단원 05

Matplotlib를 활용한 데이터 시각화 기초

인공지능소프트웨어학과

강환수 교수

DONGYANG MIRAE UNIVERSITY
Dept. of Artificial Intelligence

DMUAi 동양미래대학교 인공지능소프트웨어학과

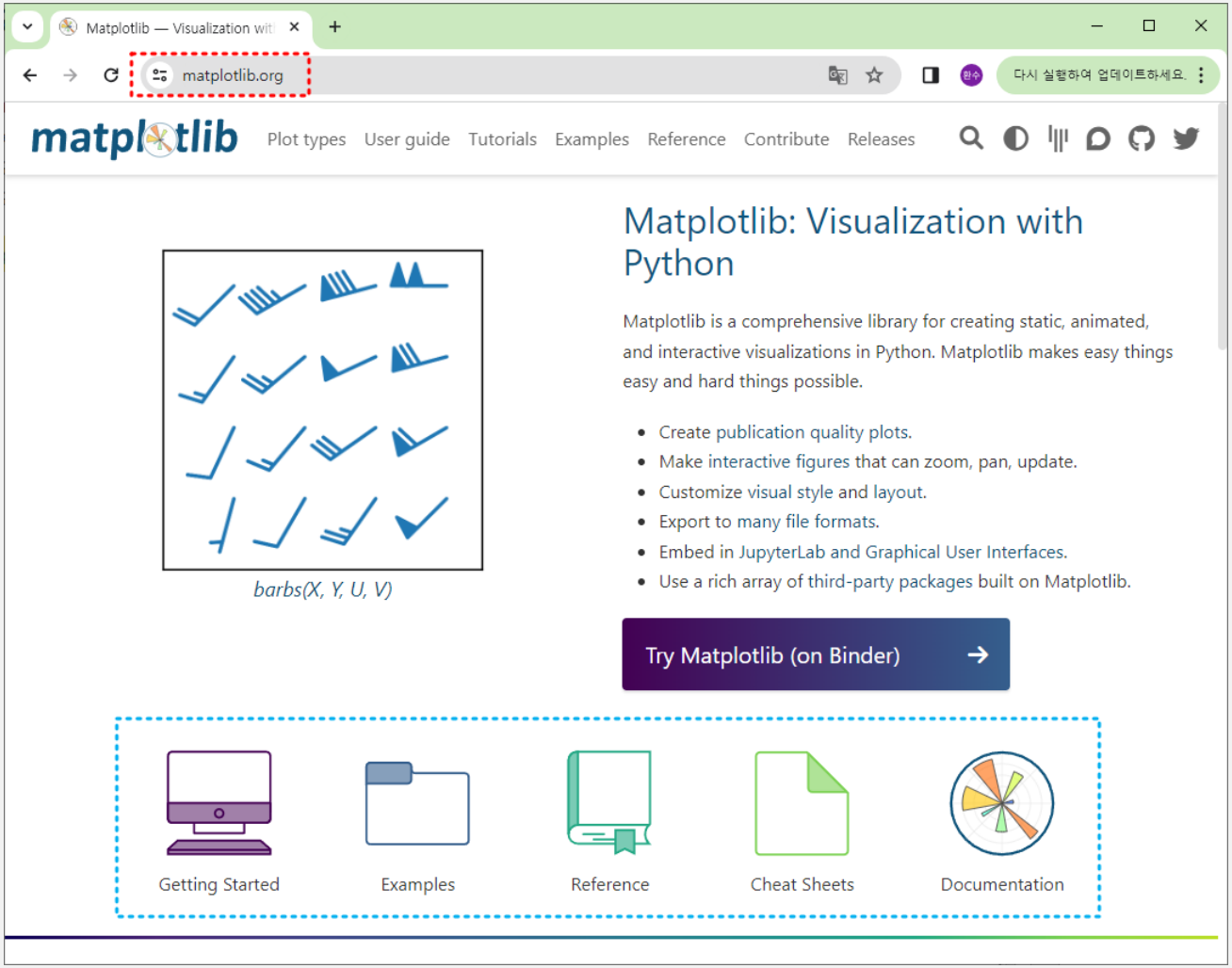
5.1 데이터 시각화 기초

-

데이터 시각화 패키지 matplotlib 개요

2D 그래픽 그림을 생성하는 데 사용되는 데이터 시각화 라이브러리

- 과학 및 엔지니어링 분야에서 데이터를 시각적으로 탐색하고 표현하는 데 널리 사용
- 다양한 차트, 플롯, 히스토그램 등을 생성



설치 확인과 한글 처리 준비

- 한글 처리 준비가 플랫폼에 상이
  - Colab 노트북 준비
    - !pip install koreanize-matplotlib
    - import koreanize_matplotlib
  - 클라이언트(PC) 노트북 준비
    - 방법 1
      - import matplotlib.pyplot as plt
      - plt.rc('font', family = 'Malgun Gothic')
      - plt.rc('axes', unicode_minus = False)
    - 방법 2
      - import matplotlib.pyplot as plt
      - plt.rcParams['font.family'] = 'Malgun Gothic'
      - plt.rcParams['axes.unicode_minus'] = False

한글 처리

- 클라이언트(PC)의 주피터 노트북에서 한글 처리

```
import matplotlib.pyplot as plt
# 방법 1: 함수 rc()로 설정
# rc(runtime configuration)
plt.rc('font', family='Malgun Gothic')
plt.rc('axes', unicode_minus=False)
```

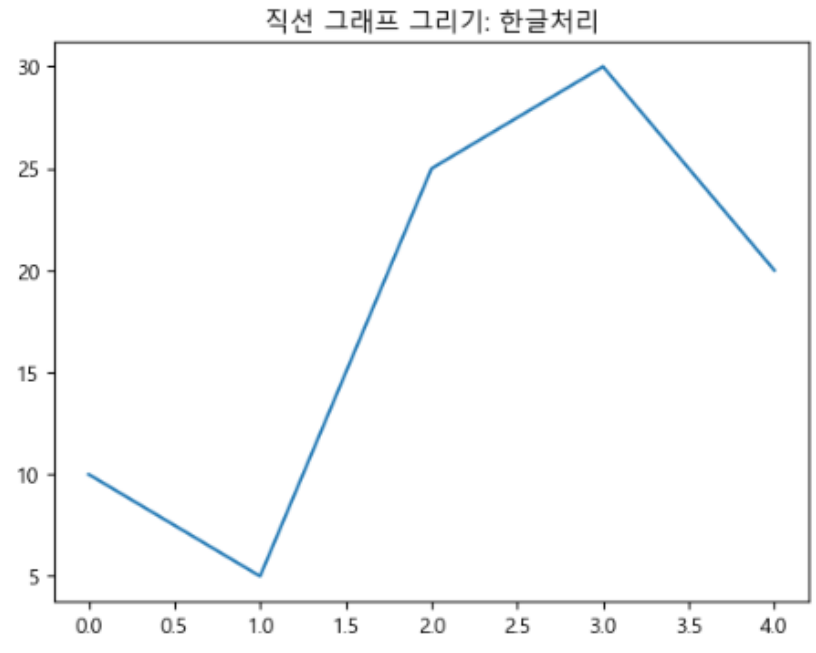
✓ 0.0s

```
import matplotlib.pyplot as plt

# 방법 2: 사전 rcParams[]에 설정
plt.rcParams['font.family'] ='Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False
```

✓ 0.0s

```
import matplotlib.pyplot as plt

plt.plot([10, 5, 25, 30, 20])
plt.title('직선 그래프 그리기: 한글처리')
plt.show()
```

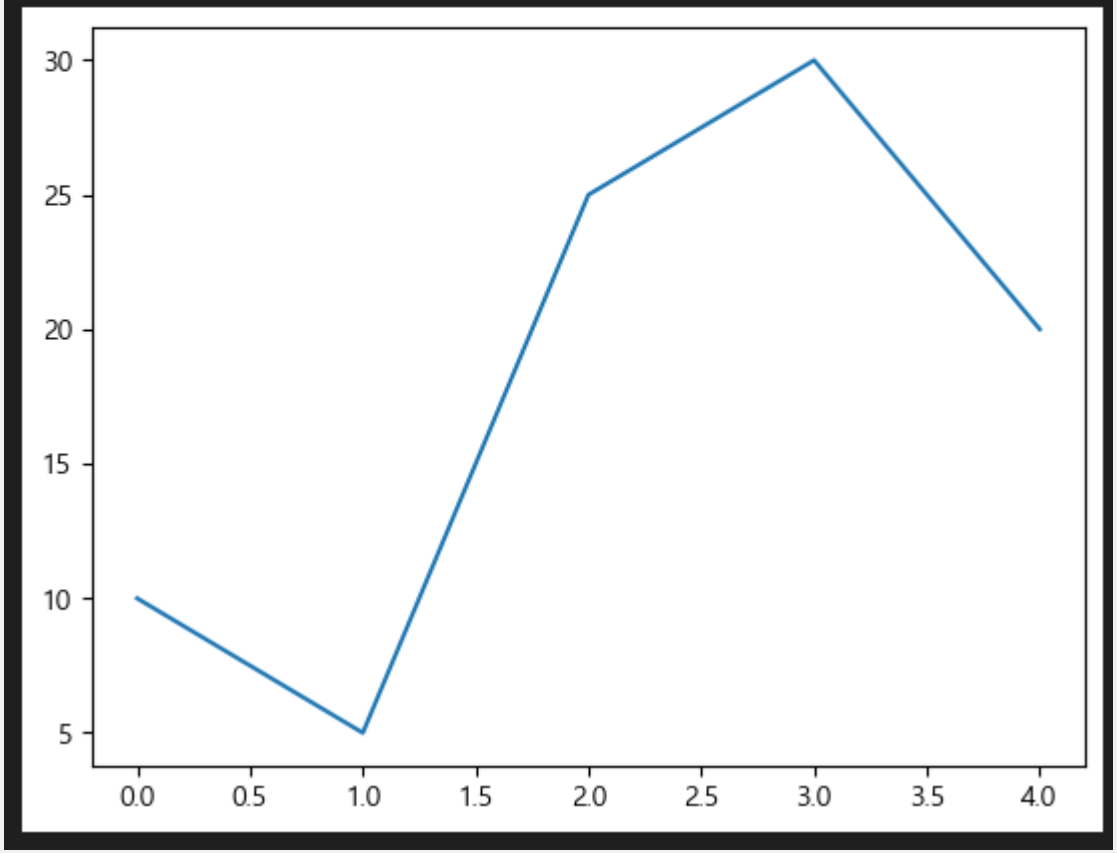
✓ 0.1s



plt.plot(y_value)

- 모듈 matplotlib.pyplot
  - 간단하고 빠른 방식으로 그래프를 그릴 수 있도록 도와주는 모듈
    - 폴더 matplotlib 하부의 pypoly.py
  - 내부적으로 Figure와 Axes 객체를 자동으로 생성하고 관리
    - 간단한 작업에 적합
      - 자동으로 axes를 생성하고 사용
      - 직접 axes 객체를 다룰 필요가 없음
      - 한 두 개의 간단한 그래프를 그릴 때 매우 유용
    - 복잡한 그래프 레이아웃을 구성할 때는 유연성이 떨어질 수 있음
- Figure와 Axes
  - Figure
    - 전체 그래프가 그려질 "캔버스"
  - Axes
    - 실제 데이터를 시각화하는 영역을 의미

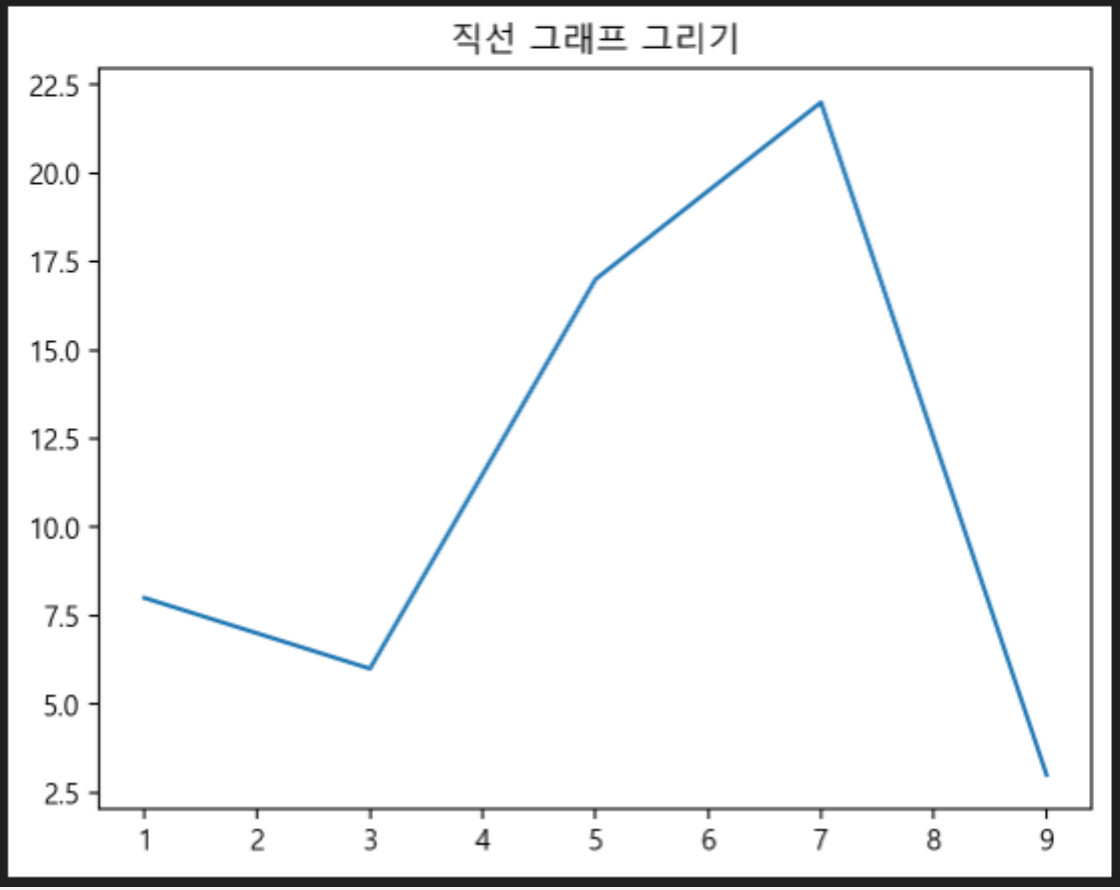
```
import matplotlib.pyplot as plt

plt.plot([10, 5, 25, 30, 20])
plt.show()
```



plt.plot(x_value, y_value)

```python
import matplotlib.pyplot as plt

plt.rcParams['font.family'] ='Malgun Gothic'
plt.rcParams['axes.unicode_minus'] = False

plt.plot([1, 3, 5, 7, 9], [8, 6, 17, 22, 3])
plt.title('직선 그래프 그리기')
plt.show()
```

Python

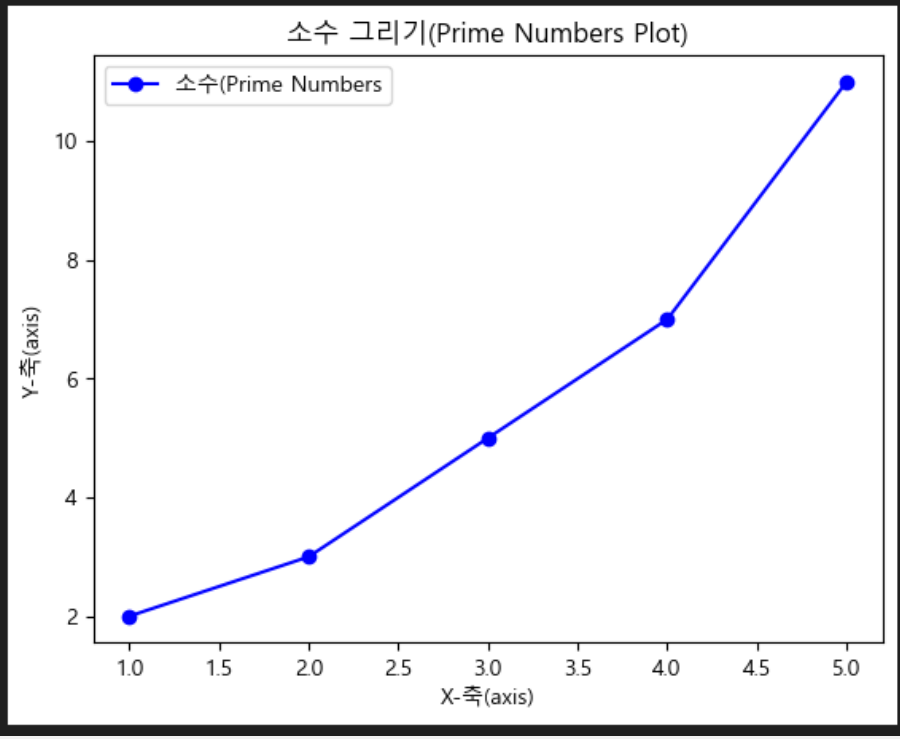


그래프의 여러 속성

- marker
- linestyle
- color
- label

```python
import matplotlib.pyplot as plt

# 데이터 생성
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# 선 플롯 생성
plt.plot(x, y, marker='o', linestyle='-', color='b', label='소수(Prime Numbers')

# 축 및 제목 설정
plt.xlabel('X-축(axis)')
plt.ylabel('Y-축(axis)')
plt.title('소수 그리기(Prime Numbers Plot)')
plt.legend()

# 플롯 표시
plt.show()
```

✓ 0.1s Python

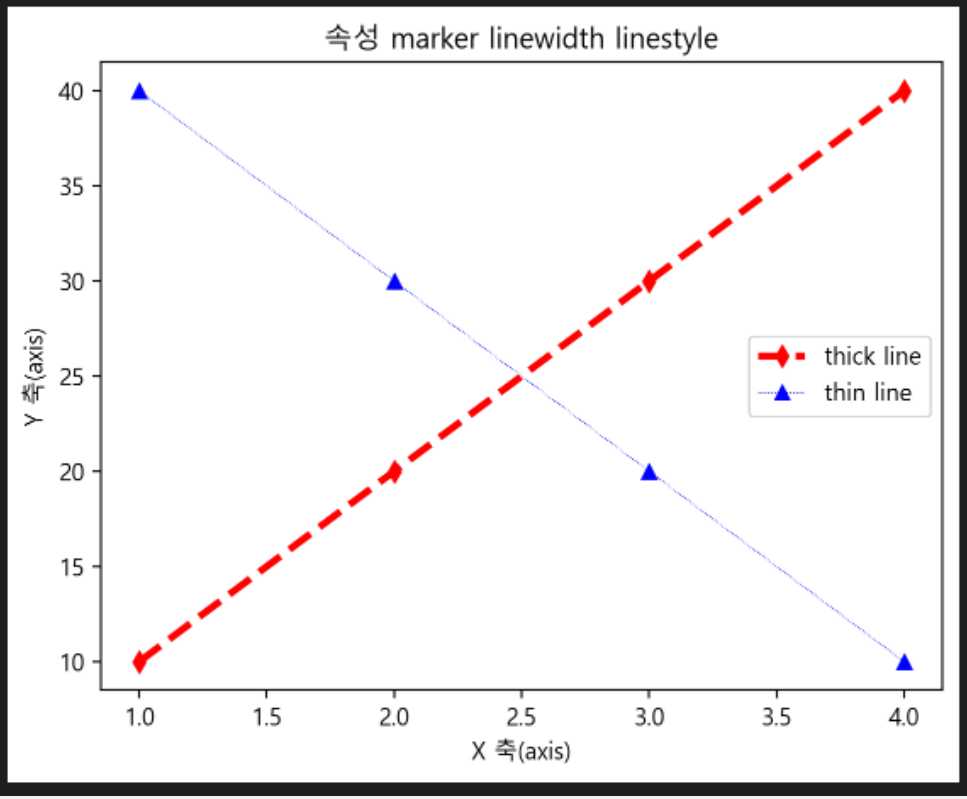


속성 marker, linewidth, linestyle, label

label과 함수 legend()

```python
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [10, 20, 30, 40], color='r', linestyle='--', marker='d', linewidth=3, label='thick line')
plt.plot([1, 2, 3, 4], [40, 30, 20, 10], 'b', ls=':', marker='^', lw=0.5, label='thin line')

plt.title('속성 marker linewidth linestyle')
plt.xlabel('X 축(axis)')
plt.ylabel('Y 축(axis)')
plt.legend()
plt.show()
```

Python

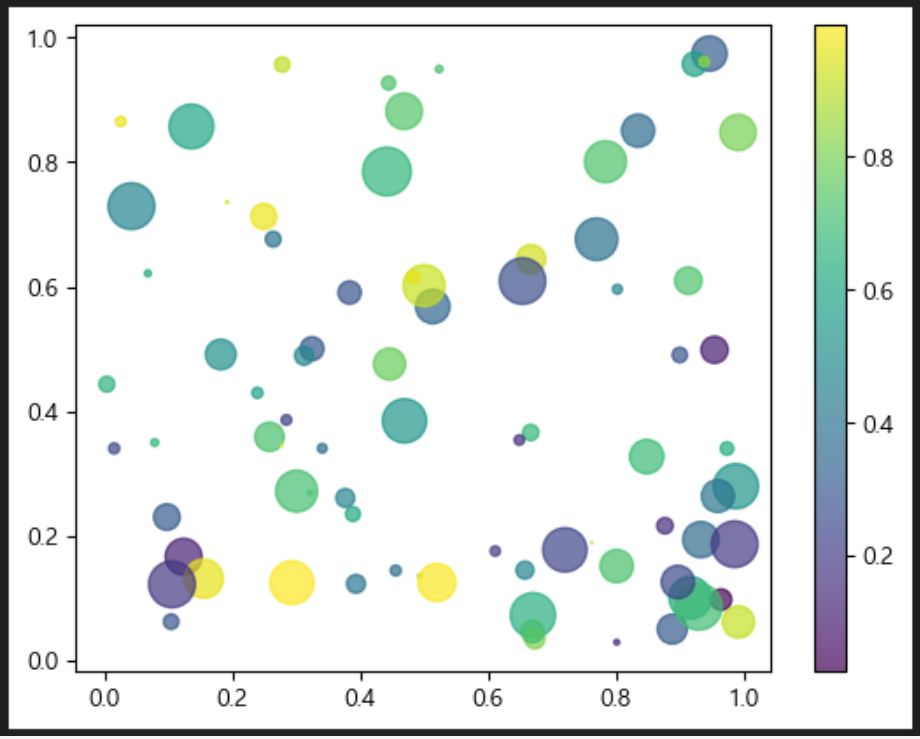


산점도

- 함수 plt.scatter(x, y)
  - 좌표 부분에 속성 marker 유형을 찍는 산점도
  - 마커의 기본은 o로 작은 원
  - 인자 s
    - 크기(size)
  - c
    - 색상(color)
  - alpha
    - 불투명도

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

# Fixing random state for reproducibility
np.random.seed(2025)

N = 80
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (20 * np.random.rand(N)) ** 2  # 0 to 15 point radii

plt.scatter(x, y, s=area, c=colors, alpha=0.7)
plt.colorbar()
plt.show()
```

✓ 0.1s Python



논리 인덱싱(Boolean indexing)

- **np.random.randint(start, stop, size)**
  - [start, end)에서 size의 난수 생성
- **논리 인덱싱**
  - 논리 값이 true인 원소만 추출

```python
import numpy as np

x = np.random.randint(-10, 10, 10) # [-10, 10) 중에서 10개의 랜덤 값 추출

print(x)        # 10개 난수 목록
print(x > 0)    # 논리 연산
print(x[x > 0]) # 논리 인덱싱(boolean indexing)
```

✓ 0.0s Python

```
[-8 -4  5  1 -1 -9  4 -8  2  5]
[False False  True  True False False  True False  True  True]
[5 1 4 2 5]
```

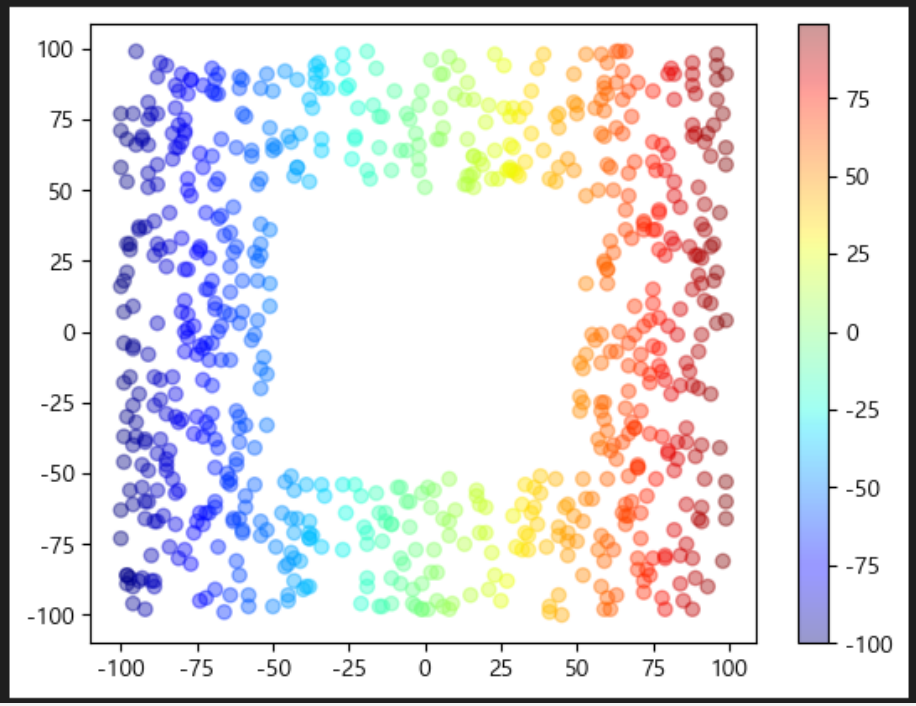
산점도

논리 색인(boolean index)를 만들어 모든 좌표 x와 y에서 mask1 또는 mask2를 만족하는 좌표 추출

- **연산자 +**
  - 논리 or 연산

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

x = np.random.randint(-100, 100, 1000) # 1000개의 랜덤 값 추출
y = np.random.randint(-100, 100, 1000) # 1000개의 랜덤 값 추출

mask1 = abs(x) > 50 # x에 저장된 값 중 절댓값이 50보다 큰 값 걸러 냄
mask2 = abs(y) > 50 # y에 저장된 값 중 절댓값이 50보다 큰 값 걸러 냄
x = x[mask1 + mask2] # mask1과 mask2 중 하나라도 만족하는 값 저장
y = y[mask1 + mask2] # mask1과 mask2 중 하나라도 만족하는 값 저장

plt.scatter(x, y, c=x, cmap='jet', alpha = 0.4)
plt.colorbar()
plt.show()
```

✓ 0.1s Python

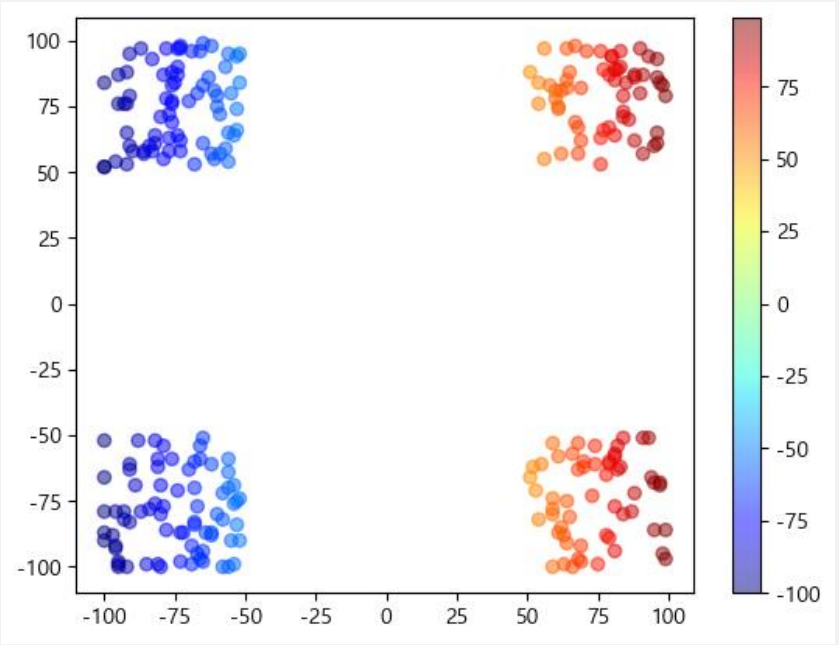


산점도

논리 색인(boolean index)를 만들어 모든 좌표 x와 y에서 mask1와 mask2를 만족하는 좌표 추출

- **연산자 ***
  - 논리 and 연산

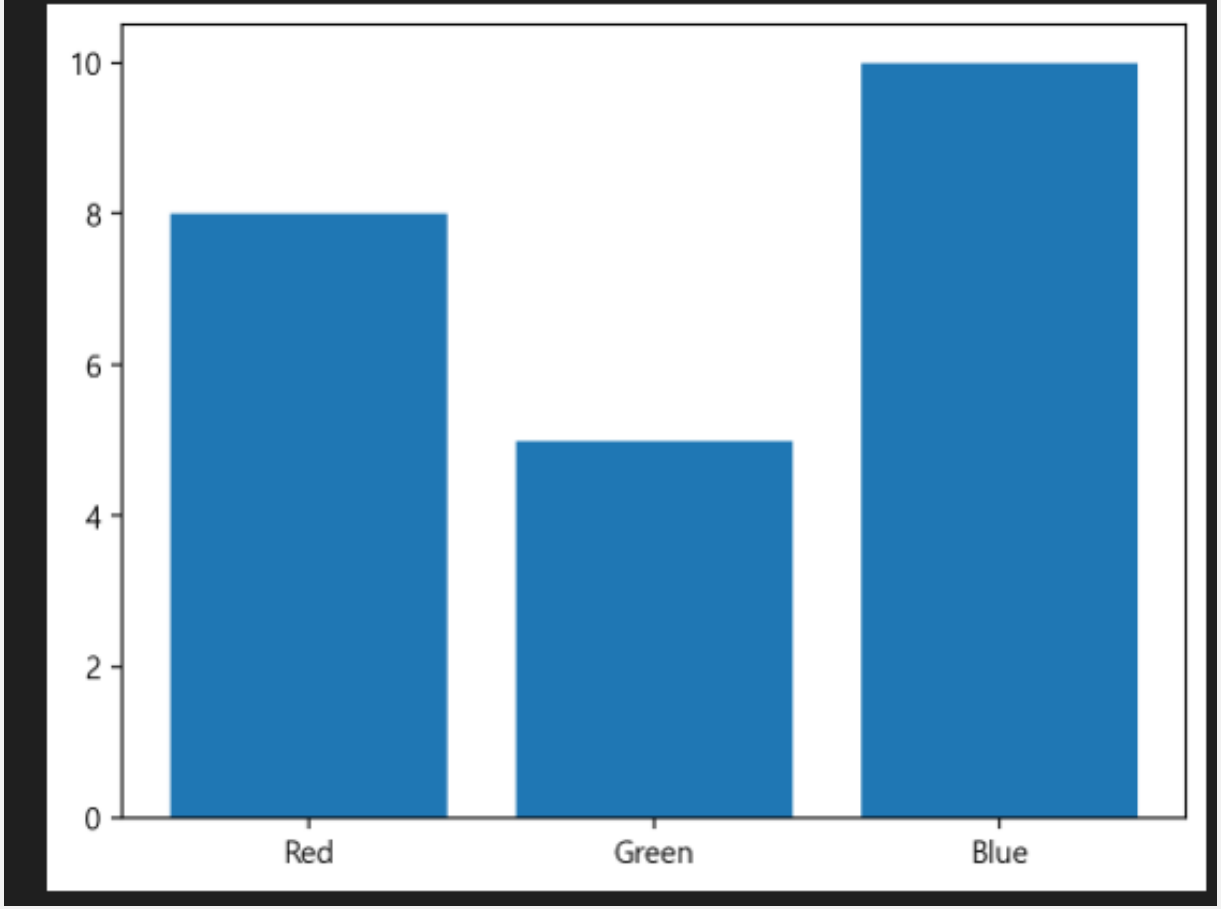
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(-100, 100, 1000) # 1000개의 랜덤 값 추출
y = np.random.randint(-100, 100, 1000) # 1000개의 랜덤 값 추출

mask1 = abs(x) > 50 # x에 저장된 값 중 절댓값이 50보다 큰 값 걸러 냄
mask2 = abs(y) > 50 # y에 저장된 값 중 절댓값이 50보다 큰 값 걸러 냄
x = x[mask1 * mask2] # mask1과 mask2 모두 만족하는 값 저장
y = y[mask1 * mask2] # mask1과 mask2 모두 하나라도 만족하는 값 저장
# x = x[mask1 & mask2] # mask1과 mask2 모두 만족하는 값 저장
# y = y[mask1 & mask2] # mask1과 mask2 모두 하나라도 만족하는 값 저장

plt.scatter(x, y, c=x, cmap='jet', alpha = 0.5)
plt.colorbar()
plt.show()
```



막대 그래프

- **plt.bar()**

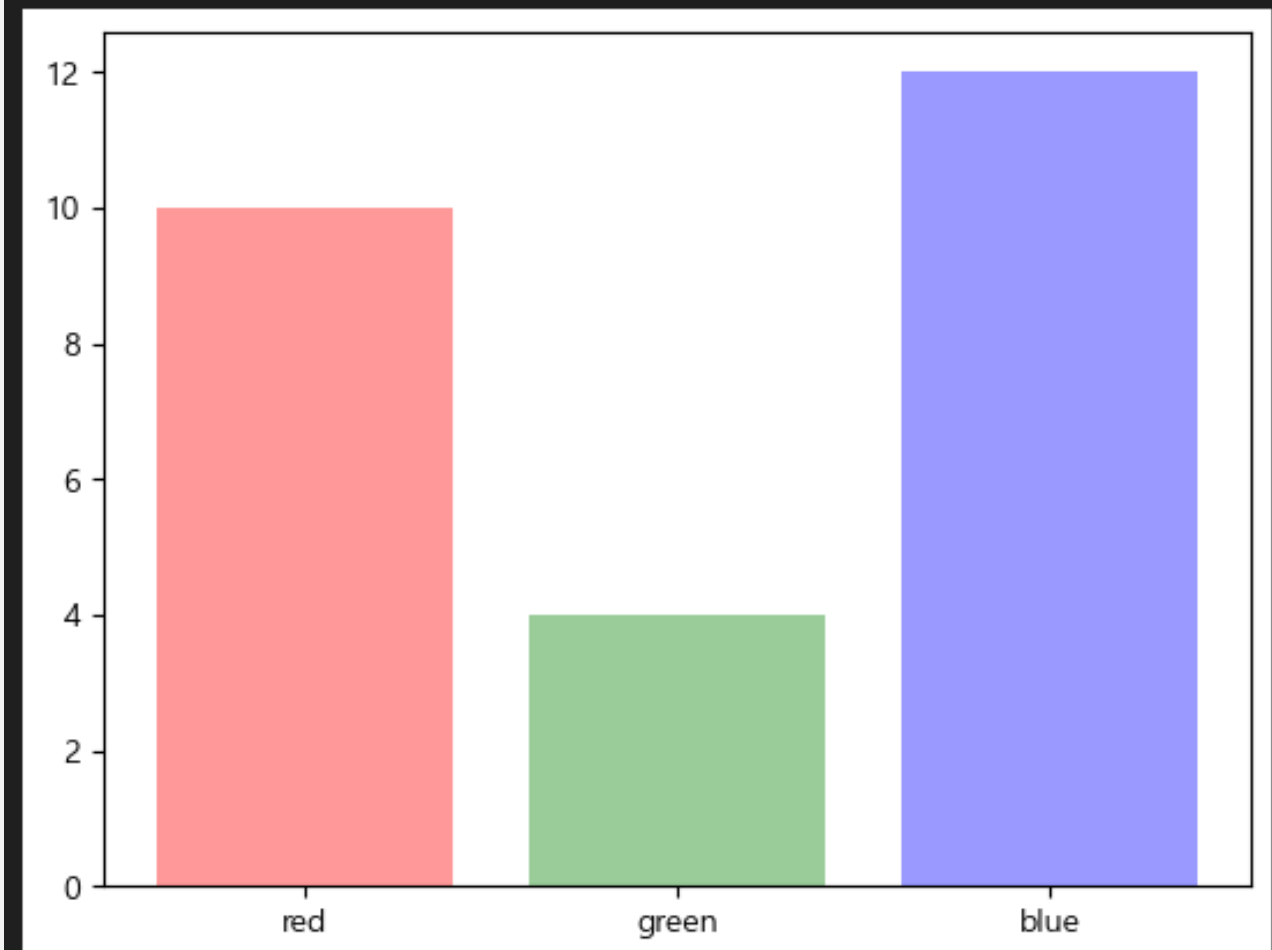
```
import matplotlib.pyplot as plt

plt.bar(["Red", "Green", "Blue"], [8, 5, 10])
plt.show()
```



속성 color

```
import matplotlib.pyplot as plt

c = ["red", "green", "blue"]
plt.bar(c, [10, 4, 12], color=c, alpha=.4)
plt.show()
```

✓ 0.0s



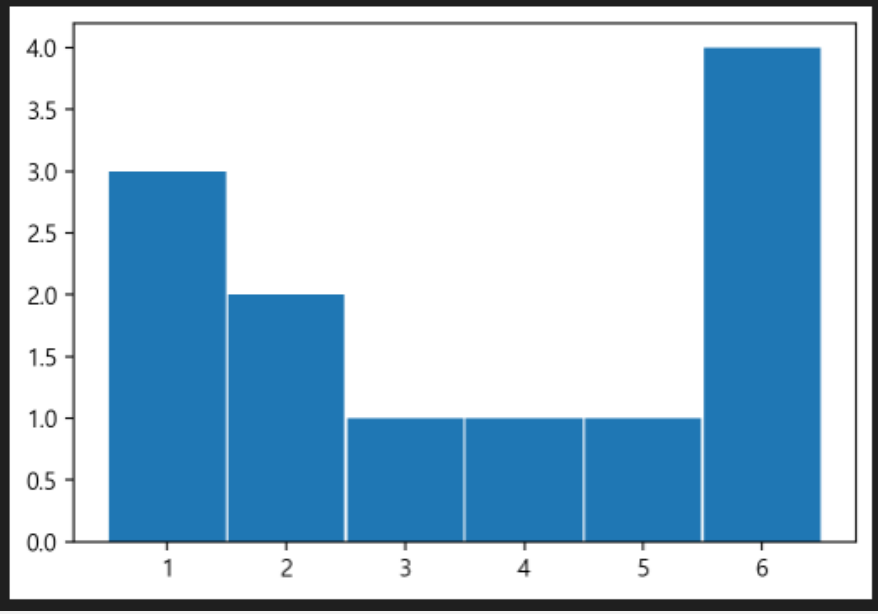
돗수분포표 그림 hist()

데이터의 분포를 시각화하는 데 유용, 데이터가 특정 구간에서 얼마나 자주 발생하는지(빈도)를 나타내는 막대그래프

- **히스토그램 구간 bins**
  - 구간(bin)의 개수 또는 구간의 경계를 지정
    - 정수 값: 구간의 개수를 지정, 기본은 10개
    - 리스트나 배열: 구간의 경계를 명시적으로 지정

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 4)) # 그림 크기 지정

plt.hist([1, 1, 1, 2, 2, 3, 4, 5, 6, 6, 6, 6], rwidth=.98, bins=[.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5])
plt.xticks(range(1, 7))
plt.show()
```

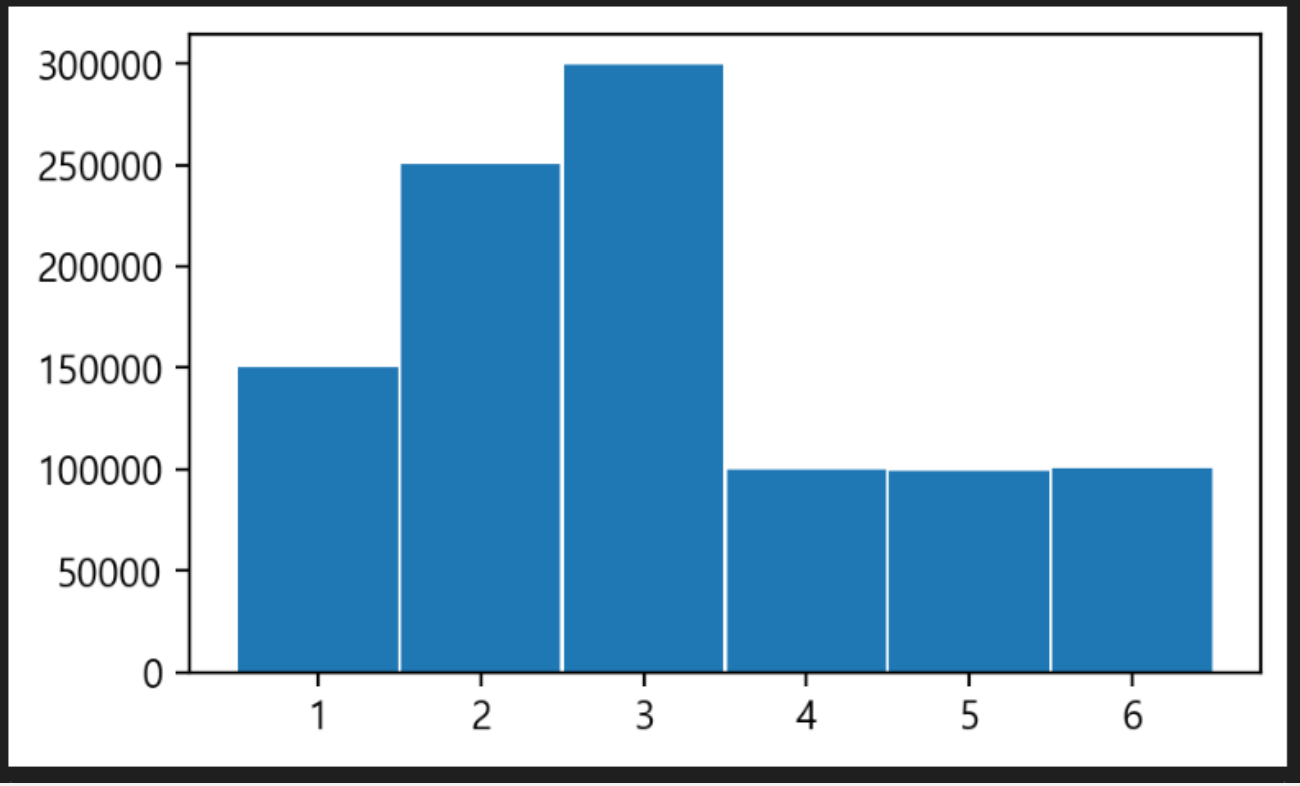
Python



주사위 모의 실험 도수분포표

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

dice = np.random.choice(range(1, 7), 1000000,
                        p = [0.15, 0.25, 0.3, 0.1, 0.1, 0.1])
plt.figure(figsize=(5, 3), dpi=200)

plt.hist(dice, rwidth=.98, bins=np.arange(.5, 7))
plt.xticks(range(1, 7))

plt.savefig('주사위.png')
plt.show()
```

✓ 0.2s Python

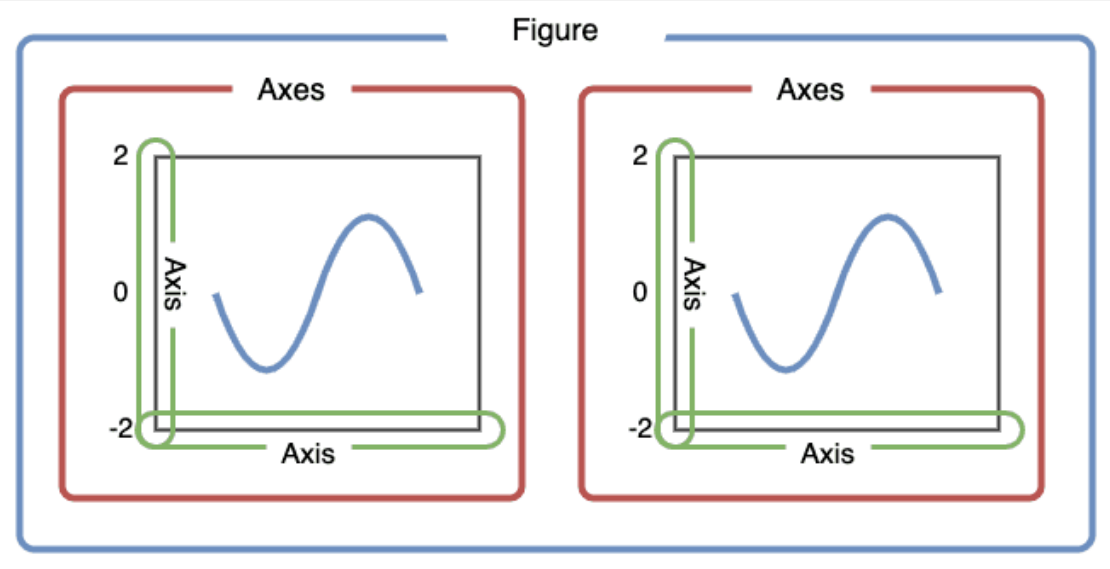


Figure와 axes, 부분그림

figure와 axes 개요

- **전체 그림의 바탕인 Figure**
  - 그래프를 담는 캔버스나 종이 시트
  - 하나의 Figure 객체에는 하나 이상의 부분그림(subplot)인 Axes 객체가 포함 가능

  - ● Figure 개체: 전체 캔버스
  - ● Axes 객체(object):
    - Figure 객체에 속하며 시각화할 데이터를 추가하는 공간
    - Figure 내부에서 실제 그래프가 그려지는 영역
  - ● Axis 객체:
    - Axes 객체에 속하는 축을 말하며, Axis는 다시 X Axis, Y Axis로 분류



서브플롯과 방법

- 하나의 Figure에 여러 개의 그래프를 동시에 그리는 레이아웃
  - 서브플롯과 부분그림이란 용어를 혼용해서 쓸 예정
- 서브플롯을 그리는 방법
  - import matplotlib.pyplot as plt
    - plt는 패키지 matplotlib 하부에 있는 모듈 이름
    - plt 역할
      - 내부적으로 Figure와 Axes 객체를 자동으로 생성하고 관리
  - plt 함수
    - plt.subplot(m, n, index): 반환 값: axes
      - 현재 그림에 세부 그림 축(axes)을 하나 추가하거나 기존 축을 참조
        - Add an Axes to the current figure or retrieve an existing Axes.
    - plt.subplots(m, n): 반환 값이 2개, Fig: Figure, Ax: Axes or array of Axes
      - 캠버스 figure에 지정한 일련의 subplots을 생성
        - Create a figure and a set of subplots.
  - figure 메소드
    - fig.add_subplot(m, n, index), 반환 값: axes
      - 하위 플롯 구성의 일부로 그림에 하나의 Axes을 추가
        - Add an Axes to the figure as part of a subplot arrangement.

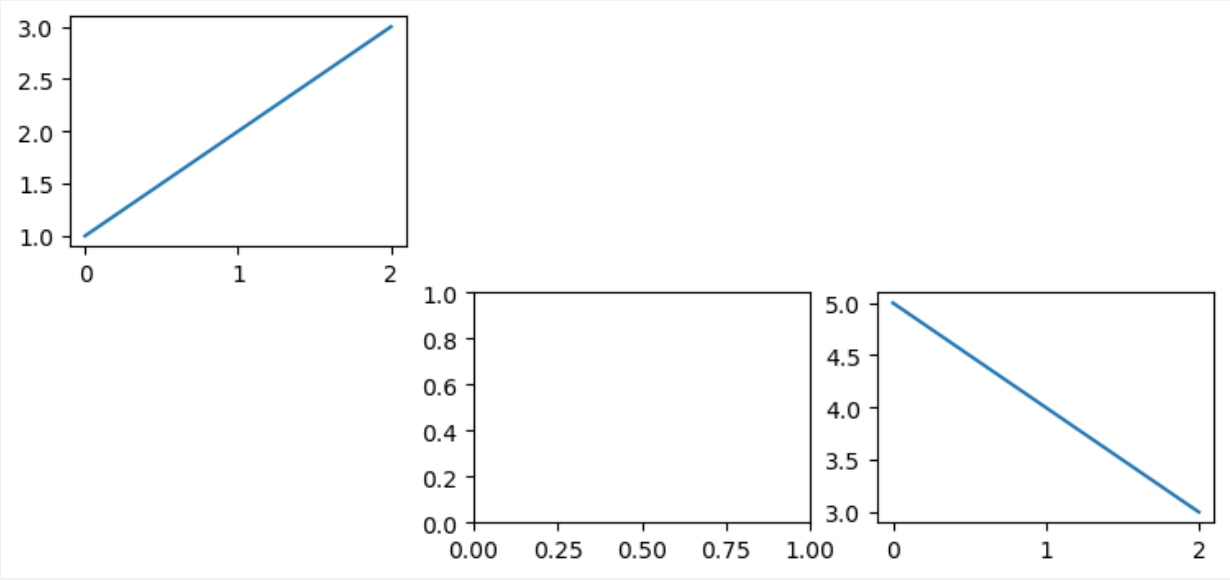
plt.subplot(m, n, index)

현재 그림에 세부 그림 축(axes)을 하나 추가

- **subplot(nrows, ncols, index)의 형식**
  - nrows: 플롯의 행(row) 개수
  - ncols: 플롯의 열(column) 개수
  - index: 현재 활성화할 플롯의 위치(1부터 시작)
- **특징**
  - 개별적으로 각 서브플롯을 지정하는 방식
    - 하나의 플롯을 설정한 후 바로 그 다음 플롯을 설정할 때 유용
  - 다양한 플롯 레이아웃을 쉽게 조작할 수 있는 유연성을 제공
    - 복잡한 다중 플롯 레이아웃에서는 코드가 길어지고 가독성이 떨어질 수 있음

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(9, 4))

plt.subplot(2, 3, 1)
plt.plot([1, 2, 3])
plt.subplot(2, 3, 5)
plt.subplot(2, 3, 6)
plt.plot([5, 4, 3])

plt.show()
```

바로 plt.plot()처럼 그림



plt.subplots()

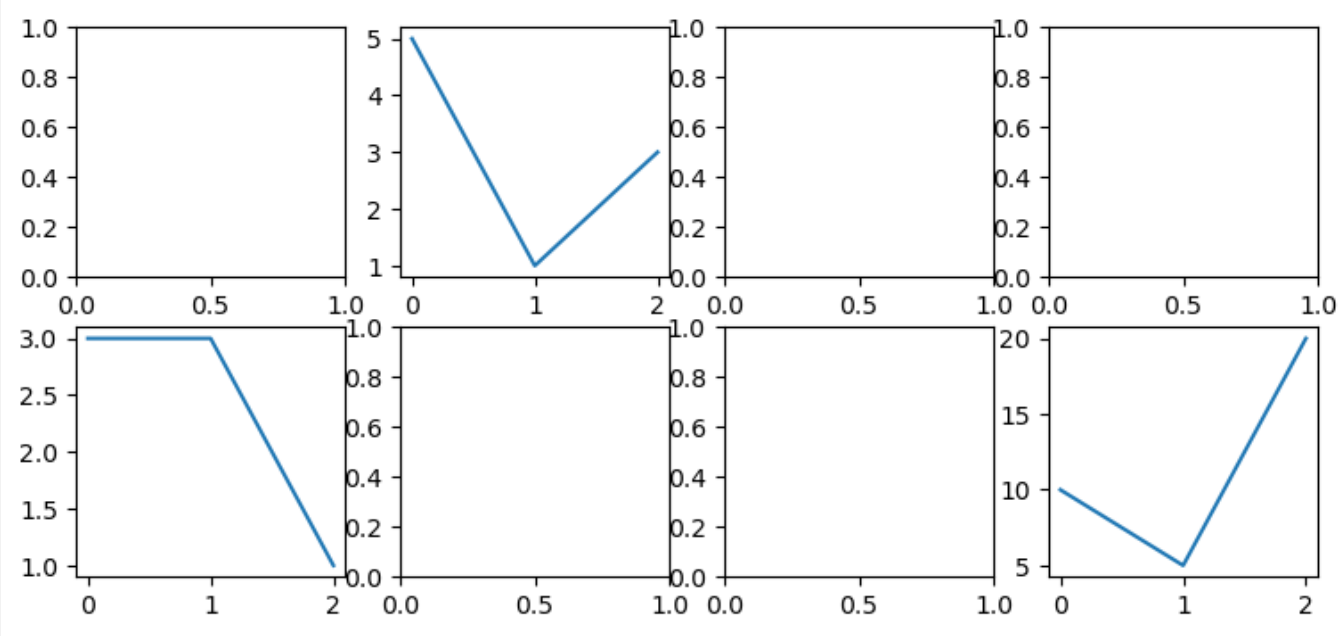
도화지 figure에 지정한 subplots을 생성

- **subplots(nrows, ncols)의 형식**
  - nrows
    - 플롯의 행(row) 개수
  - .ncols: 플롯의 열(column) 개수
- **특징**
  - 한 번의 호출로 여러 개의 Axes 객체를 한꺼번에 생성하여 반환하는 함수
  - 보통 fig와 ax라는 두 개의 객체를 반환
    - fig
      - Figure 객체
    - ax:
      - 여러 개의 Axes 객체 배열 (또는 단일 Axes 객체)
  - 더 직관적이고 효율적으로 다중 플롯을 그릴 수 있게 해 줌
    - 배열 형태로 반환된 Axes 객체를 통해 각각의 플롯에 접근

```python
import matplotlib.pyplot as plt

# plt.figure(figsize=(9, 4)) # 반영이 안됨

fig, axs = plt.subplots(2, 4) # 2행 4열의 서브플롯
fig.set_size_inches((9, 4)) # 캔버스의 크기 지정

axs[0, 1].plot([5, 1, 3])
axs[1, 0].plot([3, 3, 1])
plt.plot([10, 5, 20])
plt.show()
```

반환 받은 ax에서 첨자로 각각의 그림을 그림

마지막 그림(ax[1, 3])은 plt로도 가능



plt.subplots()에서 캔버스 figure의 크기 지정 주의점

- **plt.subplots()**
  - figure와 그 안의 Axes 객체들을 한 번에 생성하는 함수

- **다음 코드의 문제**
  - plt.figure(figsize=(9, 4))을 호출
    - 이미 figure가 생성
  - plt.subplots(2, 4)
    - 다시 새로운 figure를 생성
  - 따라서 두 번째 figure가 생성되며, 첫 번째 figure는 사실상 무시

```
import matplotlib.pyplot as plt

plt.figure(figsize=(9, 4)) # 반영이 안됨

fig, axs = plt.subplots(2, 4) # 2행 4열의 서브플롯
# fig.set_size_inches((9, 4))
plt.show()
```

plt.subplots()에서 캔버스 figure의 크기 지정

- fig의 크기를 지정하는 방법
  - 방법1
    - plt.subplots()에서 figsize=(8, 6)로 직접 지정
      - plt.subplots(2, 1, figsize=(8, 6))
  - 방법2
    - plt.subplots()에서 반환 받은 fig에서 메소드 set_size_inches(8, 6)로 지정
      - fig, axes = plt.subplots(2, 1)
      - fig.set_size_inches(10, 4)

```
# plt.subplots()에서 직접 캔버스의 크기 지정

import matplotlib.pyplot as plt

# plt.subplots()로 figure와 크기를 지정
fig, axes = plt.subplots(2, 1, figsize=(8, 6))

# 각 서브플롯에 데이터 추가
axes[0].plot([1, 2, 3], [1, 4, 9])
axes[1].plot([1, 2, 3], [1, 2, 3])

plt.show()
```

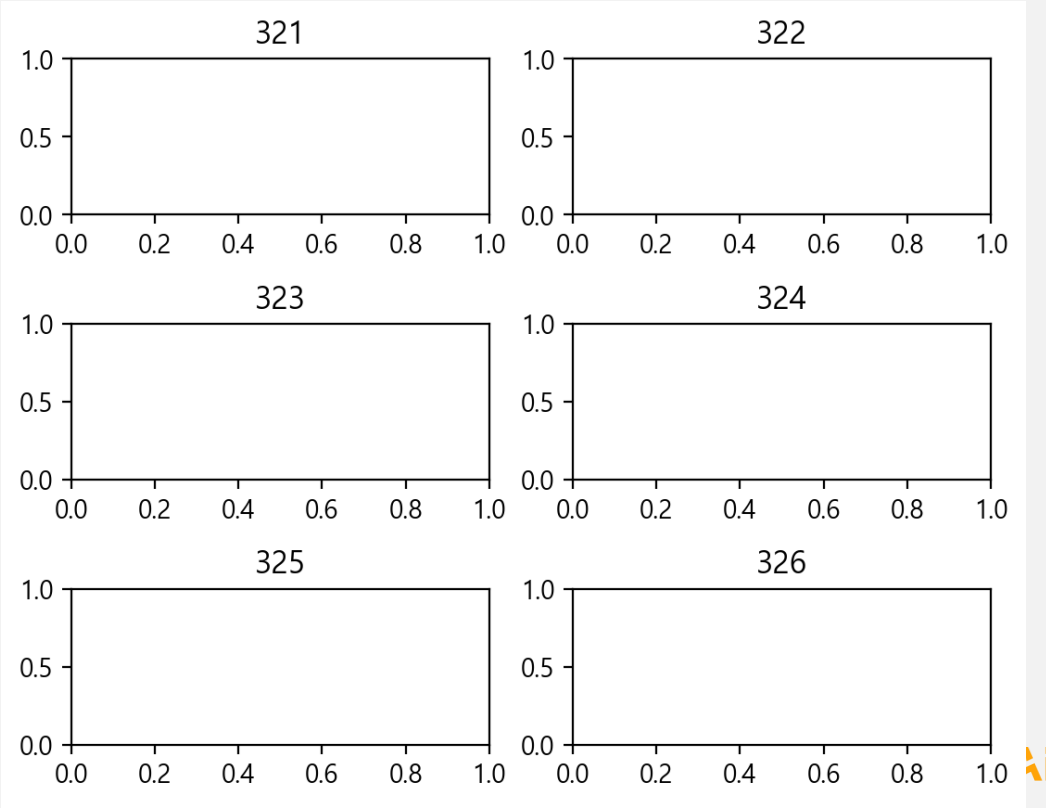
```
# plt.subplots()에서 직접 캔버스의 크기 지정

import matplotlib.pyplot as plt

# plt.subplots()로 figure와 크기를 지정
fig, axes = plt.subplots(2, 1, figsize=(8, 6))
fig.set_size_inches(10, 4)

# 각 서브플롯에 데이터 추가
axes[0].plot([1, 2, 3], [1, 4, 9])
axes[1].plot([1, 2, 3], [1, 2, 3])

plt.show()
```

서브플롯 fig.add_subplot(m, n, index) 개요

하위 플롯 구성의 일부로 그림에 하나의 Axes을 추가

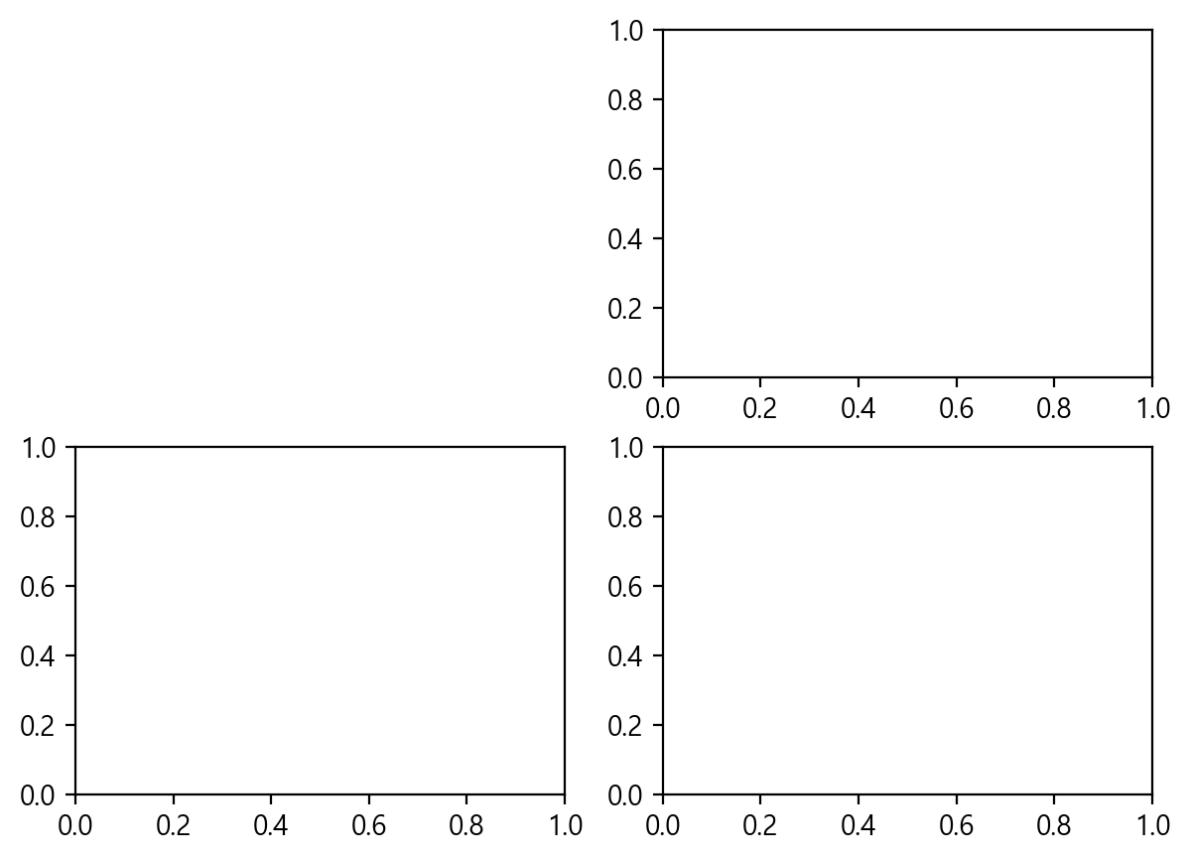
- **fig.add_subplots(m, n, index)으로 부분그림 그리기**
  - 부분그림에서 m은 행의 수, n은 열의 수
  - index는 1부터 시작하는 부분플롯의 번호
    - index는 행이 우선
  - (3, 2, 1)은 (321)로 쓰는 것이 가능
    - (321)로 사용하는 경우, 단 단위 자리만 가능하므로 1에서 9까지만 가능

```python
import matplotlib.pyplot as plt

# define figure
fig = plt.figure()

# add subplots
fig.add_subplot(321).set_title('321')
fig.add_subplot(322).set_title('322')
fig.add_subplot(323).set_title('323')
fig.add_subplot(324).set_title('324')
fig.add_subplot(325).set_title('325')
fig.add_subplot(326).set_title('326')

# adjust spaces
fig.subplots_adjust(wspace=.2, hspace=.7)
plt.show()
```



서브플롯 fig.add_subplot() 활용

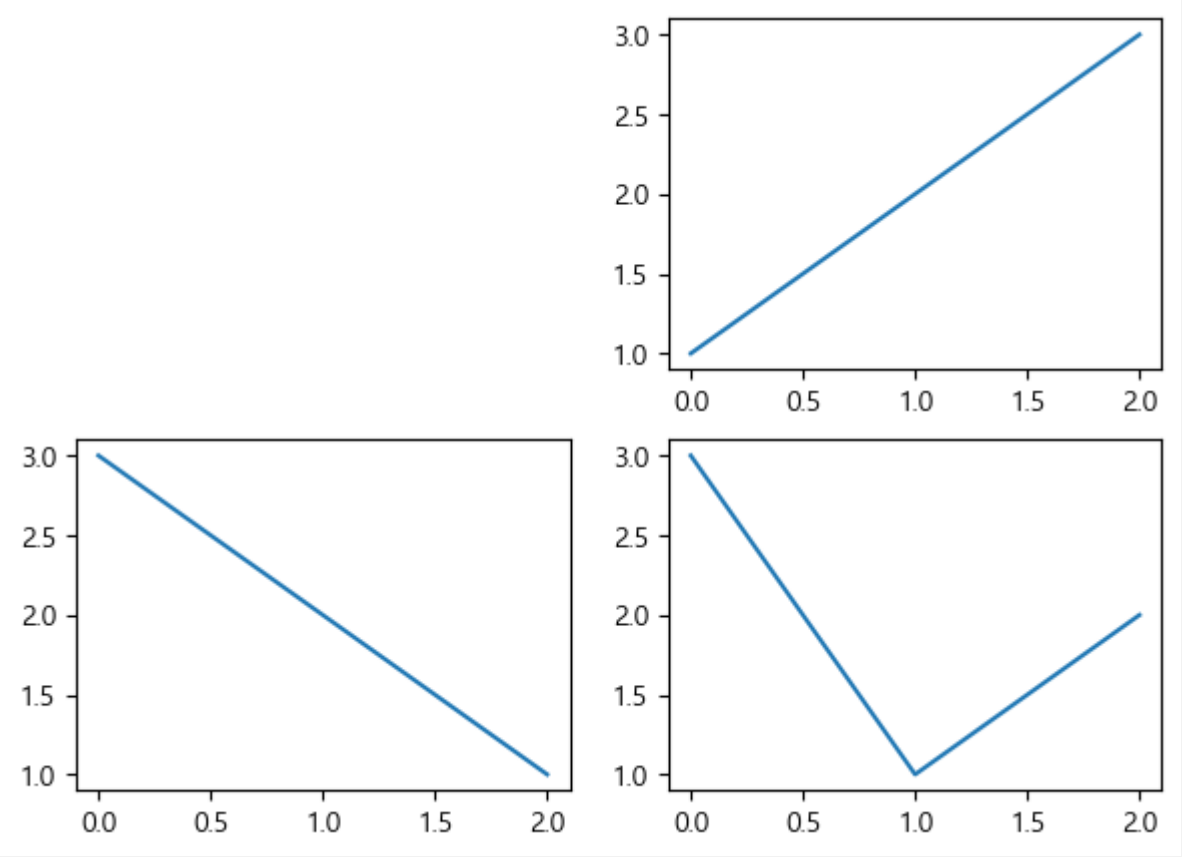
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(7, 5))

ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)
```



서브플롯 fig.add_subplot() 그리기

- 반환 값 ax로 그리기

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(7, 5))

ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

ax2.plot([1, 2, 3])
ax3.plot([3, 2, 1])
ax4.plot([3, 1, 2]);
```



plt.subplots()과 fig.add_subplot() 비교

- **plt.subplots()**
  - 간단한 다중 플롯 생성에 적합
  - 한 번에 figure와 axes 객체를 모두 생성해 간결한 코드 작성이 가능
- **fig.add_subplot()**
  - 개별적으로 Axes를 추가해야 하지만, 복잡한 레이아웃이나 더 세밀한 제어가 필요할 때 유용

특징	plt.subplots()	fig.add_subplot()
역할	figure와 axes 객체를 한 번에 생성	figure를 생성한 후 개별적으로 Axes 추가
figure 생성	자동으로 생성됨	plt.figure()로 먼저 생성해야 함
Axes 생성	한 번에 여러 Axes 객체를 배열 형태로 반환	각각의 Axes를 하나씩 추가
코드 간결성	더 간결함. 한 번의 호출로 여러 서브플롯 생성 가능	복잡해질 수 있음. 서브플롯을 하나씩 추가해야 함
figsize 설정	plt.subplots(figsize=(width, height))로 지정	plt.figure(figsize=(width, height))로 지정
반환 값	figure와 axes 배열을 동시에 반환	add_subplot() 호출 시 개별 Axes 반환
서브플롯 레이아웃	한 번에 설정 (nrows, ncols 인자 사용)	각 서브플롯을 개별적으로 추가 (add_subplot)
적합한 사용 사례	다수의 서브플롯을 그릴 때 빠르고 간결하게 사용	서브플롯을 개별적으로 세밀하게 제어할 때 유용
축 공유 기능	sharex, sharey 옵션으로 축 공유 가능	수동으로 설정해야 함
복잡한 레이아웃 제어	자동 배치. 복잡한 레이아웃에 덜 유연	복잡한 레이아웃에서 더 유연하게 제어 가능
사용 예시	fig, axes = plt.subplots(2, 2)	fig = plt.figure(); ax = fig.add_subplot(2, 2, 1)

5.2 맞춤형 figure 레이아웃 서브플롯

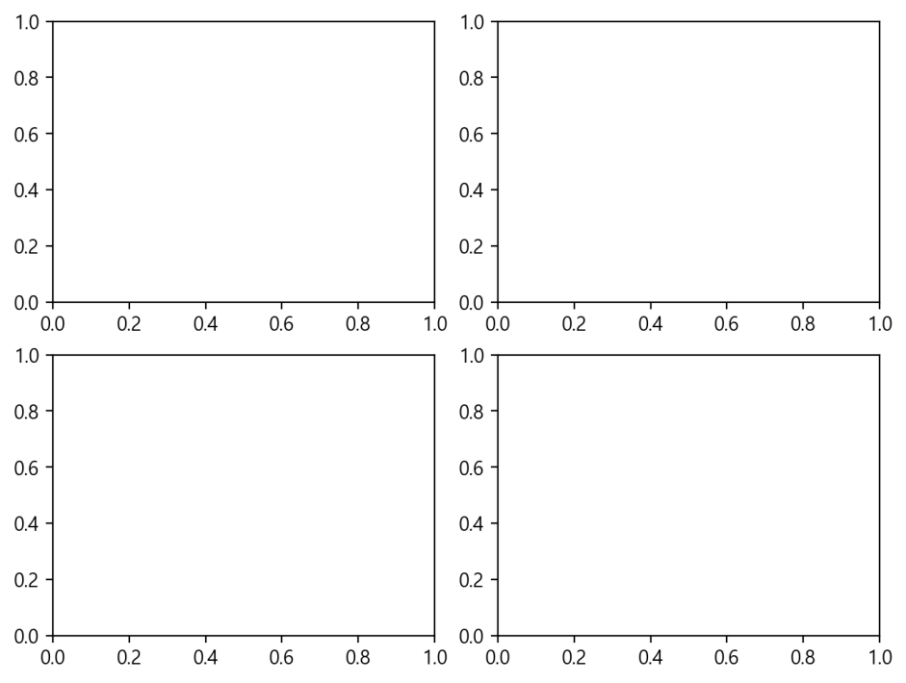
-

GridSpec 개요

서브플롯의 크기와 위치를 세밀하게 조정하고 싶을 때는 GridSpec을 사용하는 것이 적합

- **0. 캔버스 얻기**
  - fig = plt.figure()
- **1. GridSpec 객체 생성**
  - GridSpec 객체를 생성할 때 그리드의 행과 열의 개수를 설정
  - 인자 figure에 위에서 얻은 fig를 지정
- **2. add_subplot()으로 서브플롯 추가**
  - GridSpec 객체 내에서 서브플롯의 위치를 지정한 Gridspec 인스턴스를 지정
    - Gridspec의 요소는 일반적으로 numpy 배열처럼 처마 방식으로 참조
- **2행 2열의 레이아웃을 GridSpec**

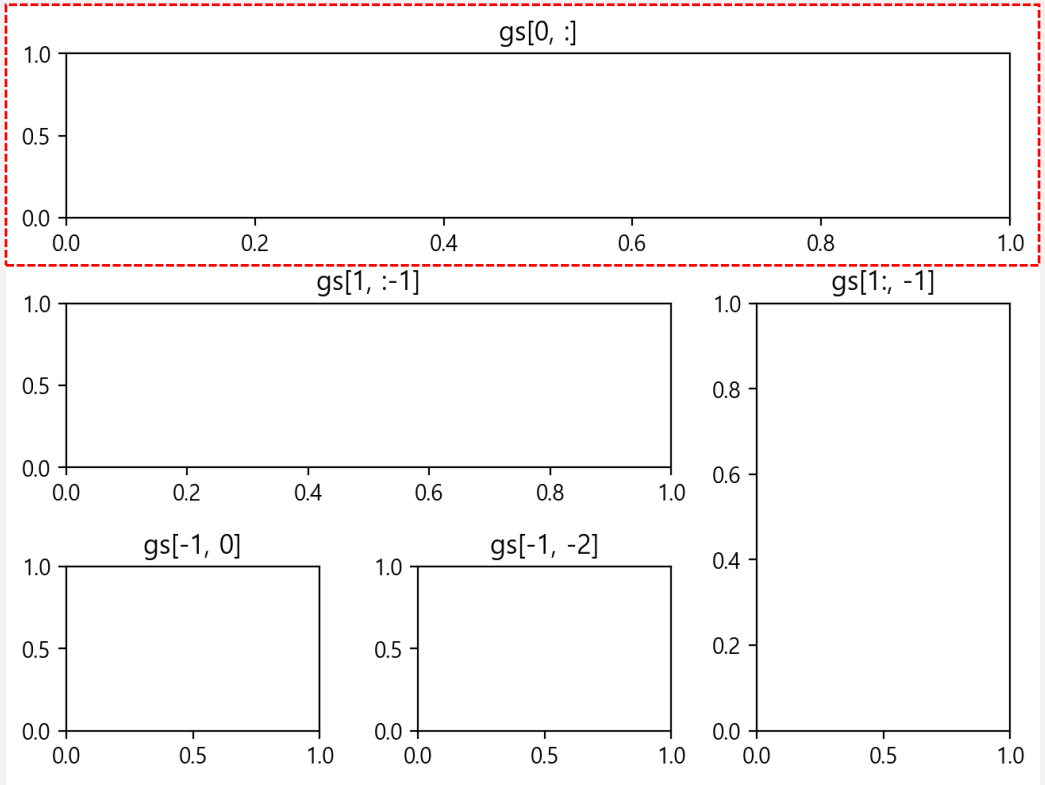
```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig2 = plt.figure(constrained_layout=True)
spec2 = gridspec.GridSpec(ncols=2, nrows=2, figure=fig2)

f2_ax1 = fig2.add_subplot(spec2[0, 0])
f2_ax2 = fig2.add_subplot(spec2[0, 1])
f2_ax3 = fig2.add_subplot(spec2[1, 0])
f2_ax4 = fig2.add_subplot(spec2[1, 1])
```



슬라이스를 활용한 GridSpec

- **Gridspec의 장점**
  - 적절히 행과 열을 합친 하위 그림을 생성
- **3행 3열의 그리드 배치**
  - fig3 = plt.figure(constrained_layout=True)
  - gs = fig3.add_gridspec(3, 3)
- **gs[0, :]은 0행과 모든 열의 배치하는 서브플롯**
  - f3_ax1 = fig3.add_subplot(gs[0, :])

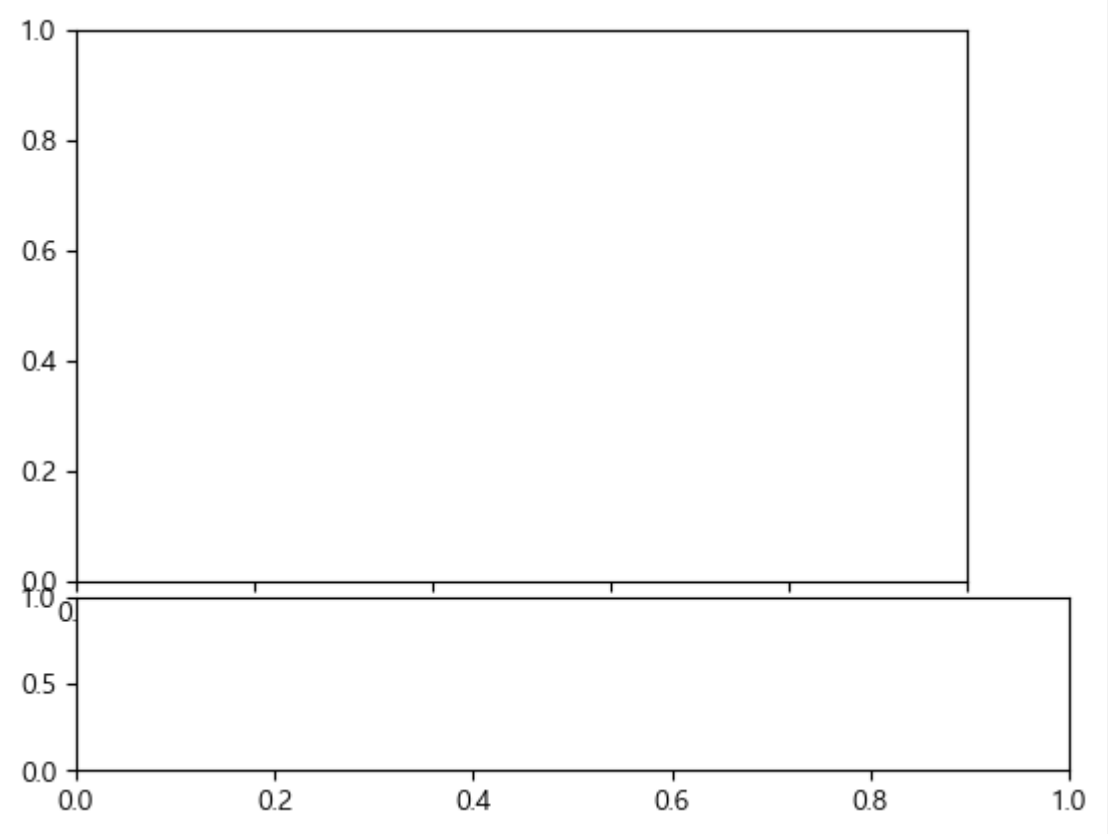
```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig3 = plt.figure(constrained_layout=True)
gs = fig3.add_gridspec(3, 3)

f3_ax1 = fig3.add_subplot(gs[0, :])
f3_ax1.set_title('gs[0, :]')
f3_ax2 = fig3.add_subplot(gs[1, :-1])
f3_ax2.set_title('gs[1, :-1]')
f3_ax3 = fig3.add_subplot(gs[1:, -1])
f3_ax3.set_title('gs[1:, -1]')
f3_ax4 = fig3.add_subplot(gs[-1, 0])
f3_ax4.set_title('gs[-1, 0]')
f3_ax5 = fig3.add_subplot(gs[-1, -2])
f3_ax5.set_title('gs[-1, -2]');
```



GridSpec을 활용한 다양한 그리드 레이아웃

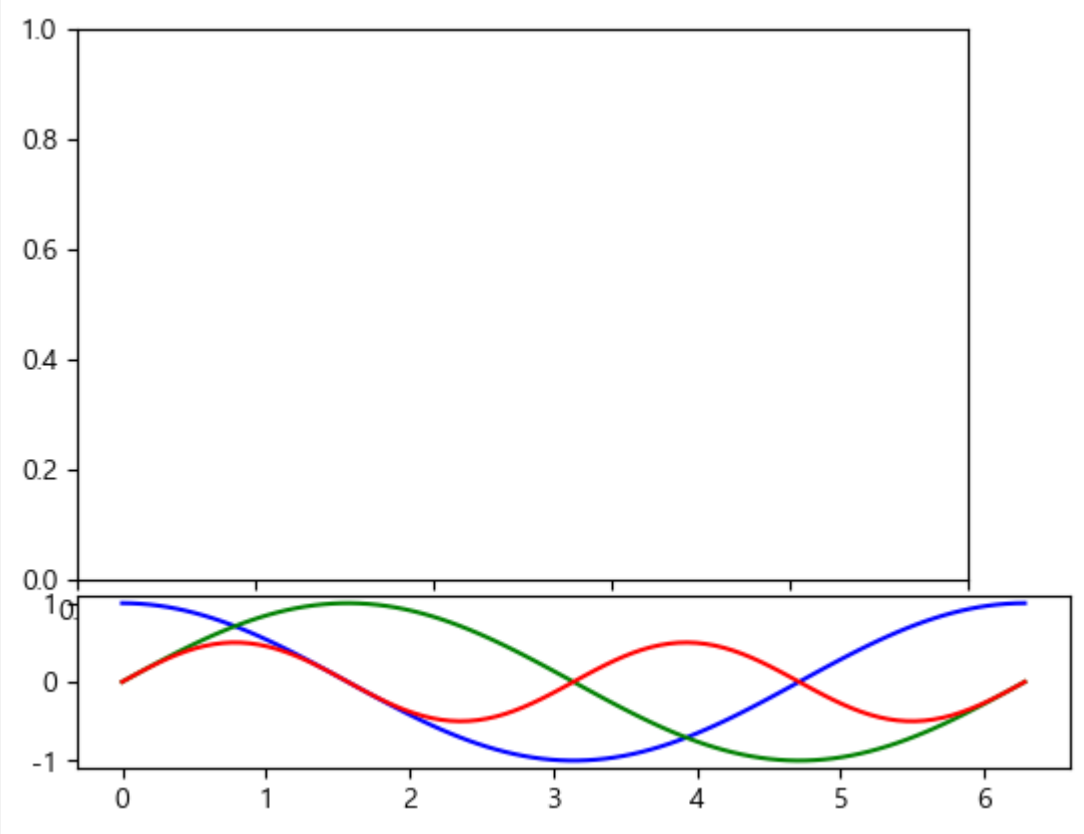
- **8행, 39열의 배치 형태의 GridSpec**
  - 서브플롯 ax1은 슬라이싱 배열 참조 gs[:6, :35]
    - 0행에서 5행까지, 0열에서 34열까지를 통합하는 배치
    - 마찬가지로 ax2는 6행에서 마지막 행까지, 모든 열을 통합하는 배치

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

gs = GridSpec(8, 39)
ax1 = plt.subplot(gs[:6, :35])
ax2 = plt.subplot(gs[6:, :])
```



삼각함수 그리기

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

gs = GridSpec(8, 39)
ax1 = plt.subplot(gs[:6, :35])
ax2 = plt.subplot(gs[6:, :])

x = np.linspace(0, 2*np.pi, 100)
ax2.plot(x, np.cos(x), color='blue')
ax2.plot(x, np.sin(x), color='green')
ax2.plot(x, np.cos(x)*np.sin(x), color='red')
```



난수 [0, 1)

- **numpy.randomn.rand()**

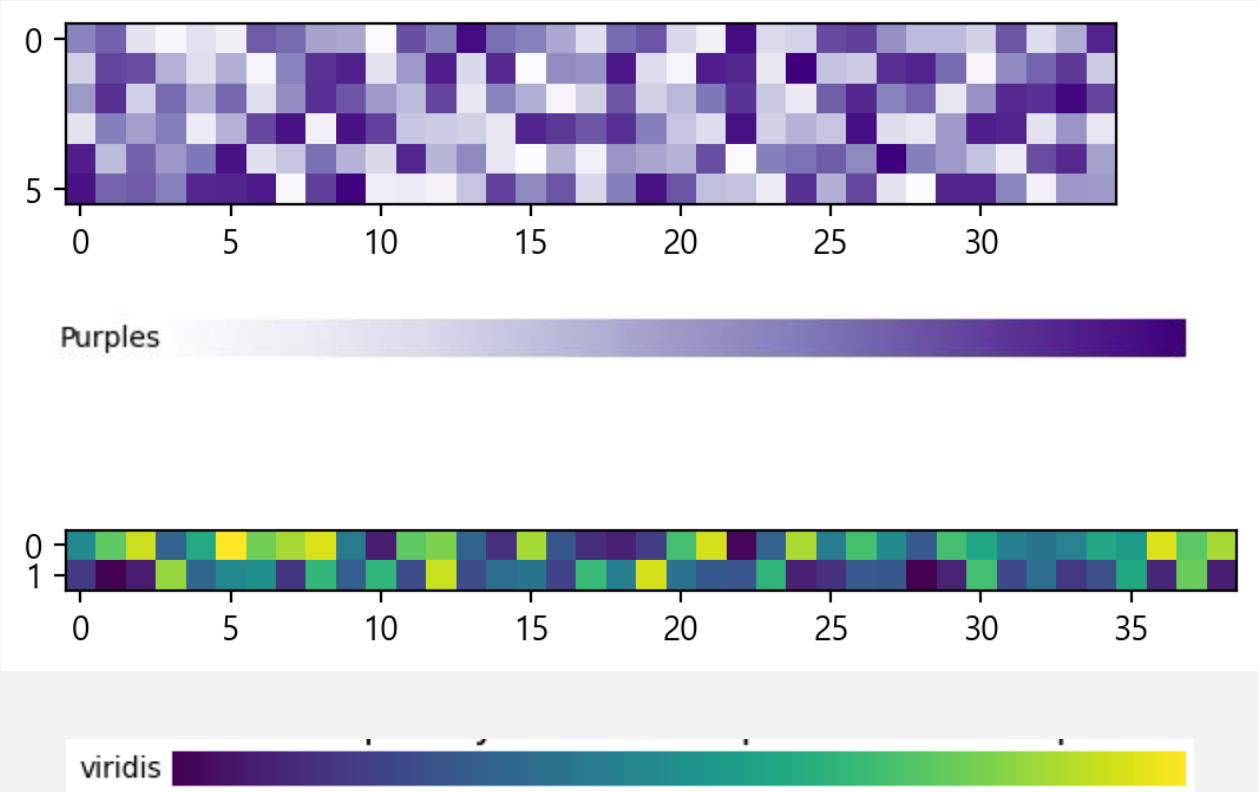
```
np.random.seed(2024)
np.set_printoptions(precision=2)
data1 = np.random.rand(6, 35) # [0, 1) 난수 6 * 35
data2 = np.random.rand(2, 39) # [0, 1) 난수 2 * 39
print(data1)
print()
print(data2)
print()
data1[4, 0], data1[5, 0]
```

```
[[0.59 0.7  0.19 0.04 0.21 0.11 0.73 0.68 0.47 0.45 0.02 0.75 0.6  0.96
  0.66 0.61 0.45 0.23 0.67 0.74 0.26 0.1  0.96 0.25 0.28 0.77 0.8  0.54
  0.38 0.38 0.29 0.74 0.24 0.44 0.88]
 [0.29 0.78 0.76 0.42 0.23 0.42 0.06 0.6  0.84 0.89 0.2  0.5  0.9  0.26
  0.87 0.02 0.55 0.53 0.92 0.25 0.06 0.9  0.87 0.16 1.   0.35 0.31 0.85
  0.88 0.68 0.05 0.56 0.69 0.82 0.31]
 [0.51 0.85 0.29 0.68 0.42 0.68 0.22 0.55 0.85 0.74 0.5  0.38 0.79 0.17
  0.59 0.43 0.06 0.29 0.73 0.29 0.39 0.64 0.83 0.32 0.16 0.71 0.87 0.59
  0.69 0.17 0.53 0.87 0.84 0.97 0.78]
 [0.2  0.61 0.48 0.62 0.14 0.41 0.78 0.94 0.1  0.94 0.8  0.33 0.31 0.29
  0.17 0.88 0.82 0.74 0.85 0.61 0.34 0.23 0.94 0.29 0.41 0.34 0.95 0.23
  0.17 0.49 0.9  0.88 0.19 0.52 0.17]
 [0.9  0.38 0.7  0.51 0.63 0.93 0.21 0.34 0.66 0.41 0.26 0.88 0.4  0.56
  0.17 0.01 0.41 0.1  0.53 0.46 0.41 0.76 0.02 0.61 0.65 0.72 0.57 1.
  0.61 0.5  0.35 0.14 0.76 0.87 0.47]
 [0.94 0.69 0.72 0.6  0.88 0.88 0.91 0.04 0.8  0.99 0.13 0.14 0.07 0.34
  0.8  0.57 0.74 0.26 0.61 0.94 0.74 0.36 0.35 0.13 0.84 0.43 0.78 0.19
  0.   0.88 0.88 0.58 0.1  0.51 0.49]]

[[0.49 0.75 0.92 0.33 0.62 1.   0.78 0.87 0.95 0.44 0.1  0.75 0.8  0.34
  0.16 0.86 0.29 0.15 0.11 0.2  0.71 0.93 0.04 0.33 0.87 0.43 0.71 0.49
  0.31 0.71 0.6  0.45 0.4  0.46 0.61 0.56 0.94 0.75 0.86]
 [0.19 0.03 0.1  0.84 0.35 0.48 0.51 0.18 0.67 0.32 0.67 0.25 0.91 0.26
  0.38 0.4  0.22 0.68 0.44 0.93 0.39 0.29 0.28 0.66 0.11 0.16 0.3  0.29
  0.02 0.12 0.71 0.24 0.38 0.18 0.26 0.61 0.14 0.77 0.11]]


(0.8995458788323055, 0.9420867848216234)
```

imshow(..., cmp="Purples")

- 6행, 35열의 난수와 2행, 38열의 이미지를 GridSpec으로 레이아웃을 배치해 그린 그림

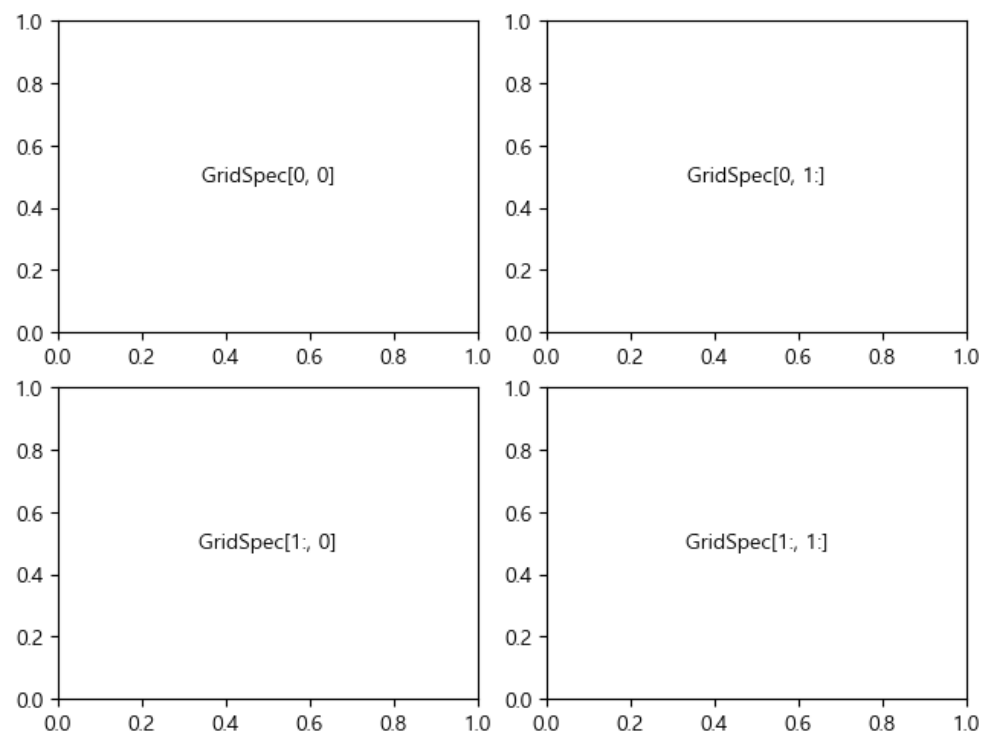
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

gs = GridSpec(8, 39)
ax1 = plt.subplot(gs[:6, :35])
ax2 = plt.subplot(gs[6:, :])

# Fixing random state for reproducibility
np.random.seed(2024)
data1 = np.random.rand(6, 35)
data2 = np.random.rand(2, 39)

ax1.imshow(data1, cmap="Purples")
ax2.imshow(data2)

plt.show()
```



axes.annotate()로 텍스트 추가

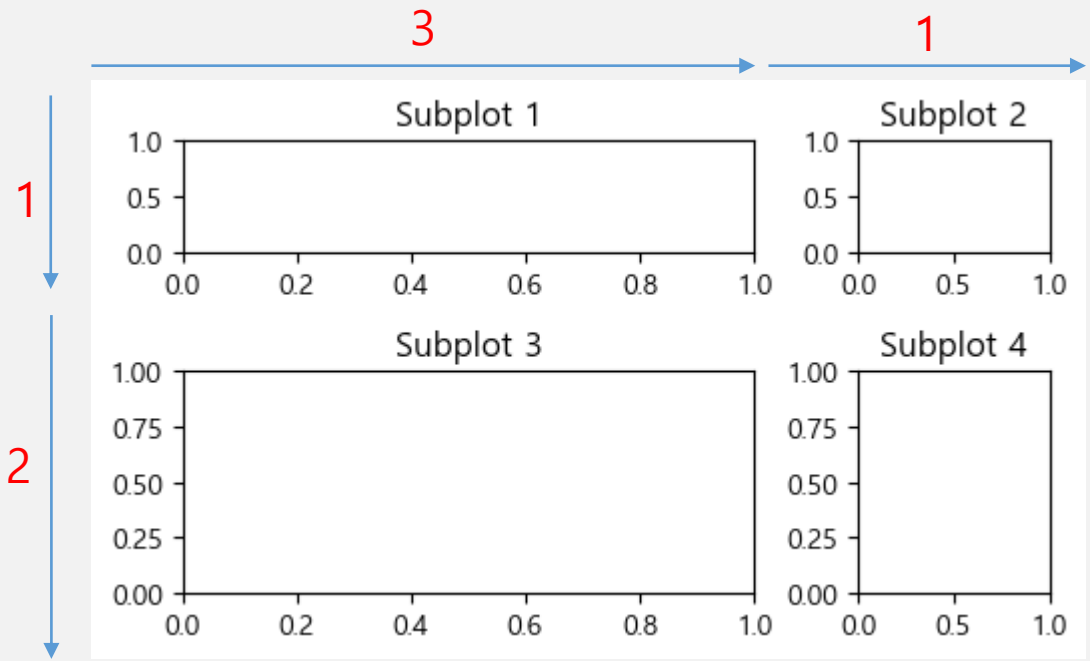
- 인자 xy=(0.5, 0.5), xycoords='axes fraction'
  - 각 그림의 중앙에 글자 쓰기
    - 'axes fraction'
      - Fraction of Axes from lower left

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig4 = plt.figure(constrained_layout=True)
spec4 = fig4.add_gridspec(ncols=2, nrows=2)

anno_opts = dict(xy=(0.5, 0.5), xycoords='axes fraction',
                 va='center', ha='center')

f4_ax1 = fig4.add_subplot(spec4[0, 0])
f4_ax1.annotate('GridSpec[0, 0]', **anno_opts)
fig4.add_subplot(spec4[0, 1]).annotate('GridSpec[0, 1:]', **anno_opts)
fig4.add_subplot(spec4[1, 0]).annotate('GridSpec[1:, 0]', **anno_opts)
fig4.add_subplot(spec4[1, 1]).annotate('GridSpec[1:, 1:]', **anno_opts);
```



가로와 세로 비율로 레이아웃 배치

- 서브플롯의 가로와 세로의 길이 비율을 각각 지정하는 옵션
  - width_ratios 및 height_ratios 매개변수
    - 인자 유형은 상대 비율의 숫자 목록
  - width_ratios=[3, 1]
    - 가로의 비율을 3: 1
  - height_ratios=[1, 2]
    - 세로의 비율을 1: 2

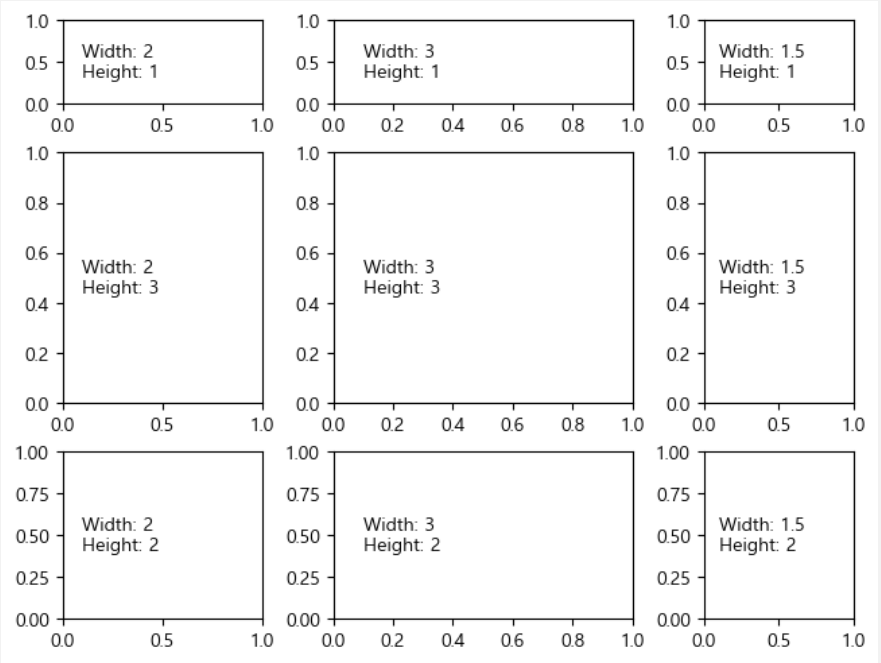



```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig = plt.figure(figsize=(5, 3))
gs = GridSpec(2, 2, width_ratios=[3, 1], height_ratios=[1, 2])

ax1 = fig.add_subplot(gs[0, 0])
ax1.set_title('Subplot 1')

ax2 = fig.add_subplot(gs[0, 1])
ax2.set_title('Subplot 2')

ax3 = fig.add_subplot(gs[1, 0])
ax3.set_title('Subplot 3')

ax4 = fig.add_subplot(gs[1, 1])
ax4.set_title('Subplot 4')

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

가로와 세로 비율로 레이아웃 배치 활용

- 서브플롯의 가로와 세로의 길이 비율을 각각 지정하는 옵션
  - width_ratios 및 height_ratios 매개변수
    - 인자 유형은 상대 비율의 숫자 목록
    - width_ratios=[2, 4, 8]과 width_ratios=[1, 2, 4]는 같은 의미
  - 3행 3열로 배치한 서브플롯
    - 가로를 각각 widths = [2, 3, 1.5]로 지정
      - 소수를 빼고 widths = [4, 6, 3]으로 해도 같은 결과
    - 세로를 heights = [1, 3, 2]로 지정

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig5 = plt.figure(constrained_layout=True)
widths = [2, 3, 1.5]
heights = [1, 3, 2]

spec5 = fig5.add_gridspec(ncols=3, nrows=3,
                          width_ratios=widths, height_ratios=heights)
for row in range(3):
    for col in range(3):
        ax = fig5.add_subplot(spec5[row, col])
        label = 'Width: {}\nHeight: {}'.format(widths[col], heights[row])
        ax.annotate(label, (0.1, 0.5), xycoords='axes fraction', va='center')
```



그리드 서브플롯의 추가와 삭제

서브플롯을 이용하여 대부분의 서브플롯을 만든 후 일부를 제거하고 결합하는 것이 편리

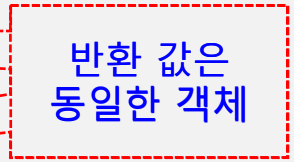
- **서브플롯과 axes.get_gridspec()**
  - 서브플롯이 속해 있는 GridSpec 객체를 반환하는 역할

- **3행 3열의 그리드 배치 생성**

```
fig7, f7_axs = plt.subplots(ncols=3, nrows=3)
```

- **메소드 get_gridspec()의 결과를 gs에 저장해 서브플롯으로 추가 가능**

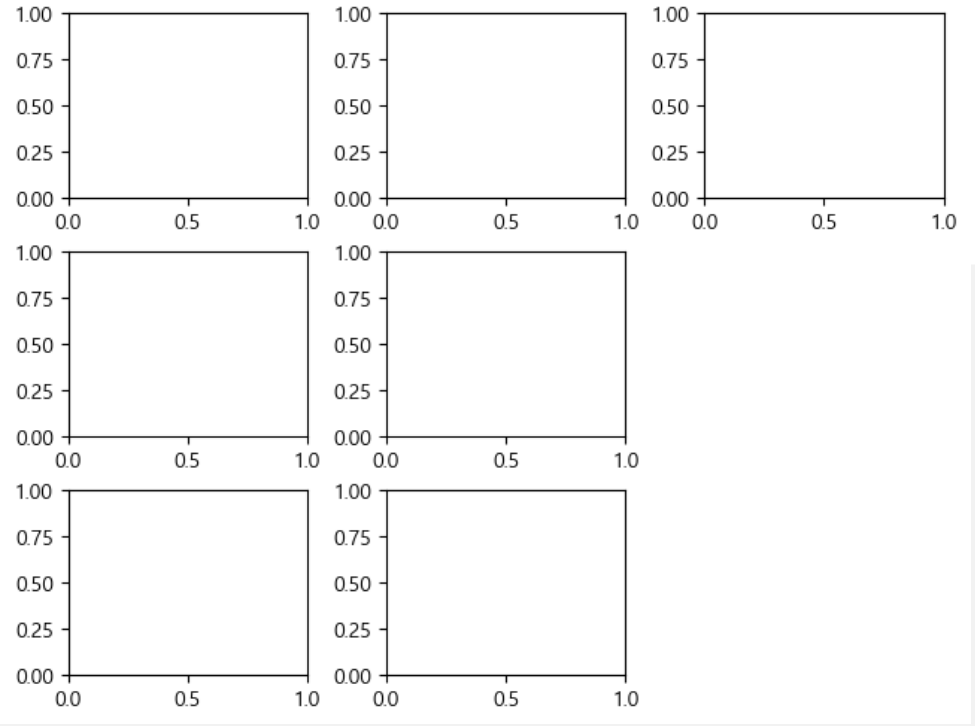
```
>>> import matplotlib.pyplot as plt
>>> from matplotlib.gridspec import GridSpec
>>>
>>> fig7, f7_axs = plt.subplots(ncols=3, nrows=3)
>>> gs = f7_axs[1, 2].get_gridspec()
>>> print(gs)
GridSpec(3, 3)
>>> gs = f7_axs[0, 2].get_gridspec()
>>> print(gs)
GridSpec(3, 3)
```



서브플롯의 추가와 삭제

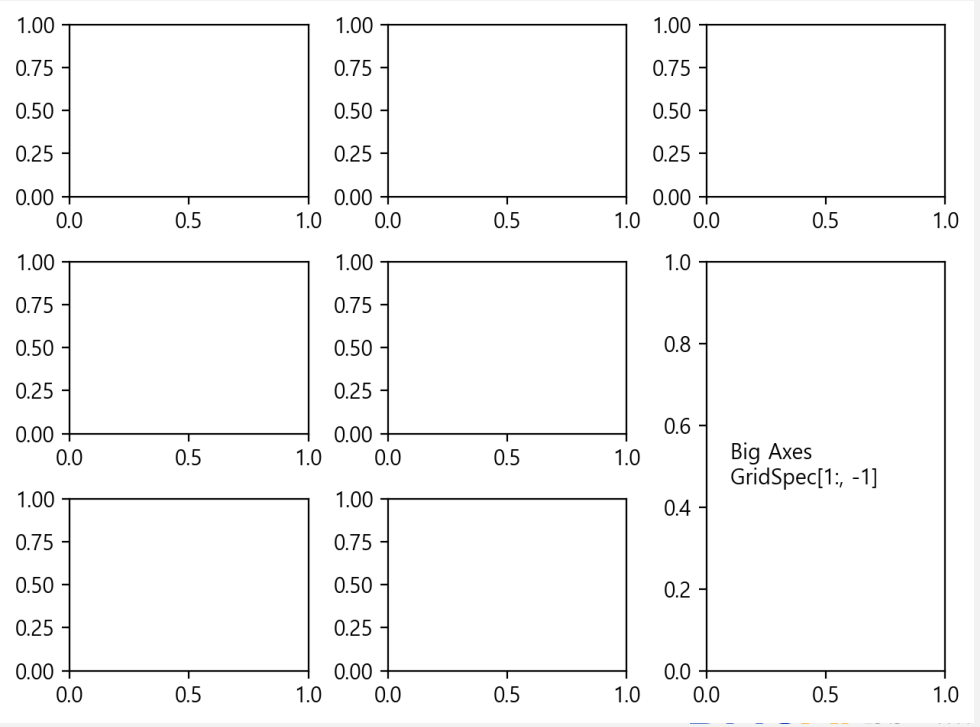
- (2행, 3열)과 (3행, 3열)의 2개 서브플롯을 제거

```
f7_axs[1, -1].remove()
f7_axs[2, -1].remove()
```



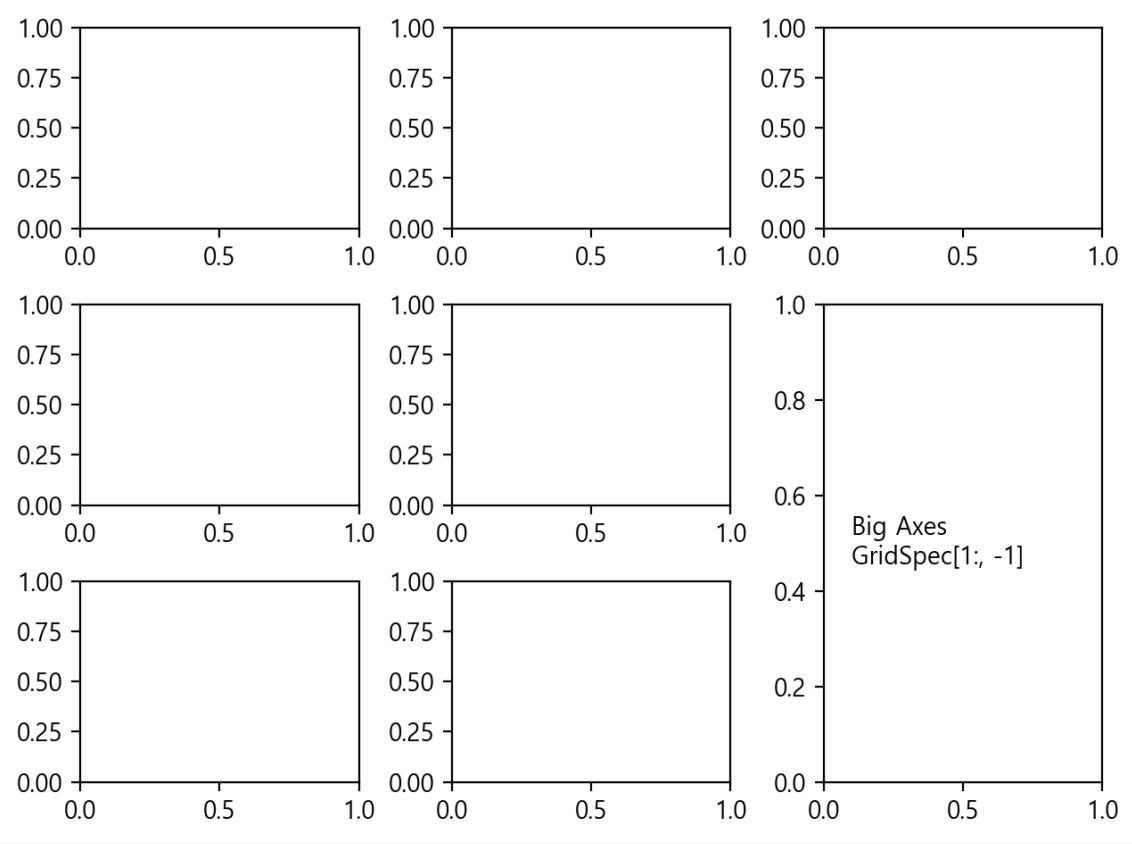
- 두 서브플롯 영역에 하나의 서브플롯을 추가

```
axbig = fig7.add_subplot(gs[1:, -1])
```



그리드 서브플롯의 추가와 삭제 결과

```python
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig7, f7_axs = plt.subplots(ncols=3, nrows=3)
gs = f7_axs[1, 2].get_gridspec()

# remove the underlying axes
# for ax in f7_axs[1:, -1]:
#     ax.remove()
f7_axs[1, -1].remove()
f7_axs[2, -1].remove()

axbig = fig7.add_subplot(gs[1:, -1])

axbig.annotate('Big Axes \nGridSpec[1:, -1]', (0.1, 0.5),
               xycoords='axes fraction', va='center')

fig7.tight_layout()
```



서브플롯에서 축 레이블, 제목, 눈금 등 요소들이 서로 겹치지 않도록

서브플롯이 figure 내에서 겹치지 않도록 자동으로 여백을 조정

- **fig.tight_layout()**
  - 서브플롯 간의 간격을 자동으로 계산
    - 축 제목, 축 레이블, 눈금 레이블 등이 겹치지 않도록 서브플롯 간의 여백을 최적화
  - 특히 여러 개의 서브플롯이 있는 경우 유용
    - 각 서브플롯의 크기와 간격이 자동으로 조정
    - figure 내의 서브플롯들이 잘 배치되도록 도와 줌

- **메소드 인자 constrained_layout=True**
  - 서브플롯 간의 간격을 자동으로 최적화
    - 서브플롯 간의 여백이나 제목, 축 레이블 등이 자동으로 최적화되어 겹치지 않게 배치
  - 사용 방법1
    - `fig = plt.figure(figsize=(4, 2), constrained_layout=True)`
  - 사용 방법2
    - `fig7, f7_axs = plt.subplots(ncols=3, nrows=3, constrained_layout=True)`

GridSpec의 세부 조정

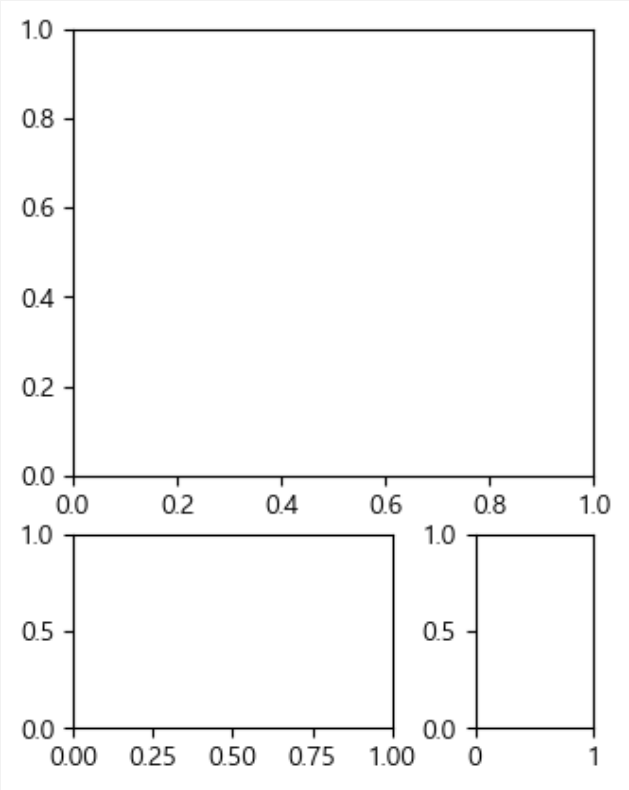
- GridSpec을 생성하면서 하위 플롯의 레이아웃 매개변수를 조정
  - figure.tight_layout이나 constrained_layout으로 붉가능
- left는 전체 그림의 왼쪽 위치를, right는 오른쪽 위치를 비율로 표시
  - left=0.05
    - 전체 그리드에서 왼쪽 가장자리와 서브플롯 사이의 여백을 “figure의 가로 길이의 5%”로 설정
  - right=0.48
    - 전체 그리드에서 전체 figure의 오른쪽 48% 위치까지 그리드가 차지
  - hspace=0.3
    - 세로 여백, 그리드 높이의 30%
  - wspace=0.7
    - 가로 여백, 그리드 너비의 70%

```
gs1 = fig8.add_gridspec(nrows=3, ncols=3, left=0.05, right=0.48,
                        hspace=0.3, wspace=0.7)
```

GridSpec의 세부 조정

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig8 = plt.figure(constrained_layout=False)
gs1 = fig8.add_gridspec(nrows=3, ncols=3, left=0.05, right=0.48,
hspace=0.3, wspace=0.7)

f8_ax1 = fig8.add_subplot(gs1[:-1, :])
f8_ax2 = fig8.add_subplot(gs1[-1, :-1])
f8_ax3 = fig8.add_subplot(gs1[-1, -1])
```

- **fig8.add_gridspec()으로 전체 그림을 조정**
  - 다시 fig8.add_subplot()으로 3개의 서브플롯을 그리는 코드와 결과
- **subplots_adjust()와 유사하지만 지정된 GridSpec에서 생성된 하위 플롯에만 영향을 미침**



48%

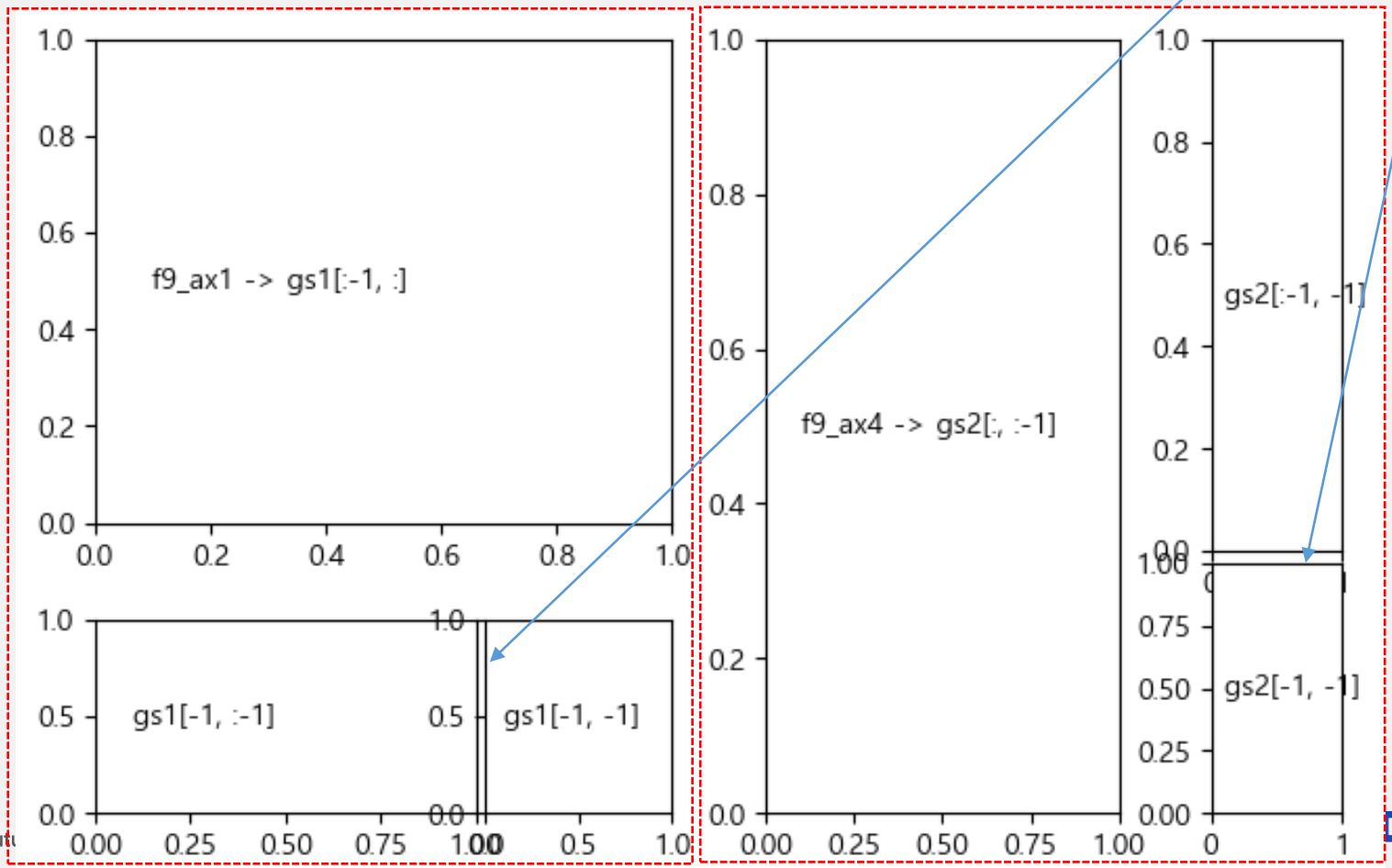
52%

결과 그림의 왼쪽과 오른쪽을 비교

왼쪽은 위와 비슷하며 오른쪽은 3행 3열에서 서브플롯을 3개를 그린 결과

- 그림의 이해를 돕기 위해 annotate()를 사용해 서브플롯을 만든 내용을 표시

```
gs1 = fig9.add_gridspec(nrows=3, ncols=3, left=0.05, right=0.48, wspace=0.05)
…
gs2 = fig9.add_gridspec(nrows=3, ncols=3, left=0.55, right=0.98, hspace=0.05)
```

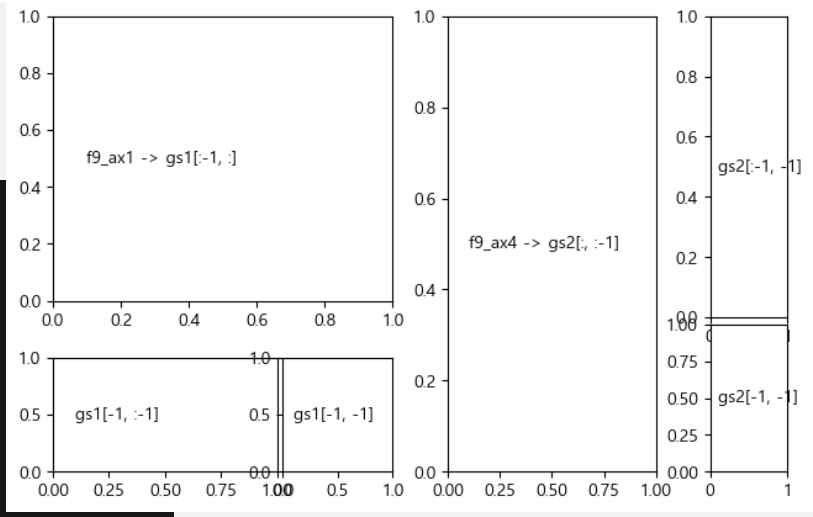


결과 그림의 왼쪽과 오른쪽을 비교

왼쪽은 위와 비슷하며 오른쪽은 3행 3열에서 서브플롯을 3개를 그린 결과

- 그림의 이해를 돕기 위해 annotate()를 사용해 서브플롯을 만든 내용을 표시

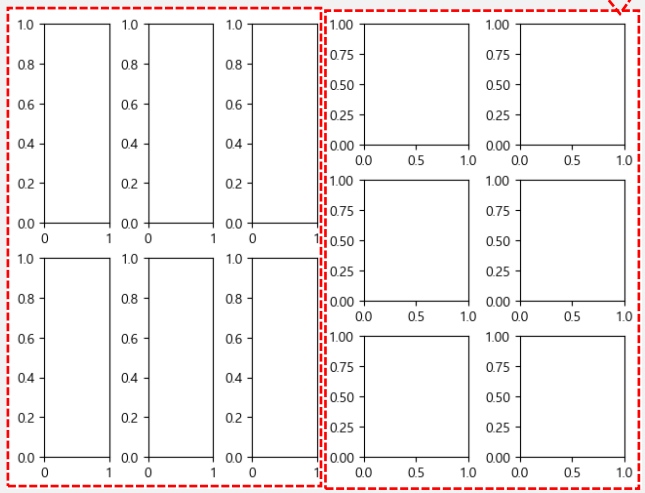
```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig9 = plt.figure(constrained_layout=False)

gs1 = fig9.add_gridspec(nrows=3, ncols=3, left=0.05, right=0.48,
                        wspace=0.05)

f9_ax1 = fig9.add_subplot(gs1[:-1, :]) # 1행 2행, 모든열에 걸친 서브플롯
f9_ax1.annotate('f9_ax1 -> gs1[:-1, :]', (0.1, 0.5), xycoords='axes fraction', va='center')
f9_ax2 = fig9.add_subplot(gs1[-1, :-1]) # 3행에 1~2열에 걸친 서브플롯
f9_ax2.annotate('gs1[-1, :-1]', (0.1, 0.5), xycoords='axes fraction', va='center')
f9_ax3 = fig9.add_subplot(gs1[-1, -1]) # 3행 3열 서브플롯
f9_ax3.annotate('gs1[-1, -1]', (0.1, 0.5), xycoords='axes fraction', va='center')

gs2 = fig9.add_gridspec(nrows=3, ncols=3, left=0.55, right=0.98,
hspace=0.05)

f9_ax4 = fig9.add_subplot(gs2[:, :-1]) # 모든행, 1~2열에 걸친 서브플롯
f9_ax4.annotate('f9_ax4 -> gs2[:, :-1]', (0.1, 0.5), xycoords='axes fraction', va='center')
f9_ax5 = fig9.add_subplot(gs2[:-1, -1]) # 1~2행과 3열에 걸친 서브플롯
f9_ax5.annotate('gs2[:-1, -1]', (0.1, 0.5), xycoords='axes fraction', va='center')
f9_ax6 = fig9.add_subplot(gs2[-1, -1]) # 3행 3열 서브플롯
f9_ax6.annotate('gs2[-1, -1]', (0.1, 0.5), xycoords='axes fraction', va='center')

plt.subplots_adjust(hspace=.5, wspace=.7)
```



SubplotSpec을 사용하는 GridSpec

클래스 SubplotSpec의 메소드 subgridspec(m, n) 사용

- 클래스 SubplotSpec()의 메소드 subgridspec()으로 세부 GridSpec을 생성
  - subgridspec(m, n)의 인자로 설정



- 먼저 1행 2열의 그리드를 배치

```
gs0 = fig10.add_gridspec(1, 2) # 1행 2열
```

  - 1열 객체인 gs0[0]
    - 메소드 subgridspec(2, 3)의 호출로 1열 내부에 2행 3열의 서브플롯 생성
  - 2열 객체인 gs0[1]
    - 메소드 subgridspec(3, 2)의 호출로 2열 내부에 3행 2열의 서브플롯 생성

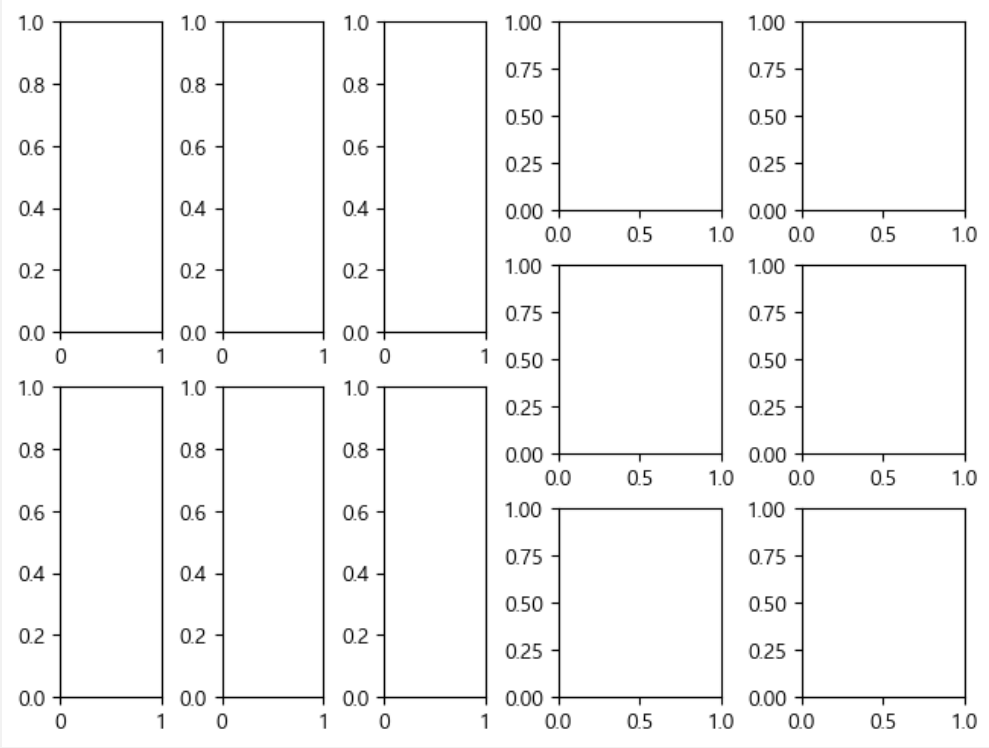
```
gs00 = gs0[0].subgridspec(2, 3) # 1열 내부에 2행 3열의 서브플롯
gs01 = gs0[1].subgridspec(3, 2) # 2열 내부에 3행 2열의 서브플롯
```

클래스 class matplotlib.gridspec.SubplotSpec

SubplotSpec을 사용하는 GridSpec 결과

- 1열 내부에 2행 3열의 서브플롯, 2열 내부에 3행 2열의 서브플롯을 그리는 코드와 결과

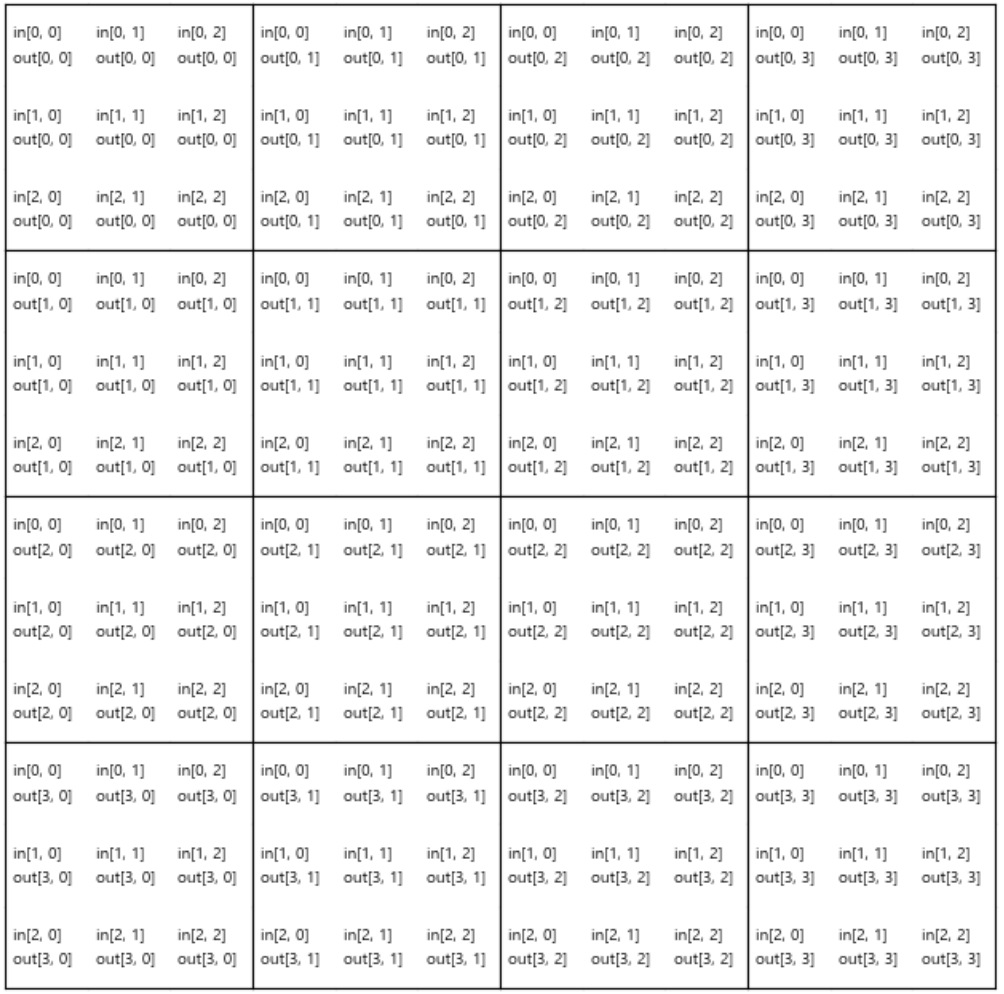
```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig10 = plt.figure(constrained_layout=True)
gs0 = fig10.add_gridspec(1, 2) # 1행 2열

gs00 = gs0[0].subgridspec(2, 3) # 1열 내부에 2행 3열의 서브플롯
gs01 = gs0[1].subgridspec(3, 2) # 2열 내부에 3행 2열의 서브플롯

for a in range(2):
    for b in range(3):
        fig10.add_subplot(gs00[a, b])
        fig10.add_subplot(gs01[b, a])
```



SubplotSpec을 사용해 중첩된 그리드 배치

- 외부 4 x 4 그리드의 각각의 내부
  - 다시 내부 3 x 3 그리드를 표현한 그림
  - 각 그리드에는 외부와 내부의 서브플롯을 참조할 수 있는 첨자를 표시
  - 외부 4 x 4 그리드를 만든 코드

```
outer_grid = fig11.add_gridspec(4, 4,
                                wspace=0, hspace=0)
```



- 16개의 outer_grid[a, b]에서 subgridspec(3, 3, ...)으로 내부에 다시 3 x 3의 그리드 생성 코드

```
inner_grid = outer_grid[a, b].subgridspec(3, 3, wspace=0, hspace=0)
axs = inner_grid.subplots()  # Create all subplots for the inner grid.
```

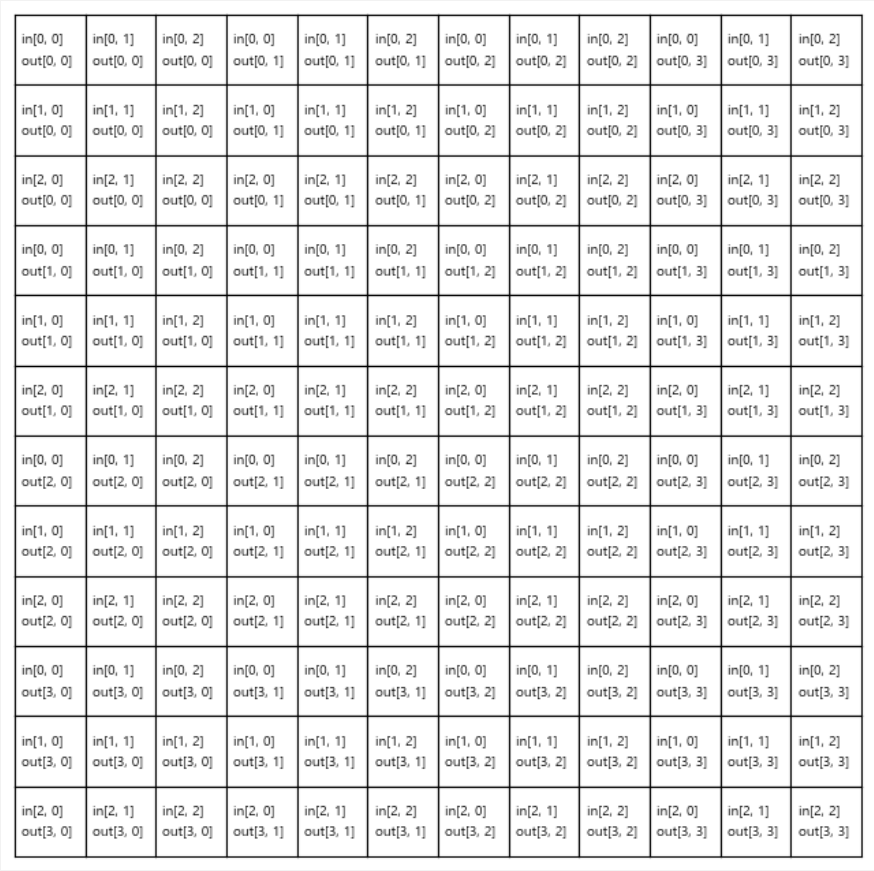
SubplotSpec을 사용해 중첩된 그리드 배치

- **numpy.ndenumerate(arr)**
  - 배열 첨자 좌표와 배열 값의 쌍을 만드는 반복자(iterator)를 반환
  - for 반복에서 두 개의 변수 index, x로 받아 출력하면 내용을 이해

```
>>> a = np.array([[1, 2], [3, 4]])
>>> for index, x in np.ndenumerate(a):
...         print(index, x)
...
(0, 0) 1
(0, 1) 2
(1, 0) 3
(1, 1) 4
```

SubplotSpec을 사용해 중첩된 그리드 배치 결과

```python
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

fig11 = plt.figure(figsize=(8, 8), constrained_layout=False)
outer_grid = fig11.add_gridspec(4, 4, wspace=0, hspace=0)

for a in range(4):
    for b in range(4):
        # gridspec inside gridspec
        inner_grid = outer_grid[a, b].subgridspec(3, 3, wspace=0, hspace=0)
        axs = inner_grid.subplots() # Create all subplots for the inner grid.
        for(c, d), ax in np.ndenumerate(axs):
            s1, s2 = f'out[{a}, {b}]', f'in[{c}, {d}]'
            ax.text(0.1, 0.3, s1, fontsize=7)
            ax.text(0.1, 0.6, s2, fontsize=7)
            ax.set(xticks=[], yticks=[]) # 눈금 없애기

plt.show()
```

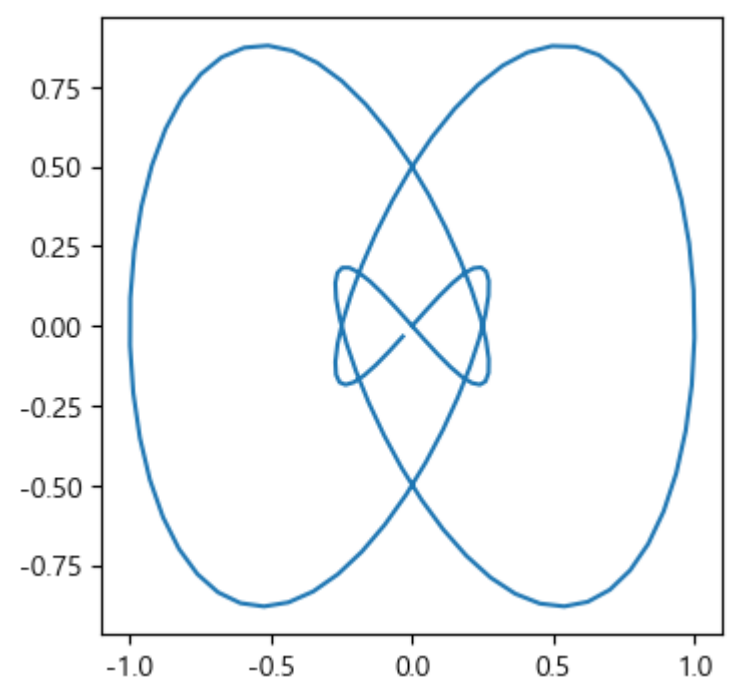


기학적인 그림 그리기 함수

sin()과 cos() 함수를 사용

- **함수 squiggle_xy(1, 2, 1, 3)으로 호출**
  - x, y의 좌표를 얻어 plot()으로 그린 기하학적인 그림
  - 네 개의 인자를 정수로 적정하게 바꾸면 다양한 기하학적인 문양이 생성

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

def squiggle_xy(a, b, c, d, i=np.arange(0.0, 2*np.pi, 0.05)):
    return np.sin(i*a)*np.cos(i*b), np.sin(i*c)*np.cos(i*d)

fig = plt.figure(figsize=(4, 4))

plt.plot(*squiggle_xy(1, 2, 1, 3))
plt.show()
```



전체 12 X 12의 그리드 기초

- 외부 4x4 그리드의 각각의 내부에 다시 내부 3x3 그리드를 표현한 그림

```
outer_grid = fig11.add_gridspec(4, 4, wspace=0, hspace=0)
```

- 16개의 outer_grid[a, b]에서 subgridspec(3, 3, ...)으로 내부에 다시 3x3의 그리드를 생성

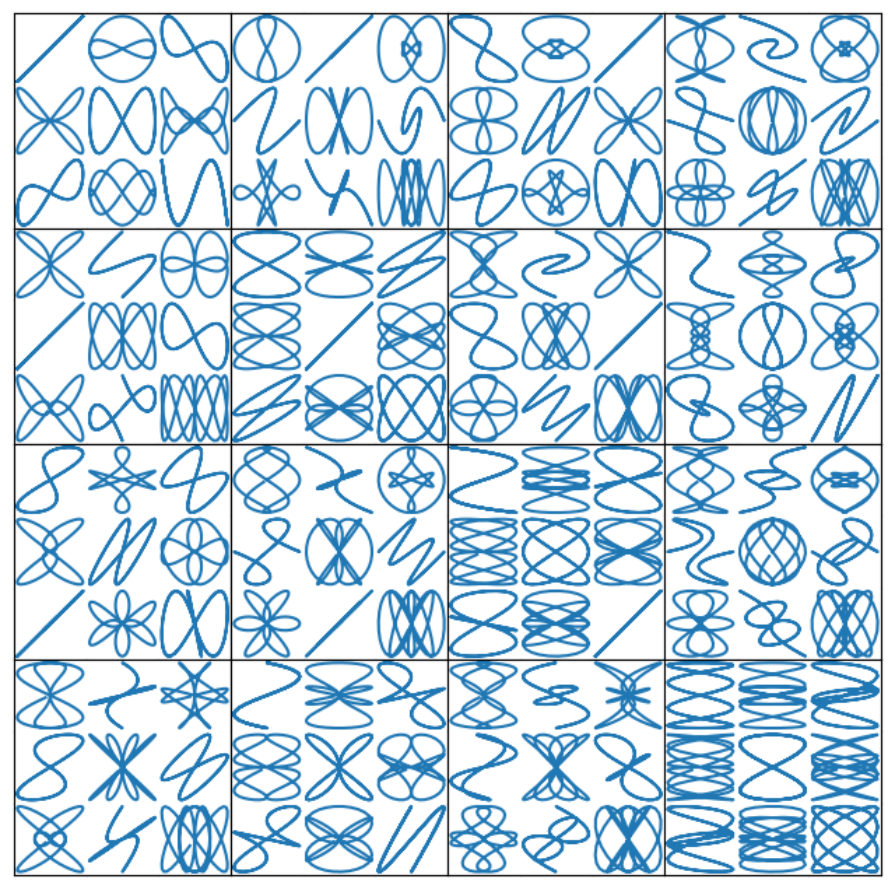
```
for a in range(4):
    for b in range(4):
        # gridspec inside gridspec
        inner_grid = outer_grid[a, b].subgridspec(3, 3, wspace=0, hspace=0)
        axs = inner_grid.subplots() # Create all subplots for the inner grid.
```

- 16개의 outer_grid[a, b]를 순회하면서 각각 3x3의 모든 서브플롯에서 기하학적인 그림을 그리고 모든 눈금(ticks)은 없도록 지정

```
for a in range(4):
    for b in range(4):
        # ...
        for(c, d), ax in np.ndenumerate(axs):
            ax.plot(*squiggle_xy(a + 1, b + 1, c + 1, d + 1))
            ax.set(xticks=[], yticks=[])
```

전체 12 X 12의 기하학 그림 그리기

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

def squiggle_xy(a, b, c, d, i=np.arange(0.0, 2*np.pi, 0.05)):
    return np.sin(i*a)*np.cos(i*b), np.sin(i*c)*np.cos(i*d)

fig11 = plt.figure(figsize=(8, 8), constrained_layout=False)
outer_grid = fig11.add_gridspec(4, 4, wspace=0, hspace=0)

for a in range(4):
    for b in range(4):
        # gridspec inside gridspec
        inner_grid = outer_grid[a, b].subgridspec(3, 3, wspace=0, hspace=0)
        axs = inner_grid.subplots() # Create all subplots for the inner grid.
        for(c, d), ax in np.ndenumerate(axs):
            ax.plot(*squiggle_xy(a + 1, b + 1, c + 1, d + 1))
            ax.set(xticks=[], yticks=[])

# show only the outside spines
for ax in fig11.get_axes():
    ss = ax.get_subplotspec()
    ax.spines.top.set_visible(ss.is_first_row())
    ax.spines.bottom.set_visible(ss.is_last_row())
    ax.spines.left.set_visible(ss.is_first_col())
    ax.spines.right.set_visible(ss.is_last_col())

plt.show()
```



내부 외곽선을 보이지 않게 처리하는 부분이 없는 경우

- 마지막 for 문장 제거

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec

def squiggle_xy(a, b, c, d, i=np.arange(0.0, 2*np.pi, 0.05)):
    return np.sin(i*a)*np.cos(i*b), np.sin(i*c)*np.cos(i*d)

fig11 = plt.figure(figsize=(8, 8), constrained_layout=False)
outer_grid = fig11.add_gridspec(4, 4, wspace=0, hspace=0)

for a in range(4):
    for b in range(4):
        # gridspec inside gridspec
        inner_grid = outer_grid[a, b].subgridspec(3, 3, wspace=0, hspace=0)
        axs = inner_grid.subplots() # Create all subplots for the inner grid.
        for(c, d), ax in np.ndenumerate(axs):
            ax.plot(*squiggle_xy(a + 1, b + 1, c + 1, d + 1))
            ax.set(xticks=[], yticks=[])

plt.show()
```