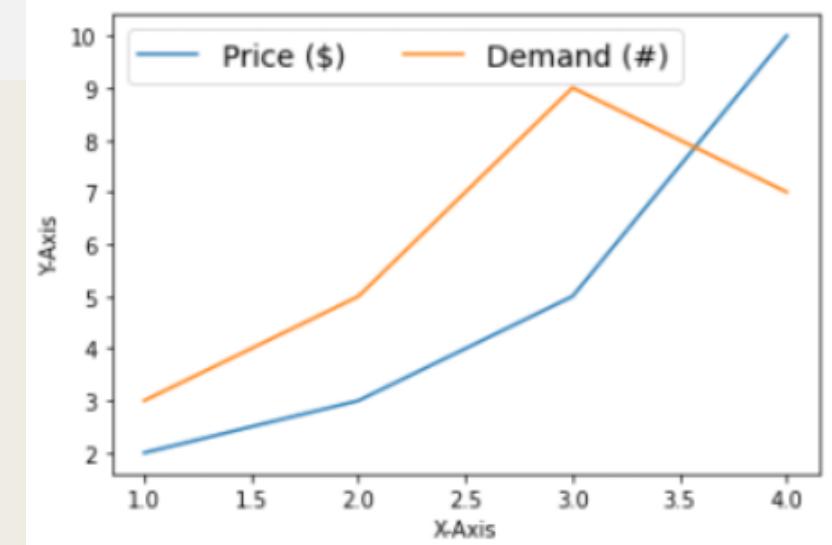


- legend() 함수의 ncol 파라미터는 범례에 표시될 텍스트의 열의 개수를 지정한다.
- 기본적으로 범례텍스트는 1개 열로 표시 ncol=2로 지정하면 2개의 열로 표시된다.

```
import matplotlib.pyplot as plt

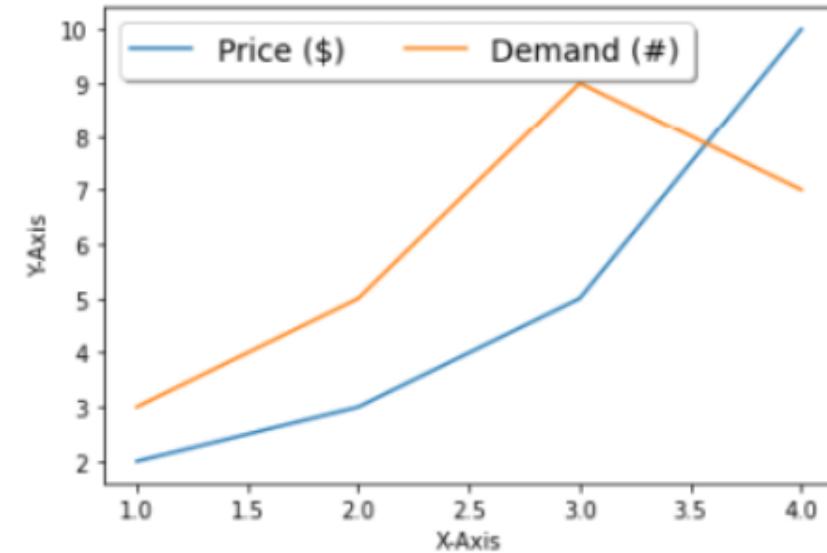
plt.plot([1, 2, 3, 4], [2, 3, 5, 10], label='Price ($)')
plt.plot([1, 2, 3, 4], [3, 5, 9, 7], label='Demand (#)')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.legend(loc='best', ncol=2, fontsize=14)
plt.show()
```

- legend()함수의 fontsize 파라미터는 범례에 표시될 폰트의 크기를 지정한다.



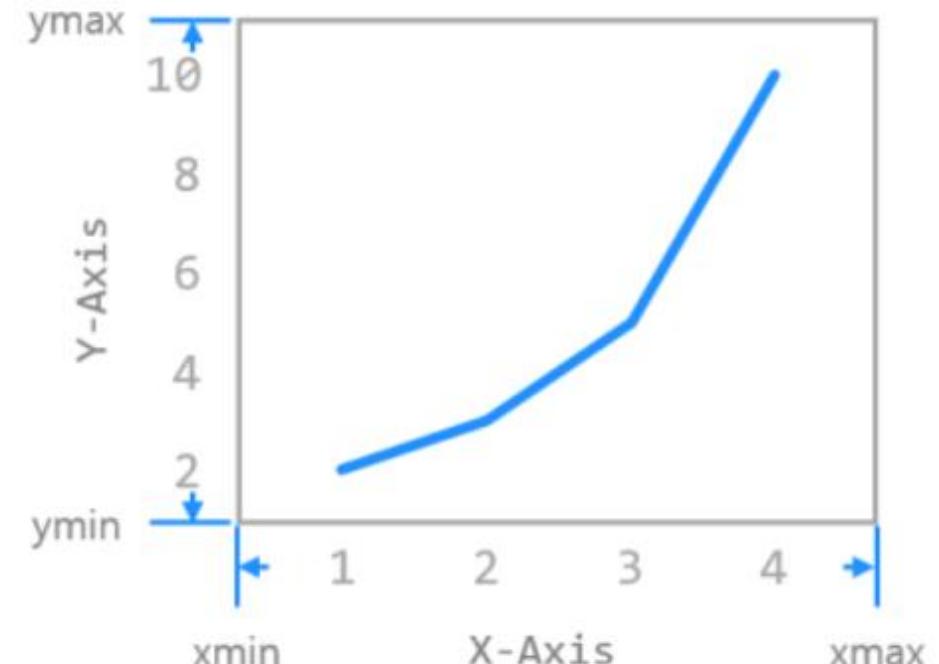
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], label='Price ($)')
plt.plot([1, 2, 3, 4], [3, 5, 9, 7], label='Demand (#)')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.legend(loc='best', ncol=2, fontsize=14,
           frameon=True, shadow=True)
plt.show()
```



- `frameon` 파라미터는 범례 상자의 테두리를 표시할지 여부를 지정한다.
`frameon =False`를 지정하면 테두리가 표시되지 않는다.
- `shadow` 파라미터를 사용해서 상자에 그림자를 추가할 수 있다.

축 설정

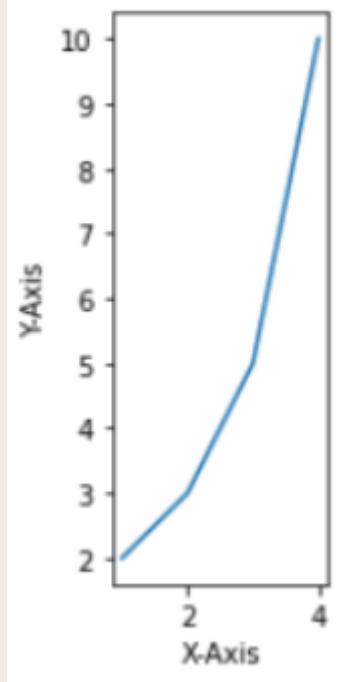
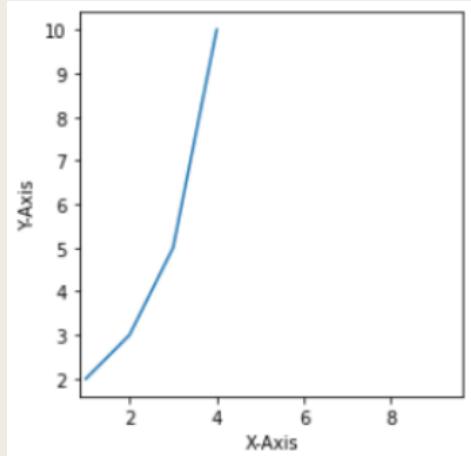


- `xlim()`, `ylim()`, `axis()` 함수들을 사용하면 그래프의 X,Y축이 표시되는 범위를 지정할 수 있다.
- `xlim()`에 $[x_{\min}, x_{\max}]$, `ylim()` $[y_{\min}, y_{\max}]$ 각각 지정하거나
- `axis()` 함수에 $[x_{\min}, x_{\max}, y_{\min}, y_{\max}]$ 의 형태로 범위를 지정할 수 있다.
- `axis()` 함수를 사용할 때는 반드시 네 개의 값이 있어야 한다.
- 입력이 없으면 자동으로 범위를 지정한다.

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10])
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
# plt.axis('square')
plt.axis('scaled')

plt.show()
```



'on' | 'off' | 'equal' | 'scaled' | 'tight' | 'auto' | 'normal' | 'image' | 'square'

- square 는 축의 길이가 동일하게 표시된다.
- scaled은 x,y축이 같은 길이 스케일로 나타나게 된다.

축 범위 반환

```
import matplotlib.pyplot as plt  
  
plt.plot([1, 2, 3, 4], [2, 3, 5, 10])  
plt.xlabel('X-Axis')  
plt.ylabel('Y-Axis')  
  
x_range, y_range = plt.xlim(), plt.ylim()  
print(x_range, y_range)  
  
axis_range = plt.axis('scaled')  
print(axis_range)  
  
plt.show()
```

(0.85, 4.15) (1.6, 10.4)
(0.85, 4.15, 1.6, 10.4)

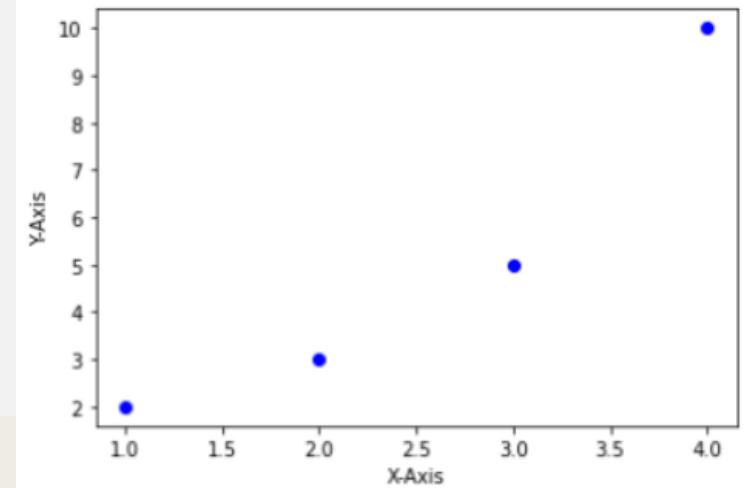
- xlim(), ylim()함수는 그래프 영역에 표시되는 x축,y축의 범위를 각각 반환한다.
- axis()함수는 그래프 영역에 표시되는 x,y축의 범위를 한번에 반환한다.

마커 지정 Marker

- 특별한 설정이 없으면 그래프가 실선으로 그려진다.

```
import matplotlib.pyplot as plt

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

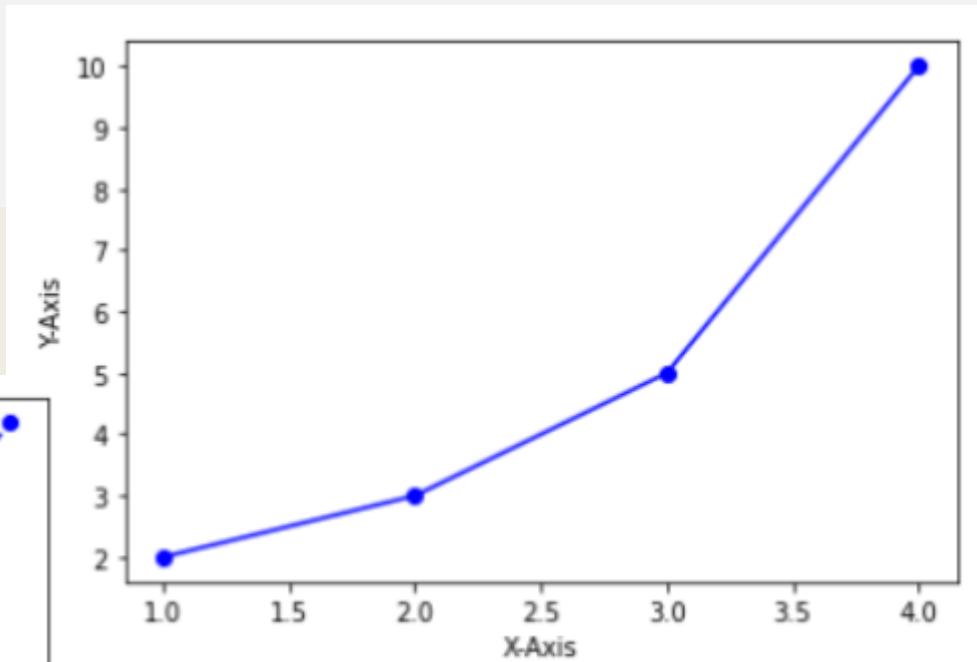
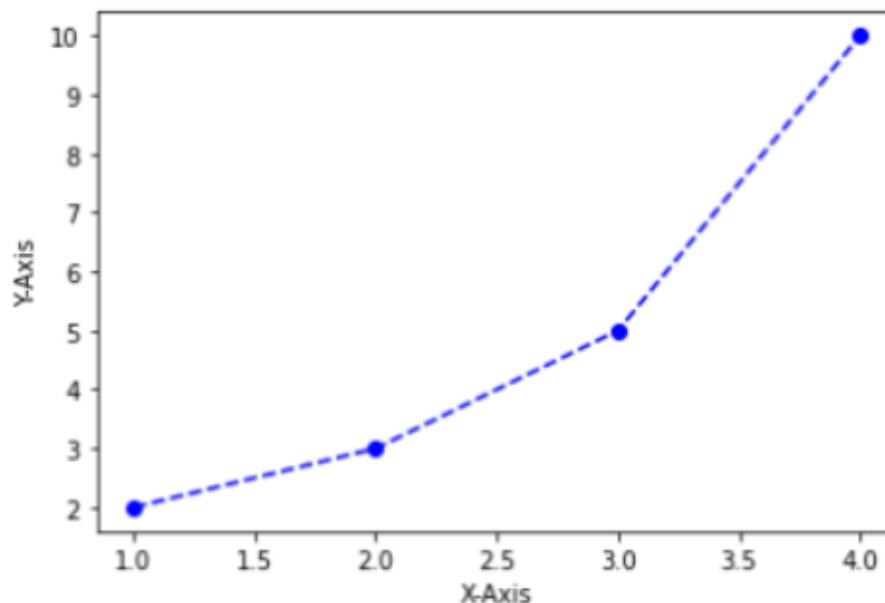


- `plot()`함수에 숫자 데이터와 함께 'bo'를 설정하면 파란색의 원형 마커로 그래프가 표시된다.

```
import matplotlib.pyplot as plt

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

- 'bo-' 파란색의 원형 마커와 실선
- 'bo--'파란색의 원형 마커와 점선



```
'b'      # blue markers with default shape
'r o'    # red circles
'g-'     # green solid line
'---'    # dashed line with default color
'k^:'   # black triangle_up markers connected by a dotted line
```

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 |

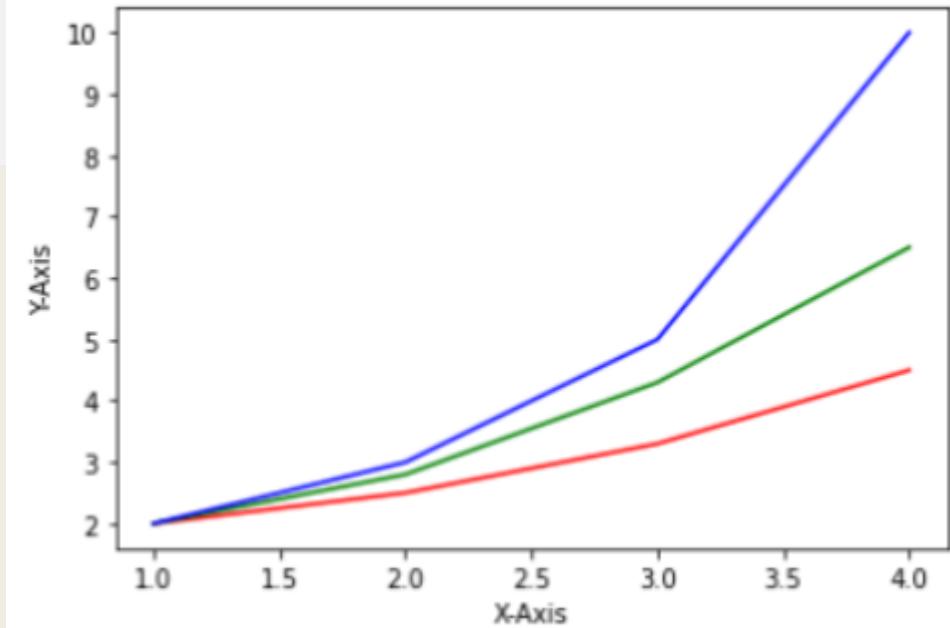
Markers

| 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 |

색상 지정

```
import matplotlib.pyplot as plt

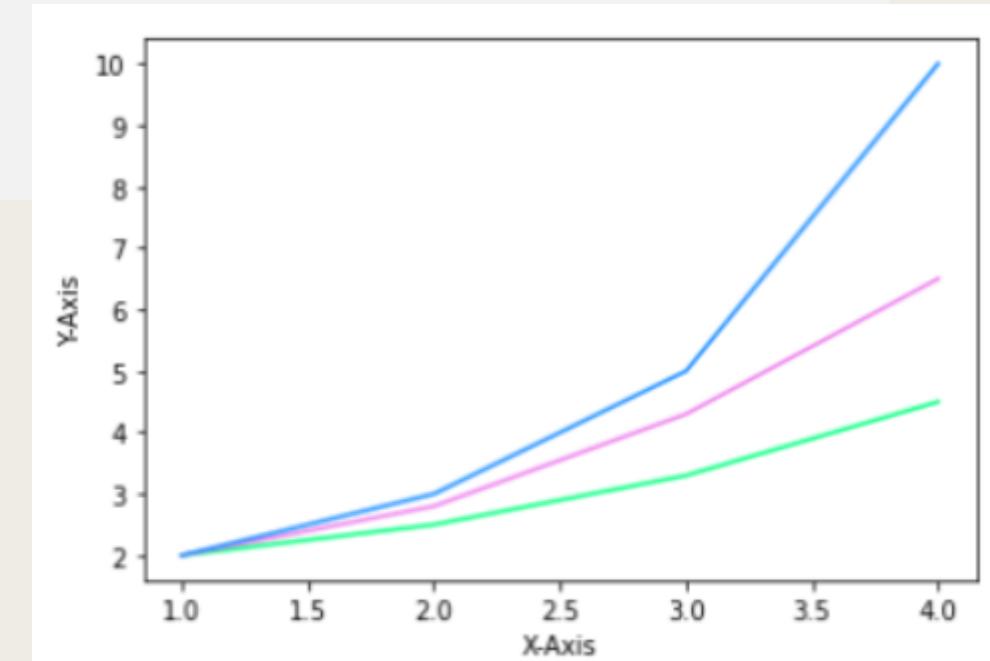
plt.plot([1, 2, 3, 4], [2.0, 2.5, 3.3, 4.5], 'r')
plt.plot([1, 2, 3, 4], [2.0, 2.8, 4.3, 6.5], 'g')
plt.plot([1, 2, 3, 4], [2.0, 3.0, 5.0, 10.0], 'b')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```



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

```
plt.plot([1, 2, 3, 4], [2.0, 2.5, 3.3, 4.5], color='springgreen')
plt.plot([1, 2, 3, 4], [2.0, 2.8, 4.3, 6.5], color='violet')
plt.plot([1, 2, 3, 4], [2.0, 3.0, 5.0, 10.0], color='dodgerblue')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```

- color키워드 인자를 사용해서 더 다양한 색상의 이름을 지정할 수 있다.

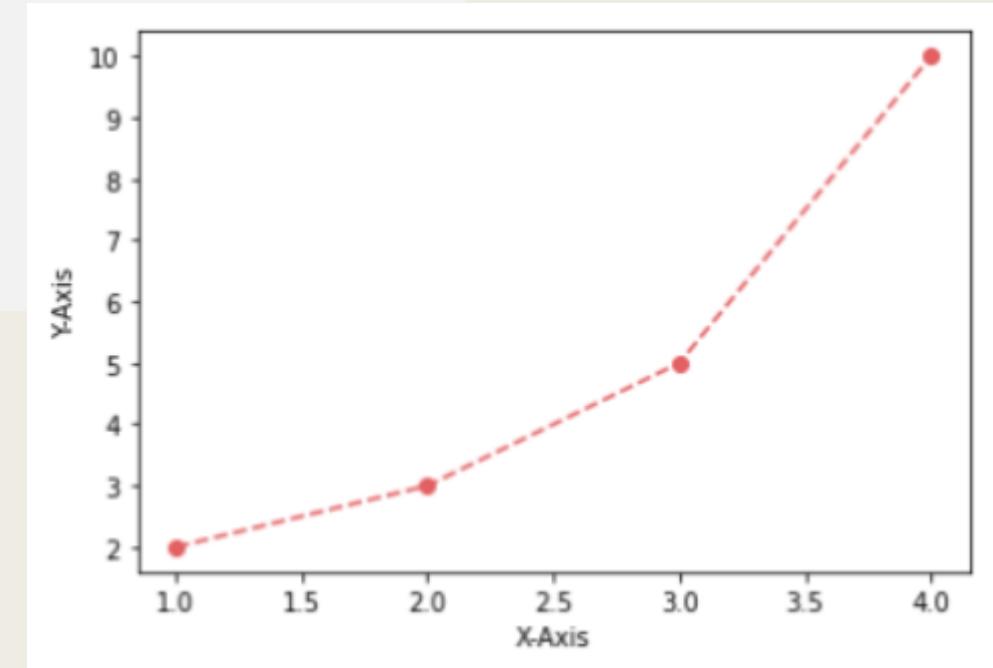
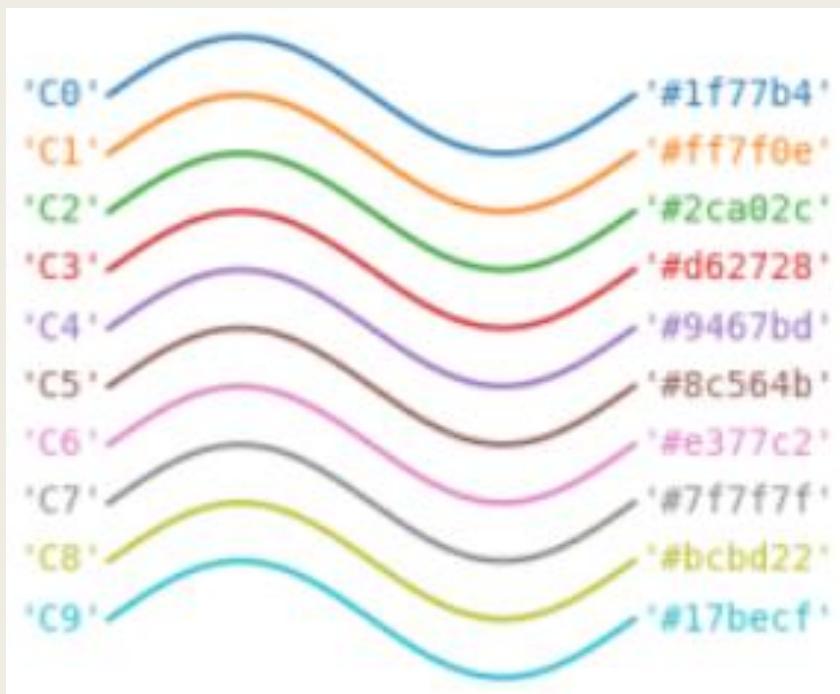


```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [2, 3, 5, 10], color='#e35f62',
         marker='o', linestyle='--')
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')

plt.show()
```

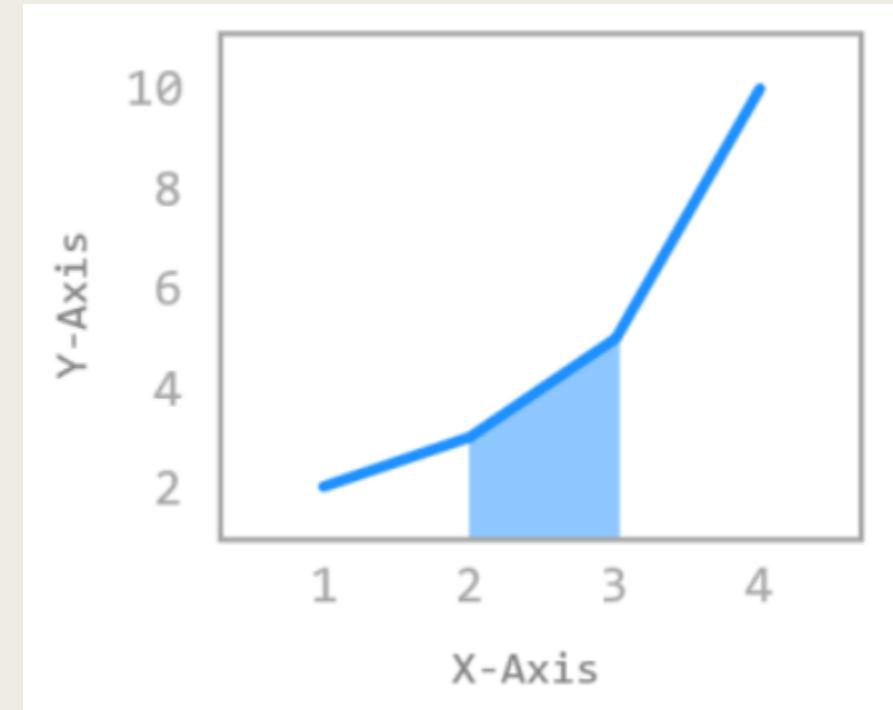
- 선의 색상은 Hex code 를 사용할 수도 있다.



| | | | |
|-------------|----------------------|-------------------|-----------------|
| black | bisque | forestgreen | slategrey |
| dimgray | darkorange | limegreen | lightsteelblue |
| dimgrey | burlywood | darkgreen | cornflowerblue |
| gray | antiquewhite | green | royalblue |
| grey | tan | lime | ghostwhite |
| darkgray | navajowhite | seagreen | lavender |
| darkgrey | blanchedalmond | mediumseagreen | midnightblue |
| silver | papayawhip | springgreen | navy |
| lightgray | moccasin | mintcream | darkblue |
| lightgrey | orange | mediumspringgreen | mediumblue |
| gainsboro | wheat | mediumaquamarine | blue |
| whitesmoke | oldlace | aquamarine | slateblue |
| white | floralwhite | turquoise | darkslateblue |
| snow | darkgoldenrod | lightseagreen | mediumslateblue |
| rosybrown | goldenrod | mediumturquoise | mediumpurple |
| lightcoral | cornsilk | azure | rebeccapurple |
| indianred | gold | lightcyan | blueviolet |
| brown | lemonchiffon | paleturquoise | indigo |
| firebrick | khaki | darkslategray | darkorchid |
| maroon | palegoldenrod | darkslategrey | darkviolet |
| darkred | darkkhaki | teal | mediumorchid |
| red | ivory | darkcyan | thistle |
| mistyrose | beige | aqua | plum |
| salmon | lightyellow | cyan | violet |
| tomato | lightgoldenrodyellow | darkturquoise | purple |
| darksalmon | olive | cadetblue | darkmagenta |
| coral | yellow | powderblue | fuchsia |
| orangered | olivedrab | lightblue | magenta |
| lightsalmon | yellowgreen | deepskyblue | orchid |
| sienna | darkolivegreen | skyblue | mediumvioletred |
| seashell | greenyellow | lightskyblue | deeppink |
| chocolate | chartreuse | steelblue | hotpink |
| saddlebrown | lawngreen | aliceblue | lavenderblush |
| sandybrown | honeydew | dodgerblue | palevioletred |
| peachpuff | darkseagreen | lightslategray | crimson |
| peru | palegreen | lightslategrey | pink |
| linen | lightgreen | slategray | lightpink |

그래프 일부 영역 채우기

- 특정 영역을 강조할 때 사용
- 특정 영역만 채우는 세가지 함수,
- `fill_between()`, 두 수평 방향의 곡선 사이를 채운다.
- `fill_betweenx()`, 두 수직 방향의 곡선 사이를 채운다.
- `fill()`, 다각형 영역을 채운다.



```

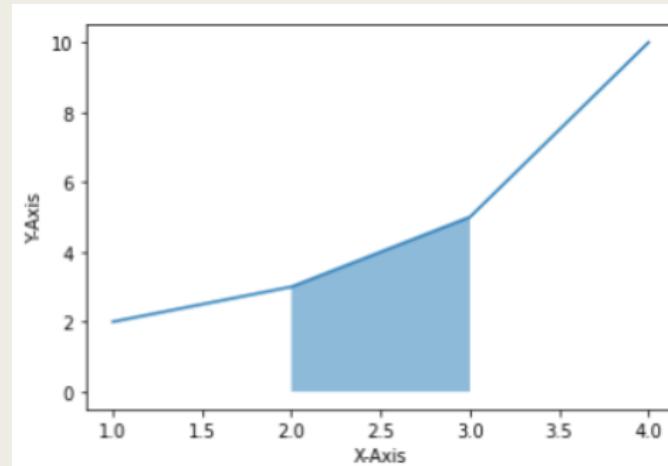
import matplotlib.pyplot as plt

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

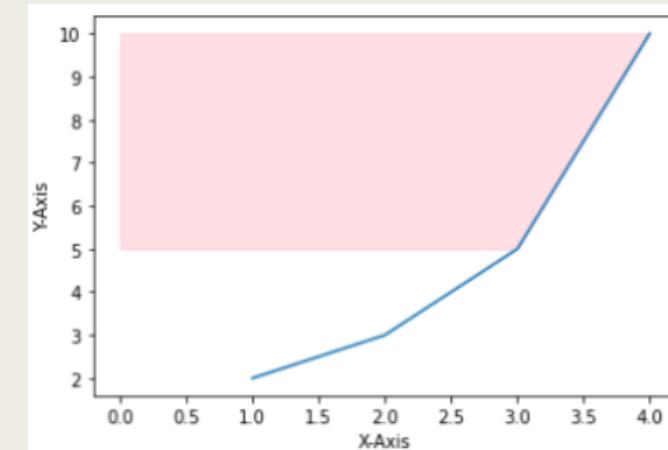
plt.plot(x, y)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
# plt.fill_between(x[1:3], y[1:3], alpha=0.5)
plt.fill_betweenx(y[2:4], x[2:4], color='pink', alpha=0.5)
plt.show()

```

- fill_between() x[1:3] y[1:3]을 입력하면
- 네 점 (x[1], y[1]), (x[2] ,y[2]), (x[1],0), (x[2],0) 영역이 채워진다.



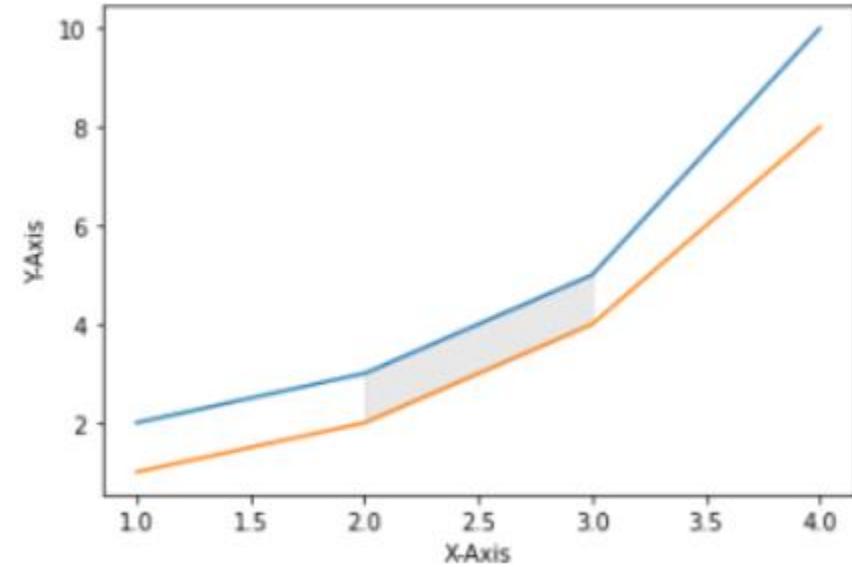
- fill_betweenx()에 y[2:4],x[2:4]를 입력하면
- 네 점 (x[2], y[2]),(x[3], y[3]),(0,y[2]),(0,y[3]) 영역이 채워진다.



```
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_between(x[1:3], y1[1:3], y2[1:3], color='lightgray',
                 alpha=0.5)
plt.show()
```

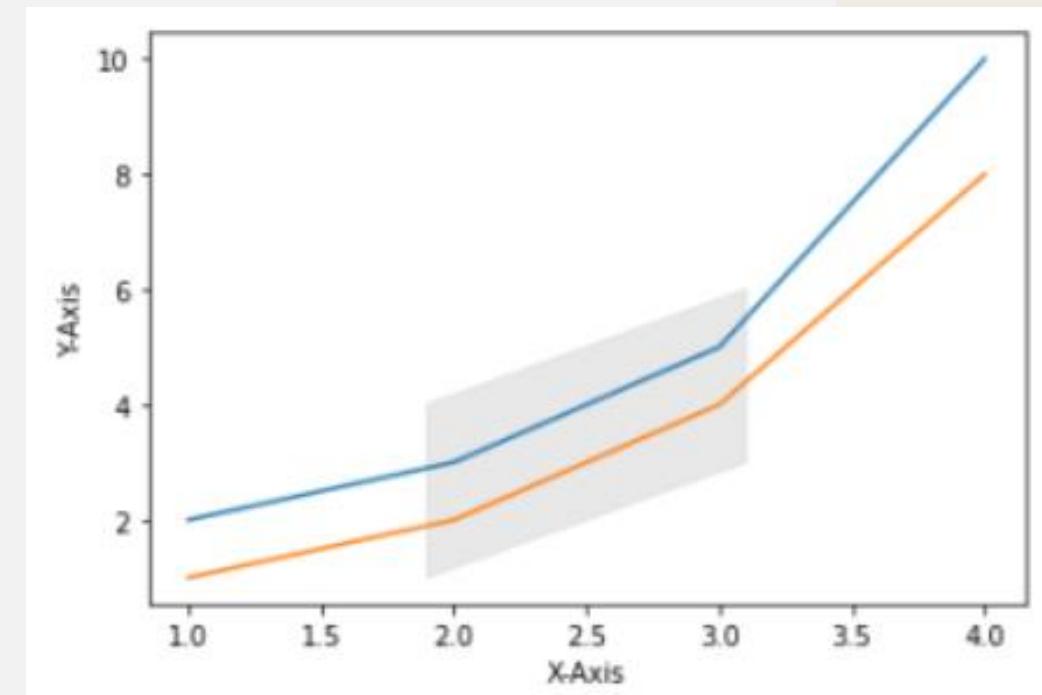


두 그래프 사이 영역을 채우기 위해 두개의 y값의 리스트 y1, y2를 입력한다.
네점 (x[1] ,y[1]), (x[1] ,y[2]), (x[2] ,y[1]), (x[2] ,y[2]) 사이의 영역을 채운다.

```
import matplotlib.pyplot as plt

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

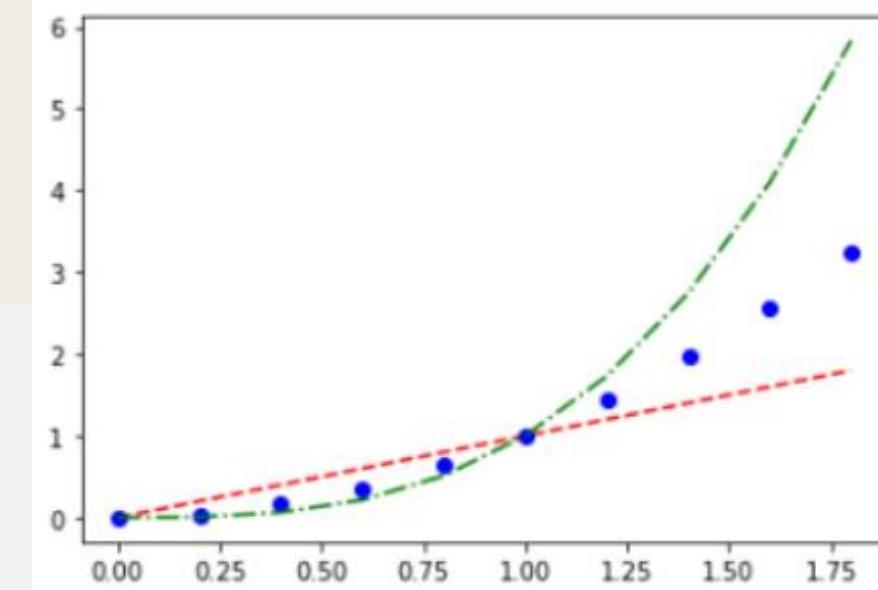
plt.plot(x, y1)
plt.plot(x, y2)
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.fill([1.9, 1.9, 3.1, 3.1], [1.0, 4.0, 6.0, 3.0],
         color='lightgray', alpha=0.5)
plt.show()
```



- fill() 함수에 x, y 값의 리스트를 입력해주면
- 각 x,y 점들로 정의되는 다각형 영역을 자유롭게 지정해서 채울 수 있다.

여러곡선 그리기

```
import matplotlib.pyplot as plt  
import numpy as np  
  
a = np.arange(0, 2, 0.2)  
  
plt.plot(a, a, 'r--', a, a**2, 'bo', a, a**3, 'g-.')  
plt.show()
```



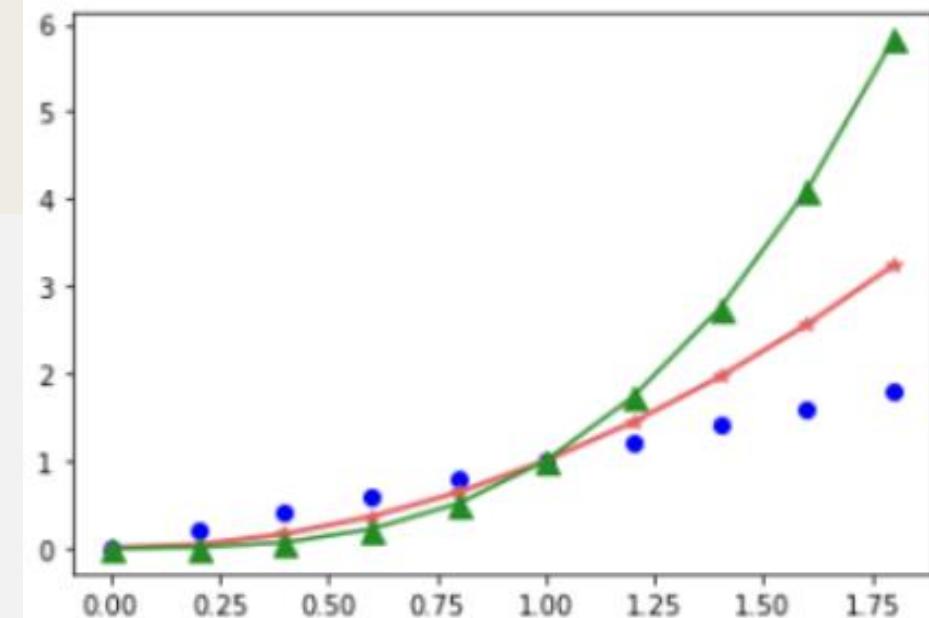
Matplotlib 여러 곡선 그리기 - 기본 사용

- 하나의 그래프 영역에 여러 개 스타일의 곡선을 나타내는 방법을 소개
- Numpy Array[0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8]
- 세개의 곡선($y=x$, $y = x^2$, $y = x^3$)이 동시에 그려진다.
- r- 빨간색 대시 , bo 파란색 원형, g- 녹색 대시-닷 스타일 선을 의미한다.

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

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

plt.plot(a, a, 'bo')
plt.plot(a, a**2, color="#e35f62", marker='*', linewidth=2)
plt.plot(a, a**3, color='forestgreen', marker='^', markersize=9)
plt.show()
```



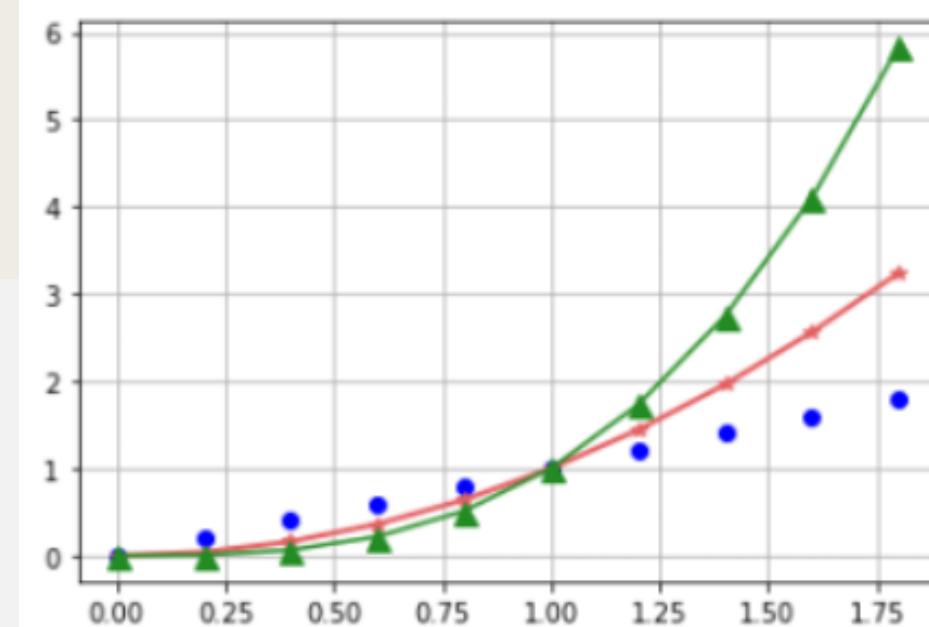
- 첫 번째 곡선의 모양은 ‘bo’,
- 두 번째 곡선의 모양은 색상 #e35f62 마커는 '*' 두께는 2로 설정
- 세 번째 곡선의 모양은 색상 forestgreen 마커는 ^ 마커의 크기는 9로 설정

그리드 Grid 설정

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

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

plt.plot(a, a, 'bo')
plt.plot(a, a**2, color='#e35f62', marker='*', linewidth=2)
plt.plot(a, a**3, color='springgreen', marker='^', markersize=9)
plt.grid(True)
plt.show()
```

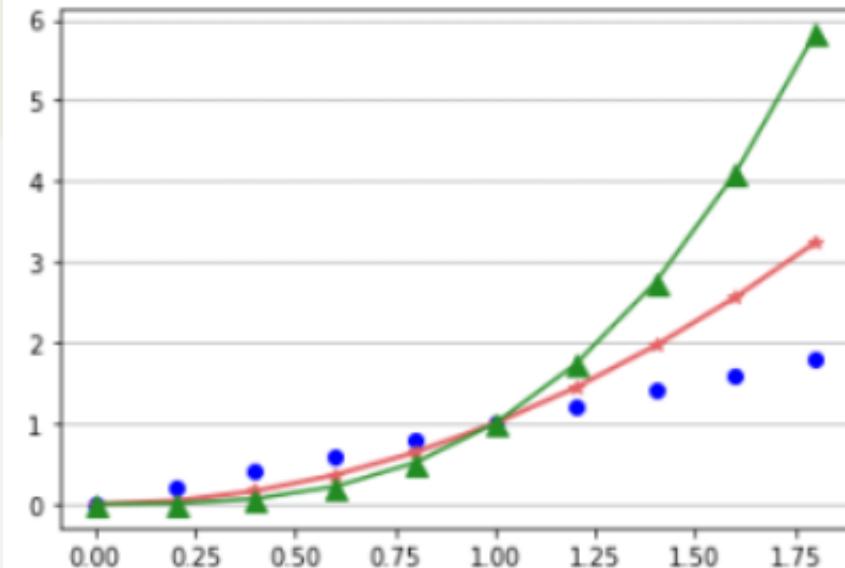


- 데이터의 위치를 더 명확하게 나타내기 위해 그리드(격자, Grid)를 사용한다.
- grid()함수를 이용해서 그래프에 다양한 그리드를 설정할 수 있다.

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

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

plt.plot(a, a, 'bo')
plt.plot(a, a**2, color='#e35f62', marker='*', linewidth=2)
plt.plot(a, a**3, color='forestgreen', marker='^', markersize=9)
plt.grid(True, axis='y')
plt.show()
```



- axis =‘y’로 설정하면 가로 방향의 그리드만 표시된다.
- ‘both’,‘x’,‘y’중 선택할 수 있고 디폴트 ‘both’

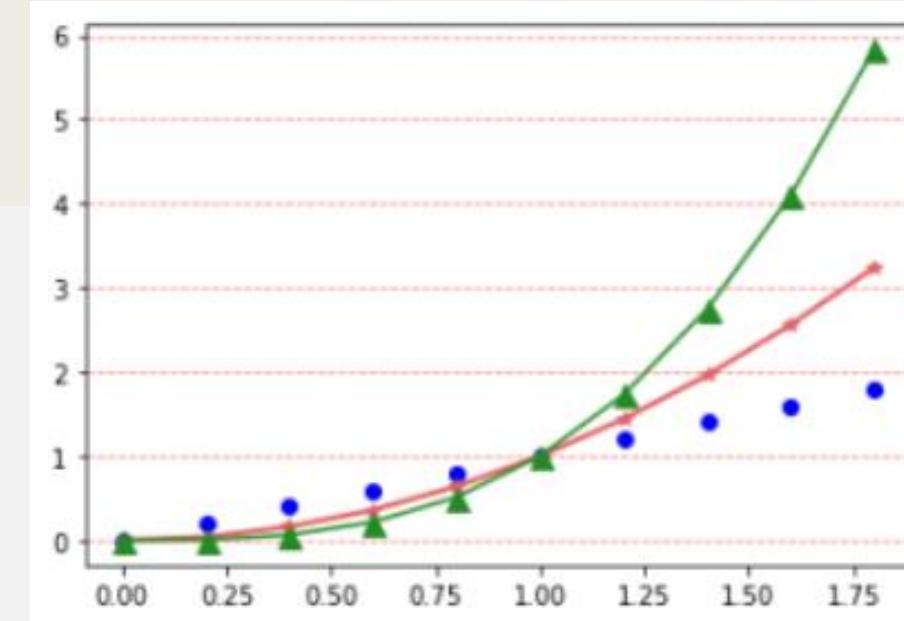
```

import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(a, a, 'bo')
plt.plot(a, a**2, color='#e35f62', marker='*', linewidth=2)
plt.plot(a, a**3, color='springgreen', marker='^', markersize=9)
plt.grid(True, axis='y', color='red', alpha=0.5, linestyle='--')
plt.show()

```



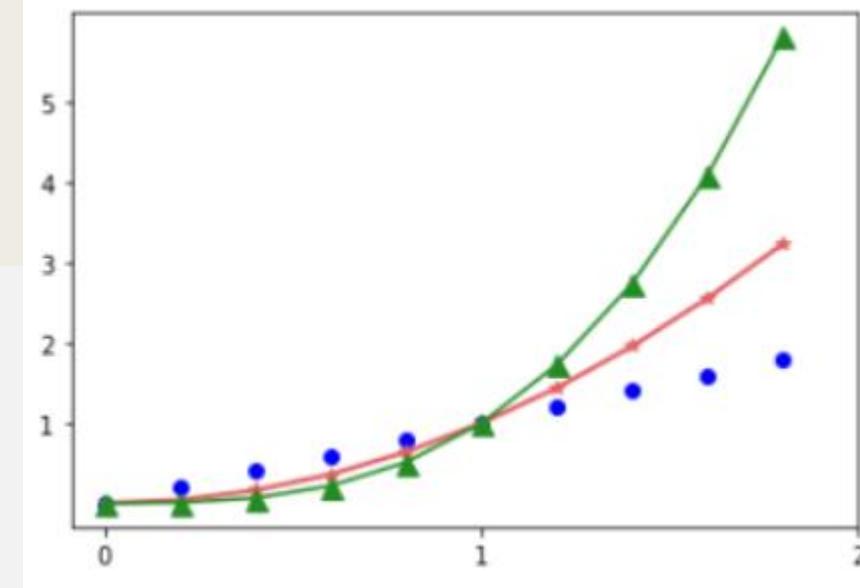
- color, alpha, linestyle 파라미터를 사용해서 그리드 선의 스타일을 설정한다.
- which 파라미터 ‘major’,‘minor’,‘both’ 등으로 설정하면 주눈금, 보조눈금에 각각 그리드를 표시할 수도 있다.

눈금 표시하기

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

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

plt.plot(a, a, 'bo')
plt.plot(a, a**2, color="#e35f62", marker='*', linewidth=2)
plt.plot(a, a**3, color='forestgreen', marker='^', markersize=9)
plt.xticks([0, 1, 2])
plt.yticks(np.arange(1, 6))
plt.show()
```



- Tick 틱 은 그래프의 축에 간격을 구분하기 위해 표시하는 눈금
- xticks(), yticks(), tick_params()함수를 이용해서 그래프에 눈금을 그린다.
- plot함수는 리스트의 값들이 y값들이라고 가정하고 x 값들은 자동으로 만들어진다.

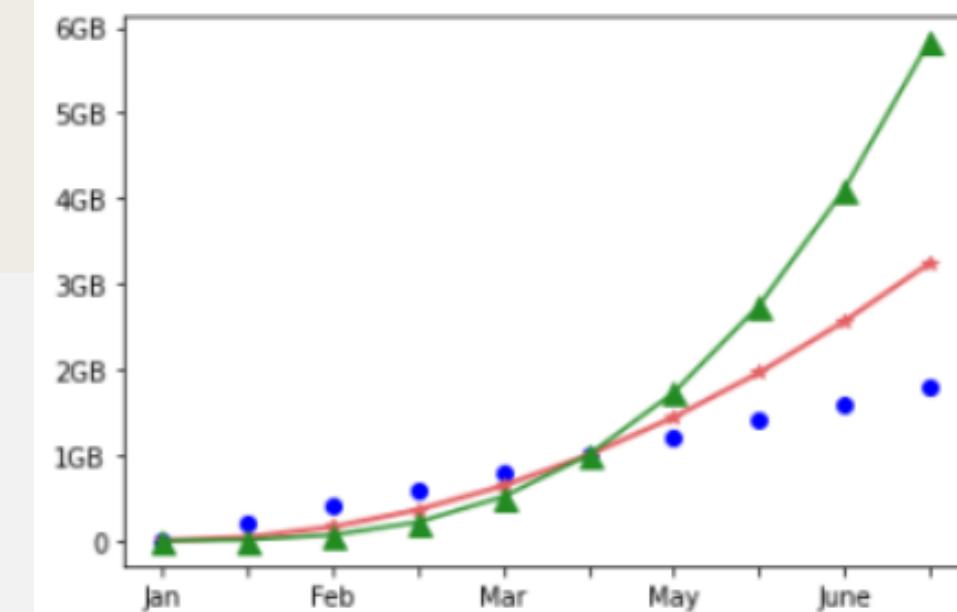
눈금 레이블 지정

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

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

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



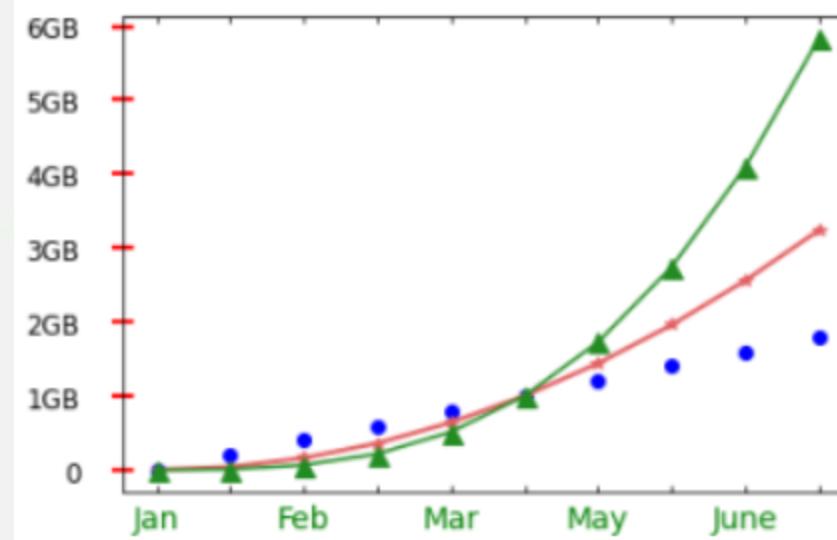
- `plot()`함수의 `labels`를 사용하면 눈금 레이블을 문자열의 형태로 지정할 수 있다.
- 입력해준 눈금의 개수와 같은 개수의 레이블을 지정해 준다.

```

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

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

```



- tick_params()함수를 사용해서 눈금의 스타일을 다양하게 설정할 수 있다.
- axis는 설정이 적용될 축을 지정한다. 'x','y','both'중 선택할 수 있다.
- direction 'in','out','inout'으로 설정하면 눈금이 안/밖으로 표시된다.
- length 는 눈금의 길이를 지정한다. pad는 눈금과 레이블의 거리를 지정한다.
- labelsize 레이블의 크기를 지정한다. labelcolor는 레이블의 색상을 지정한다.
- top/bottom/left/right를 True/False로 지정하면 눈금이 표시될 위치를 선택
- width 는 눈금의 너비를 지정한다. color는 눈금의 색상을 지정한다.

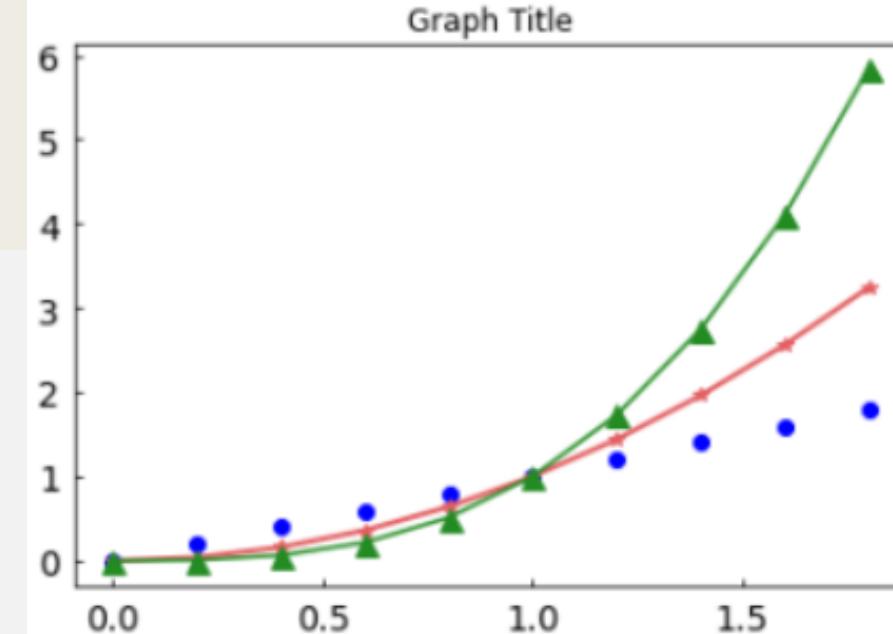
타이틀 설정

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

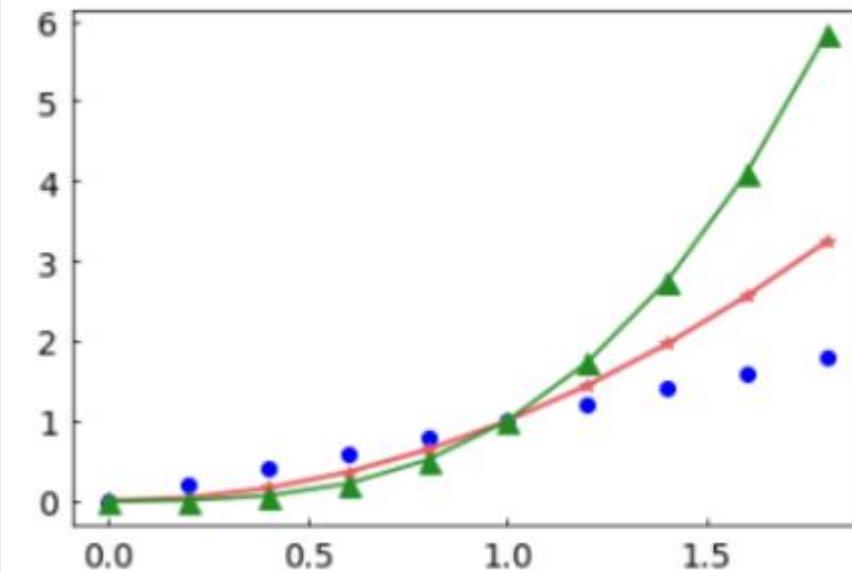
a = np.arange(0, 2, 0.2)

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

plt.tick_params(axis='both', direction='in', length=3, pad=6,
                labelsize=14)
plt.title('Graph Title')
plt.show()
```



- `title()`함수를 사용해서 그래프의 타이틀을 설정할 수 있다.

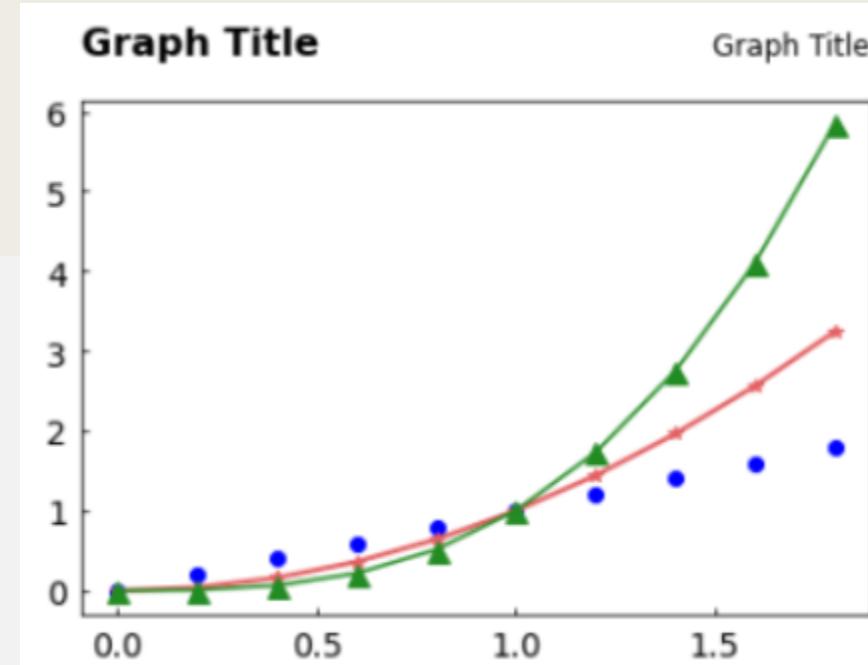


```
plt.title('Graph Title', loc='right', pad=20)
```

- loc='right'로 설정하면 타이틀이 그래프의 오른쪽 위에 나타나게 된다.
- 'left','center','right' 중 선택할 수 있으며 디폴트는 center 이다.
- pad = 20 은 타이틀과 그래프와의 간격을 포인트 단위로 설정한다.

```
plt.title('Graph Title', loc='right', pad=20)

title_font = {
    'fontsize': 16,
    'fontweight': 'bold'
}
plt.title('Graph Title', fontdict=title_font, loc='left', pad=20)
```



- fontdict에 딕셔너리 형태로 폰트에 대한 설정을 입력한다.
- 글자 크기를 16, 글자 두께는 두껍게로 설정했다.
- fontsize는 포인트 단위 숫자로 입력하거나 smaller, x-large로 설정할 수 있다.
- fontweight은 normal, bold, heavy, light, ultrabold, ultralight로 설정할 수 있다.

수직선과 수평선 표시

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

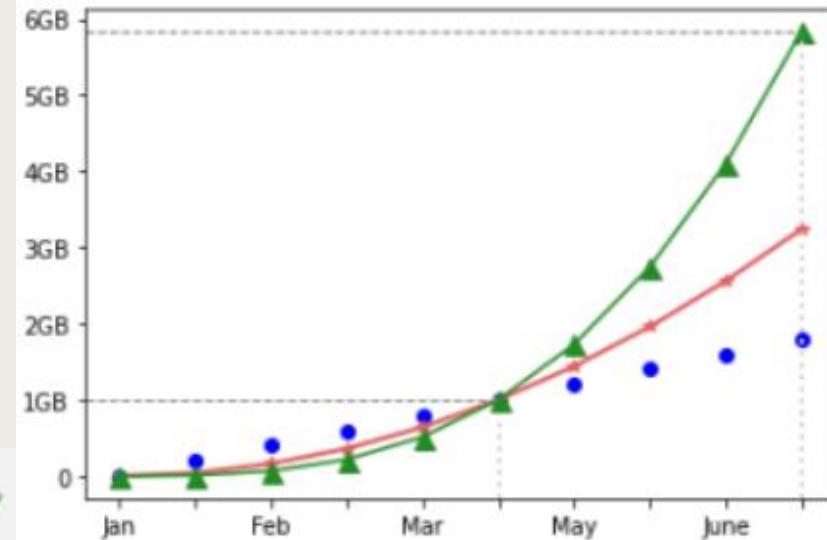
a = np.arange(0, 2, 0.2)

plt.plot(a, a, 'bo')
plt.plot(a, a**2)
plt.axhline(1, 0, 0.55, color='gray', linestyle='--',
            linewidth='1')
plt.xticks(np.arange('Jan', 'Jun'))
plt.yticks(np.arange('0', '4GB', 1GB))

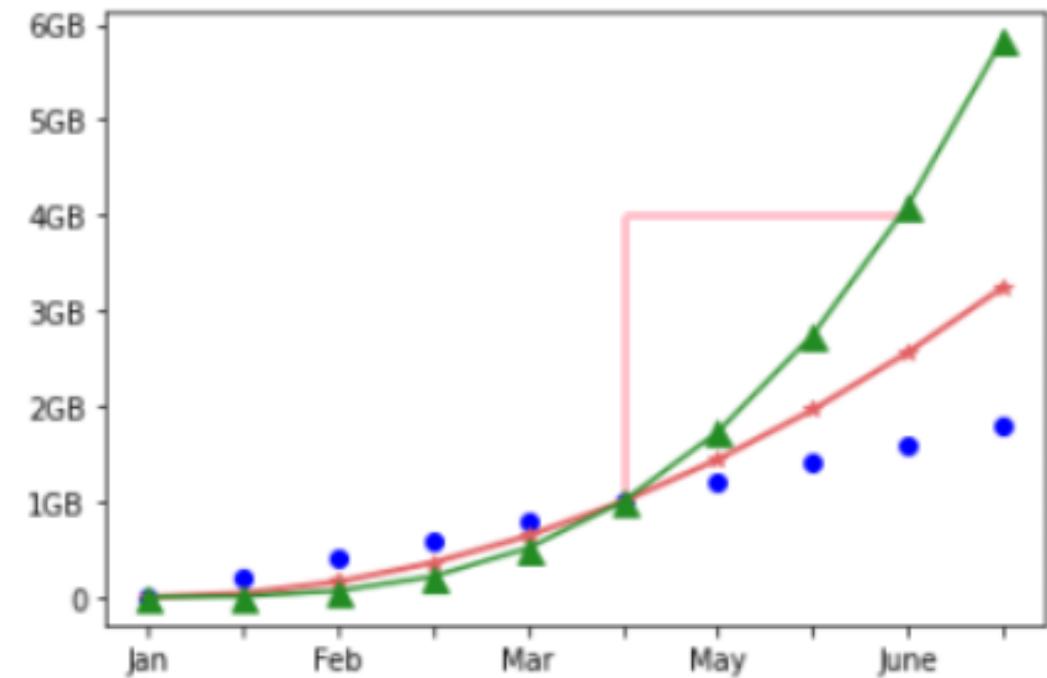
plt.axvline(1, 0, 0.16, color='lightgray', linestyle=':',
            linewidth='2')

plt.axhline(5.83, 0, 0.95, color='gray', linestyle='--',
            linewidth='1')
plt.axvline(1.8, 0, 0.95, color='lightgray', linestyle=':',
            linewidth='2')

plt.show()
```



- axhline() : 축을 따라 수평선을 표시 axvline() : 축을 따라 수직선을 표시
- hlines() : 점을 따라 수평선을 표시 vlines() : 점을 따라 수직선을 표시
- axhline() 함수의 첫번째 인자는 y 값으로서 수평선의 위치가 된다. 두, 세번째 인자는 xlim, xmax의 값으로 0에서 1사이의 값을 입력한다. 0은 왼쪽 끝 1은 오른쪽 끝을 의미한다.
- axvline() 함수의 첫번째 인자는 x 값으로서 수직선의 위치가 된다. 두, 세번째 인자는 ylim, ymax의 값으로 0에서 1사이의 값을 입력한다. 0은 아래쪽 끝 1은 위쪽 끝을 의미한다.

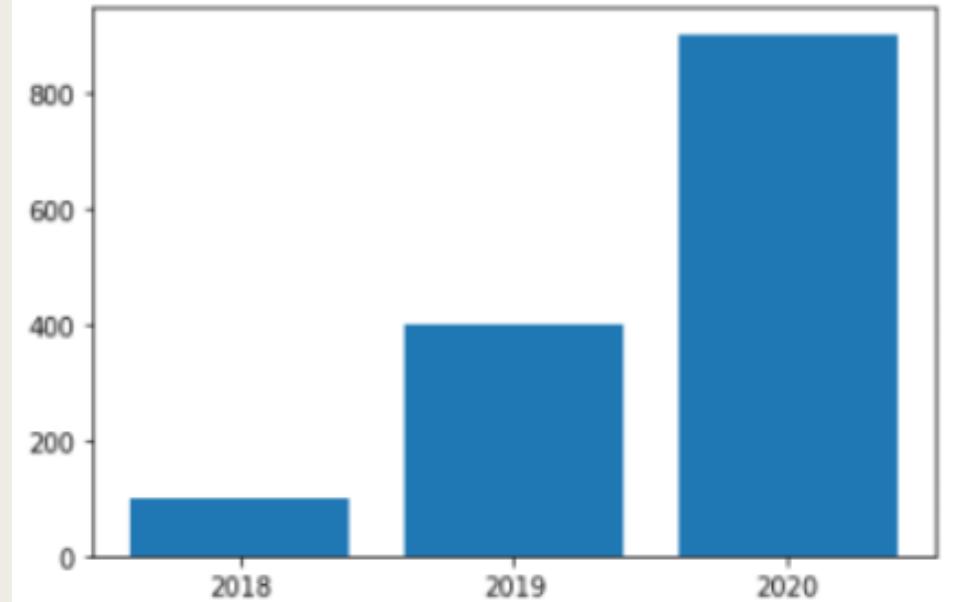


```
plt.hlines(4, 1, 1.6, colors='pink', linewidth=3)
plt.vlines(1, 1, 4, colors='pink', linewidth=3)
plt.show()
```

- hlines()함수에 y, xmin, xmax를 순서대로 입력하면 점 (xmin, y)에서 점(xmax,y)을 따라 수평선을 표시한다.
- vlines()함수에 x, ymin, ymax를 순서대로 입력하면 점 (x, ymin)에서 점(x, ymax)을 따라 수직선을 표시한다.

수직 막대 그래프

```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.arange(3)  
years = ['2018', '2019', '2020']  
values = [100, 400, 900]  
  
plt.bar(x, values)  
plt.xticks(x, years)  
  
plt.show()
```



- 막대 그래프는 (Bar Graph)는 범주가 있는 데이터 값을 직사각형의 막대로 표현하는 그래프이다. `bar()`함수를 사용해서 막대 그래프를 간단하게 표현할 수 있다.
- 연도별로 변화하는 값을 갖는 데이터를 막대그래프로 표현하였다.
- `np.arange()`는 주어진 범위와 간격에 따라 균일한 값을 갖는 어레이를 생성했다.
- `years` 는 x축에 표시될 연도이고 `values`는 막대 그래프의 높이로 표시될 Y 값이다.

```

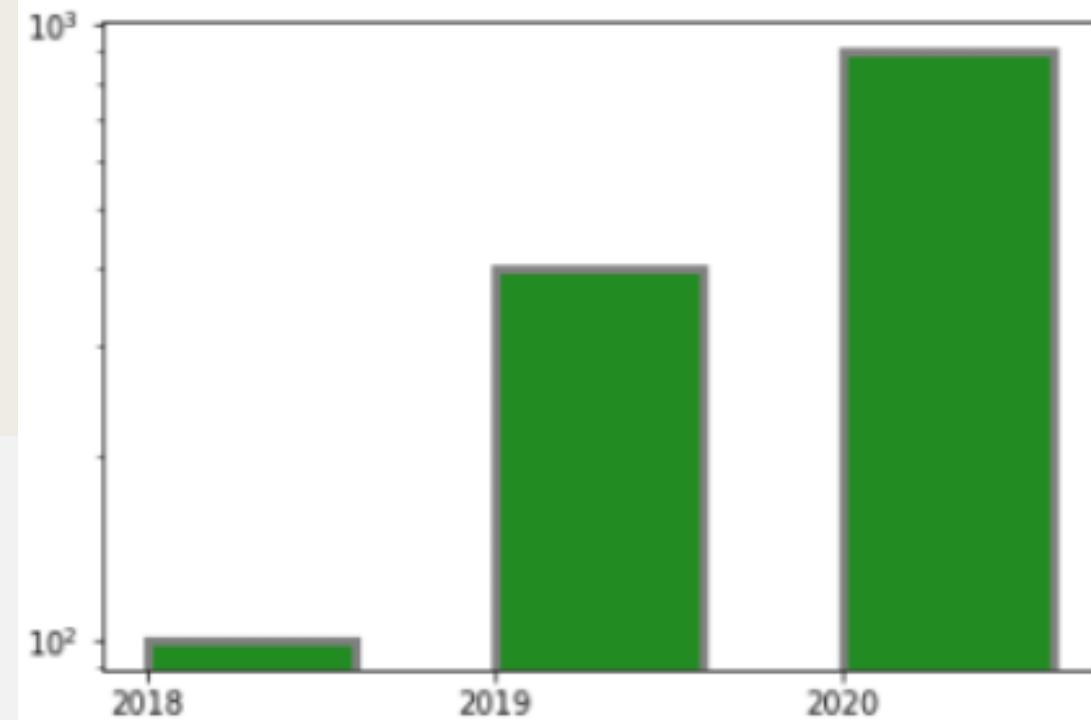
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(3)
years = ['2018', '2019', '2020']
values = [100, 400, 900]

plt.bar(x, values, width=0.6, align='edge', color='forestgreen',
        edgecolor="gray", linewidth=3, tick_label=years, log=True)

plt.show()

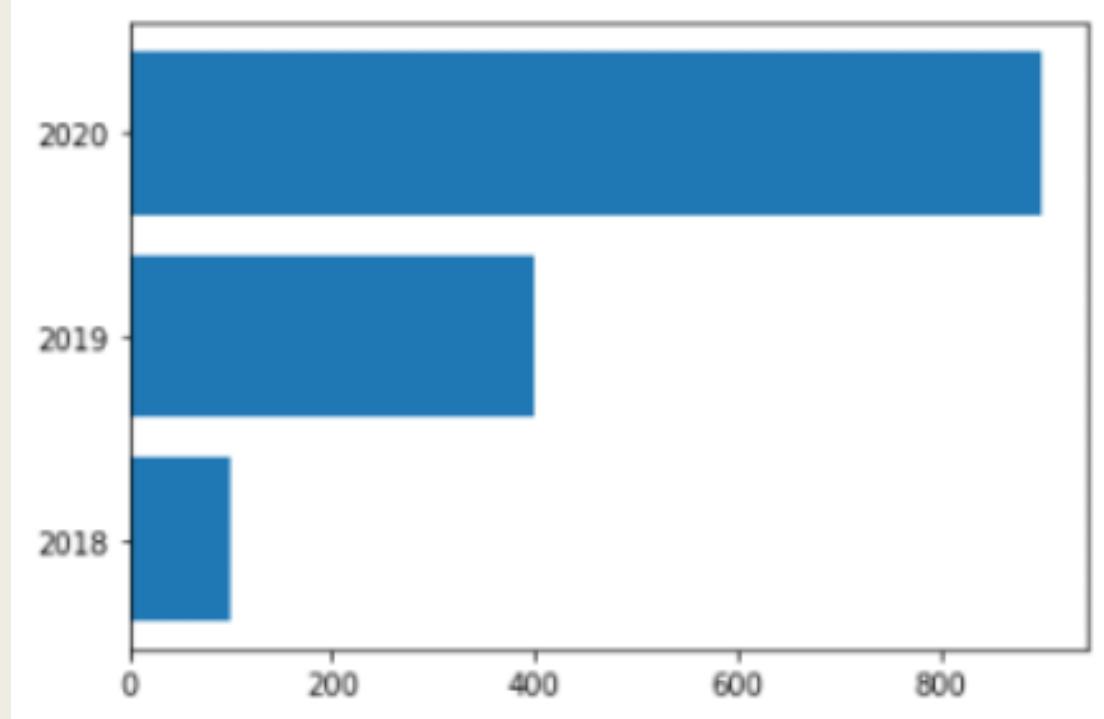
```



- align은 틱 tick 과 막대의 위치를 조정한다. 디폴트 값은 center
- color는 막대의 색을 지정한다. edgecolor는 막대의 테두리색
- linewidth는 테두리의 두께를 지정한다. tick_label 을 어레이로 지정하면 틱에 어레이의 문자열을 나타낸다.
- log=True 로 설정하면 y축이 로그 스케일로 표시된다.

수평 막대 그래프

```
import matplotlib.pyplot as plt  
import numpy as np  
  
y = np.arange(3)  
years = ['2018', '2019', '2020']  
values = [100, 400, 900]  
  
plt.barh(y, values)  
plt.yticks(y, years)  
  
plt.show()
```



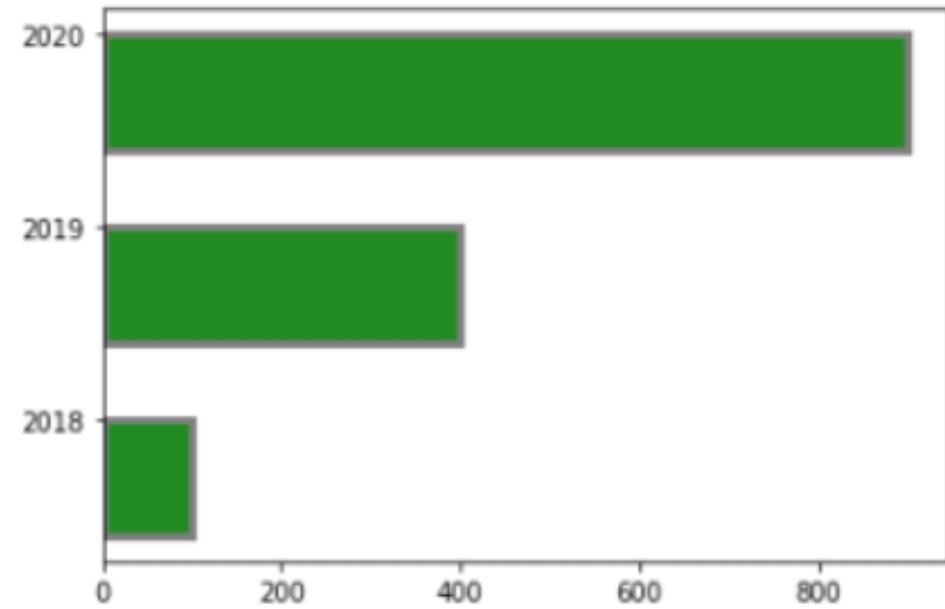
- 수평 막대 그래프 Horizontal bar graph 은 범주가 있는 데이터 값을 수평 막대로 표현하는 그래프이다. barh() 함수를 사용하여 그린다.
- years는 y축에 표시될 연도, values는 막대 그래프의 너비로 표시될 x값이다.

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

y = np.arange(3)
years = ['2018', '2019', '2020']
values = [100, 400, 900]

plt.barh(y, values, height=-0.6, align='edge', color='forestgreen',
         edgecolor="gray", linewidth=3, tick_label=years, log=False)

plt.show()
```



- height 막대의 높이 디폴트는 0.8 이다.
- align 은 tick과 막대의 위치 조절이다. 디폴트는 center 이다.
- color는 막대의 색을 지정하고 edgecolor는 테두리의 색을 지정한다.
- linewidth는 테두리의 두께이고 log = False로 설정하면 x축이 선형 스케일로 표시된다. 디폴트는 False이다.

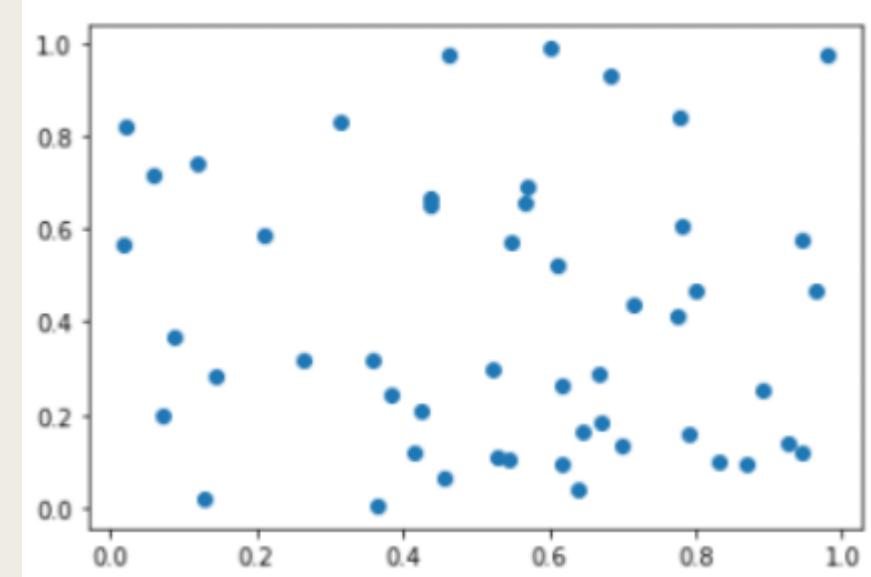
산점도 Scatter plot

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

np.random.seed(0)

n = 50
x = np.random.rand(n)
y = np.random.rand(n)

plt.scatter(x, y)
plt.show()
```



- 산점도는 두 변수의 상관 관계를 직교 좌표계의 평면에 점으로 표현하는 그래프이다. scatter()함수를 사용하여 산점도를 그린다.
- NumPy의 random 모듈로 0,1범위의 난수를 생성하였다.

```

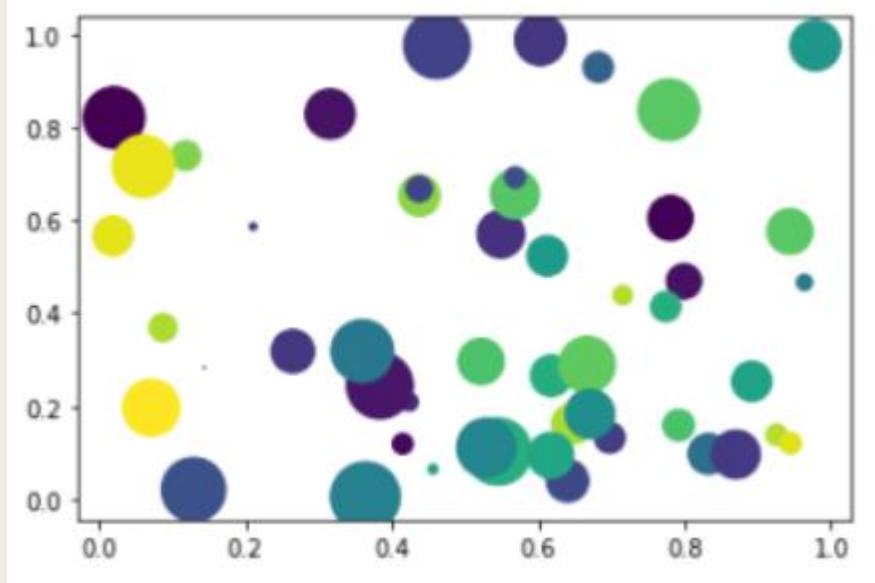
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)

n = 50
x = np.random.rand(n)
y = np.random.rand(n)
area = (30 * np.random.rand(n)) ** 2
colors = np.random.rand(n)

plt.scatter(x, y, s=area, c=colors)
plt.show()

```

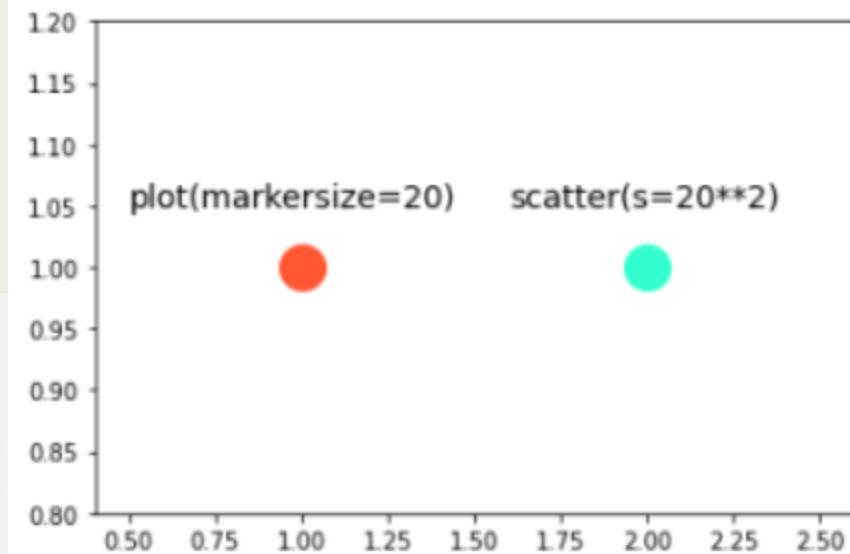


- scatter()함수의 s와 c 파라미터는 각각 마커의 크기와 색상을 지정한다.
- 마커의 크기는 size**2의 형태로 지정한다.
- 마커의 색상은 데이터의 길이와 같은 크기의 숫자 시퀀스 또는 rgb 그리고 hex code색상을 입력해서 지정한다.

```
import matplotlib.pyplot as plt

plt.plot([1], [1], 'o', markersize=20, c='#FF5733')
plt.scatter([2], [1], s=20**2, c='#33FFCE')

plt.text(0.5, 1.05, 'plot(markersize=20)', fontdict={'size': 14})
plt.text(1.6, 1.05, 'scatter(s=20**2)', fontdict={'size': 14})
plt.axis([0.4, 2.6, 0.8, 1.2])
plt.show()
```



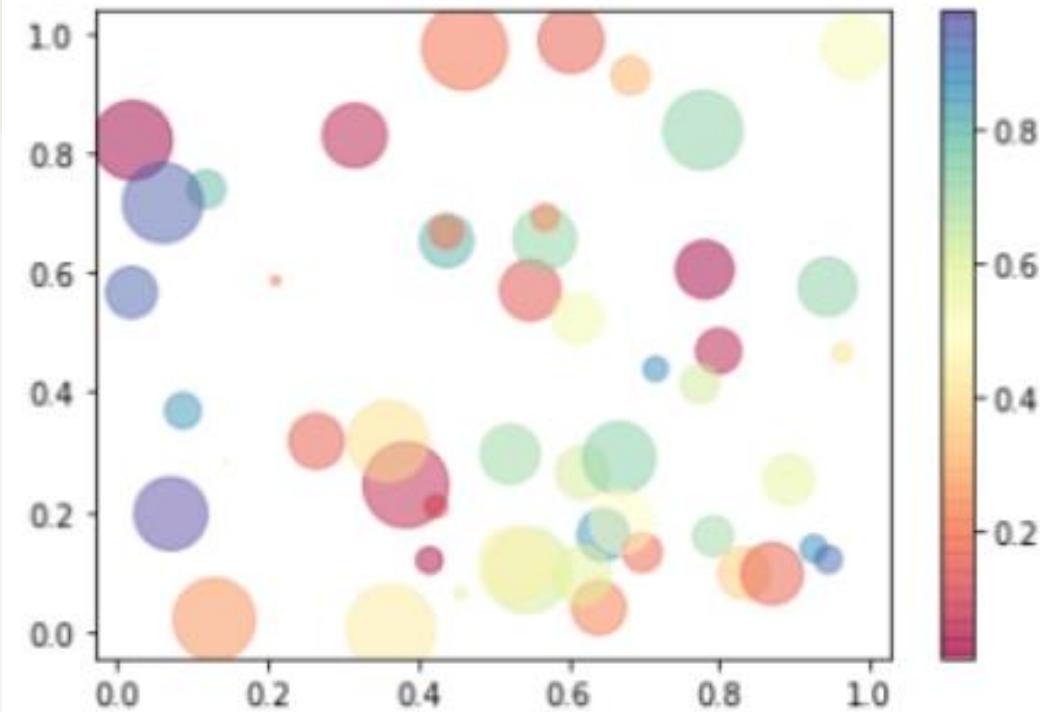
- plot함수의 markersize를 20으로 scatter()함수의 s값을 20^{**2} 으로 지정
- 동일한 크기의 마커가 표시된다.

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

np.random.seed(0)

n = 50
x = np.random.rand(n)
y = np.random.rand(n)
area = (30 * np.random.rand(n))**2
colors = np.random.rand(n)

plt.scatter(x, y, s=area, c=colors, alpha=0.5, cmap='Spectral')
plt.colorbar()
plt.show()
```



- `alpha` 파라미터는 마커의 투명도를 지정한다. 0 ~ 1사이의 값으로 입력한다.
- `cmap` 파라미터는 컬러맵에 해당하는 문자열을 지정할 수 있다.

```

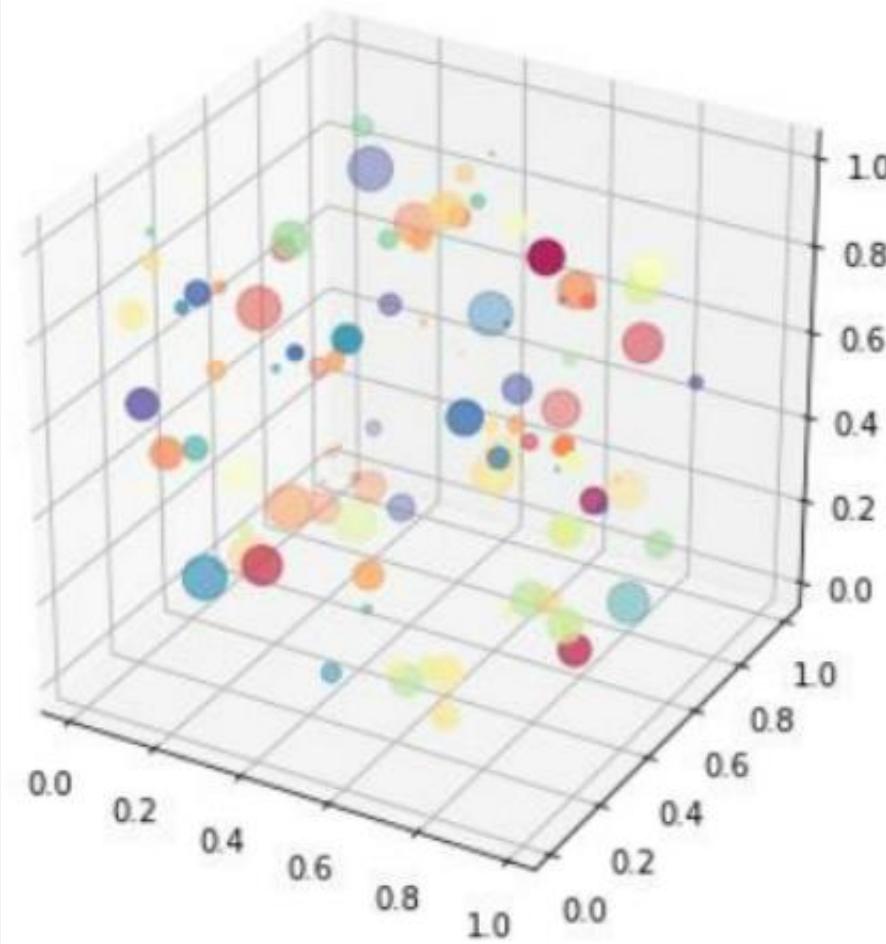
# from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(0)

n = 100
x = np.random.rand(n)
y = np.random.rand(n)
z = np.random.rand(n)
area = (15 * np.random.rand(n)) ** 2
colors = np.random.rand(n)

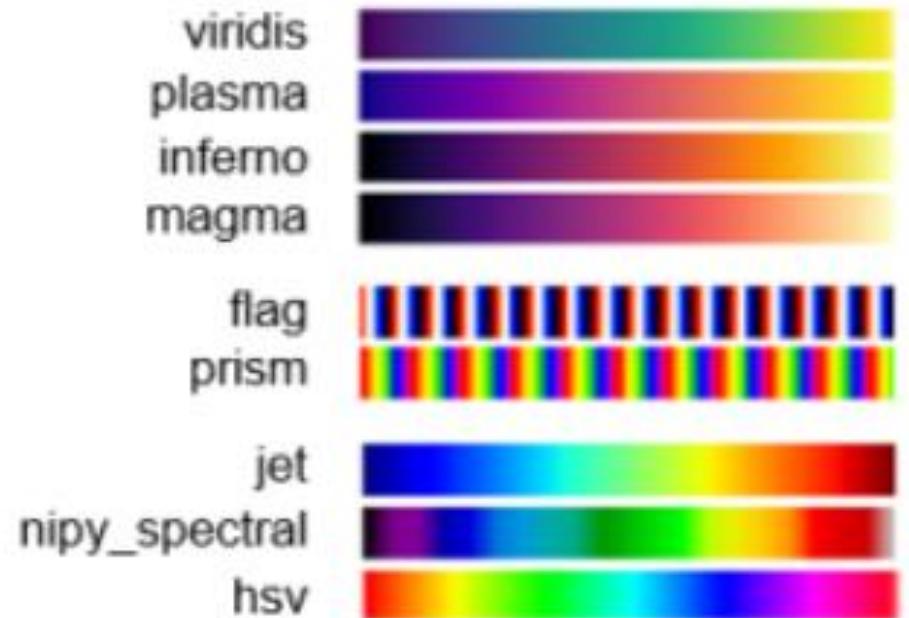
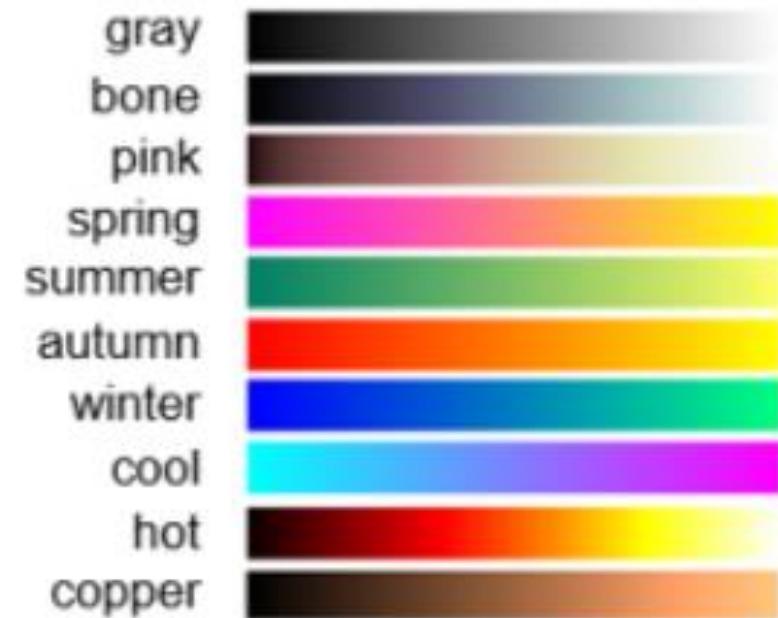
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, s=area, c=colors, cmap='Spectral')
plt.show()

```



- 3차원 그래프를 그리기 위해서 `from mpl_toolkits.mplot3d import Axes3D`를 추가한다.
- Matplotlib 3.1.0버전부터는 기본 포함이기 때문에 추가하지 않아도 동작한다.
- 3d axes를 만들기 위해서 `add_subplot()`에 `projection = '3d'` 키워드를 입력한다.

컬러맵



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

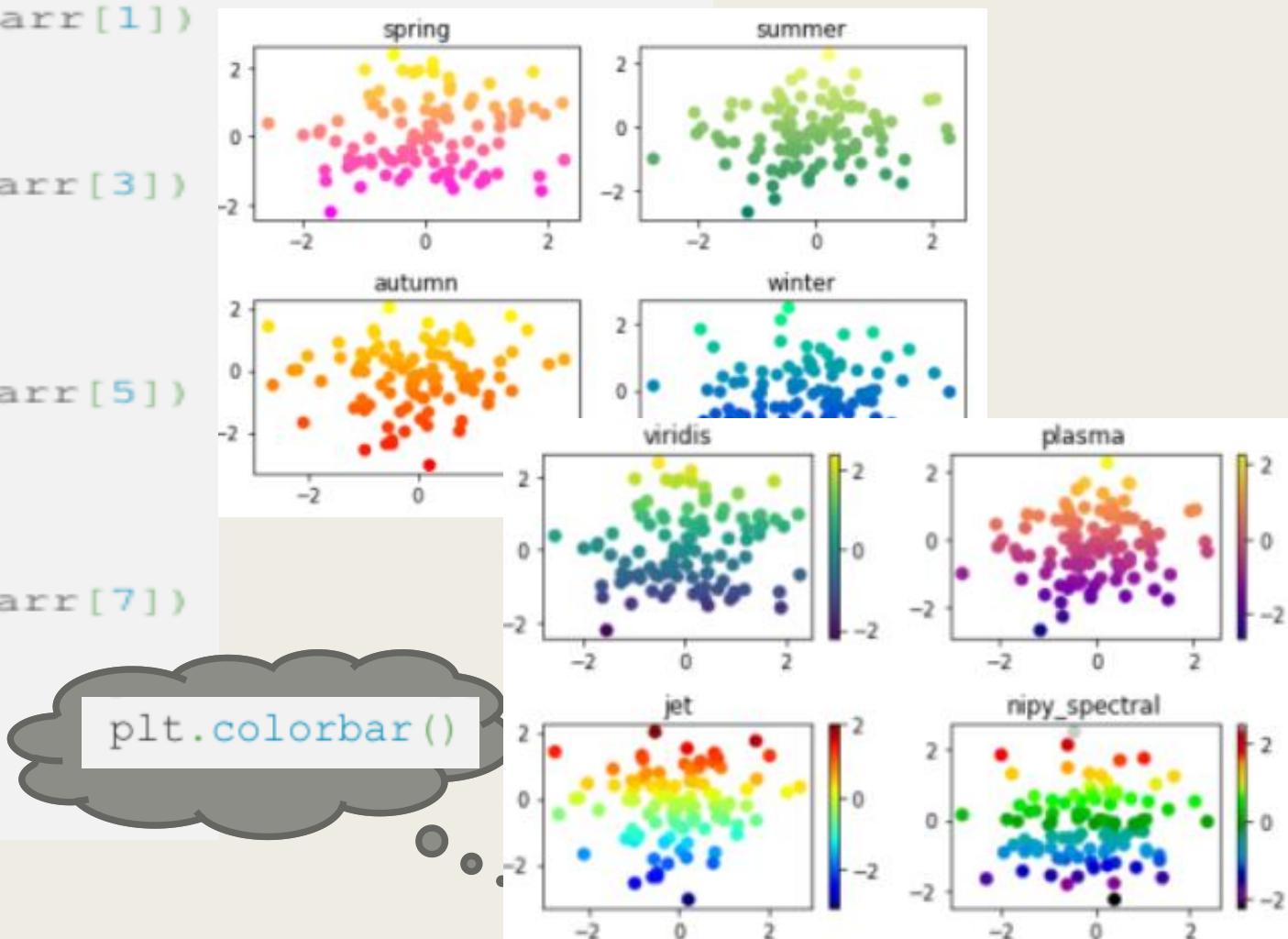
np.random.seed(0)
arr = np.random.standard_normal((8, 100))

plt.subplot(2, 2, 1)
# plt.scatter(arr[0], arr[1], c=arr[1], cmap='spring')
plt.scatter(arr[0], arr[1], c=arr[1])
plt.spring()
plt.title('spring')
plt.subplot(2, 2, 2)
plt.scatter(arr[2], arr[3], c=arr[3])
plt.summer()
plt.title('summer')

plt.subplot(2, 2, 3)
plt.scatter(arr[4], arr[5], c=arr[5])
plt.autumn()
plt.title('autumn')

plt.subplot(2, 2, 4)
plt.scatter(arr[6], arr[7], c=arr[7])
plt.winter()
plt.title('winter')

plt.tight_layout()
plt.show()
```



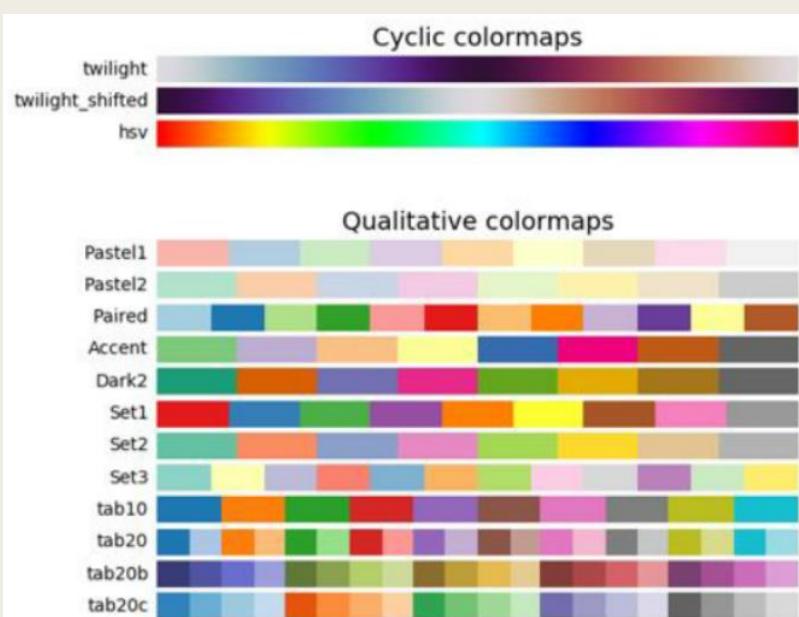
```

import matplotlib.pyplot as plt
from matplotlib import cm

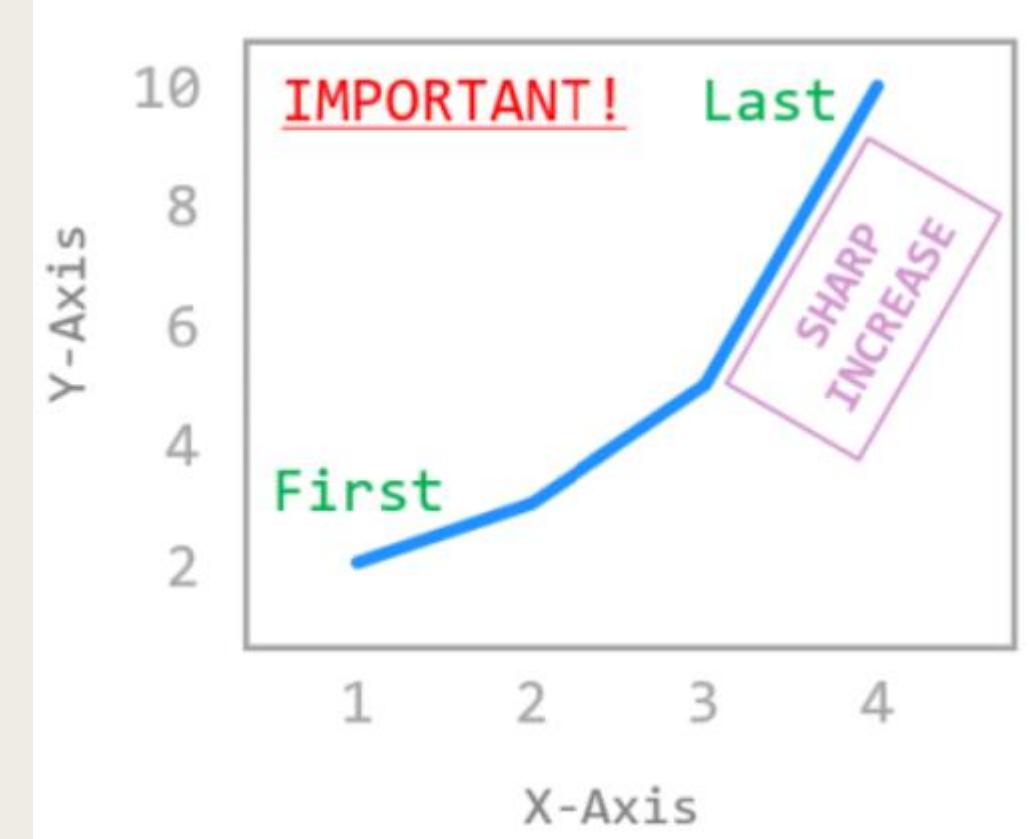
cmmaps = plt.colormaps()
for cm in cmmaps:
    print(cm)

```

- colomaps() 함수를 사용해서 Matplotlib에서 사용할 수 있는 컬러맵의 모든 이름을 얻을 수 있다.
- winter와 winter_r은 순서가 앞뒤로 뒤집어진 컬러맵이다.



텍스트 삽입하기



- `text()`함수는 그래프의 적절한 위치에 텍스트를 삽입하도록 한다.
- `text()`함수를 사용해서 그래프의 영역에 텍스트를 삽입하고 다양하게 꾸미는 방법에 대해 알아보자.

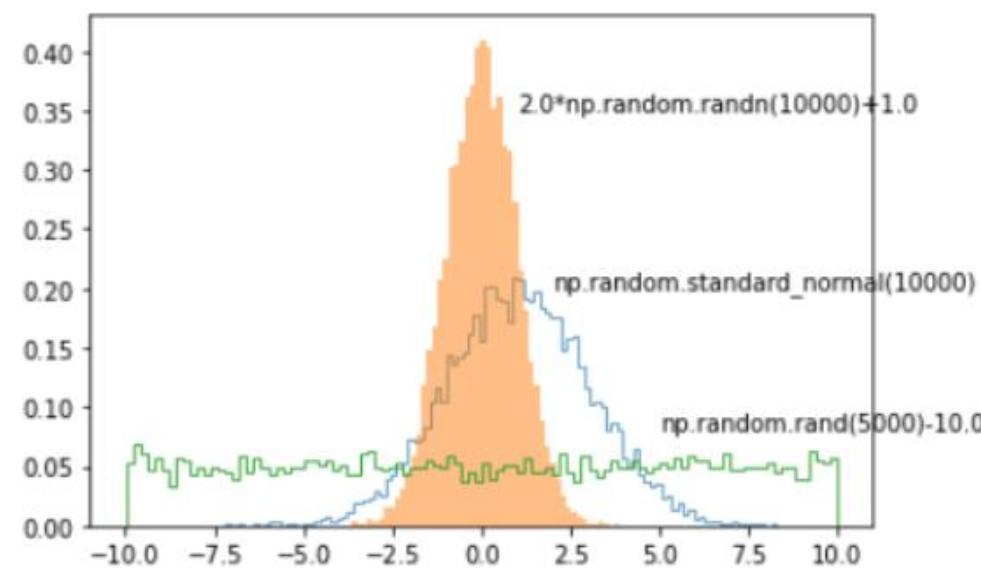
```

import matplotlib.pyplot as plt
import numpy as np

a = 2.0 * np.random.randn(10000) + 1.0
b = np.random.standard_normal(10000)
c = 20.0 * np.random.rand(5000) - 10.0

plt.hist(a, bins=100, density=True, alpha=0.7, histtype='step')
plt.text(1.0, 0.35, '2.0*np.random.randn(10000)+1.0')
plt.hist(b, bins=50, density=True, alpha=0.5, histtype='stepfilled')
plt.text(2.0, 0.20, 'np.random.standard_normal(10000)')
plt.hist(c, bins=100, density=True, alpha=0.9, histtype='step')
plt.text(5.0, 0.08, 'np.random.rand(5000)-10.0')
plt.show()

```



- `text()`함수를 이용해서 3개의 히스토그램 그래프에 설명을 위한 텍스트를 추가한다.
- `text()`함수로 그래프 상의 x위치 y위치 그리고 삽입할 텍스트의 순서대로 입력한다.

```

import matplotlib.pyplot as plt
import numpy as np

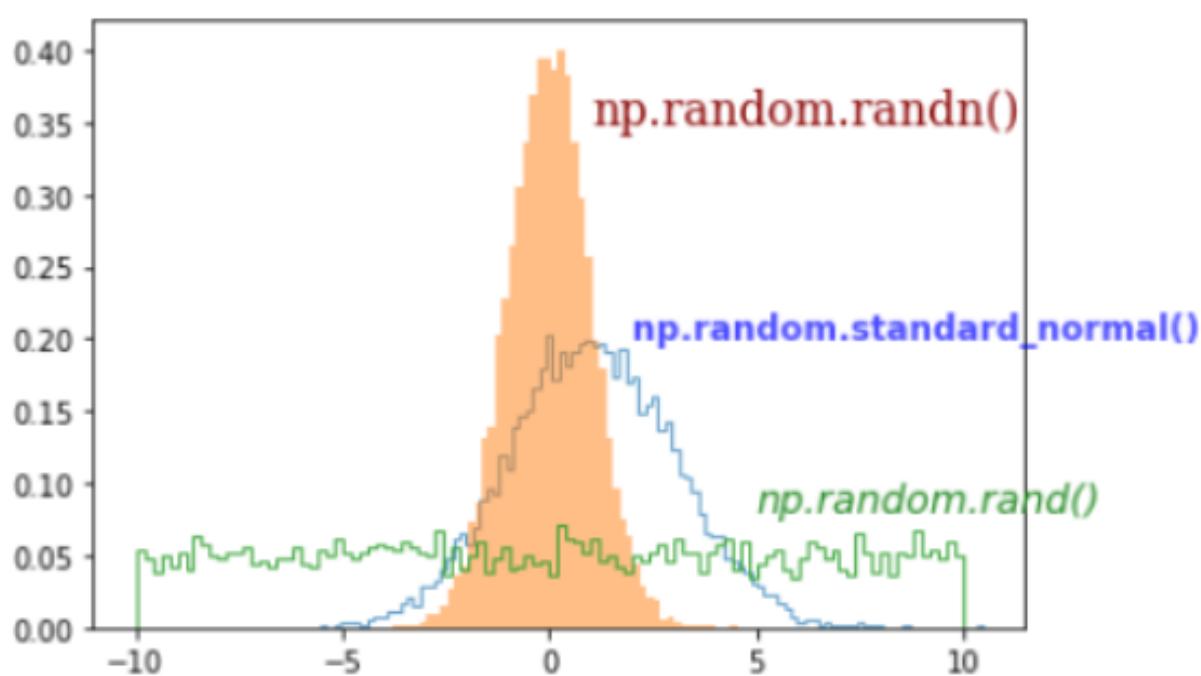
a = 2.0 * np.random.randn(10000) + 1.0
b = np.random.standard_normal(10000)
c = 20.0 * np.random.rand(5000) - 10.0

font1 = {'family': 'serif',
          'color': 'darkred',
          'weight': 'normal',
          'size': 16}
font2 = {'family': 'Times New Roman',
          'color': 'blue',
          'weight': 'bold',
          'size': 12,
          'alpha': 0.7}
font3 = {'family': 'Arial',
          'color': 'forestgreen',
          'style': 'italic',
          'size': 14}

plt.hist(a, bins=100, density=True, alpha=0.7, histtype='step')
plt.text(1.0, 0.35, 'np.random.randn()', fontdict=font1)
plt.hist(b, bins=50, density=True, alpha=0.5, histtype='stepfilled')

plt.text(2.0, 0.20, 'np.random.standard_normal()', fontdict=font2)
plt.hist(c, bins=100, density=True, alpha=0.9, histtype='step')
plt.text(5.0, 0.08, 'np.random.rand()', fontdict=font3)
plt.show()

```

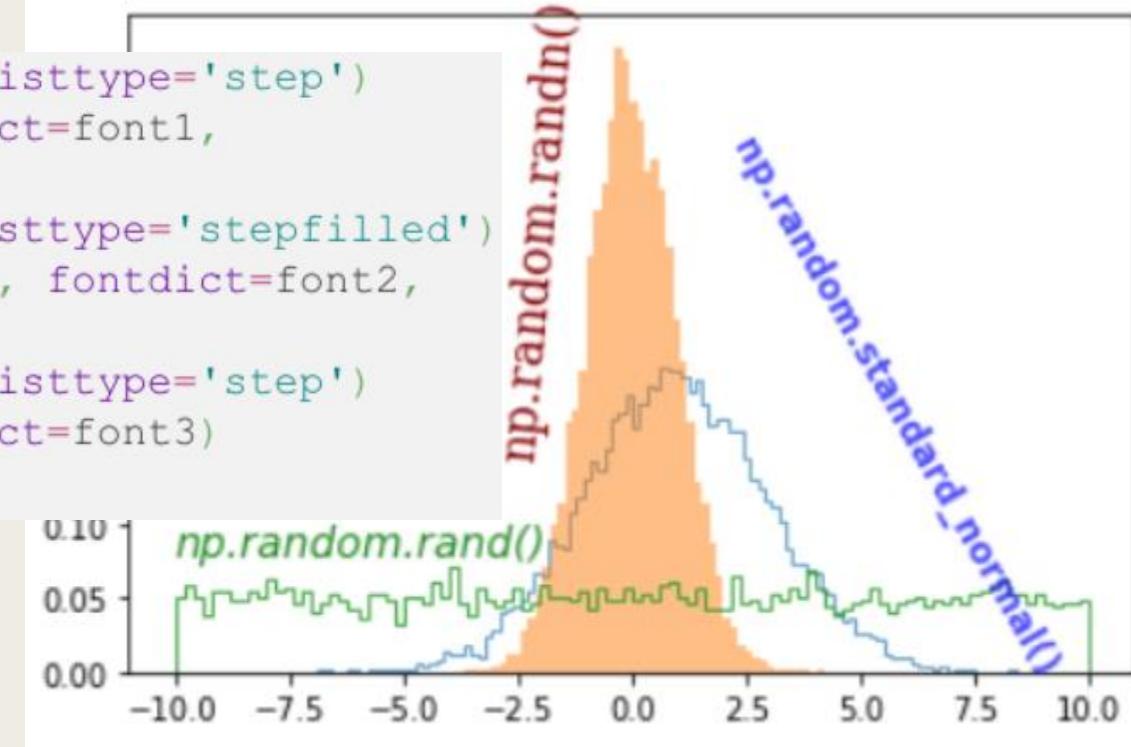


- fontdict 키워드를 이용하면 font의 종류 크기 색상 투명도 weight등의 텍스트 스타일을 설정할 수 있다.
- font1, font2, font3와 같이 미리지정한 폰트 딕셔너리를 입력해 준다.

```

plt.hist(a, bins=100, density=True, alpha=0.7, histtype='step')
plt.text(-3.0, 0.15, 'np.random.randn()', fontdict=font1,
         rotation=85)
plt.hist(b, bins=50, density=True, alpha=0.5, histtype='stepfilled')
plt.text(2.0, 0.0, 'np.random.standard_normal()', fontdict=font2,
         rotation=-60)
plt.hist(c, bins=100, density=True, alpha=0.9, histtype='step')
plt.text(-10.0, 0.08, 'np.random.rand()', fontdict=font3)
plt.show()

```



- rotation 키워드를 이용해서 텍스트를 회전할 수 있다.
- 첫번째 두번째 텍스트를 각각 85도 -65도 만큼 회전 시켰다.

```

plt.hist(a, bins=100, density=True, alpha=0.7, histtype='step')
plt.text(-3.0, 0.15, 'np.random.randn()', fontdict=font1,
         rotation=85, bbox=box1)
plt.hist(b, bins=50, density=True, alpha=0.5, histtype='stepfilled')
plt.text(2.0, 0.0, 'np.random.standard_normal()', fontdict=font2,
         rotation=-60, bbox=box2)
plt.hist(c, bins=100, density=True, alpha=0.9, histtype='step')
plt.text(-10.0, 0.08, 'np.random.rand()', fontdict=font3, bbox=box3)

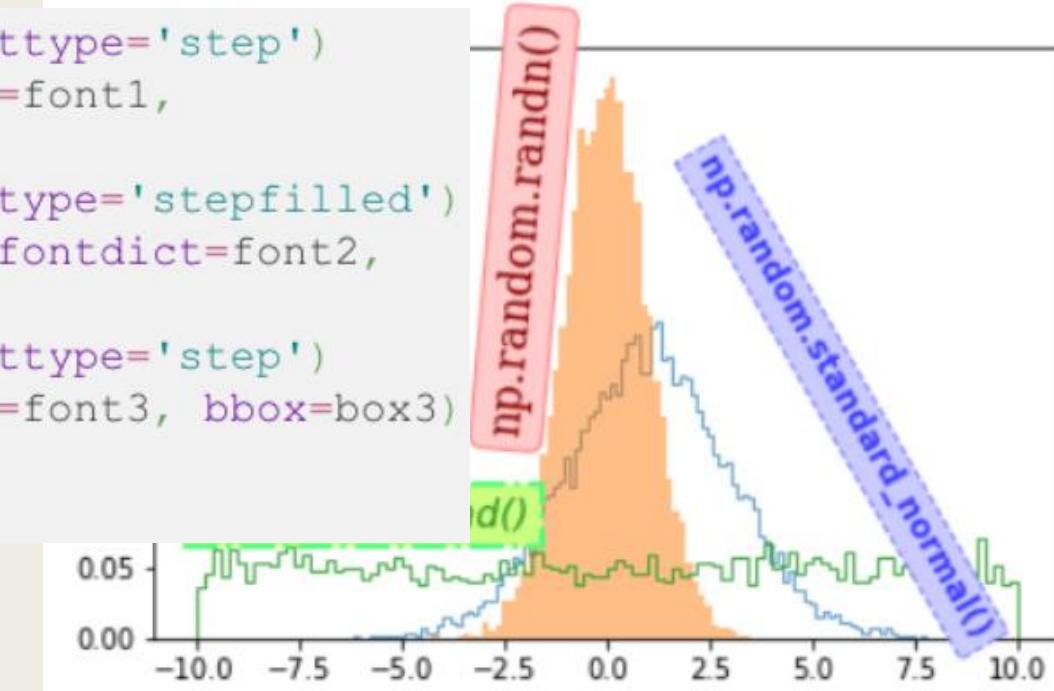
plt.show()

```

```

box1 = {'boxstyle': 'round',
        'ec': (1.0, 0.5, 0.5),
        'fc': (1.0, 0.8, 0.8)}
box2 = {'boxstyle': 'square',
        'ec': (0.5, 0.5, 1.0),
        'fc': (0.8, 0.8, 1.0),
        'linestyle': '--'}
box3 = {'boxstyle': 'square',
        'ec': (0.3, 1.0, 0.5),
        'fc': (0.8, 1.0, 0.5),
        'linestyle': '-.',
        'linewidth': 2}

```



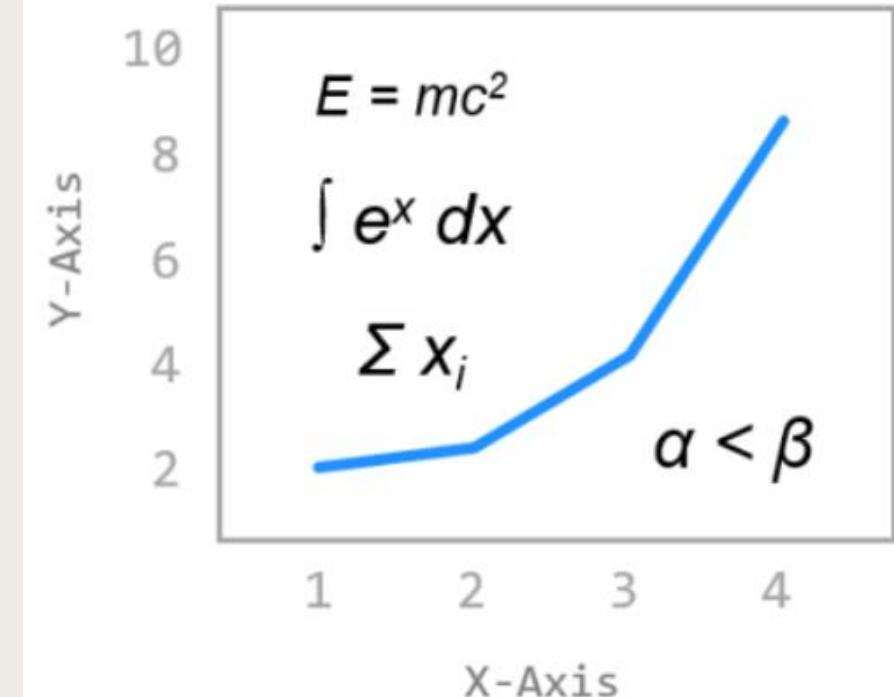
- bbox키워드를 사용해서 텍스트 상자의 스타일을 설정할 수 있다.
- ec 는 edgecolor , fc 는 facecolor와 같다. 각각 텍스트 상자의 테두리와 면의 색을 지정하는 속성이다.

```

ax.text(0.5, 0.8, 'Test', color='red',
       bbox=dict(facecolor='none', edgecolor='red'))

```

수학적 표현 사용



- 달러 기호 \$ 사이에 위치하는 TeX 마크업 표현을 통해 그래프 제목, 축 레이블 그리고 데이터 곡선을 설명하는 텍스트 상자에서도 수학적 표현을 사용할 수 있다.
- 그리스 문자, 위첨자, 아래 첨자, 분수, 거듭제곱, 액센트, 표준함수, 대형 기호등을 사용한다.

r for raw string 달러 기호 (dollar sign)

r '\$\alpha > \beta\$'

TeX 마크업

- 문자열에 수학적 표현을 사용하기 위해서는 세 가지 표현이 필요하다.
- ‘r’은 파이썬 문자열을 raw string으로 표현하도록 한다.
- 수학적 표현은 두 개의 달러 기호 \$ 사이에 위치하도록 한다.
- TeX 마크업 언어를 사용해서 각각의 수학적 표현과 기호를 사용한다.

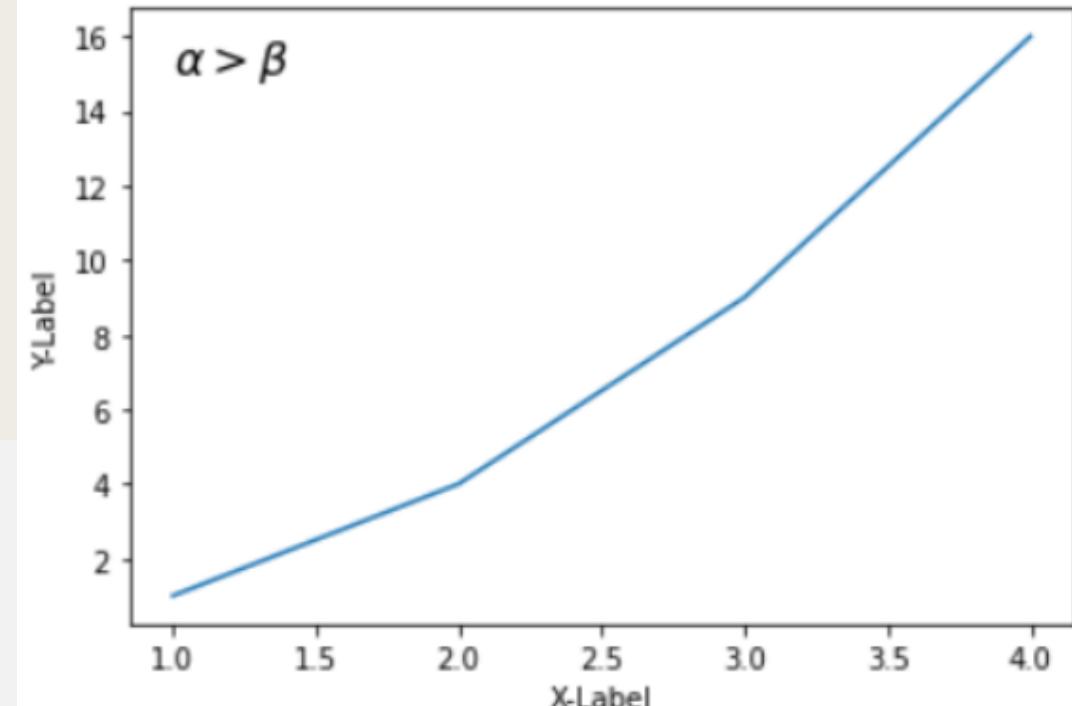
```

import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\alpha > \beta$', fontdict={'size': 16})

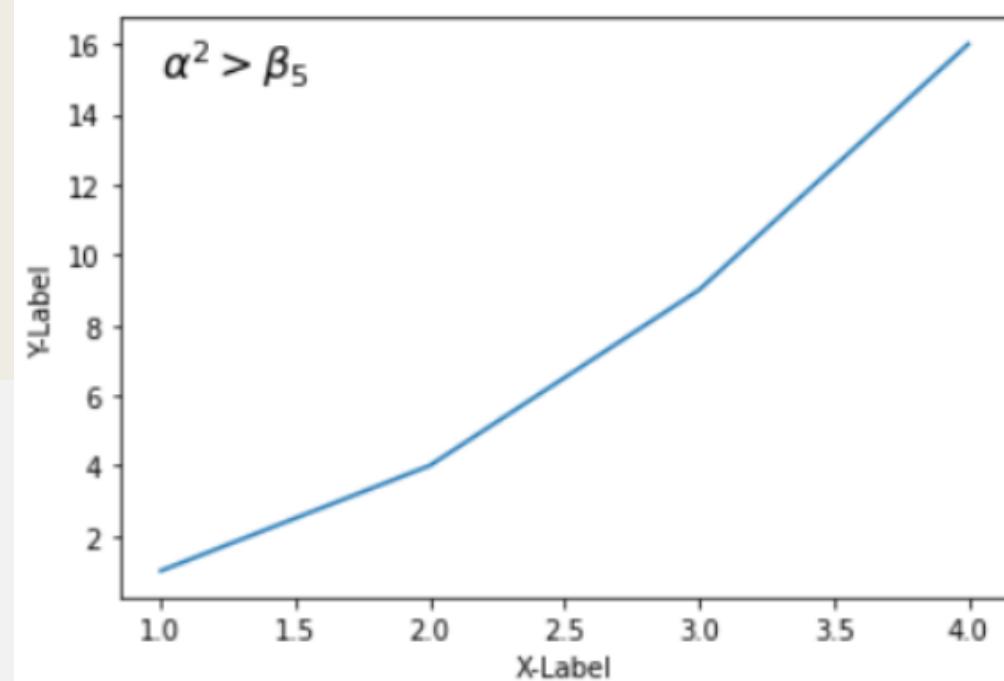
plt.show()

```



- `text()`함수를 사용해서 $x=1, y=15$ 위치에 그리스 문자를 포함하는 문자열을 삽입했다.
- 그래프에 그리스 문자로 α, β (`alpha`와 `beta`)가 삽입된다.

$$\begin{array}{ccc} a^{\textcolor{green}{\wedge}} b & \rightarrow & a^b \\ a_{\textcolor{green}{_}} b & \rightarrow & a_b \end{array}$$



```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\alpha^2 > \beta_5$', fontdict={'size': 16})

plt.show()
```

- 위 첨자는 '^', 아래첨자는 '_' 기호를 사용한다.

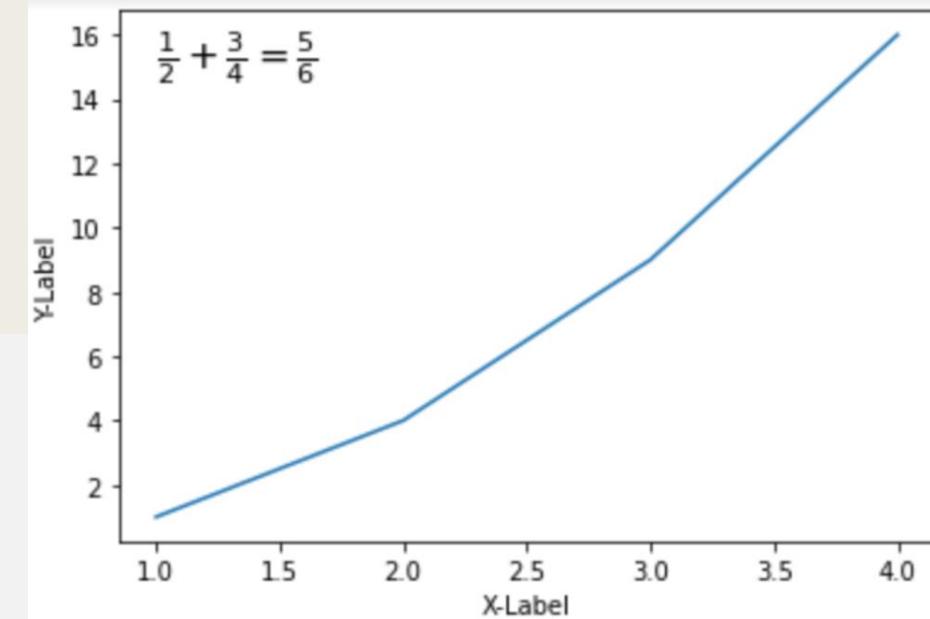
$$\frac{a}{b}$$

$$\frac{a + \frac{1}{x}}{b}$$

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\frac{1}{2} + \frac{3}{4} = \frac{5}{6}$',
         fontdict={'size': 16})

plt.show()
```



- 분수 표현을 사용하기 위해서는 `\frac{}{}` 표현을 사용한다.

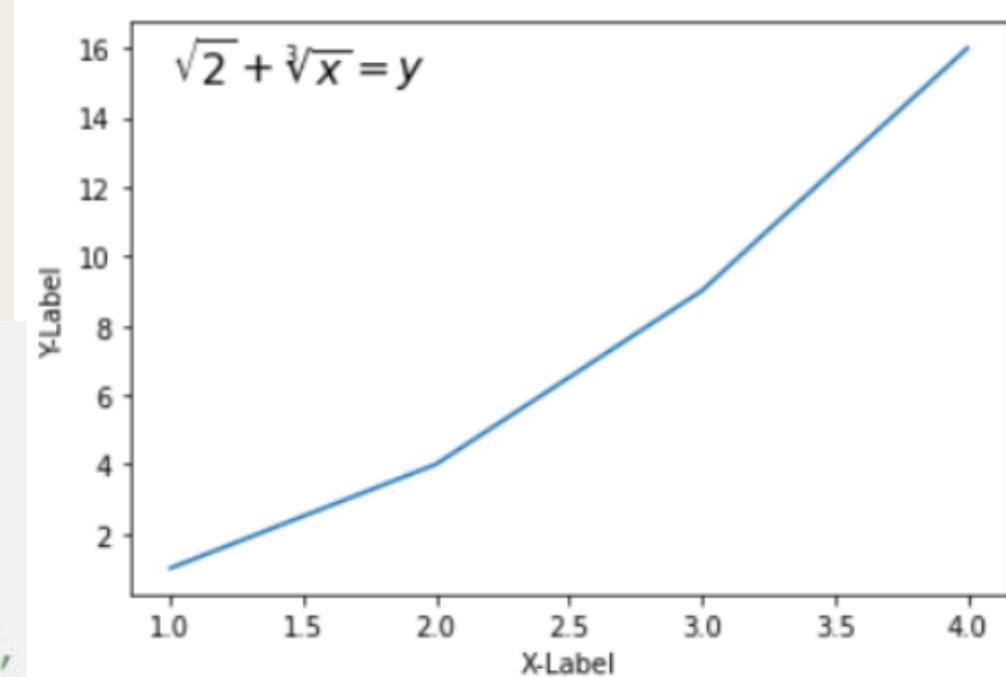
$$\sqrt{2}$$

$$\sqrt[3]{x}$$

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\sqrt{2} + \sqrt[3]{x} = y$',  
        fontdict={'size': 16})

plt.show()
```



- 거듭제곱 근호를 표현하기 위해서는 `\sqrt{}` 또는 `\sqrt[]{}`를 사용한다.

| | | | |
|-----------|---|---|---|
| \acute{a} | a | → | á |
| \bar{a} | a | → | ā |
| \ddot{a} | a | → | ä |
| \hat{a} | a | → | â |

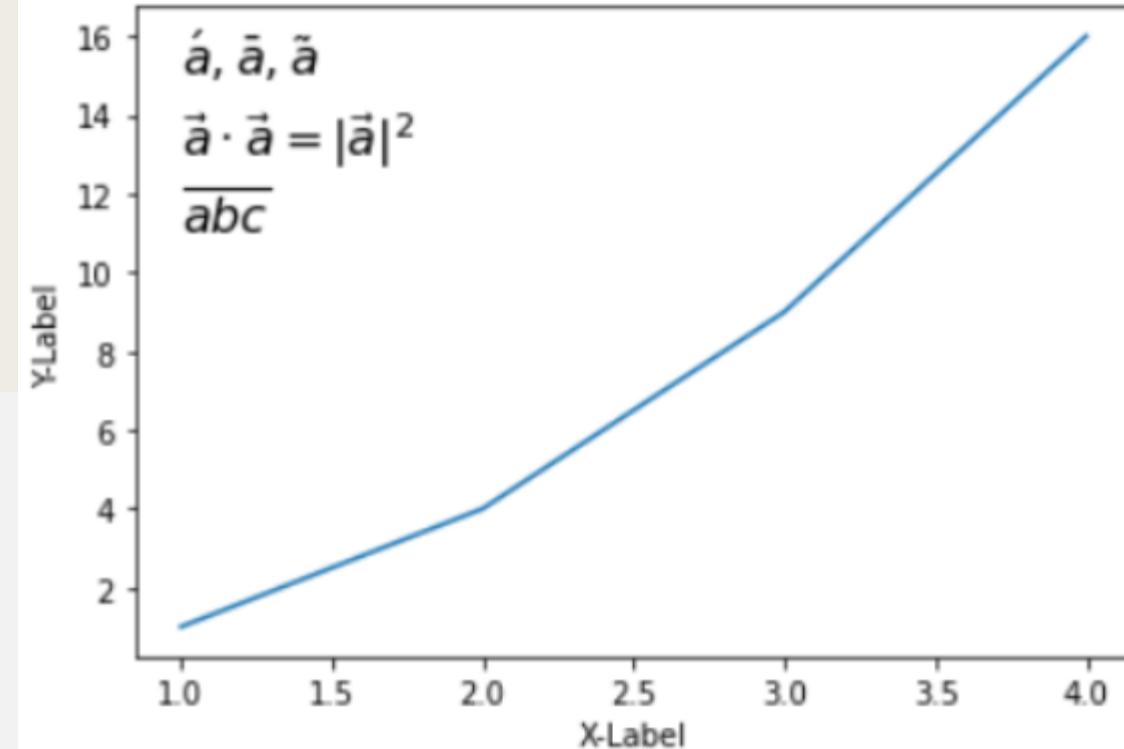
```

import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\acute{a}, \bar{a}, \ddot{a}$',
         fontdict={'size': 16})
plt.text(1, 13, r'$\vec{a} \cdot \vec{a} = |\vec{a}|^2$',
         fontdict={'size': 16})
plt.text(1, 11, r'$\overline{abc}$', fontdict={'size': 16})

plt.show()

```

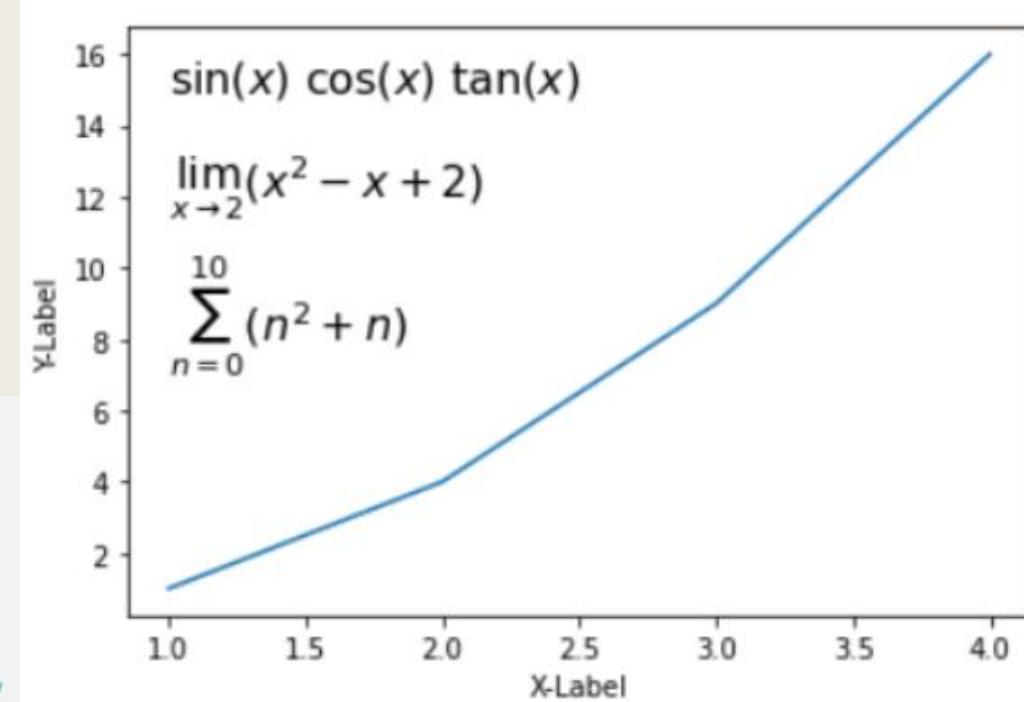


- 액센트는 강세, 억양을 나타내거나 수학적 표현을 위해 문자의 위에 표시되는 기호이다.

| | | |
|-------------------------|---------------|--------------|
| $\backslash \sin(a)$ | \rightarrow | $\sin(a)$ |
| $\backslash \lim_a b$ | \rightarrow | $\lim_a b$ |
| $\backslash \sum_a^b x$ | \rightarrow | $\sum_a^b x$ |

```
import matplotlib.pyplot as plt

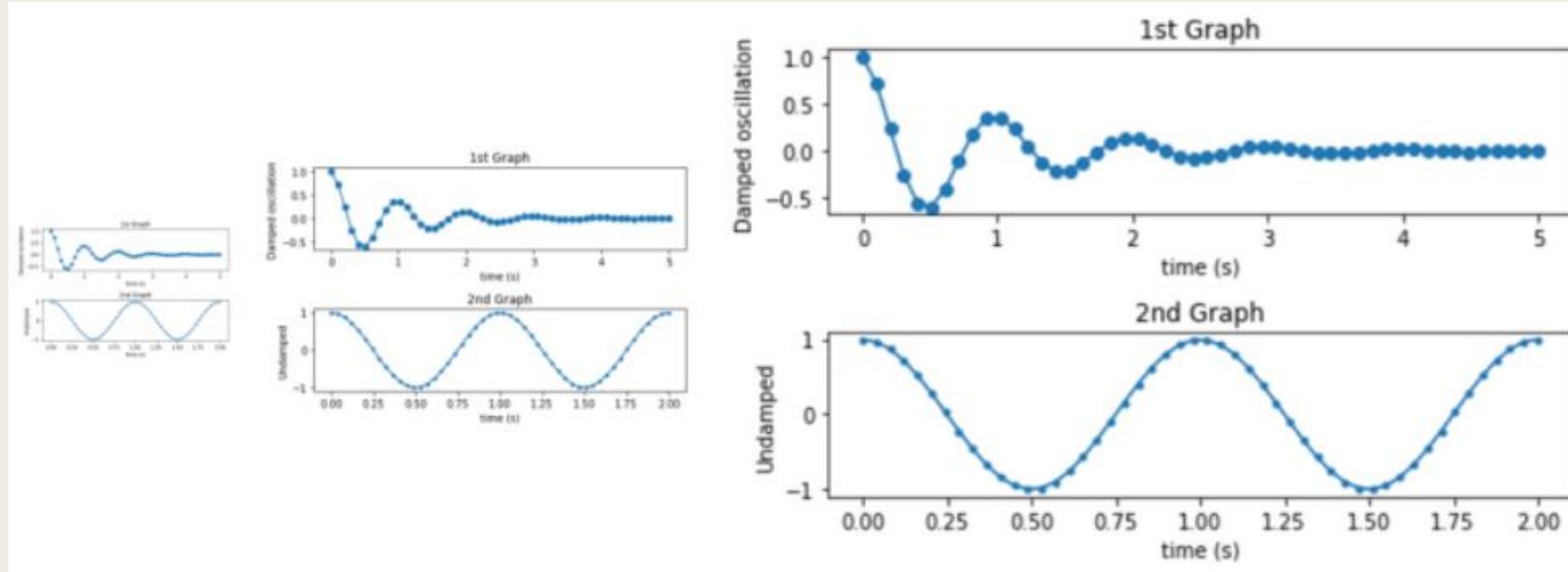
plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.xlabel('X-Label')
plt.ylabel('Y-Label')
plt.text(1, 15, r'$\sin(x) \cos(x) \tan(x)$',
        fontdict={'size': 16})
plt.text(1, 12, r'$\lim_{x \rightarrow 2} (x^2 - x + 2)$',
        fontdict={'size': 16})
plt.text(1, 8, r'$\sum_{n=0}^{10} (n^2 + n)$',
        fontdict={'size': 16})
plt.show()
```



- 그래프에 여러가지 삼각함수와 극한기호 그리고 합 기호 등을 표시 할 수 있다.

| Command | Result | Command | Result | Command | Result | Command | Result |
|------------|--------------|----------------|------------------|---------------|-----------------|-----------------|-------------------|
| \bigcap | \bigcap | \bigcup | \bigcup | \bigodot | \odot | \bigoplus | \oplus |
| \bigotimes | \bigotimes | \biguplus | \biguplus | \bigvee | \bigvee | \bigwedge | \bigwedge |
| \coprod | \coprod | \int | \int | \oint | \oint | \prod | \prod |
| \sum | \sum | | | | | | |
| Command | Result | Command | Result | Command | Result | Command | Result |
| \Pr | \Pr | \arccos | \arccos | \arcsin | \arcsin | \arctan | \arctan |
| \arg | \arg | \cos | \cos | \cosh | \cosh | \cot | \cot |
| \coth | \coth | \csc | \csc | \deg | \deg | \det | \det |
| \dim | \dim | \exp | \exp | \gcd | \gcd | \hom | \hom |
| \inf | \inf | \ker | \ker | \lg | \lg | \lim | \lim |
| \liminf | \liminf | \limsup | \limsup | \ln | \ln | \log | \log |
| \max | \max | \min | \min | \sec | \sec | \sin | \sin |
| \sinh | \sinh | \sup | \sup | \tan | \tan | \tanh | \tanh |
| Command | Result | Command | Result | Command | Result | Command | Result |
| \acute{a} | \acute{a} | \bar{a} | \bar{a} | \breve{a} | \breve{a} | \ddot{a} | \ddot{a} |
| \dot{a} | \dot{a} | \grave{a} | \grave{a} | \hat{a} | \hat{a} | \tilde{a} | \tilde{a} |
| \vec{a} | \vec{a} | \overline{abc} | \overline{abc} | \widehat{abc} | \widehat{abc} | \widetilde{abc} | \widetilde{abc} |

그림 저장



```
plt.savefig('savefig_default.png')
plt.savefig('savefig_default.png')
plt.savefig('savefig_50dpi.png', dpi=50)
plt.savefig('savefig_200dpi.png', dpi=200)
```

- `savefig()`를 사용해서 그림을 이미지 파일로 저장할 수 있다.
- 저장시 `dpi` (dots per inch) 옵션을 사용해서 이미지의 해상도를 설정한다.
- 디폴트는 `dpi = 100`

```
plt.figure(linewidth=2)
```

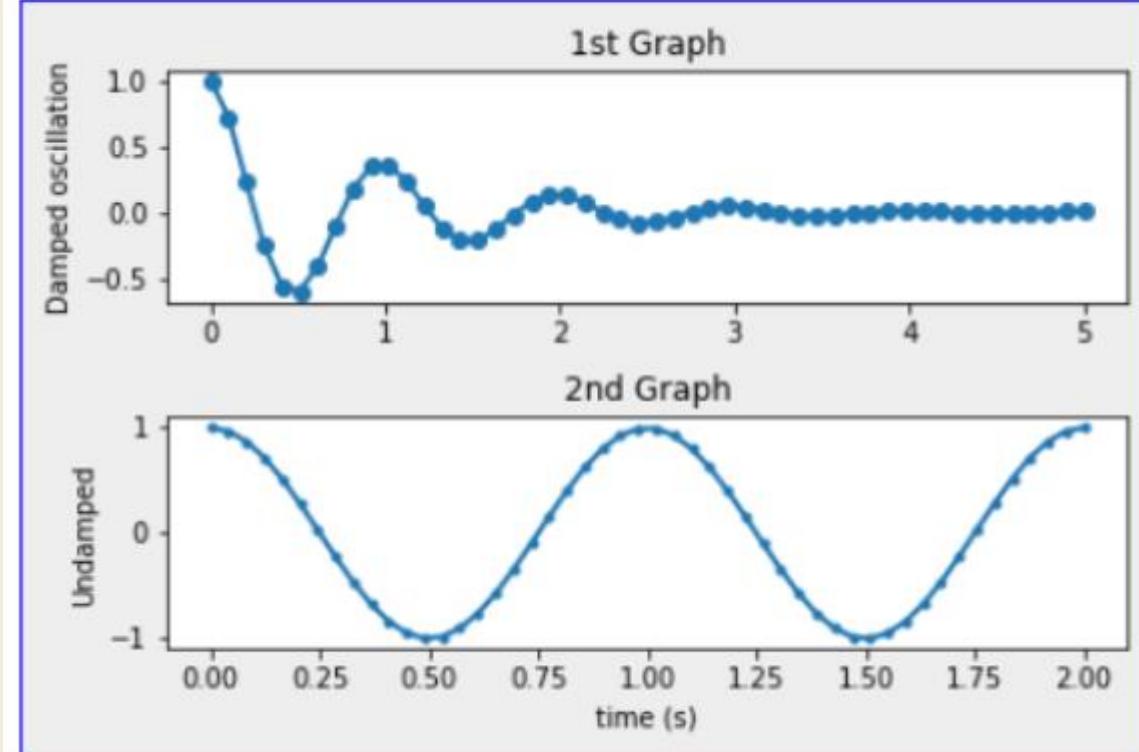
```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)

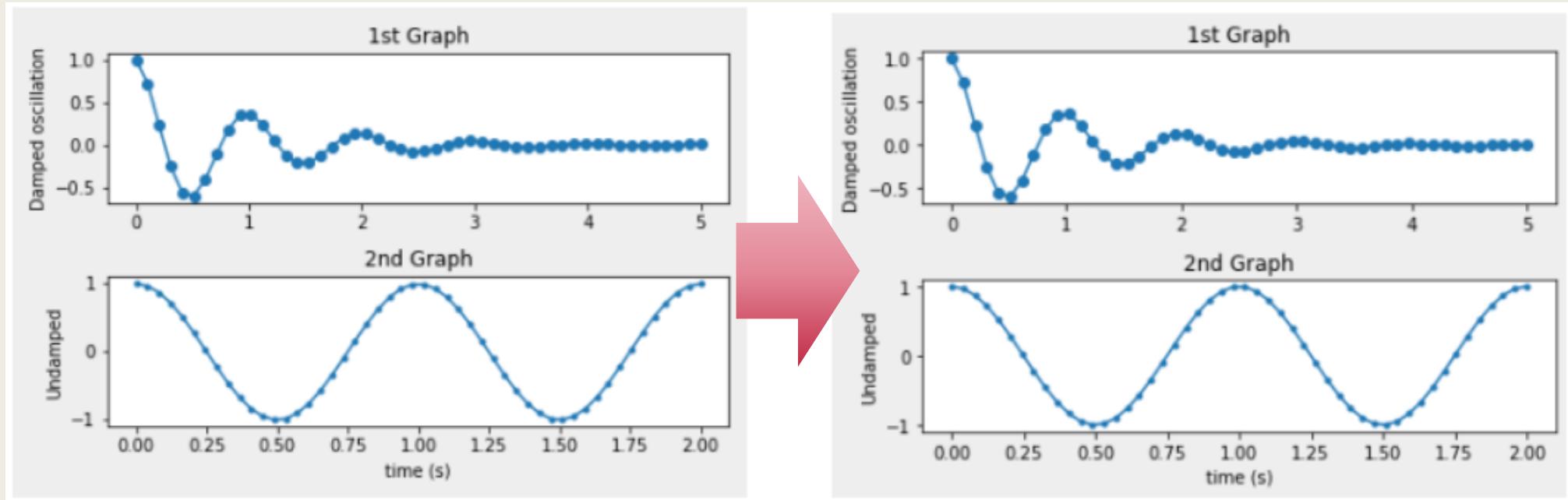
plt.subplot(2, 1, 1) # nrows=2, ncols=1, index=1
plt.plot(x1, y1, 'o-')
plt.title('1st Graph')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2) # nrows=2, ncols=1, index=2
plt.plot(x2, y2, '-.')
plt.title('2nd Graph')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
```

```
plt.savefig('savefig_edgecolor.png', facecolor='eeeeee',
            edgecolor='blue')
```

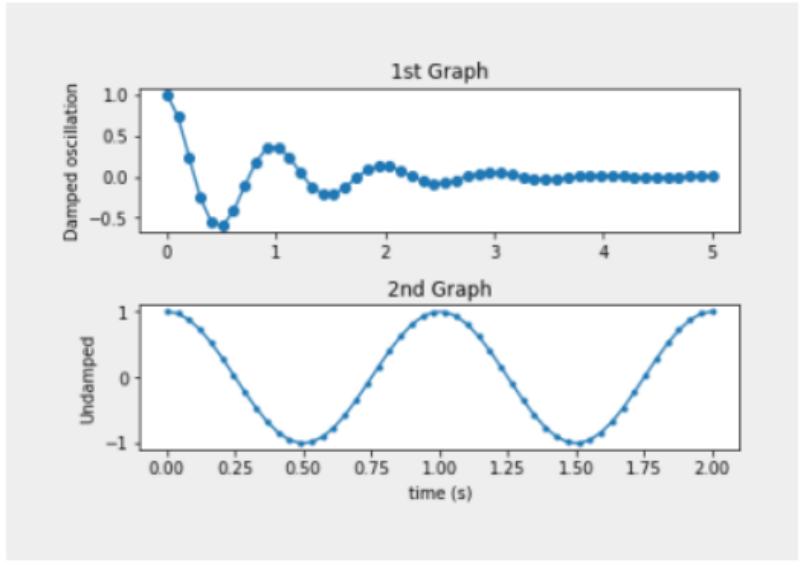
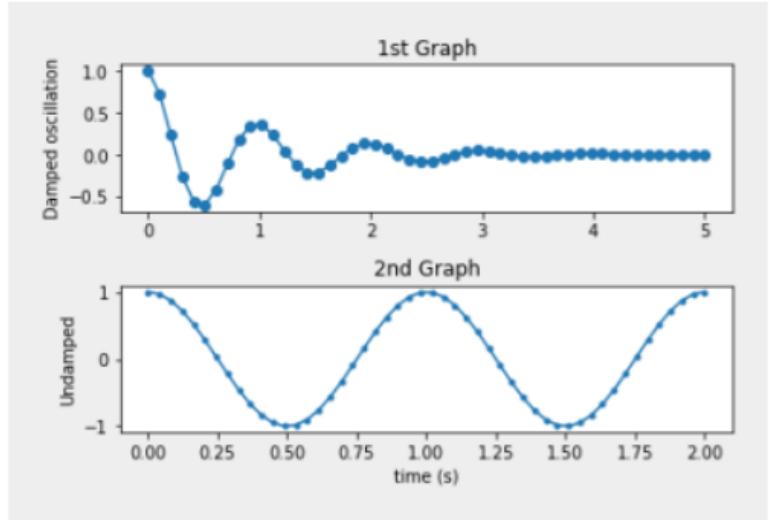
- `edgecolor`는 이미지의 테두리선의 색상을 설정한다.
- 테두리선의 너비가 기본적으로는 0이기 때문에 먼저 코드의 상단 부분에서 `linewidth = 2`로 설정해 준 다음 테두리 색을 지정한다.





```
plt.savefig('savefig_bbox_inches2.png', facecolor='eeeeee',
bbox_inches='tight')
```

- bbox_inches 는 그래프로 저장할 영역을 설정한다.
- tight을 지정하면 여백을 최소화하고 그래프 영역만 저장한다.



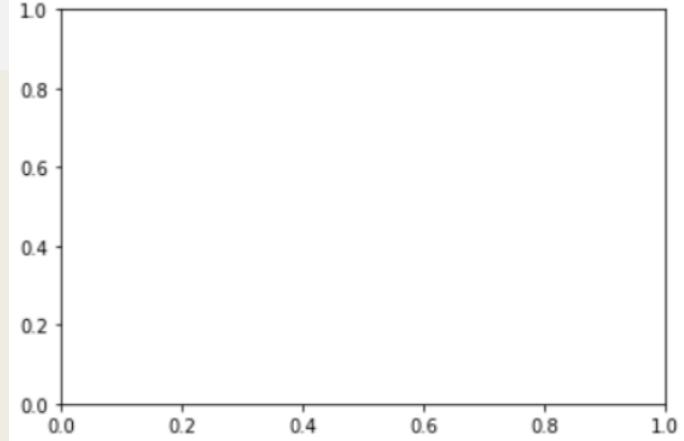
```
plt.savefig('savefig_pad_inches.png', facecolor='#eeeeee',
            bbox_inches='tight', pad_inches=0.3)
plt.savefig('savefig_pad_inches2.png', facecolor='#eeeeee',
            bbox_inches='tight', pad_inches=0.5)
```

- bbox_inches='tight'로 지정하면 pad_inches를 함께 사용해서 여백 paddin을 지정할 수 있다.
- pad_inches의 디폴트 값은 0.1이며 0.3과 0.5로 지정했을 때의 결과를 위 그림에서 확인 할 수 있다.

객체 지향 인터페이스

- 지금까지 matplotlib.pyplot 모듈의 다양한 함수들을 이용해서 간편하게 그래프를 작성했다.
- Matplotlib는 그래프를 다루는 두 가지 인터페이스를 제공한다.
 - 첫 번째가 `pyplot` 모듈을 사용하는 방식이고
 - 두 번째는 객체 지향 인터페이스이다.
- 더욱 커스터마이즈 된 그래프를 표현하기 위해 객체 지향 인터페이스를 사용할 수 있다.

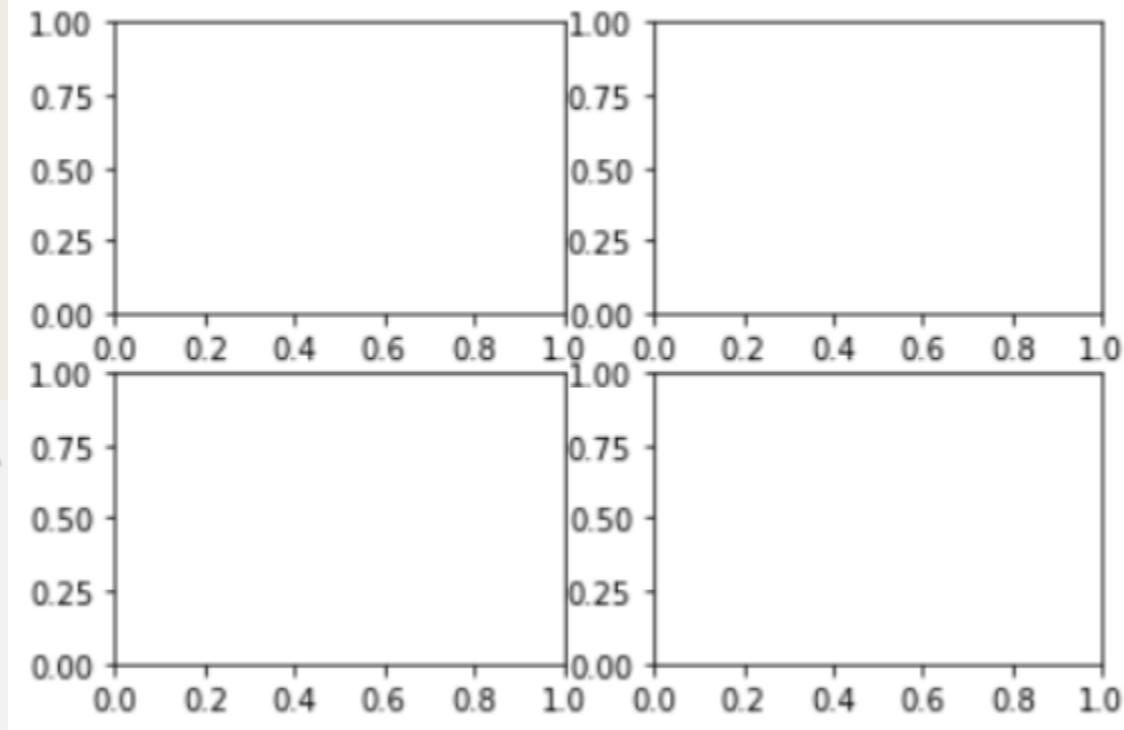
```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots()  
plt.show()
```



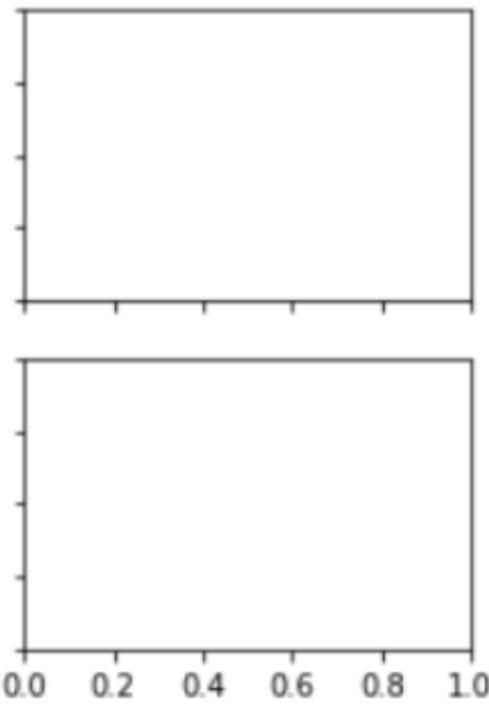
```
import matplotlib.pyplot as plt  
  
# fig, ax = plt.subplots()  
fig = plt.figure()  
ax = fig.add_axes([0, 0, 1, 1])  
plt.show()
```

- subplot()함수를 호출하면 figure(fig)와 subplot(ax)객체를 생성해서 튜플 형태로 반환한다.
- plt.figure()는 Figure클래스의 인스턴스를 반환한다.
- Figure 클래스의 인스턴스 fig의 메서드 add_axes()는 axes를 하나 추가한다.
- add_axes([left, bottom, width, height]) 의 형태로 0에서 1 사이의 값을 입력하면 그 값에 따라 이미지의 크기가 결정된다.
- 왼쪽 형태의 기본형 문법을 주로 사용한다.

```
import matplotlib.pyplot as plt  
  
fig, ax = plt.subplots(2, 2)  
plt.show()
```



- plt.subplots(행, 열) 형태로 행과 열의 개수를 지정할 수 있다.
- 만약 지정하지 않으면 기본적으로 행과 열의 개수는 모두 1이다.



```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(2, 2, sharex=True, sharey=True)
plt.show()
```

- `sharex=True, sharey=True` 를 설정하여 중복된 축을 한번만 표시하도록 할 수 있다.
- `sharex, sharey`에 `True, False` 이외에도 `all, none, row, col` 등을 지정하여 사용한다.

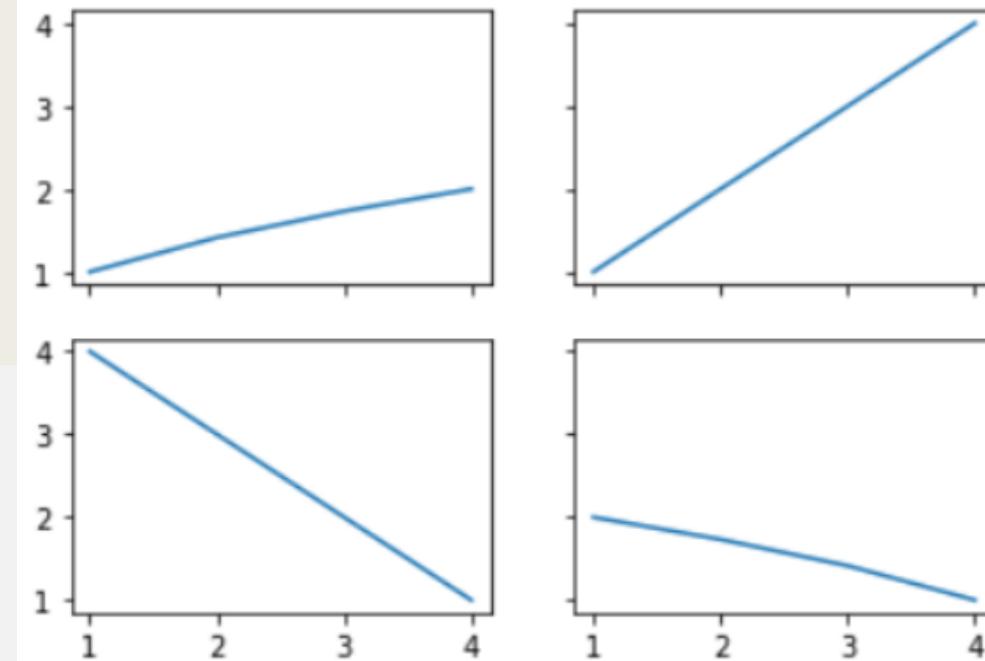
```

import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1, 5)      # [1, 2, 3, 4]

fig, ax = plt.subplots(2, 2, sharex=True, sharey=True, squeeze=True)
ax[0][0].plot(x, np.sqrt(x))      # left-top
ax[0][1].plot(x, x)              # right-top
ax[1][0].plot(x, -x+5)           # left-bottom
ax[1][1].plot(x, np.sqrt(-x+5))  # right-bottom
plt.show()

```



- NumPy를 이용해서 x값을 만들어 네 개의 그래프 영역에 각각 다른 y값을 그래프로 나타냈다.
- plt.subplots()이 반환하는 ax는 Matplotlib의 Axes 클래스의 인스턴스이다.
- 행과 열을 각각 2,2로 지정했기 때문에 as에는 2*2의 형태를 갖는 Numpy 어레이가 된다. 위치에 따라서 각각 ax[0][0], ax[0][1]....와 같이 접근해서 사용한다.

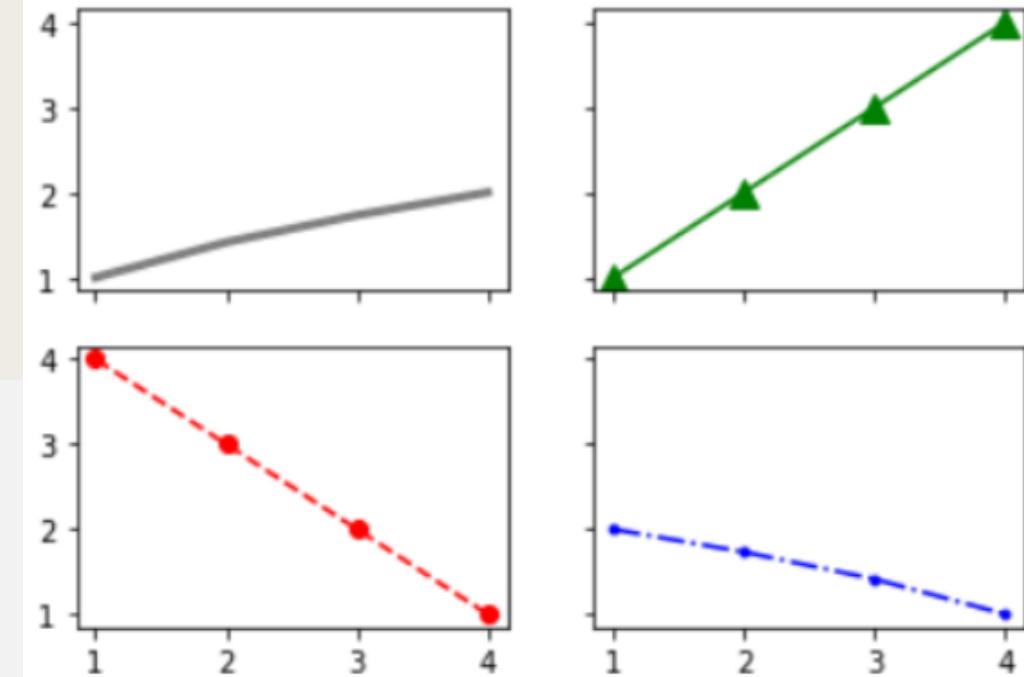
```

import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1, 5)      # [1, 2, 3, 4]

fig, ax = plt.subplots(2, 2, sharex=True, sharey=True, squeeze=True)
ax[0][0].plot(x, np.sqrt(x), 'gray', linewidth=3)
ax[0][1].plot(x, x, 'g^-', markersize=10)
ax[1][0].plot(x, -x+5, 'ro--')
ax[1][1].plot(x, np.sqrt(-x+5), 'b.-.')
plt.show()

```



- `plot()`함수에 각각의 그래프 스타일을 커스터마이즈 하도록 설정한다.

```

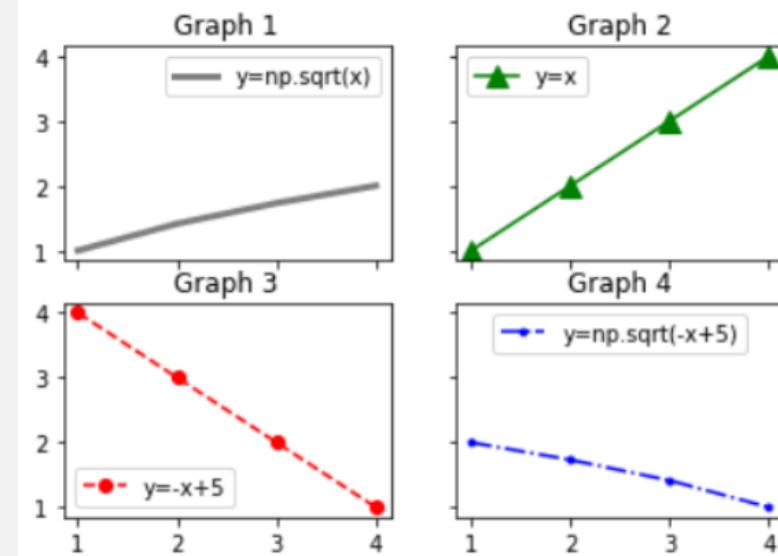
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1, 5)      # [1, 2, 3, 4]

fig, ax = plt.subplots(2, 2, sharex=True, sharey=True, squeeze=True)
ax[0][0].plot(x, np.sqrt(x), 'gray', linewidth=3,
               label='y=np.sqrt(x)')
ax[0][0].set_title('Graph 1')
ax[0][0].legend()
ax[0][1].plot(x, x, 'g^-', markersize=10, label='y=x')
ax[0][1].set_title('Graph 2')
ax[0][1].legend(loc='upper left')
ax[1][0].plot(x, -x+5, 'ro--', label='y=-x+5')
ax[1][0].set_title('Graph 3')
ax[1][0].legend(loc='lower left')
ax[1][1].plot(x, np.sqrt(-x+5), 'b.-.', label='y=np.sqrt(-x+5)')
ax[1][1].set_title('Graph 4')
ax[1][1].legend(loc='upper center')

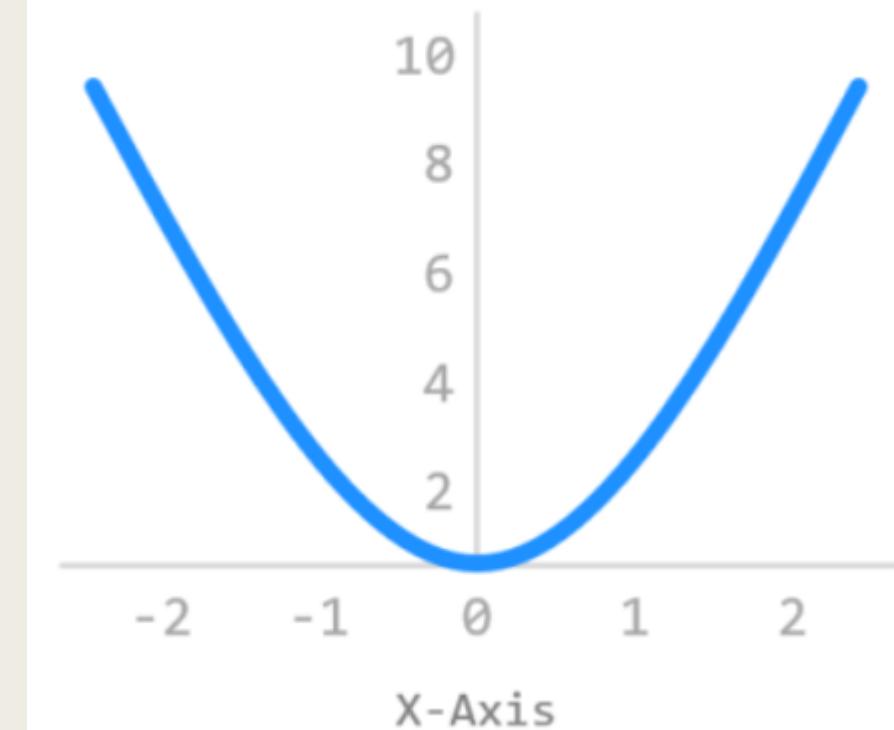
plt.show()

```



- `set_title()`은 입력한 문자열을 그래프의 제목으로 나타낸다.
- `legend()`는 `plot()`에서 `label`을 이용해서 지정한 문자열을 범례에 나타낸다.
- 그래프의 영역에서 범례가 표시될 위치를 지정하지 않으면 최적의 위치에 범례를 표시한다.

축 위치 조절



- Matplotlib의 기본적인 레이아웃은 그래프 상하좌우의 네 방향에 데이터 영역을 나타내는 직선이 그려지는 형태이다.
- 데이터 영역을 나타내는 직선을 필요한 위치에 선택적으로 표시하는 방법을 알아보자.

```

import matplotlib.pyplot as plt
import numpy as np

plt.style.use('default')
plt.rcParams['figure.figsize'] = (6, 3)
plt.rcParams['font.size'] = 12

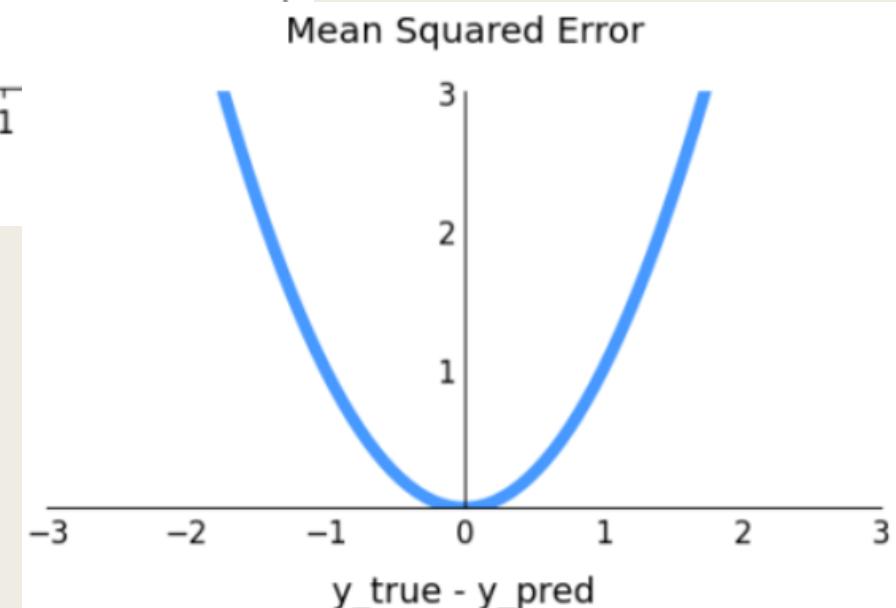
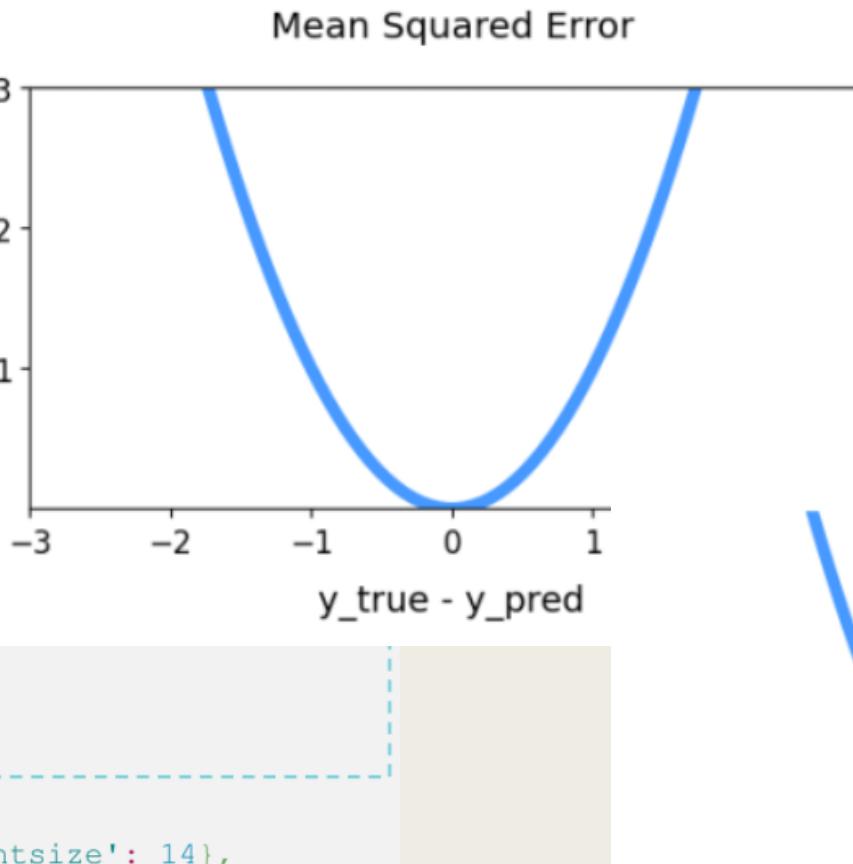
fig, ax = plt.subplots()

ax.set_title('Mean Squared Error', pad=20)
ax.set_xlim(-3, 3)
ax.set_ylim(0, 3)
ax.set_xticks([-3, -2, -1, 0, 1, 2, 3])
ax.set_yticks([1, 2, 3])

ax.spines['left'].set_position('center')
ax.spines['right'].set_visible('none')
ax.spines['top'].set_visible('none')
ax.spines['bottom'].set_position(('data', 0))
ax.tick_params('both', length=0)

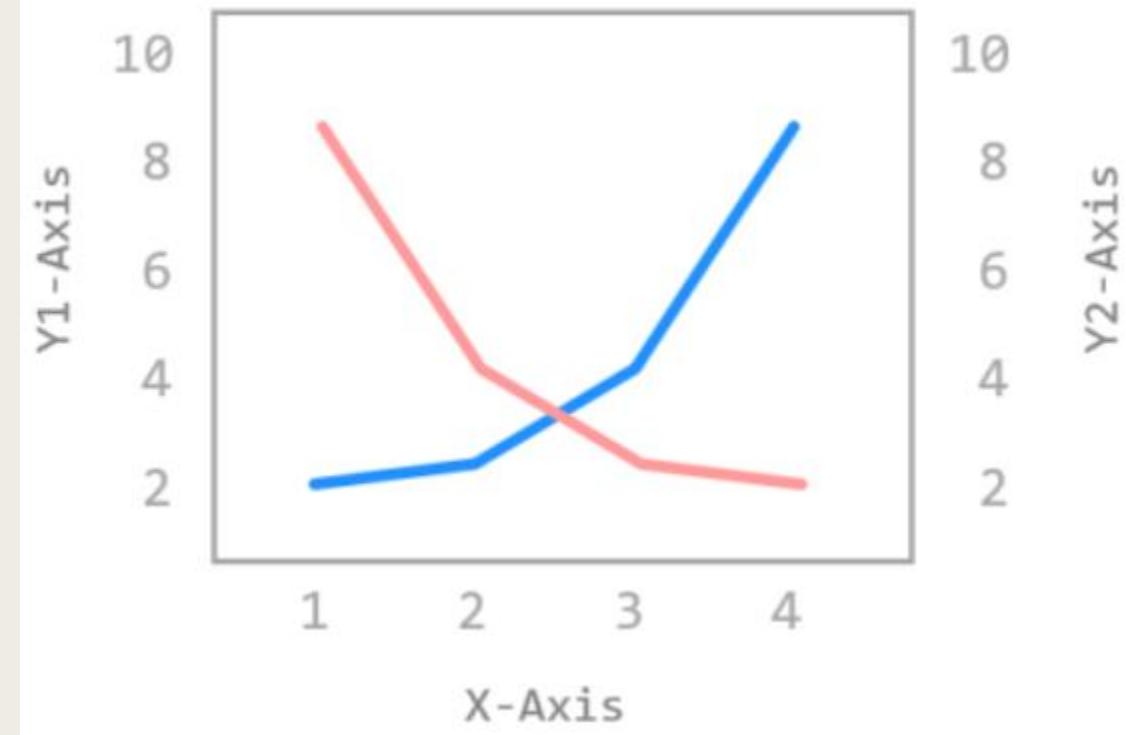
x = np.linspace(-3, 3, 100)
ax.set_xlabel('y_true - y_pred', fontdict={'fontsize': 14},
              labelpad=10)
ax.plot(x, x**2, color='#4799FF', linewidth=5)
plt.show()

```



- spines는 데이터 영역의 경계를 나타내는 선을 말한다.
- spines 클래스의 set_position()메서드를 사용하면 이 직선의 위치를 조정할 수 있다.
- set_position('center')와 같이 데이터 영역의 가운데에 위치시키거나
- set_position(('data',0))과 같이 특정 데이터 좌표의 위치에 직선을 표시할 수 있다.
- set_visible(False)는 spine이 그래프에 표시되지 않도록 한다.

이중 Y축 표시



- 두 종류의 데이터를 동시에 하나의 그래프에 표시하기 위해 이중 축을 사용

```

import matplotlib.pyplot as plt
import numpy as np

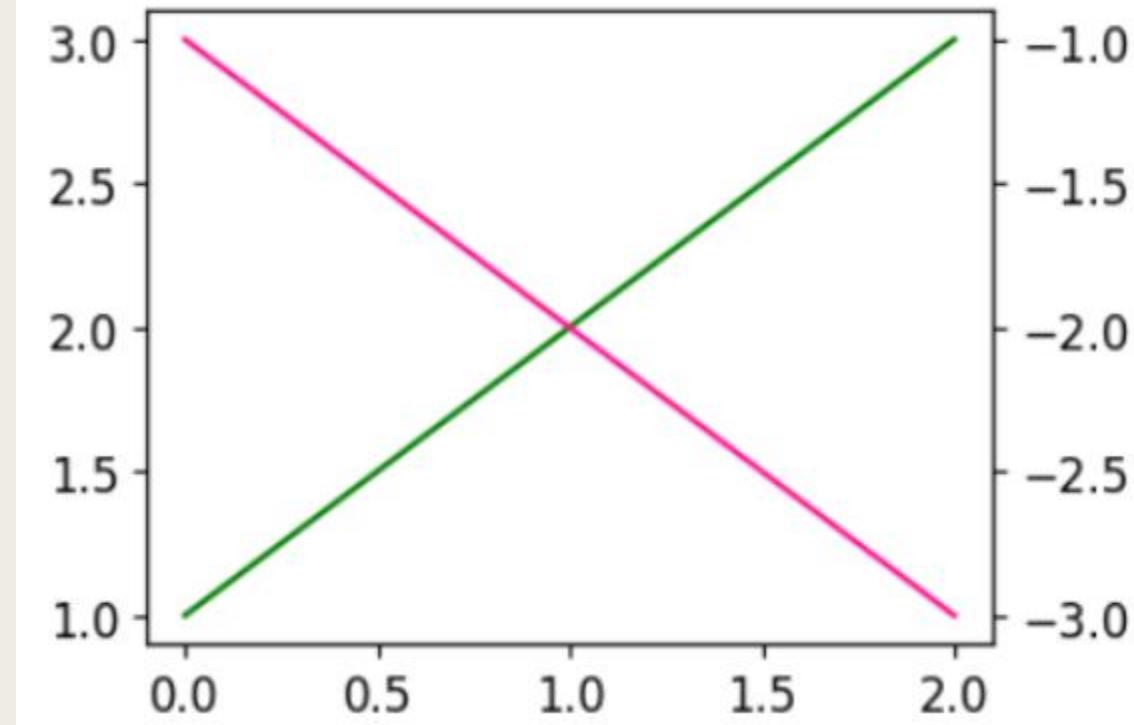
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

x = np.arange(0, 3)
y1 = x + 1
y2 = -x - 1

fig, ax1 = plt.subplots()
ax1.plot(x, y1, color='green')

ax2 = ax1.twinx()
ax2.plot(x, y2, color='deeppink')
plt.show()

```



- `ax1.plot(x, y1, color='green')`은 첫번째 축에 $(x, y1)$ 데이터를 나타낸다.
- `ax1.twinx()`는 `ax1`과 x축만 공유하는 새로운 axes 객체를 만든다.
- `ax2.plot(x, y2)`는 새로운 Axes 객체에 $(x, y2)$ 데이터를 나타낸다.

```

import matplotlib.pyplot as plt
import numpy as np

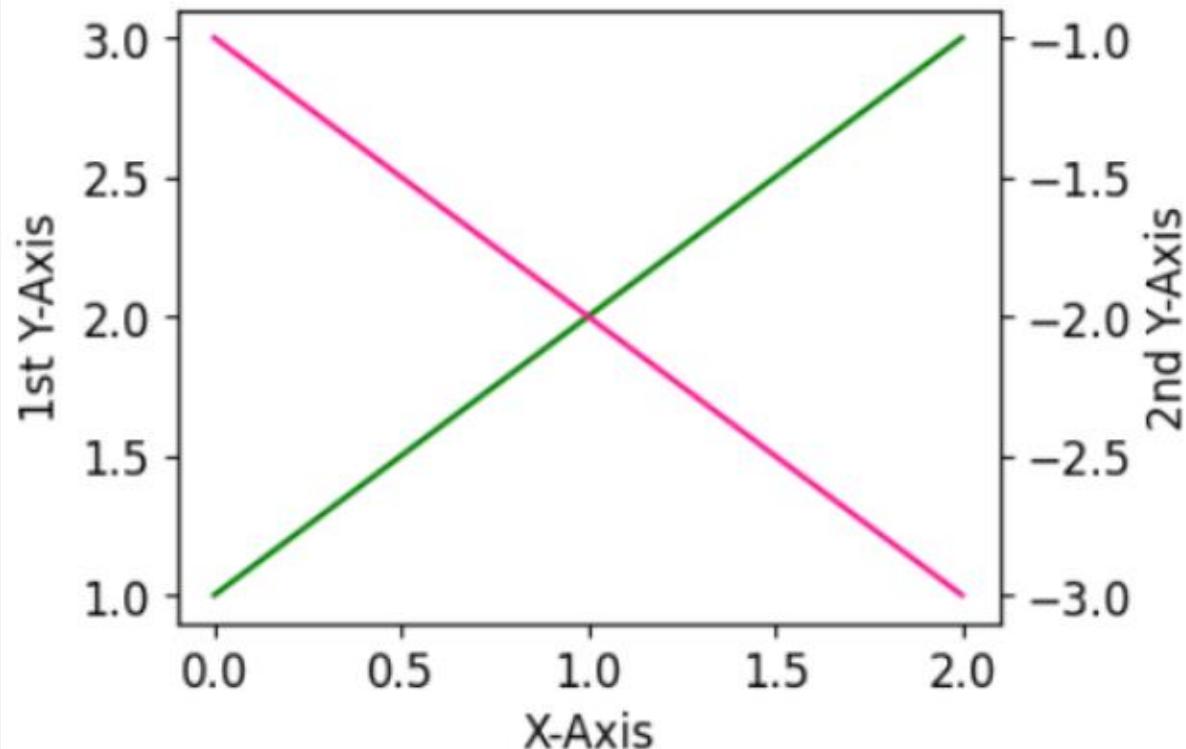
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

x = np.arange(0, 3)
y1 = x + 1
y2 = -x - 1

fig, ax1 = plt.subplots()
ax1.set_xlabel('X-Axis')
ax1.set_ylabel('1st Y-Axis')
ax1.plot(x, y1, color='green')

ax2 = ax1.twinx()
ax2.set_ylabel('2nd Y-Axis')
ax2.plot(x, y2, color='deeppink')
plt.show()

```



- `set_xlabel()`, `setylabel()` 메서드는 각 축에 대한 레이블을 표시하도록 설정한다.

```

import matplotlib.pyplot as plt
import numpy as np

plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 14

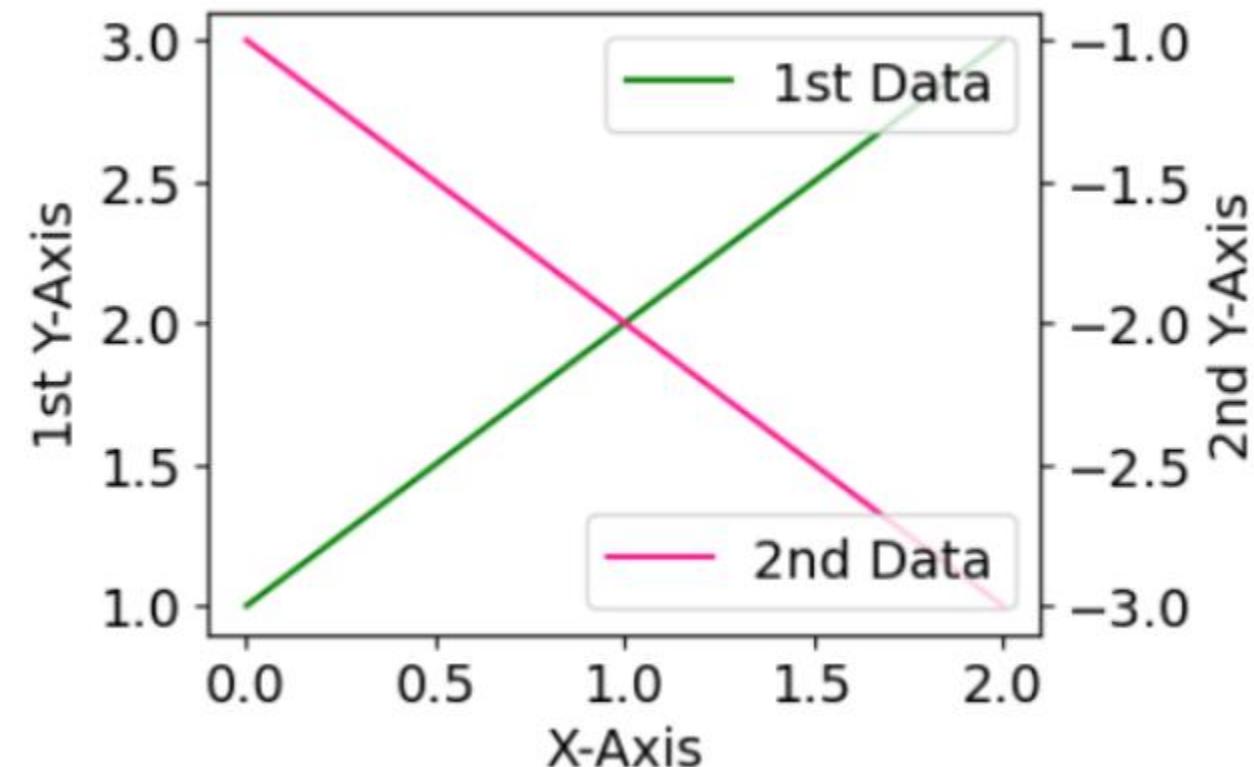
x = np.arange(0, 3)
y1 = x + 1
y2 = -x - 1

fig, ax1 = plt.subplots()
ax1.set_xlabel('X-Axis')
ax1.set_ylabel('1st Y-Axis')
ax1.plot(x, y1, color='green', label='1st Data')
ax1.legend(loc='upper right')

ax2 = ax1.twinx()
ax2.set_ylabel('2nd Y-Axis')
ax2.plot(x, y2, color='deeppink', label='2nd Data')
ax2.legend(loc='lower right')

plt.show()

```



- 각 축의 데이터 곡선에 대한 범례를 나타내기 위해 legend()매서드를 사용한다.

```

import matplotlib.pyplot as plt
import numpy as np

plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 14

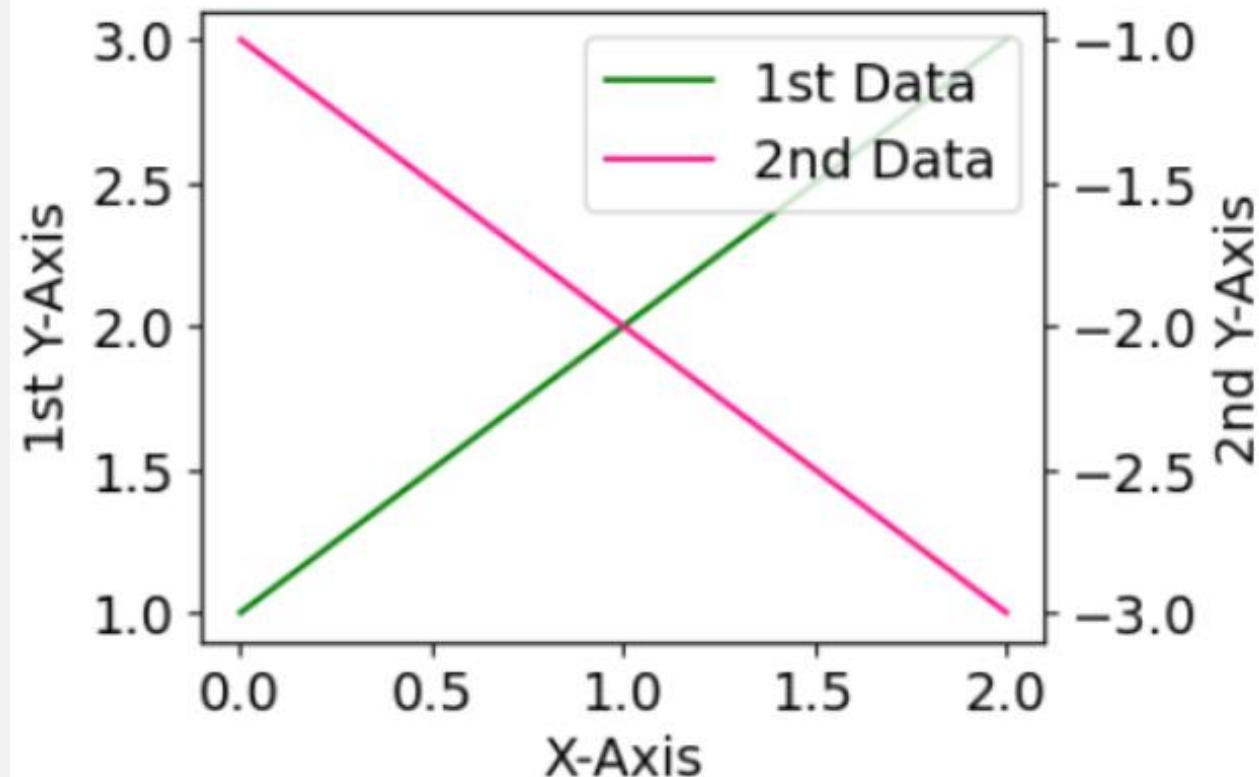
x = np.arange(0, 3)
y1 = x + 1
y2 = -x - 1

fig, ax1 = plt.subplots()
ax1.set_xlabel('X-Axis')
ax1.set_ylabel('1st Y-Axis')
line1 = ax1.plot(x, y1, color='green', label='1st Data')

ax2 = ax1.twinx()
ax2.set_ylabel('2nd Y-Axis')
line2 = ax2.plot(x, y2, color='deeppink', label='2nd Data')

lines = line1 + line2
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='upper right')
plt.show()

```



- 두 축에 대한 범례를 하나의 텍스트 상자에 표시하기 위해서는 두 곡선을 하나로 합친 후 legend() 매서드를 사용한다.

두 종류의 그래프 그리기

- 이중 Y축 표시하기를 이용해서 두 종류의 그래프를 하나의 그래프 영역에 표현 할 수 있다.

```

import matplotlib.pyplot as plt
import numpy as np

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
x = np.arange(2020, 2027)
y1 = np.array([1, 3, 7, 5, 9, 7, 14])
y2 = np.array([1, 3, 5, 7, 9, 11, 13])

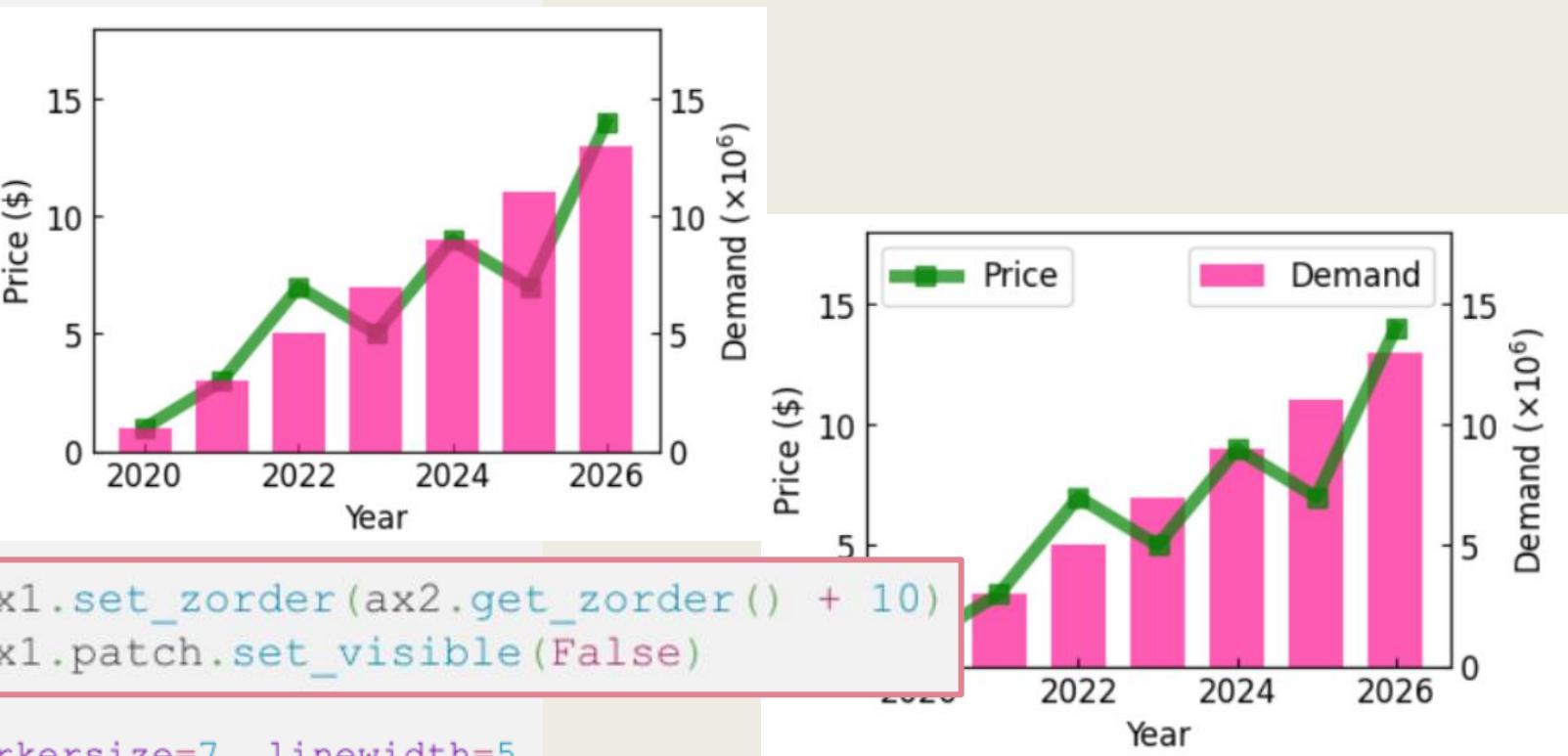
# 3. 그래프 그리기
fig, ax1 = plt.subplots()

ax1.plot(x, y1, '-s', color='green', markersize=7, linewidth=5,
          alpha=0.7, label='Price')
ax1.set_xlim(0, 18)
ax1.set_xlabel('Year')
ax1.set_ylabel('Price ($)')
ax1.tick_params(axis='both', direction='in')

ax2 = ax1.twinx()
ax2.bar(x, y2, color='deeppink', label='Demand', alpha=0.7,
        width=0.7)
ax2.set_xlim(0, 18)
ax2.set_ylabel(r'Demand ($\times 10^6$)')
ax2.tick_params(axis='y', direction='in')

plt.show()

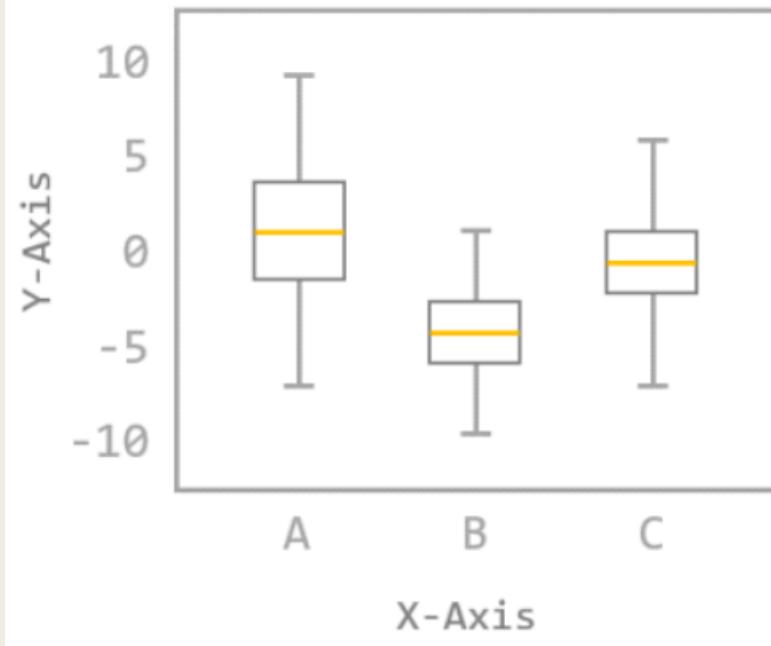
```



- 우선 `ax1.twinx()`로 x축을 공유하는 이중 Y축을 만들고
- `ax1.plot()` 과 `ax2.bar()`을 사용해서 `y1`, `y2`데이터를 각각 깍은선 그래프와 막대 그래프의 형태로 나타낼 수 있다.
- `set_zorder()`매서드를 사용해서 그래프가 표시되는 순서를 지정할 수 있다.
- `zorder`가 낮을 수록 먼저 그려진다.

박스 플롯 Box plot

- 박스 플롯 또는 박스 위스커 플롯은 수치 데이터를 표현하는 하나의 방법이다. 일반적으로 박스 플롯은 전체 데이터로 부터 얻어진 5가지 요약 수치를 사용해서 그려진다.
 - 최소값
 - 제 1사분위 수
 - 제 2사분위 수
 - 제 3사분위 수
 - 최대값
- 사분위 수는 데이터를 4등분한 지점을 의미한다. 예를 들어 제 1사분위 수는 전체 데이터 중 하위 25%에 해당하는 값이다. 제 3사분위 수는 전체 데이터 중 상위 25%에 해당하는 값이다.



```

import matplotlib.pyplot as plt
import numpy as np

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

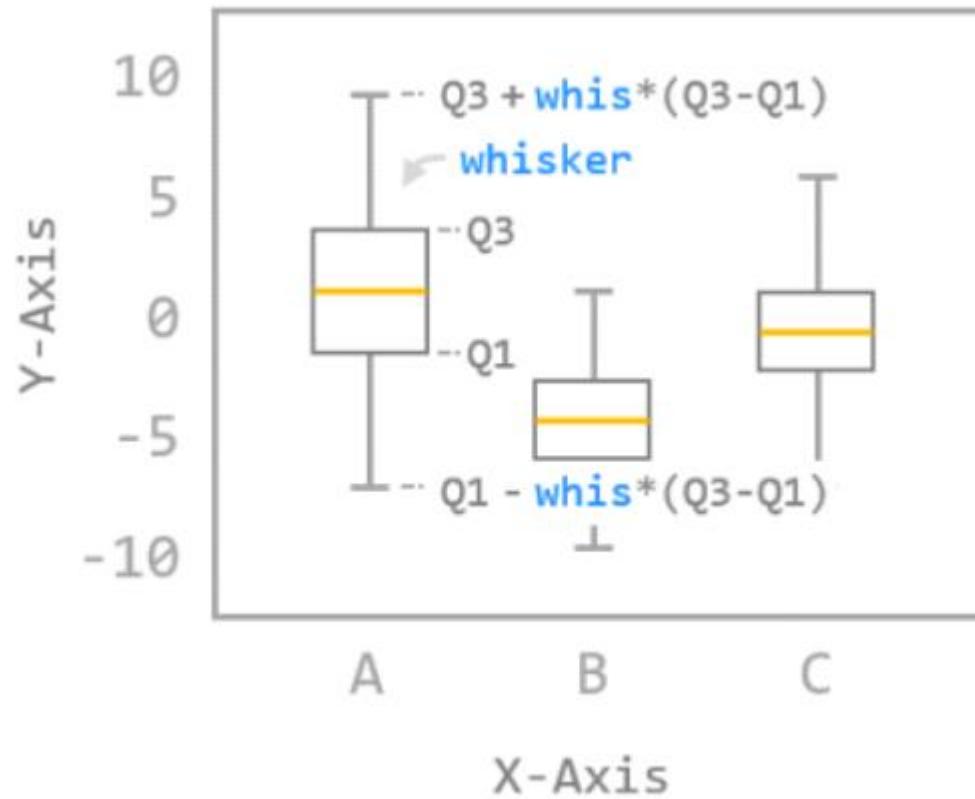
# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()
ax.boxplot([data_a, data_b, data_c], notch=True, whis=2.5)

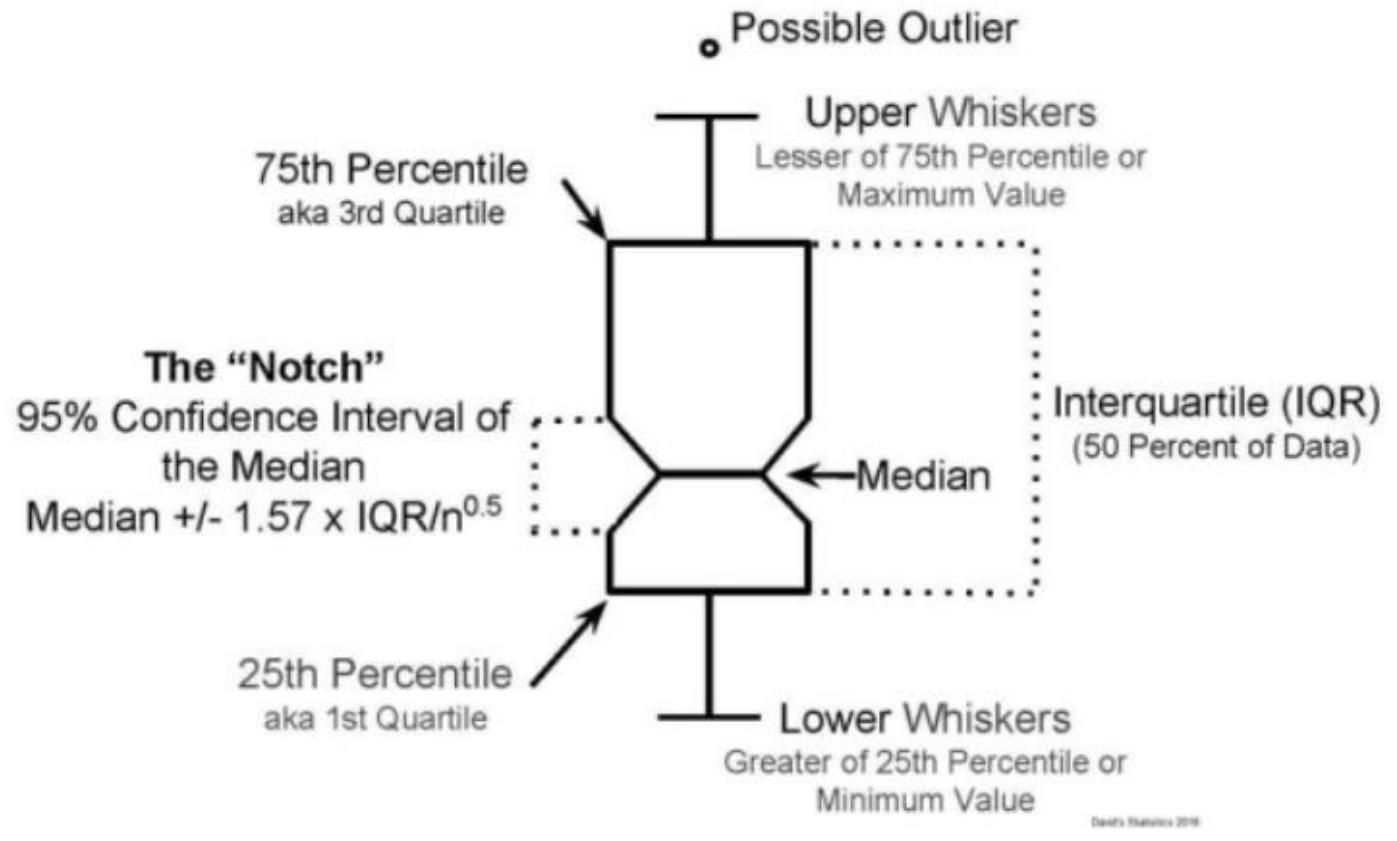
ax.boxplot([data_a, data_b, data_c])
ax.set_xlim(-10.0, 10.0)
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')

plt.show()

```



- `ax.boxplot()`은 주어진 데이터 어레이로부터 얻어진 요약 수치를 박스 형태로 나타낸다.
- 가운데 박스 모양으로부터 그려지는 중심 선을 수염(Whisker)이라고 한다.
- `boxplot()` 함수는 기본적으로 박스의 위쪽 경계 위쪽 그리고 박스의 아래쪽 경계 아래 부분을 수염 Whisker 으로 나타낸다.



- notch 파라미터를 True 지정하면 중앙값(Median)의 95% 신뢰 구간을 노치 형태로 표한다.
- whis = 2.5 수염의 길이가 길어지고 수염 밖의 데이터가 표시되지 않는다.

3. 그래프 그리기

```
fig, ax = plt.subplots()

ax.boxplot([data_a, data_b, data_c], notch=True, whis=2.5)
ax.set_xlim(-10.0, 10.0)
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')
plt.show()
```

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

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()

box = ax.boxplot([data_a, data_b, data_c], notch=True, whis=1.5)
ax.set_xlim(-10.0, 10.0)
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')

whiskers = [item.get_ydata() for item in box['whiskers']]
medians = [item.get_ydata() for item in box['medians']]
fliers = [item.get_ydata() for item in box['fliers']]

# print('whiskers:', whiskers)
# print('medians:', medians)
# print('fliers:', fliers)
plt.show()

whiskers: [array([-1.39684012, -5.31123819]), array([1.2139
medians: [array([-0.11605607, -0.11605607]), array([-2.8907
fliers: [array([-5.54518551, -5.31834448, -5.47935433, -6.0
5.39244811, 5.18884918, 5.51871023]), array([-7.1
0.78955236, 0.96440358, 1.02085626]), array([-3.4
```

- `ax.boxplot()`은 박스 플롯의 각 구성요소에 해당하는 인스턴스의 리스트를 딕셔너리 형태로 반환한다.
- 예를 들어, 예제에서 `box['whiskers']`은 Q1, Q3, max, min 값 `box['medians']`는 중앙값에 대한 정보를 갖는 인스턴스의 리스트이다.
- `box['filter']`는 수염 범위 밖의 데이터 포인트를 반환하는데 세개의 어레이를 갖는 리스트가 출력된다.
- `get_ydata()`매서드를 사용해서 y 값 위치를 얻을 수 있다.

```

import matplotlib.pyplot as plt
import numpy as np

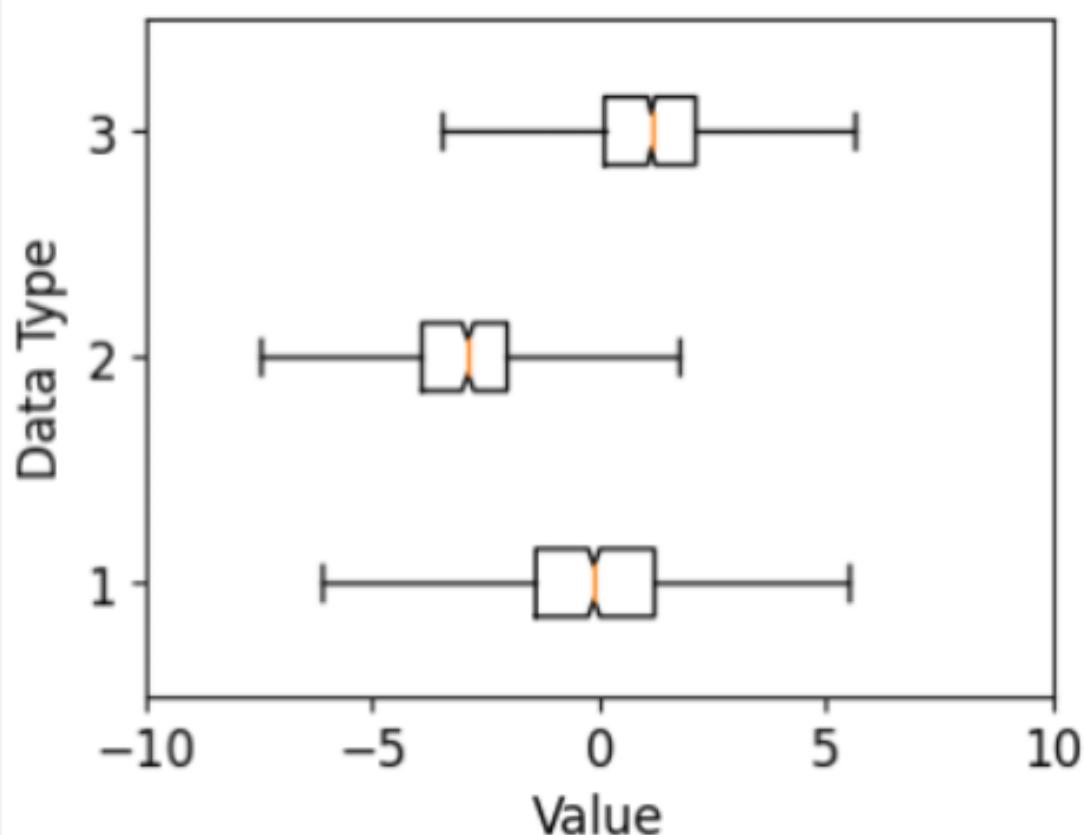
# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()

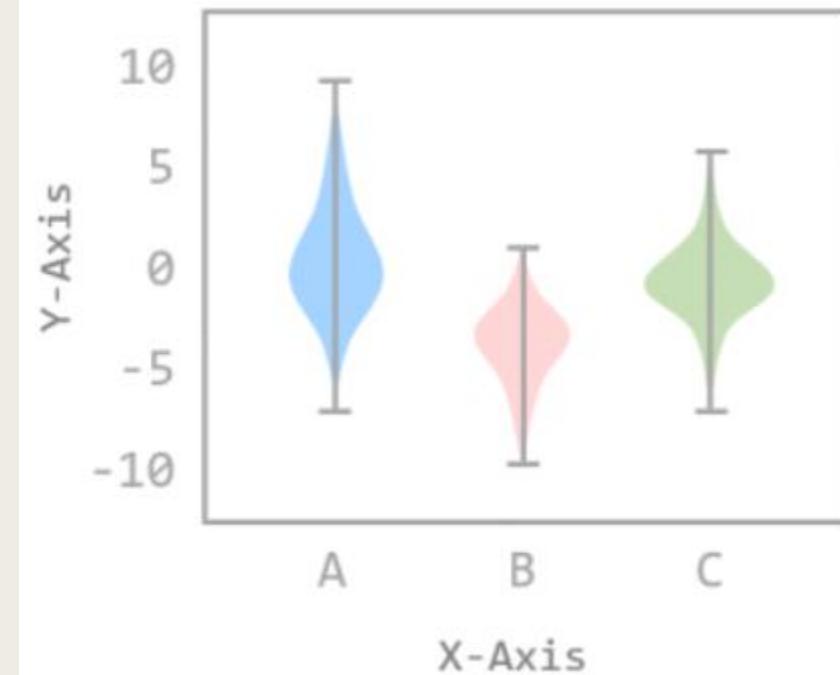
ax.boxplot([data_a, data_b, data_c], notch=True, whis=2.5,
           vert=False)
ax.set_xlim(-10.0, 10.0)
ax.set_xlabel('Value')
ax.set_ylabel('Data Type')
plt.show()

```



- `vert` 파라미터를 `False`로 지정하면 수평 방향의 박스플롯이 나타난다.
- 디폴트는 수직 방향의 박스플롯이다.

바이올린 플롯



- 바이올린 플롯 Violin plot 은 데이터의 분포와 범위를 한눈에 보기 쉽게 나타내는 그래프이다.
- 박스 플롯과 비슷하지만 더 실제에 가까운 분포를 알 수 있는 장점이 있다.

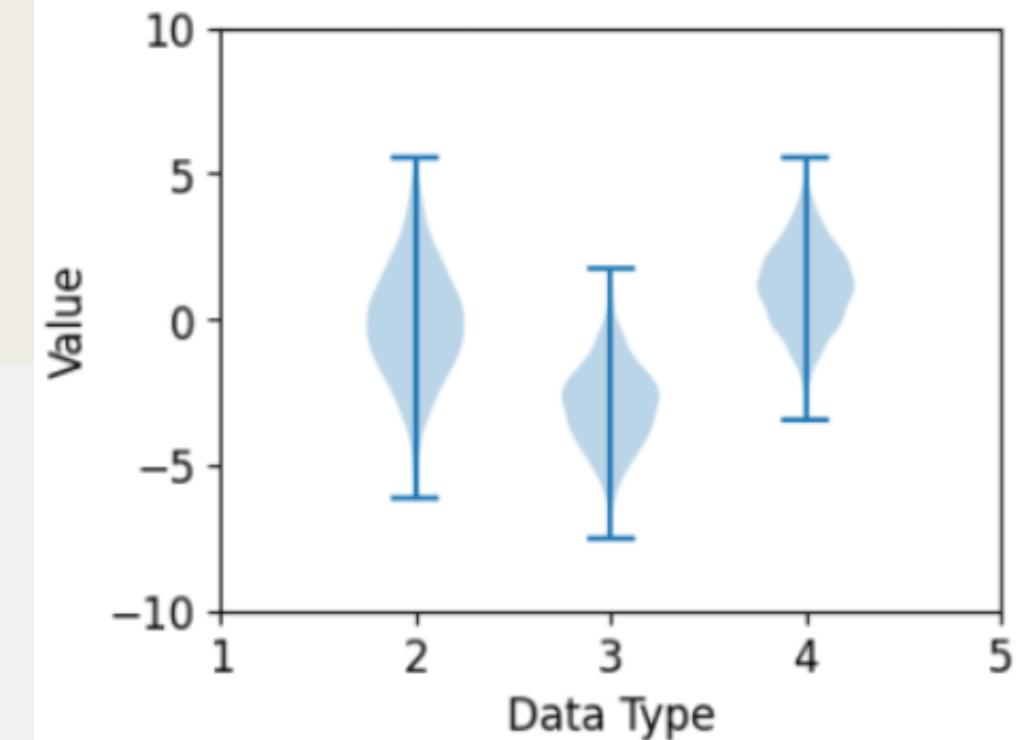
```

# 기본 사용
import matplotlib.pyplot as plt
import numpy as np

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12
# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()
violin = ax.violinplot([data_a, data_b, data_c], positions=[2, 3, 4])
ax.set_ylim(-10.0, 10.0)
ax.set_xticks([1, 2, 3, 4, 5])
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')
plt.show()

```



- `ax.violinplot()`은 주어진 데이터 어레이의 분포를 바이올린 형태로 시각화 한다.
- `positions` 파라미터는 바이올린 플롯의 x위치를 지정한다. 지정하지 않으면 1, 2, 3의 순서로 그려진다.
- `ax.set_xticks()`로 x축의 눈금을 지정했고 [2,3,4]의 위치에 그래프가 그려진다.

```

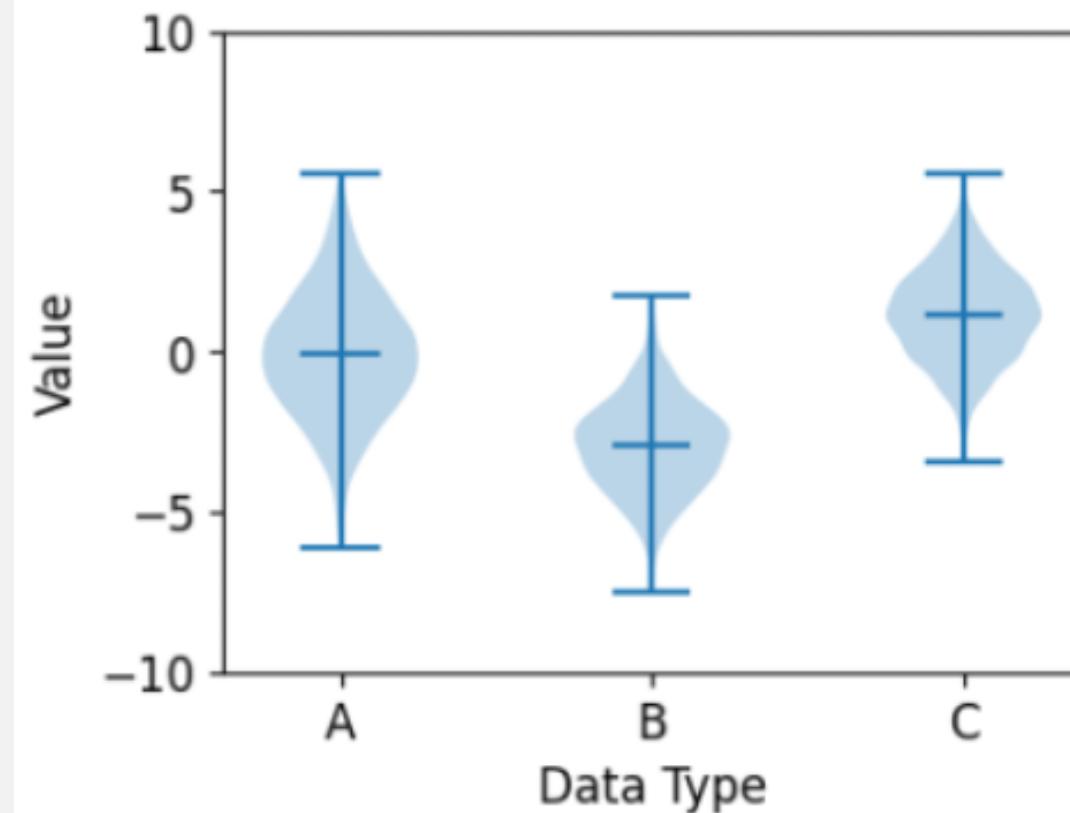
import matplotlib.pyplot as plt
import numpy as np

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()
violin = ax.violinplot([data_a, data_b, data_c], showmeans=True)
ax.set_xlim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')
plt.show()

```



- `showmeans` 파라미터는 데이터 분포에서 평균값의 위치에 직선을 표시한다. 디폴트는 `False`이다.
- `showextrema` 는 최대값.최소값에 직선을 표시한다. 디폴트가 `True`이다.
- `showmedians` 은 중간값에 직선을 표시한다. 디폴트는 `False`이다.

```

import matplotlib.pyplot as plt
import numpy as np

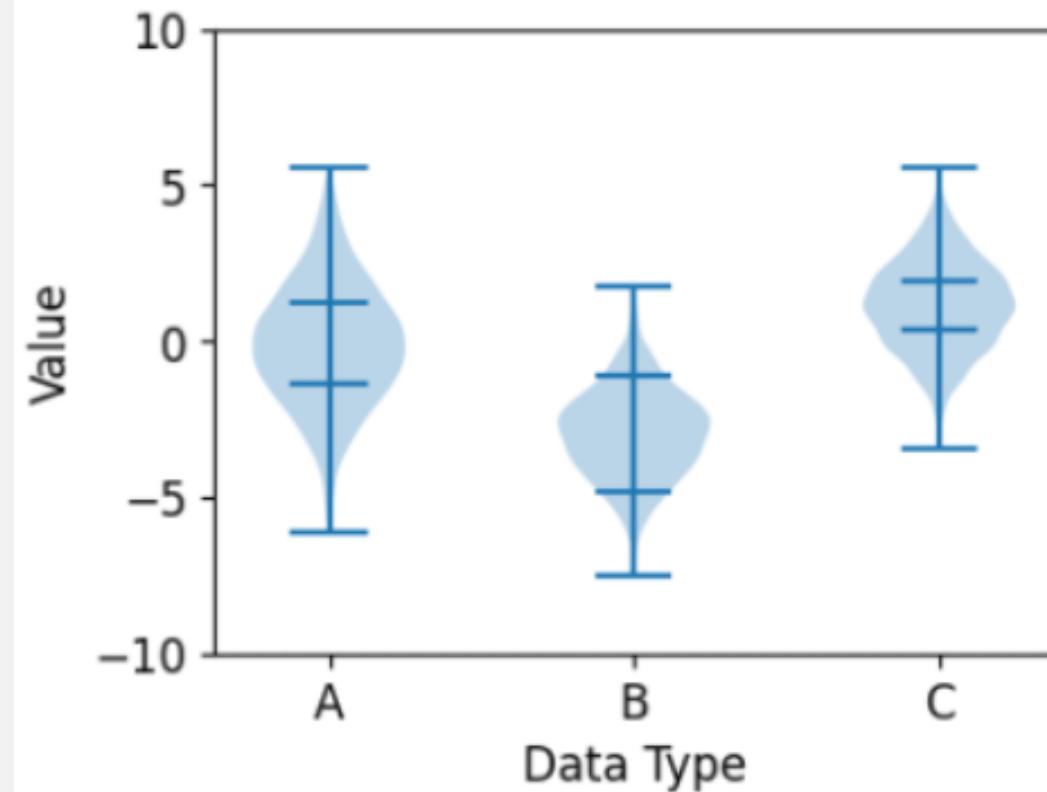
# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()

violin = ax.violinplot([data_a, data_b, data_c],
                       quantiles=[[0.25, 0.75], [0.1, 0.9], [0.3, 0.7]])
ax.set_xlim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')
plt.show()

```



- `quantiles` 파라미터는 데이터 분포의 분위수를 표시한다. 0.0에서 1.0 사이의 숫자를 리스트 형태로 입력한다.
- `data_a`에는 25%, 75%,
- `data_b`에는 10%, 90%
- `data_c`에는 30%, 70%

```

import matplotlib.pyplot as plt
import numpy as np

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

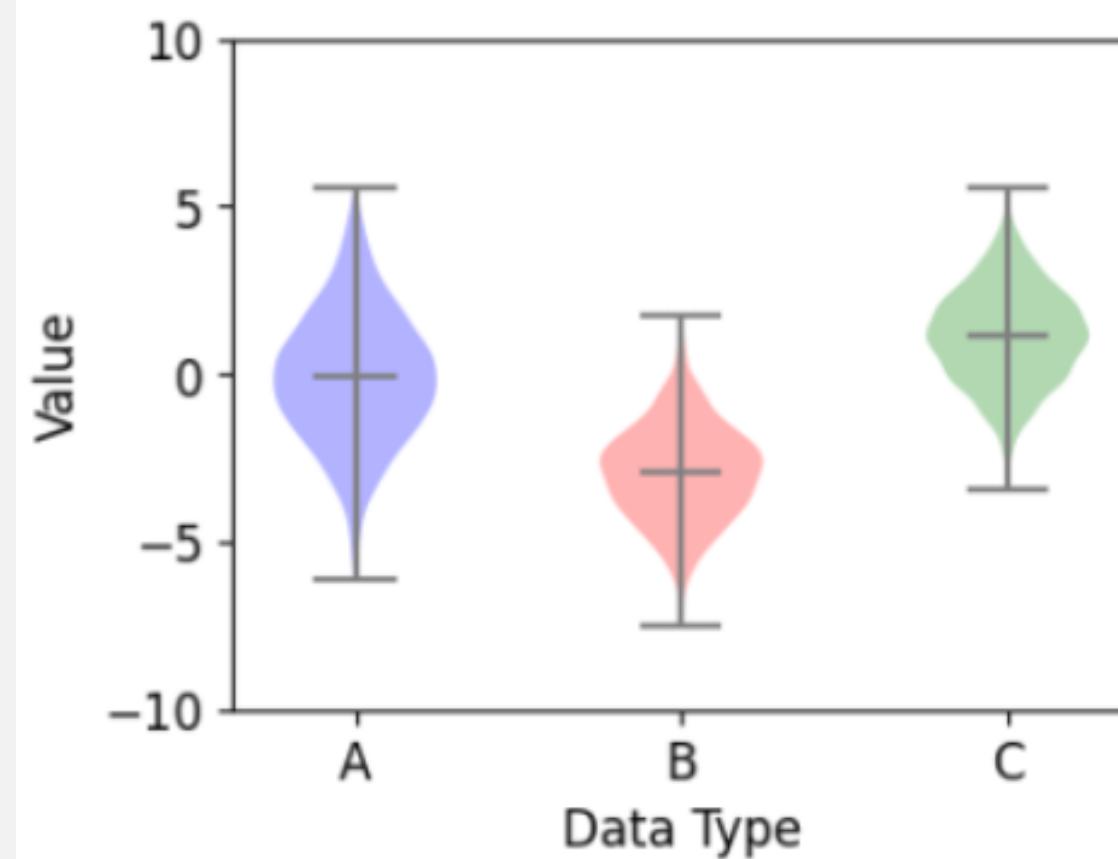
# 3. 그래프 그리기
fig, ax = plt.subplots()

violin = ax.violinplot([data_a, data_b, data_c], showmeans=True)
ax.set_xlim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')

violin['bodies'][0].set_facecolor('blue')
violin['bodies'][1].set_facecolor('red')
violin['bodies'][2].set_facecolor('green')

violin['cbars'].set_edgecolor('gray')
violin['cmaxes'].set_edgecolor('gray')
violin['cmins'].set_edgecolor('gray')
violin['cmeans'].set_edgecolor('gray')
plt.show()

```



- violin['bodies'][n] set_facecolor() 매서드는 n번째 바이올린 분포 영역의 색상을 지정한다.
- violin['cbars'] 중심을 표시하는 직선의 색상을 지정한다.
- violin['cmins'] 분포의 최소값을 표시하는 직선의 색상을 지정한다.
- violin['cmaxes'] 분포의 최대값을 표시하는 직선의 색상을 지정한다.

다양한 도형 삽입

| |
|---|
| Arc(xy, width, height[, angle, theta1, theta2]) |
| Arrow(x, y, dx, dy[, width]) |
| Circle(xy[, radius]) |
| CirclePolygon(xy[, radius, resolution]) |
| ConnectionPatch(xyA, xyB, coordsA[, ...]) |
| Ellipse(xy, width, height[, angle]) |
| FancyArrow(x, y, dx, dy[, width, ...]) |
| Polygon(xy[, closed]) |
| Rectangle(xy, width, height[, angle]) |
| RegularPolygon(xy, numVertices[, radius, ...]) |
| Wedge(center, r, theta1, theta2[, width]) |



- `matplotlib.patches` 모듈은 그래프에 다양한 2D도형을 표현하기 위한 클래스를 포함하고 있다.

```
# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

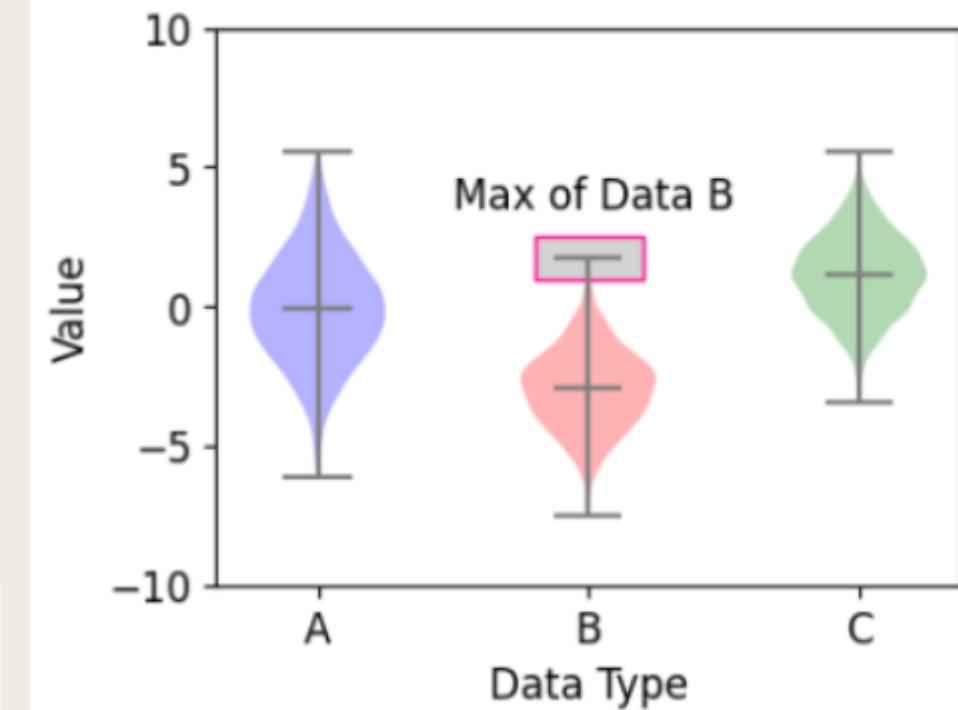
# 3. 그래프 그리기
fig, ax = plt.subplots()

violin = ax.violinplot([data_a, data_b, data_c], showmeans=True)
ax.set_ylim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Value')

# 4. 사각형 그리기
ax.add_patch(
    patches.Rectangle(
        (1.8, 1.0),
        0.4, 1.5,
        edgecolor = 'deeppink',
        facecolor = 'lightgray',
        fill=True,
    )
)

# 5. 텍스트 삽입하기
ax.text(1.5, 3.5, 'Max of Data B')

plt.show()
```



- ax.add_patch() 매서드는 입력한 Patch 를 그래프 영역에 추가한다.
 - patches.Rectangle() 패치는 그래프에 사각형을 표현하기 위해 사용된다.
 - (x,y) 위치와 width,height 을 순서대로 입력하고 edgecolor와 facecolor를 지정한다. fill=true로 지정하면 도형의 면에 색이 채워진다.
 - text() 매서드로 텍스트를 삽입했다.

```

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()

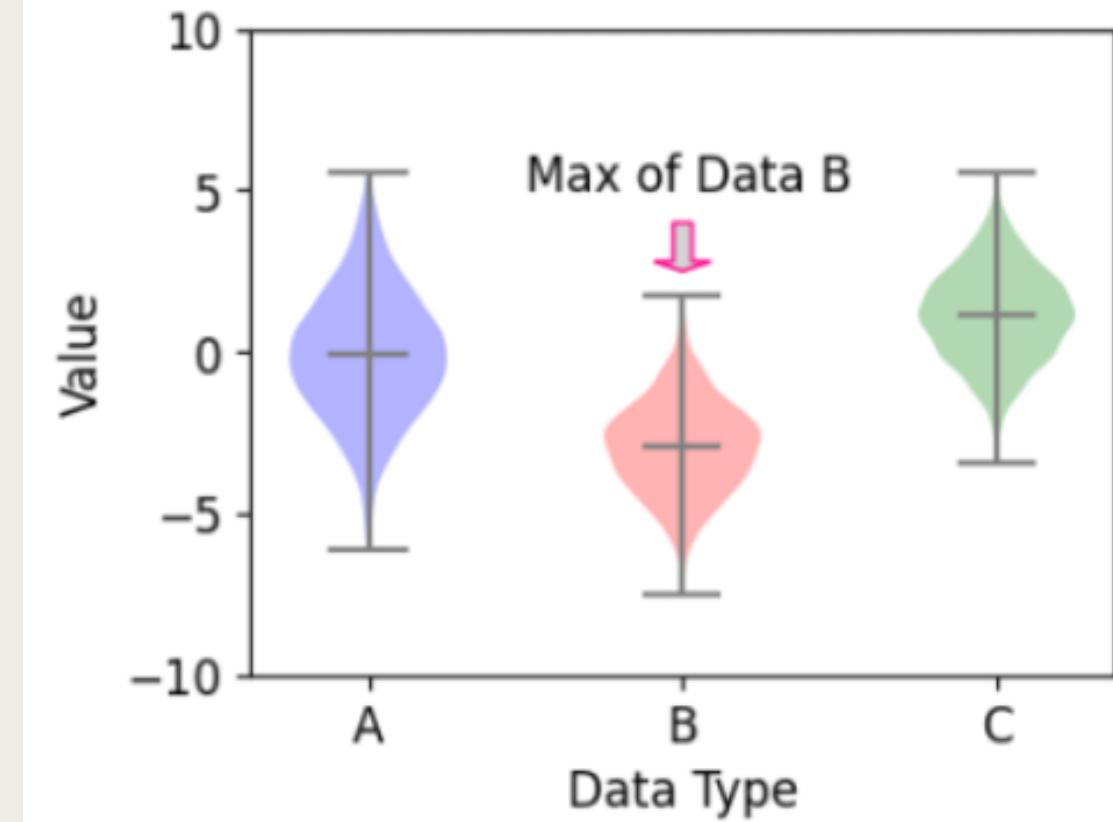
violin = ax.violinplot([data_a, data_b, data_c], showmeans=True)
ax.set_ylim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Distribution')

violin['bodies'][0].set facecolor('blue')
violin['bodies'][1].set facecolor('red')
violin['bodies'][2].set facecolor('green')

# 4. 화살표 그리기
ax.add_patch(
    patches.Arrow(
        2.0, 4.0,
        0.0, -1.5,
        width=0.3,
        edgecolor = 'deeppink',
        facecolor = 'lightgray'
    )
)

# 5. 텍스트 삽입하기
ax.text(1.5, 5.0, 'Max of Data B')
plt.show()

```



- `patches.Arrow()` 패치는 그래프에 화살표 도형을 표현하기 위해 사용한다.
- 화살표 시작점 위치 x,y와 위치 변화량 dx, dy를 순서대로 입력한다.
- `width`는 화살표 도형의 너비

```

# 1. 기본 스타일 설정
plt.style.use('default')
plt.rcParams['figure.figsize'] = (4, 3)
plt.rcParams['font.size'] = 12

# 2. 데이터 준비
np.random.seed(0)
data_a = np.random.normal(0, 2.0, 1000)
data_b = np.random.normal(-3.0, 1.5, 500)
data_c = np.random.normal(1.2, 1.5, 1500)

# 3. 그래프 그리기
fig, ax = plt.subplots()

violin = ax.violinplot([data_a, data_b, data_c], showmeans=True)
ax.set_ylim(-10.0, 10.0)
ax.set_xticks(np.arange(1, 4))
ax.set_xticklabels(['A', 'B', 'C'])
ax.set_xlabel('Data Type')
ax.set_ylabel('Distribution')
violin['bodies'][0].set_facecolor('blue')
violin['bodies'][1].set_facecolor('red')
violin['bodies'][2].set_facecolor('green')

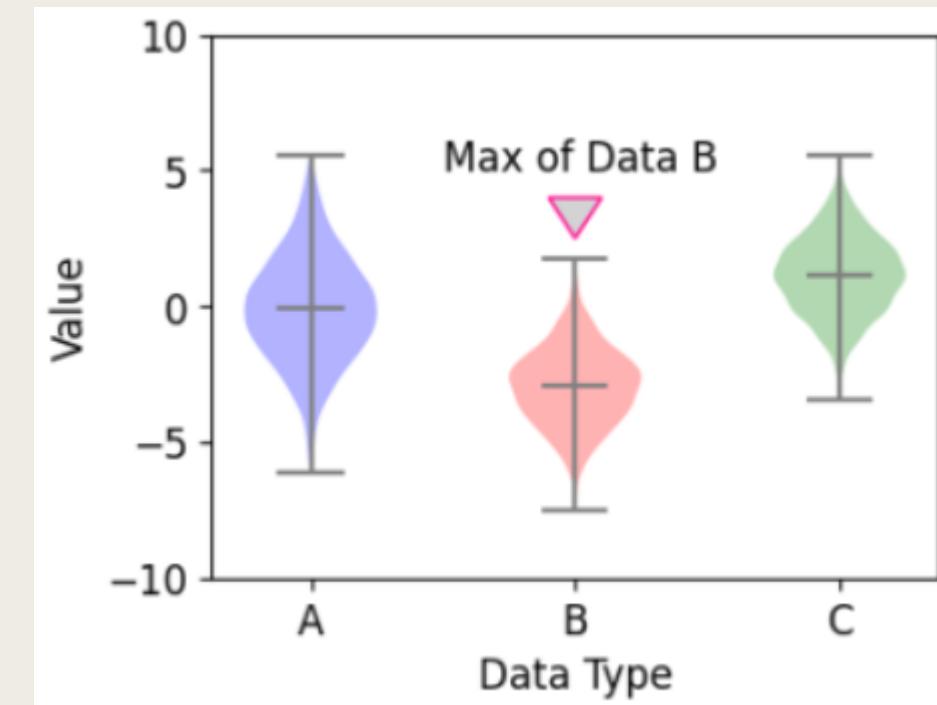
```

```

# 4. 다각형 그리기
violin['c']
violin['c']
violin['c']
violin['c'] ax.add_patch(
    patches.Polygon(
        ((1.9, 4.0), (2.0, 2.5), (2.1, 4.0)),
        closed=True,
        edgecolor = 'deeppink',
        facecolor = 'lightgray'
    )
)

# 5. 텍스트 삽입하기
ax.text(1.5, 5.0, 'Max of Data B')
plt.show()

```



- `patches.Polygon()` 패치는 그래프에 다각형을 표현하기 위해 사용한다.
- 그림과 같이 각각의 꼭지점 위치(x,y) 튜플 어레이를 입력한다.
- `closed = True`로 지정하면 닫힌 다각형의 형태로 그려진다.

다양한 패턴

```
import matplotlib.pyplot as plt

plt.style.use('default')
plt.rcParams['figure.figsize'] = (6, 5)
plt.rcParams['font.size'] = 12

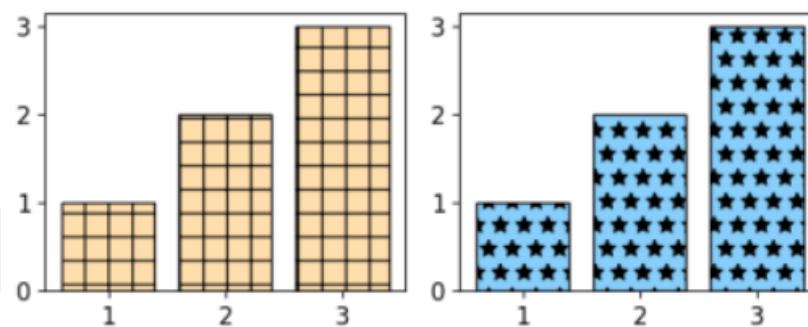
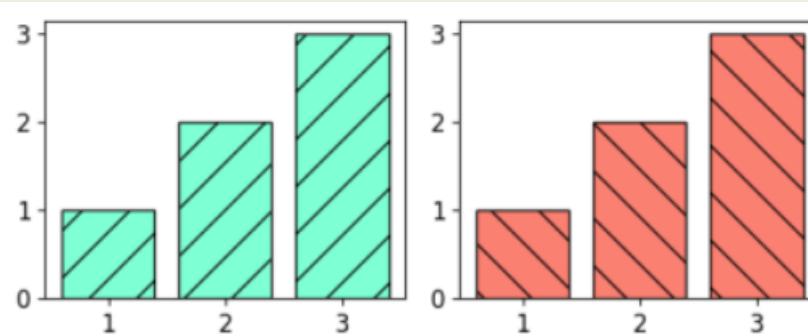
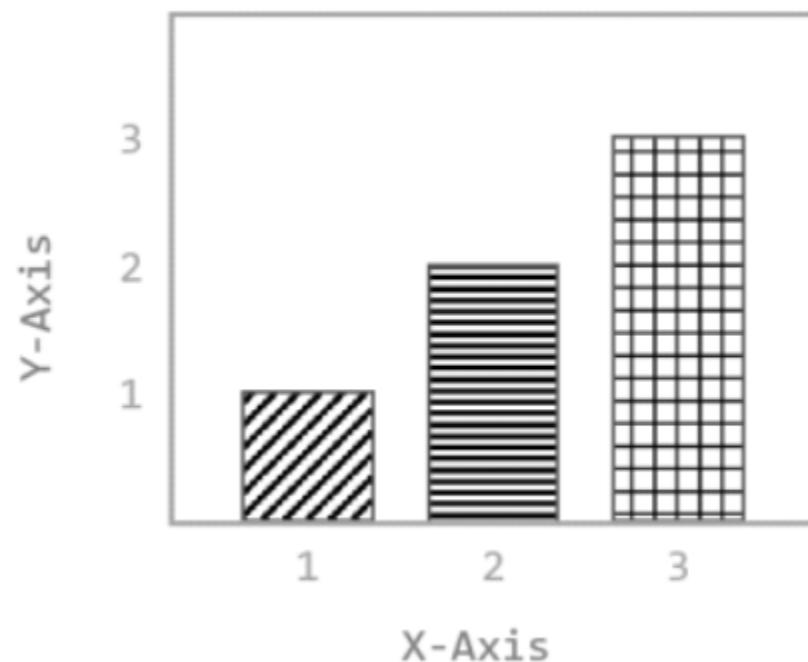
x = [1, 2, 3]
y = [1, 2, 3]

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

ax1.bar(x, y, color='aquamarine', edgecolor='black', hatch='//')
ax2.bar(x, y, color='salmon', edgecolor='black', hatch='\\\'')
ax3.bar(x, y, color='navajowhite', edgecolor='black', hatch='+')
ax4.bar(x, y, color='lightskyblue', edgecolor='black', hatch='*' )

plt.tight_layout()
plt.show()
```

{'//', '\\\'', '+', '-.', 'x', 'o', 'o', '.', '*'}



- `bar()`함수의 `hatch`는 그래프의 내부에 표시할 패턴을 지정한다.

```

import matplotlib.pyplot as plt
import numpy as np

plt.style.use('default')
plt.rcParams['figure.figsize'] = (8, 5)
plt.rcParams['font.size'] = 12

x = [1, 2, 3]
y = [1, 2, 3]

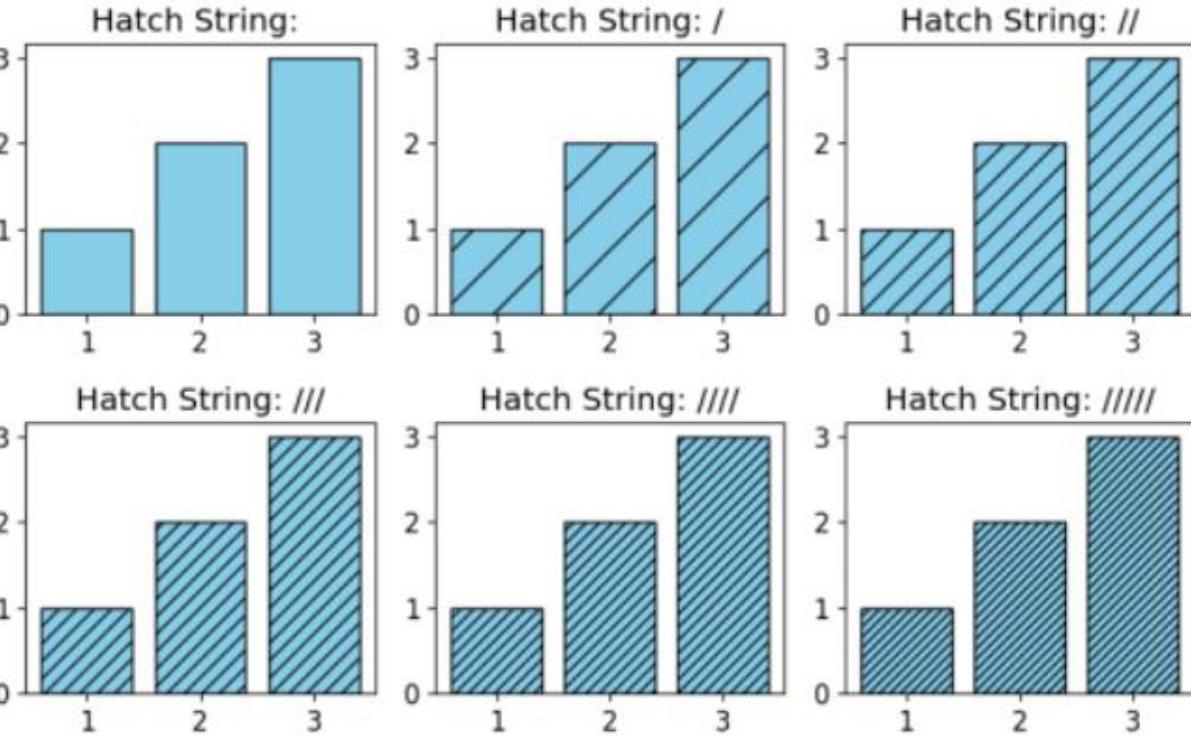
for i in range(6):
    hatch_str = "/" * i

    ax = plt.subplot(2, 3, i + 1)
    ax.set_title("Hatch String: " + hatch_str)
    bars = ax.bar(x,y,facecolor='skyblue', edgecolor='black')

    for bar in bars:
        bar.set_hatch(hatch_str)

plt.tight_layout()
plt.show()

```



- hatch 문자열의 개수를 한 개 이상 입력함으로써 패턴의 밀도를 지정할 수 있다.
- 예를 들면 hatch 문자열'/'에 대해 개수를 0-5개로 조절했다.