

CHAPTER 05

데이터 시각화

01 맷플롯립

02 시분

03 플롯리



학습목표

- 맷플롯립의 구조에 대해 알아보고,
맷플롯립으로 그래프를 꾸미는 방법을 실습한다.
- 맷플롯립에서 사용하는 그래프 타입에 대해 살펴본다.
- 시본을 통해 그래프를 작성하는 방법을 실습한다.
- 플롯리를 통해 그래프를 작성하는 방법을 실습한다.

01

맷플롯립

- 데이터 시각화(data visualization) : 데이터 분석 결과를 쉽게 이해할 수 있도록 시각적으로 표현하고 전달
- 맷플롯립, 시본, 플롯리 외에도 bokeh, pygal, ggplot 등

1. 맷플롯립의 구조

- 맷플롯립(matplotlib) : 매트랩(matlab) 기능을 파이썬에서 그대로 사용하도록 하는 시각화 모듈
 - 엑셀의 정형화된 차트나 그래프 작성, 다양한 함수 지원
 - 매트랩을 포장(wrapping)해서 맷플롯립을 지원

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

1.1 파이플롯

- 맷플롯립을 이용할 때 가장 기본이 되는 객체
- 파이플롯(pyplot) 위에 그림.figure) 객체를 올리고 그 위에 그래프에 해당하는 축(axes)을 올림
- 그림 위에 축을 여러 장 올리면 여러 개의 그래프 작성

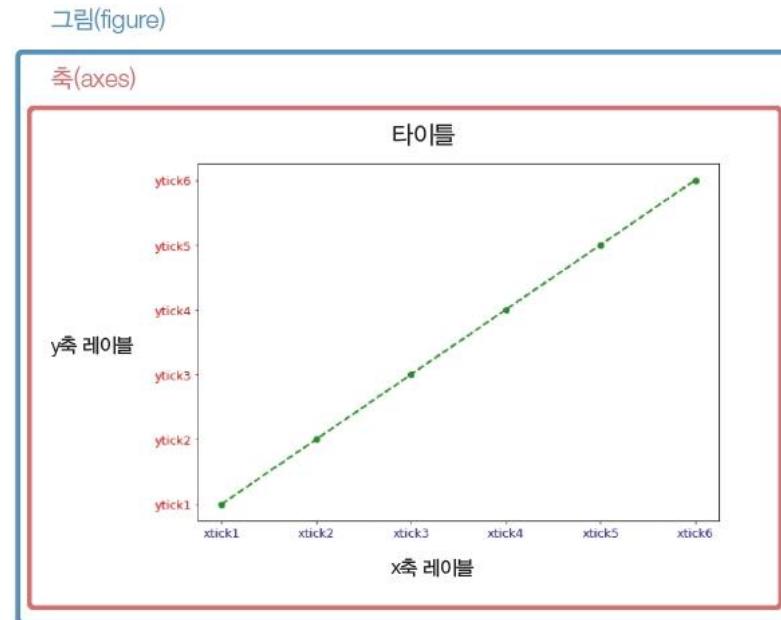
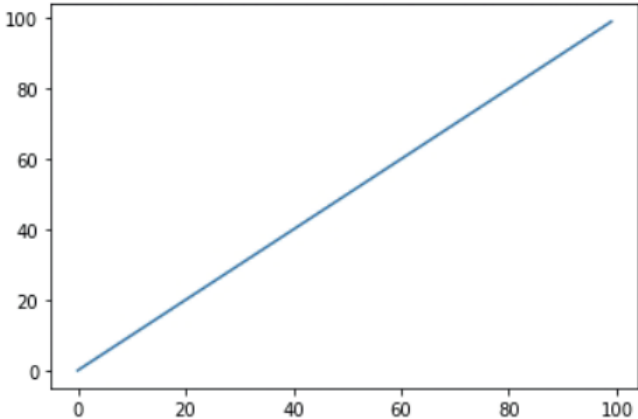
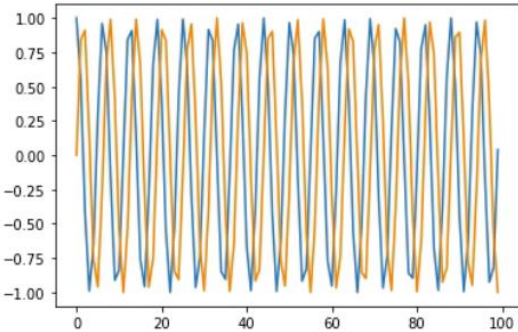


그림 5-1 파이플롯(pyplot), 그림.figure), 축(axes)의 개념

In [1]:	<pre>import matplotlib.pyplot as plt # matplotlib 모듈 호출 X = range(100) Y = range(100) plt.plot(X, Y)</pre>
Out [1]:	

- X 객체와 Y 객체 값 쌍으로 좌표평면 위에 점을 찍음
- plot 함수로 점들을 연결

In [2]:	<pre>import numpy as np # numpy 모듈 호출 X_1 = range(100) Y_1 = [np.cos(value) for value in X] X_2 = range(100) Y_2 = [np.sin(value) for value in X] plt.plot(X_1, Y_1) plt.plot(X_2, Y_2) plt.show()</pre>
Out [2]:	

- pyplot 객체 내부에 있는 하나의 그림 객체 위에 코사인 그래프와 사인 그래프를 그림

1.2 그림과 축

- 그림은 그래프를 작성하는 밑바탕이 됨
- 축은 실제로 그래프를 작성하는 공간

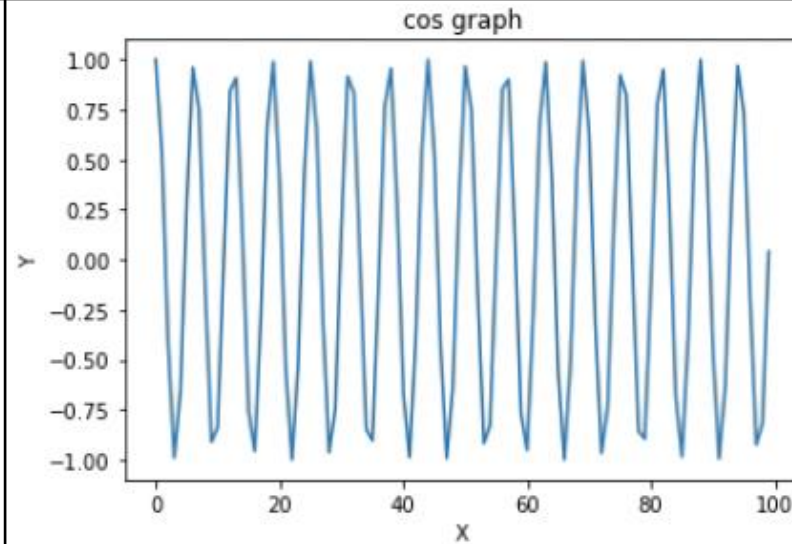
```
In [3]: fig, ax = plt.subplots() # (1) figure와 axes 객체 할당

X_1 = range(100)
Y_1 = [np.cos(value)
        for value in X]

ax.plot(X_1, Y_1) # (2) plot 함수를 사용하여 그래프 생성
ax.set(title='cos graph', # (3) 그래프 제목, X축 라벨, Y축 라벨 설정
        xlabel='X',
        ylabel='Y');

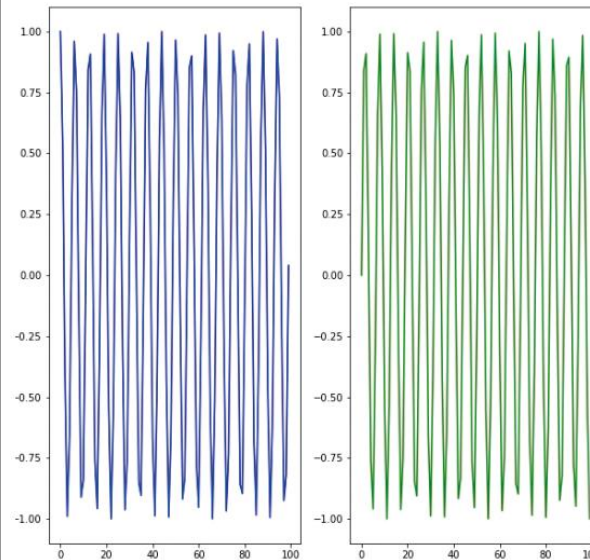
plt.show() # (4) 그래프 출력
```


Out [3]:



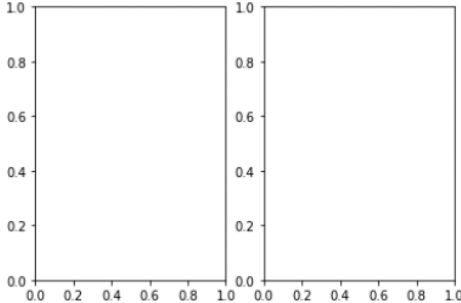
```
In [4]: fig = plt.figure() # (1) figure 반환  
fig.set_size_inches(10,10) # (2) figure의 크기 지정  
  
ax_1 = fig.add_subplot(1,2,1) # (3) 첫 번째 그래프 생성  
ax_2 = fig.add_subplot(1,2,2) # (4) 두 번째 그래프 생성  
  
ax_1.plot(X_1, Y_1, c="b") # (5) 첫 번째 그래프 설정  
ax_2.plot(X_2, Y_2, c="g") # (6) 두 번째 그래프 설정  
plt.show() # (7) 그래프 출력
```

Out [4]:



1.3 서브플롯 행렬

- 축을 여러 개 만들 때 서브플롯으로 축 객체 공간 확보
 - 그림 객체에서 `add_subplot` 함수 사용
 - 또는 `plot` 객체에서 `subplots` 함수 사용

In [5]:	<code>fig, ax = plt.subplots(nrows=1, ncols=2)</code> <code>print(ax)</code>
Out [5]:	
In [6]:	<code>print(type(ax))</code>
Out [6]:	<code><class 'numpy.ndarray'></code>

- `ax` 변수에 축 객체가 넘파이 배열 타입으로 생성됨

```
In [7]: import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,100) # (1) x 값과 y_n 값 생성
y_1 = np.sin(x)
y_2 = np.cos(x)
y_3 = np.tan(x)
y_4 = np.exp(x)

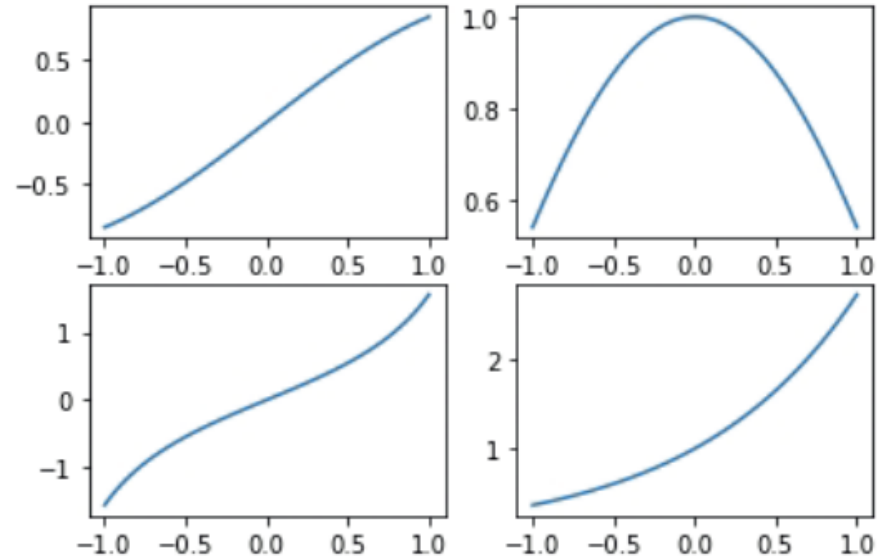
fig, ax = plt.subplots(2, 2) # (2) 2x2 figure 객체를 생성

ax[0, 0].plot(x, y_1) # (3) 첫 번째 그래프 생성
ax[0, 1].plot(x, y_2) # (4) 두 번째 그래프 생성
ax[1, 0].plot(x, y_3) # (5) 세 번째 그래프 생성
ax[1, 1].plot(x, y_4) # (6) 네 번째 그래프 생성

plt.show()
```

- # (2) subplots 함수에서 2x2 행렬 그림 객체가 생성되어
ax 변수에 4개의 축 객체가 2x2 넘파이 배열 형태로 들어가 있음
- 넘파이 배열의 인덱스로 각 축 객체에 접근하여 그래프를 생성

Out [7]:



- 행과 열을 지정하고, 세 번째 숫자는 축의 위치

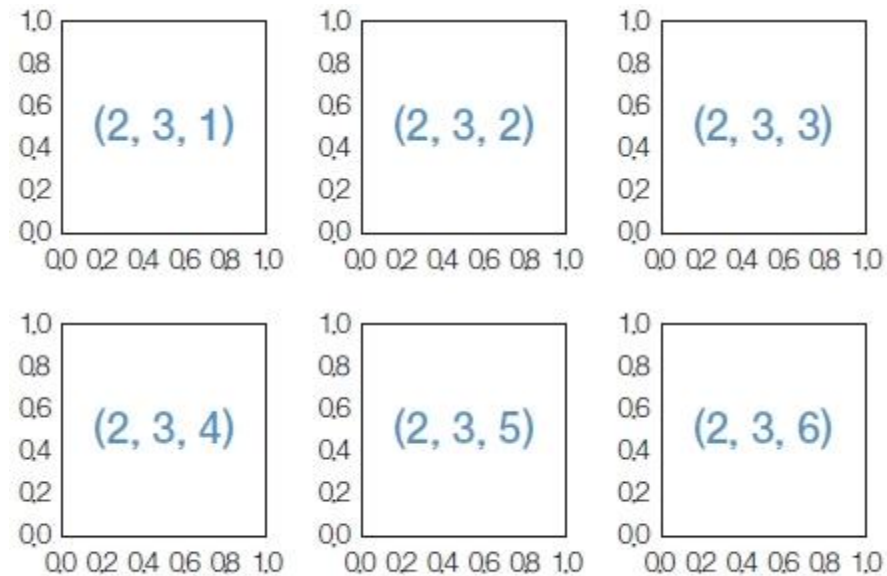
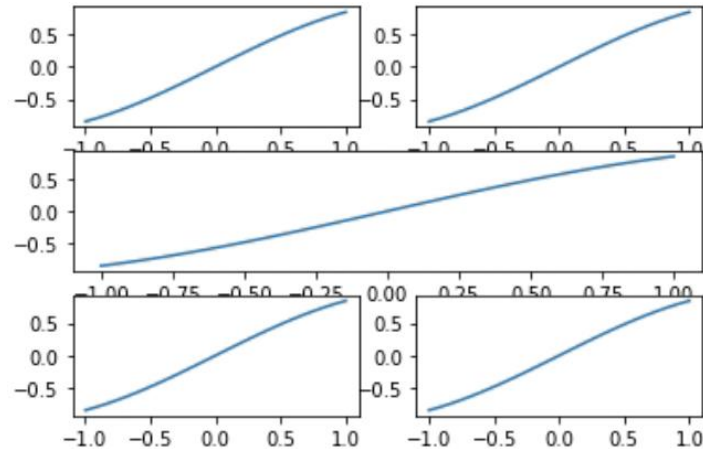


그림 5-2 지정된 행렬에서 축(axes)의 위치 배열

```
In [8]: ax1 = plt.subplot(321) # (1) 첫 번째 공간에 axes 생성  
plt.plot(x, y_1)  
ax2 = plt.subplot(322) # (2) 두 번째 공간에 axes 생성  
plt.plot(x, y_1)  
ax3 = plt.subplot(312) # (3) 두 번째 공간에 axes 생성  
plt.plot(x, y_1)  
ax4 = plt.subplot(325) # (4) 다섯 번째 공간에 axes 생성  
plt.plot(x, y_1)  
ax5 = plt.subplot(326) # (5) 여섯 번째 공간에 axes 생성  
plt.plot(x, y_1)  
  
plt.show()import num
```

Out [8]:



2. 맷플롯립으로 그래프 꾸미기

2.1 색상

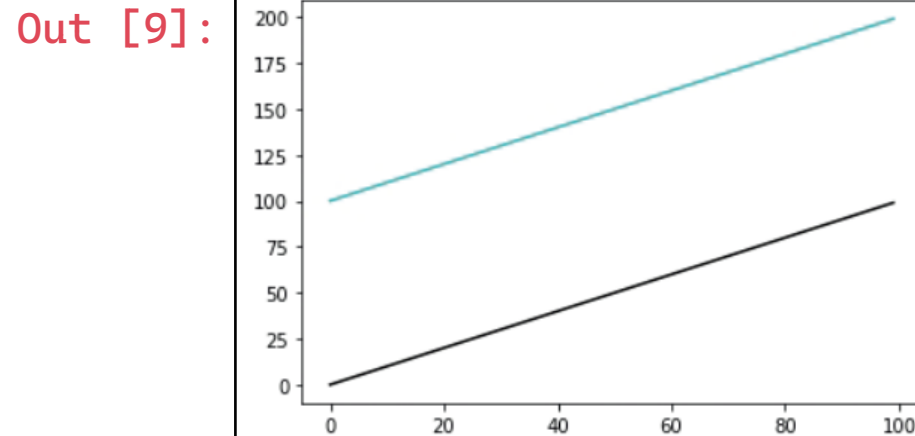
- color 또는 c 매개변수로 색상 변경
 - RGB 값을 사용해서 #을 붙여 16진법으로 색상 표현
 - 또는 b, g, r, c, m, y, k, w 등 약어 입력


```
In [9]: X_1 = range(100)
        Y_1 = [value for value in X]

        X_2 = range(100)
        Y_2 = [value + 100 for value in X]

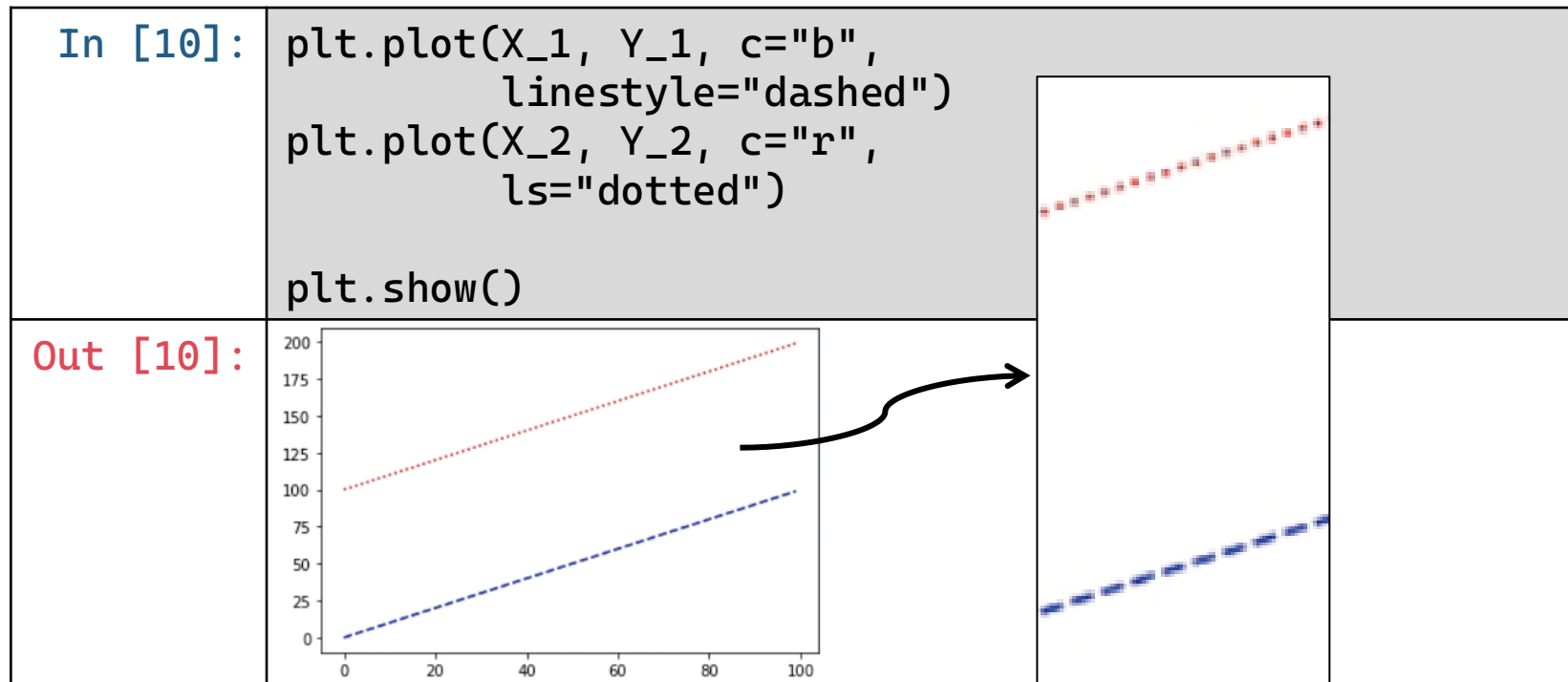
        plt.plot(X_1, Y_1, color="#000000")
        plt.plot(X_2, Y_2, c="c")

        plt.show()
```



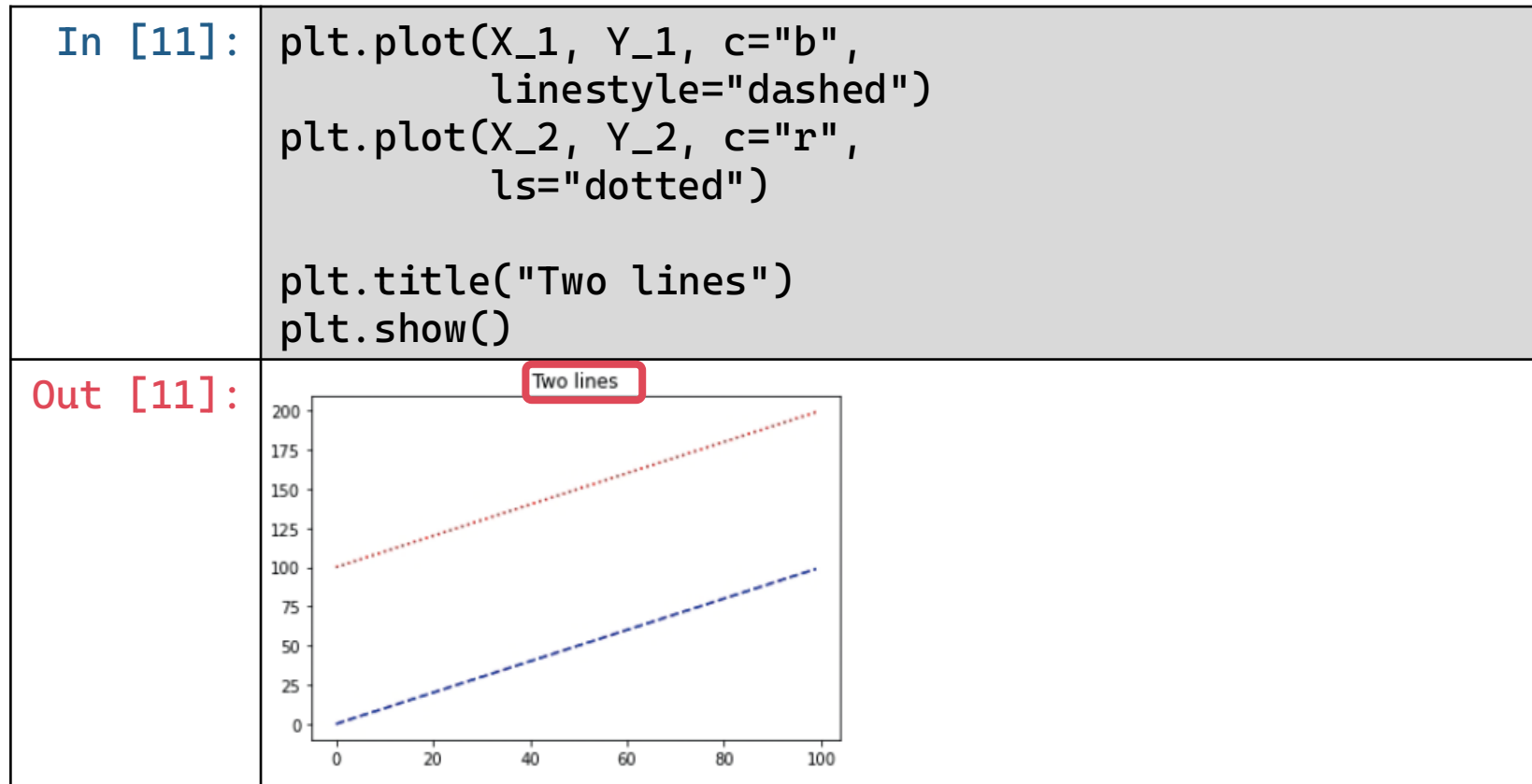
2.2 선의 형태

- linestyle 또는 ls로 선의 형태를 정의
 - dashed : 점선 형태 solid : 실선 형태



2.3 제목

- 축 객체마다 제목을 달 수 있음



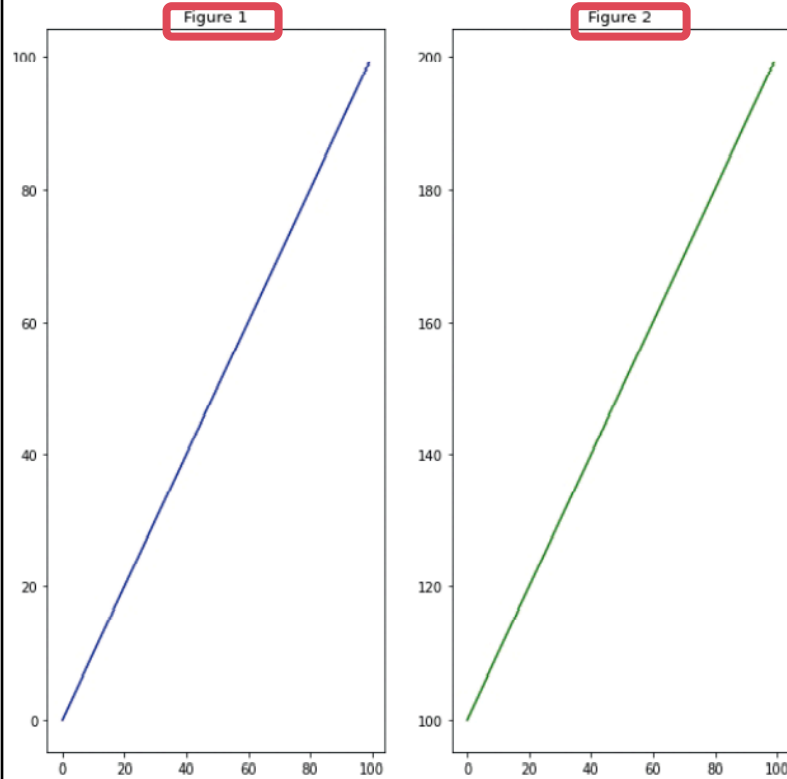
```
In [12]: fig = plt.figure()
fig.set_size_inches(10,10)

ax_1 = fig.add_subplot(1,2,1)
ax_2 = fig.add_subplot(1,2,2)

ax_1.plot(X_1, Y_1, c="b")
ax_1.set_title("Figure 1")
ax_2.plot(X_2, Y_2, c="g")
ax_2.set_title("Figure 2")

plt.show()
```

Out [12]:



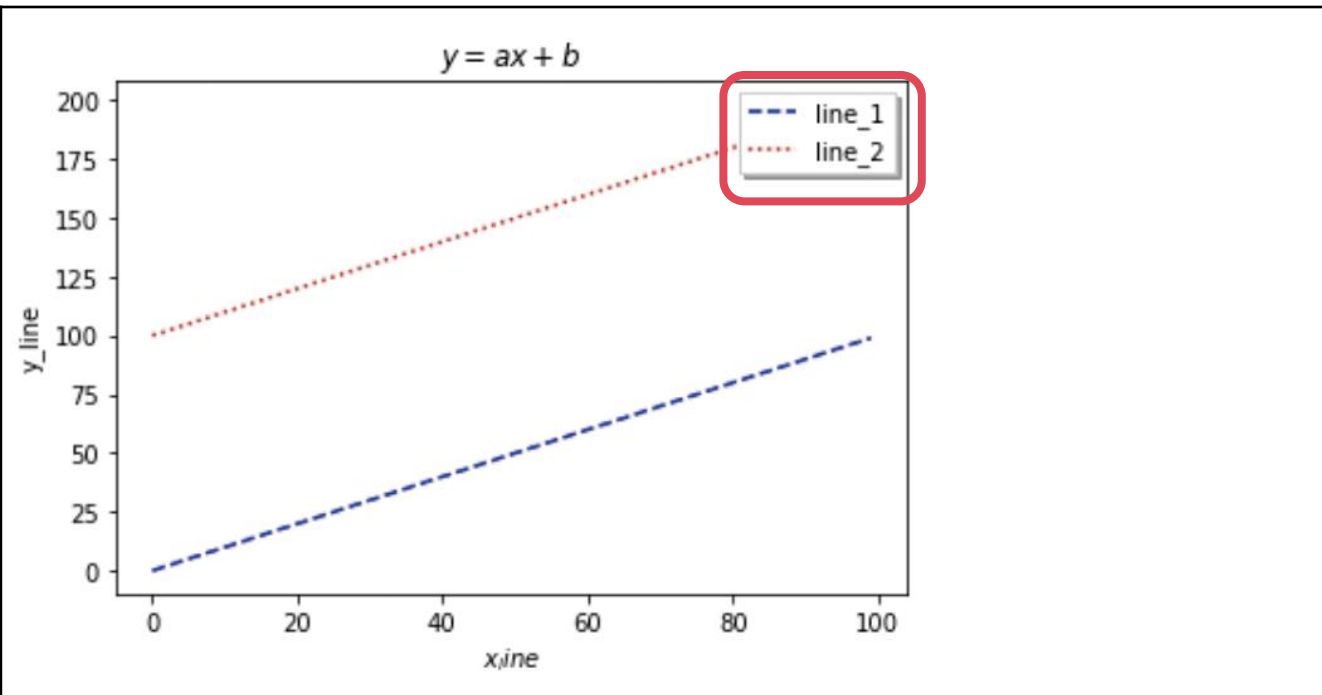
2.4 범례

- 축 객체마다 범례를 설정할 수 있음
- legend 함수 사용하여 생성
 - shadow 매개변수로 범례에 그림자 효과 추가
 - loc 매개변수로 범례의 위치 지정
 - 값은 center, upper right 등 총 11가지
 - best라고 지정하면 적절한 위치에 범례가 놓임

```
In [13]: plt.plot(X_1, Y_1,
                  color="b",
                  linestyle="dashed",
                  label='line_1')
plt.plot(X_2, Y_2,
          color="r",
          linestyle="dotted",
          label='line_2')
plt.legend(
    shadow=True,
    fancybox=False,
    loc="upper right")

plt.title('$y = ax+b$')
plt.xlabel('$x_{line}$')
plt.ylabel('y_line')
```

Out [13]:



3. 맷플롯립에서 사용하는 그래프

3.1 산점도

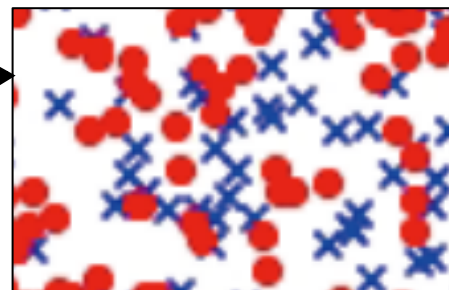
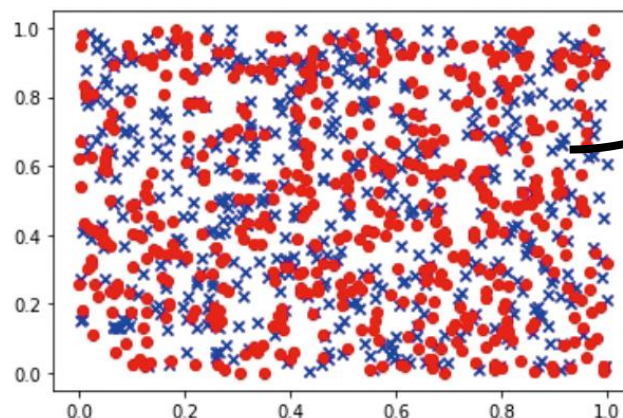
- 산점도(scatter plot) : 데이터 분포를 2차원 평면에 표현
 - 매개변수 `c`는 포인트 색상을 지정
 - `marker`는 포인트 모양을 지정
 - `size`는 포인트 크기를 지정
 - `alpha`는 포인트 불투명도를 지정

```
In [14]: data_1 = np.random.rand(512, 2)
data_2 = np.random.rand(512, 2)

plt.scatter(data_1[:,0],
            data_1[:,1],
            c="b", marker="x")
plt.scatter(data_2[:,0],
            data_2[:,1],
            c="r", marker="o")

plt.show()
```

Out [14]:

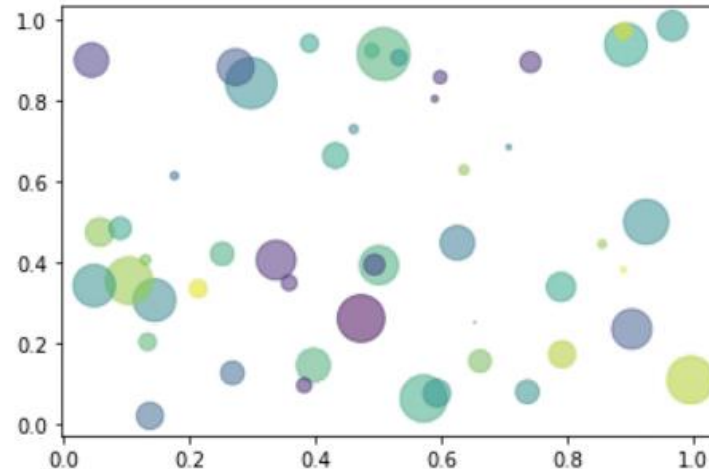


```
In [15]: N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi * (
    15 * np.random.rand(N))**2

plt.scatter(x, y,
            s=area, c=colors,
            alpha=0.5)

plt.show()
```

Out [15]:



3.2 막대그래프

- 막대그래프(bar graph) : 데이터의 개수나 크기를 비교

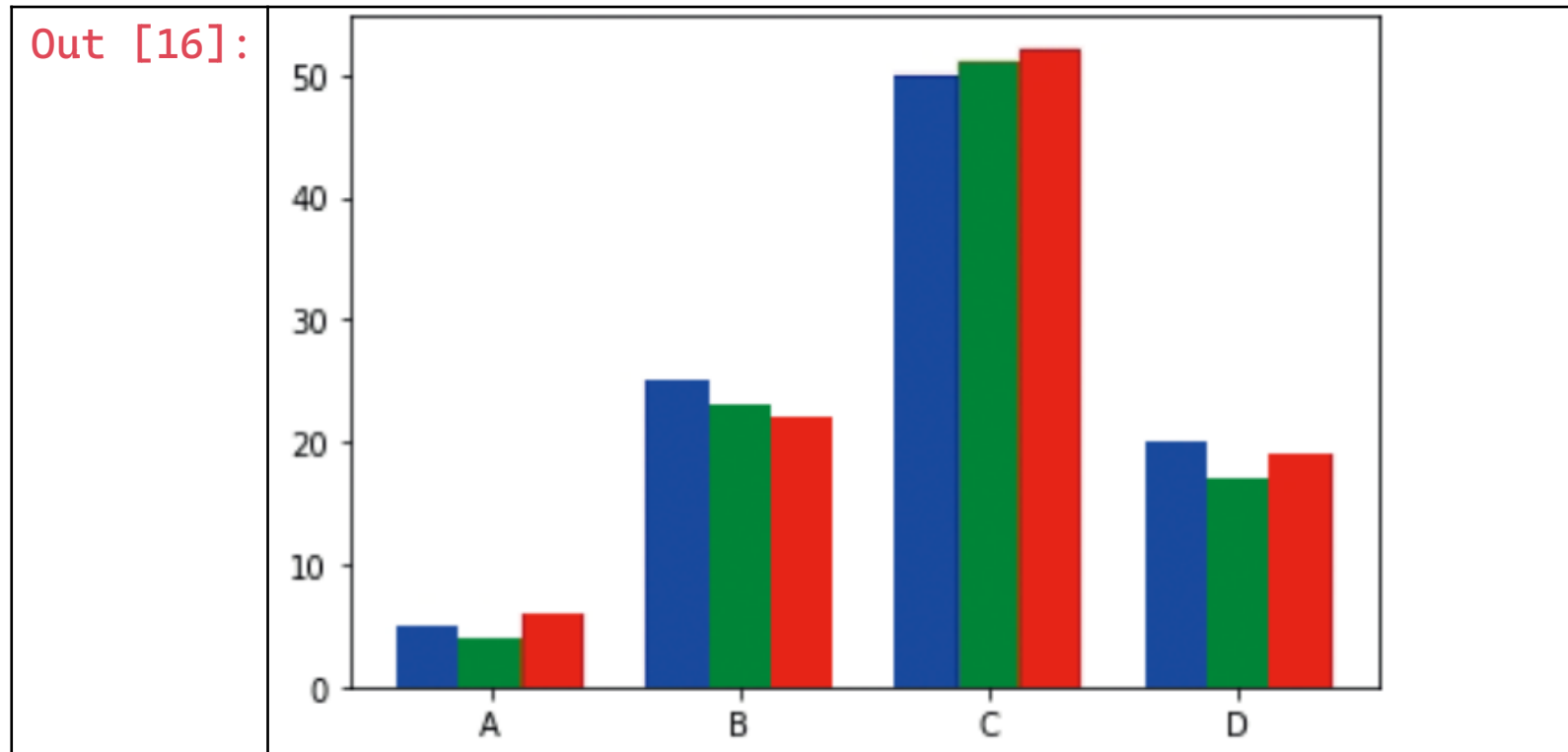
```
In [16]: # (1) 데이터 생성
data = [[5., 25., 50., 20.],
        [4., 23., 51., 17],
        [6., 22., 52., 19]]

# (2) X 좌표 시작점
X = np.arange(0,8,2)

# (3) 3개의 막대그래프 생성
plt.bar(X + 0.00, data[0], color = 'b', width = 0.50)
plt.bar(X + 0.50, data[1], color = 'g', width = 0.50)
plt.bar(X + 1.0, data[2], color = 'r', width = 0.50)

# (4) X축에 표시될 이름과 위치 설정
plt.xticks(X+0.50, ("A", "B", "C", "D"))

# (5) 막대그래프 출력
plt.show()
```



3.3 누적 막대그래프

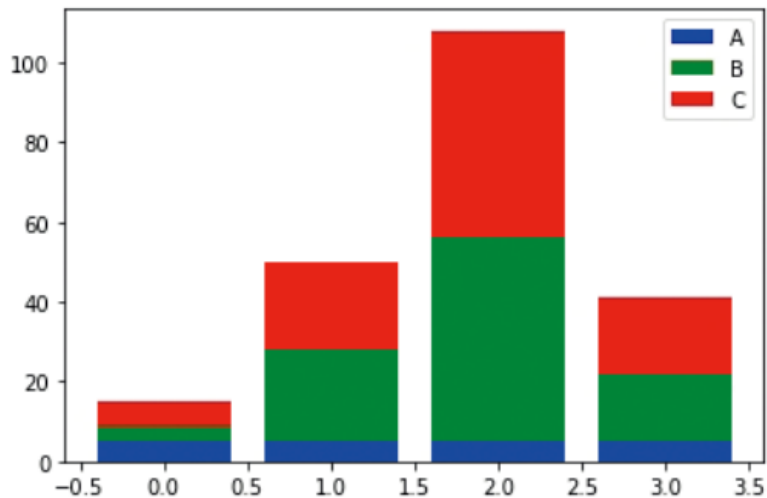
- 누적 막대그래프(stacked bar graph) :
데이터를 밑에서부터 쌓아올려 데이터를 표현

```
In [17]: data = np.array([[5., 25., 50., 20.],  
                        [4., 23., 51., 17],  
                        [6., 22., 52., 19]])  
  
color_list = ['b', 'g', 'r']  
data_label = ["A", "B", "C"]  
X = np.arange(data.shape[1])  
  
data = np.array([[5., 5., 5., 5.],  
                [4., 23., 51., 17],  
                [6., 22., 52., 19]])
```

```
for i in range(3):  
    plt.bar(X, data[i],  
            bottom = np.sum(  
                data[:i], axis=0),  
            color = color_list[i],  
            label=data_label[i])
```

```
plt.legend()  
plt.show()
```

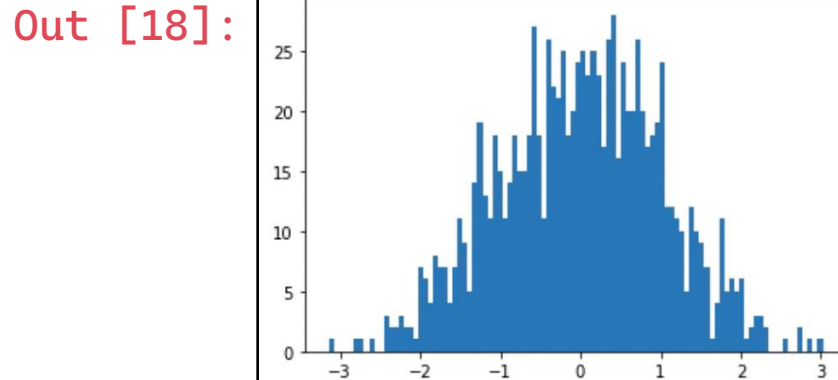
Out [17]:



3.4 히스토그램

- 히스토그램(histogram) : 데이터의 분포를 표현
 - hist 함수로 히스토그램 생성, 매개변수 bins로 막대 개수 지정

```
In [18]: N = 1000  
X = np.random.normal(size=N)  
  
plt.hist(X, bins=100)  
plt.show()
```



3.4 상자그림

- 상자그림(boxplot) : 사분위수를 시각화하여 데이터의 분포와 밀집 정도를 표현

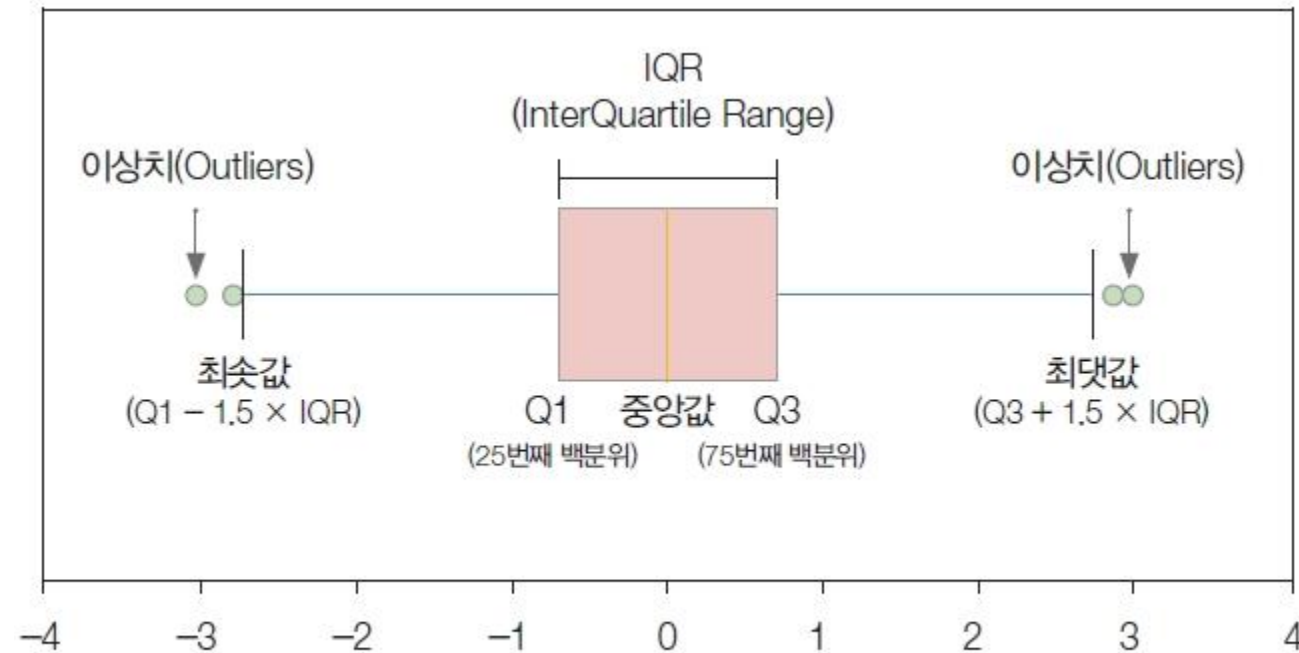
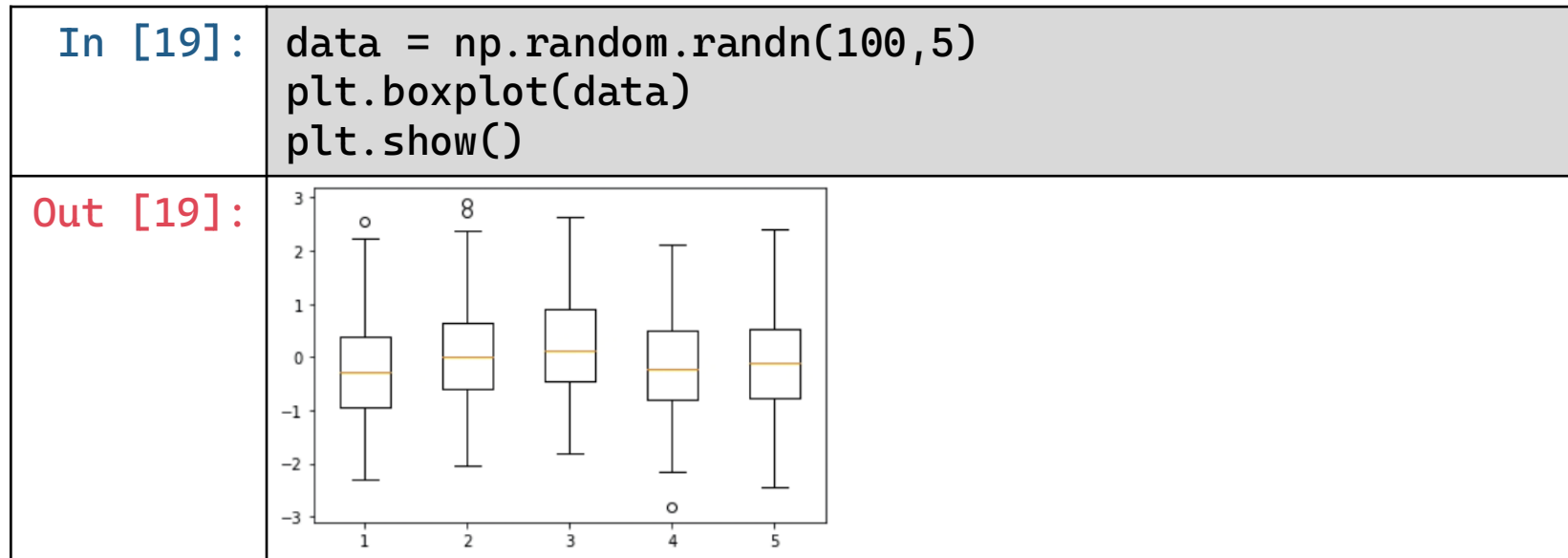


그림 5-3 상자그림의 구성 요소

- 데이터를 작은 데이터부터 큰 데이터까지 정렬
- Q1(25%)부터 Q3(75%)까지 박스 형태로 위치시킴
- IQR(InterQuatile Range) : Q1 - Q3
- $Q1 - 1.5 \times IQR$ 을 하단 값으로, $Q3 + 1.5 \times IQR$ 을 상단 값으로
- 이상치(outlier) : 상단 값과 하단 값을 넘어가는 값들



02 시본

1. 시본의 기본

- 시본(seaborn) : 맷플롯립을 바탕으로 다양한 함수 사용을 돕는 일종의 래퍼(wrapper) 모듈

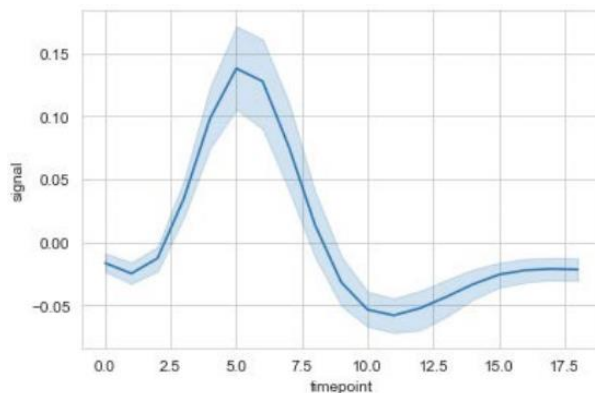
```
import seaborn as sns
```

- 맷플롯립과 동일한 결과물이 나오며, 작성 과정이 간단
 - 그림 객체나 축 객체 같은 복잡한 개념이 없음
 - xticks 설정하지 않아도 각 축에 라벨 자동으로 생성
 - 데이터프레임과 x, y에 해당하는 열 이름만 지정하면 됨

```
In [1]: import numpy as np          # numpy 모듈 호출
import pandas as pd              # pandas 모듈 호출
import matplotlib.pyplot as plt  # matplotlib 모듈 호출
import seaborn as sns            # (1)seaborn 모듈 호출

fmri = sns.load_dataset("fmri")  # (2)fmri 데이터셋 사용
sns.set_style("whitegrid")        # (3) 기본 스타일 적용
sns.lineplot(x="timepoint", y="signal", data=fmri)
                                   # (4) 선그래프 작성
```

Out [1]:



- fmri 데이터는 연속형 값 외에도 다양한 범주형 값 가짐
 - 이럴 때 맷플롯으로 표현하기는 상당히 복잡하고,
시본은 hue 매개변수만 추가하면 그래프 그릴 수 있음

In [2]:

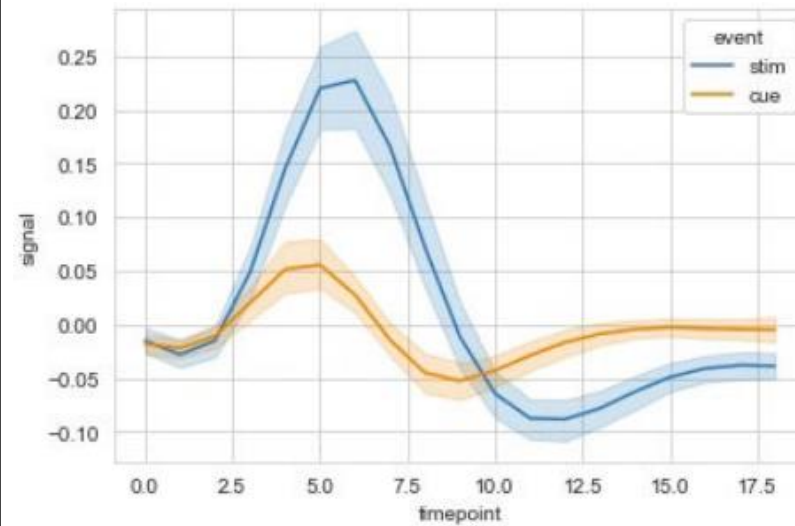
fmri.sample(n=10, random_state=1)

Out [2]:

	subject	timepoint	event	region	signal
806	s6	18	cue	parietal	0.019532
691	s5	15	cue	frontal	-0.019507
148	s5	8	stim	parietal	0.006805
676	s13	0	cue	parietal	-0.018394
156	s11	7	stim	parietal	0.254042
27	s1	17	stim	parietal	-0.038021
200	s11	4	stim	parietal	0.087175
262	s3	0	stim	parietal	-0.008576
94	s4	12	stim	parietal	-0.090036
339	s4	5	stim	frontal	0.455575

In [3]: `sns.lineplot(x="timepoint", y="signal", hue="event", data=fmri)`

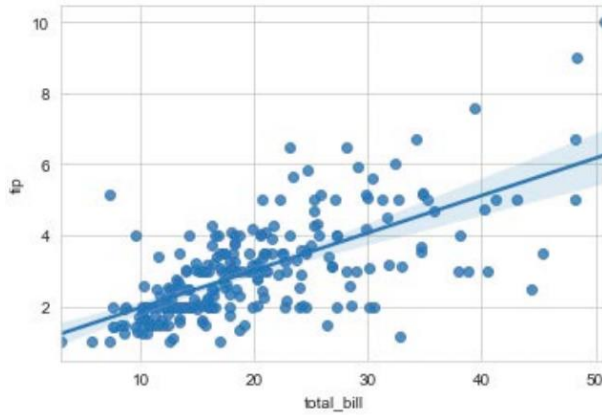
Out [3]:



2. 시본에서 사용하는 그래프

2.1 회귀 그래프

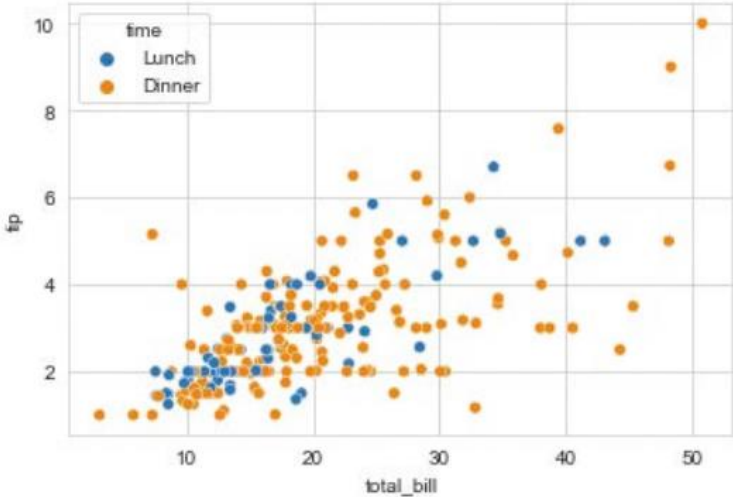
- 회귀 그래프(regression plot) : 회귀식을 적용하여 선형회귀 추세선을 그래프에 함께 작성
 - 선형회귀 추세선 : 데이터를 기반으로 데이터의 x값 대비 y값 변화를 예측하는 직선
- 함수 regplot 사용

In [4]:	<pre>tips = sns.load_dataset("tips") sns.regplot(x="total_bill", y="tip", data=tips, x_ci=95)</pre>
Out [4]:	

- 매개변수 `x_ci`는 신뢰구간의 비율을 나타냄

2.2 산점도

- 산점도(scatter plot) : x, y를 기준으로 데이터의 분포 표현
- 함수 scatterplot 사용

In [5]:	<pre>tips = sns.load_dataset("tips") sns.scatterplot(x="total_bill", y="tip", hue="time", data=tips)</pre>
Out [5]:	

2.3 비교 그래프

- 비교 그래프(counter plot) : 범주형 데이터의 항목별 개수

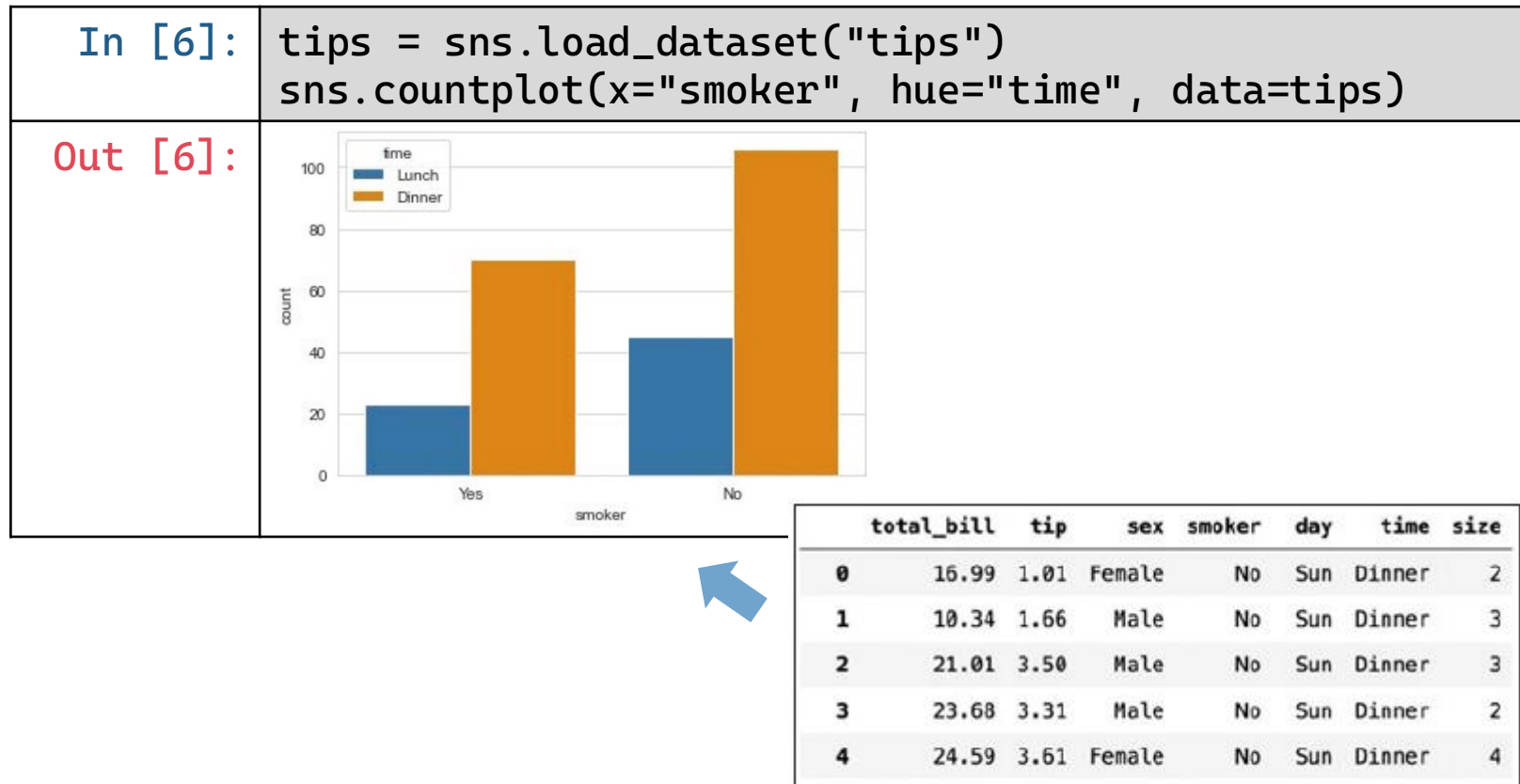
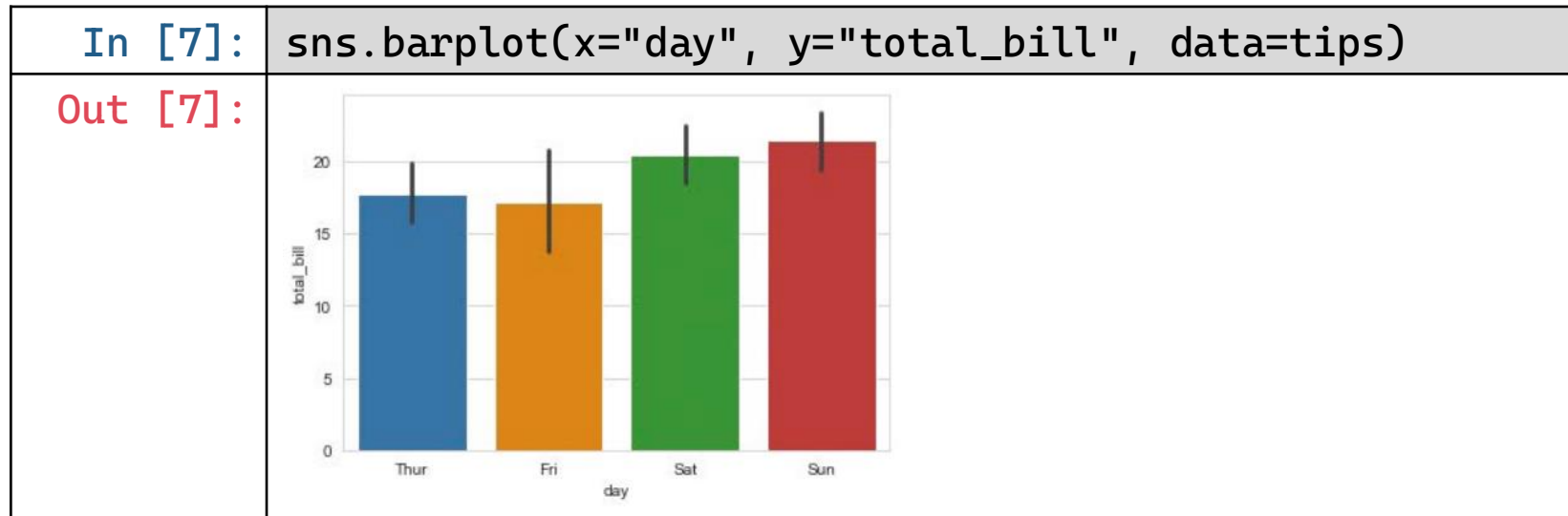


그림 5-4 비교 그래프 작성을 위한 데이터

2.4 막대그래프

- y 값이 연속형 값일 경우 해당 값들의 평균을 나타냄
- 데이터의 신뢰구간을 검은색 막대로 표현
- 함수 barplot 사용



3. 사전 정의된 그래프

- 맷플롯립 관점에서 여러 그래프들을 합쳐 정보를 추출
- 특히 범주형 데이터에 유용

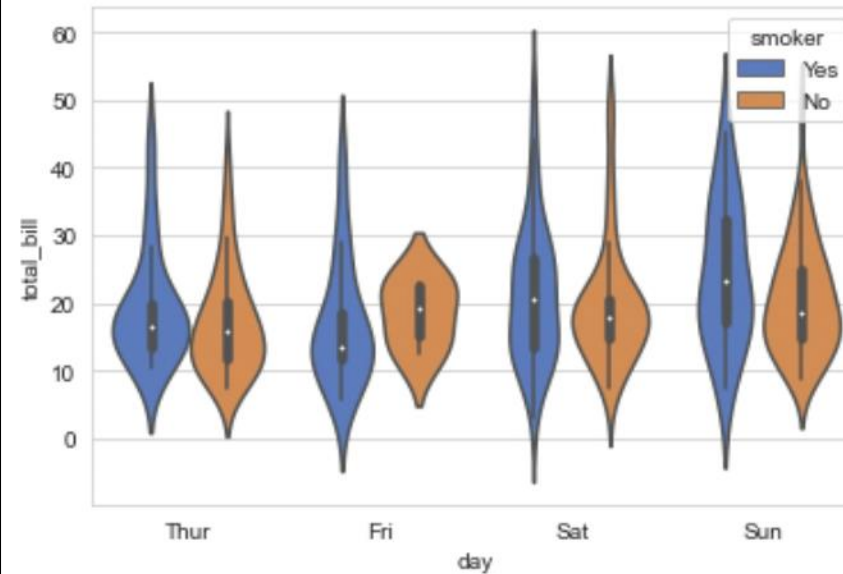
3.1 분포를 나타내는 그래프 : 바이올린 플롯과 스웜 플롯

- 바이올린 플롯(violin plot) : 상자그림과 분포도를 한 번에 나타낼 수 있음
 - x축에는 범주형 데이터, y축에는 연속형 데이터

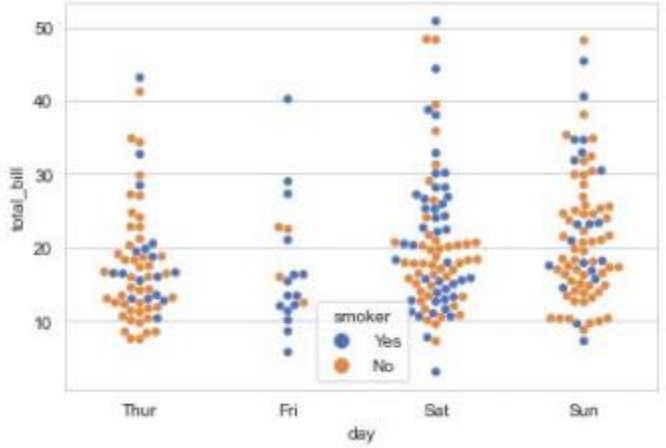
In [8]:

```
sns.violinplot(x="day", y="total_bill",  
hue="smoker", data=tips, palette="muted")
```

Out [8]:



- 스웜 플롯(swarm plot) : 바이올린 플롯과 같은 형태에 산점도로 데이터 분포를 나타냄
- 매개변수 hue로 두 개 이상의 범주형 데이터를 점이 겹치지 않게 정리
 - 영역별 데이터 양을 직관적으로 보여줌

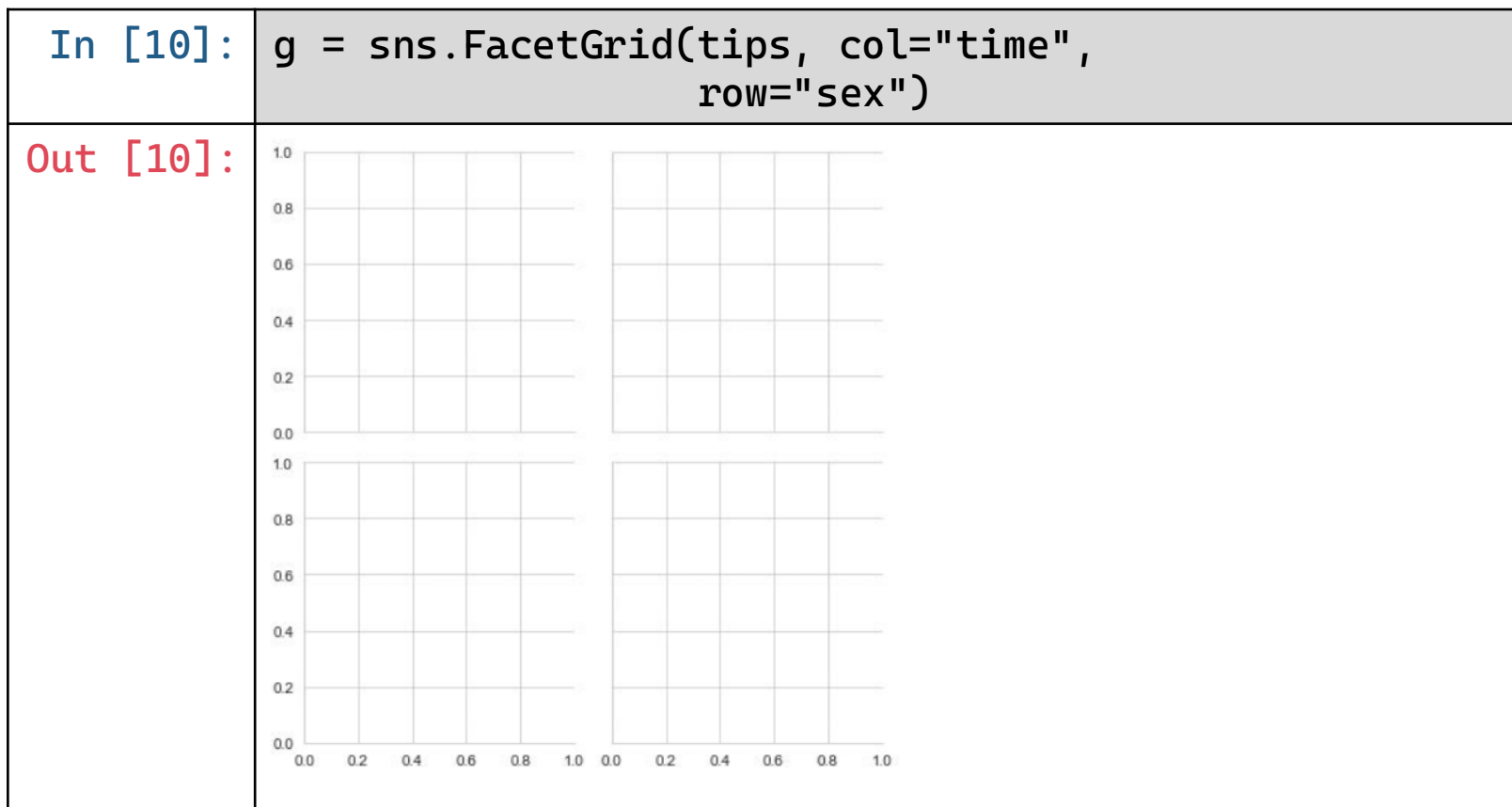
In [9]:	<pre>sns.swarmplot(x="day", y="total_bill", hue="smoker", data=tips, palette="muted")</pre>
Out [9]:	

3.2 다양한 범주형 데이터를 나타내는 패싯그리드

- 패싯그리드(FacetGrid) : 그래프의 틀만 제공하여 적당한 그래프를 그려주는 클래스

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

그림 5-5 다음 코드에서 다룰 데이터

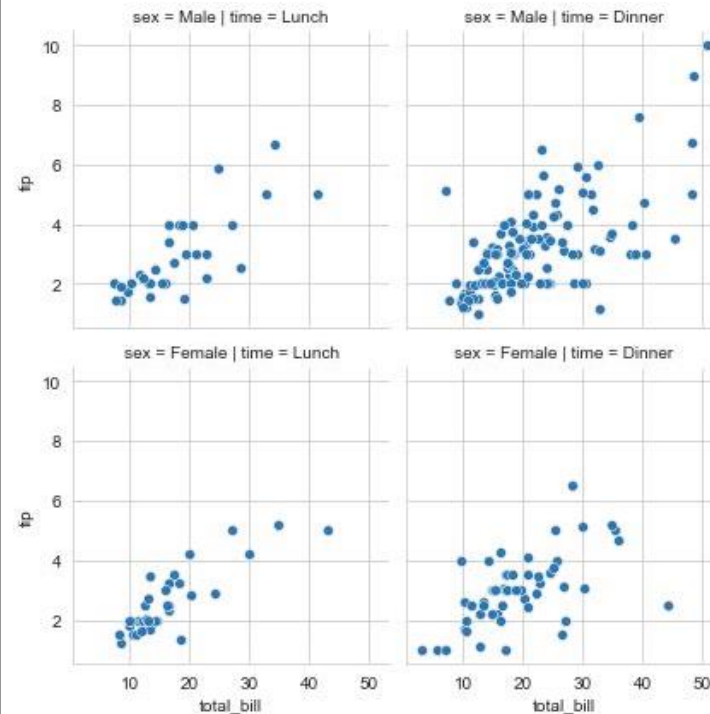


- 기본적인 데이터 표현 틀을 만들
- 매개변수 col과 row에 범주형 데이터를 넣으면 데이터 종류만큼 'm×n'의 그래프 틀 생성

- 그리드가 생성된 후 맵(map)을 사용하여 그래프 만듦
- 각 FacetGrid에 있는 개별 그래프 영역에
그래프를 집어넣는 구조
- 전체 데이터를 범주형 데이터의 다양한 관점에서 나눠서 볼 수 있음

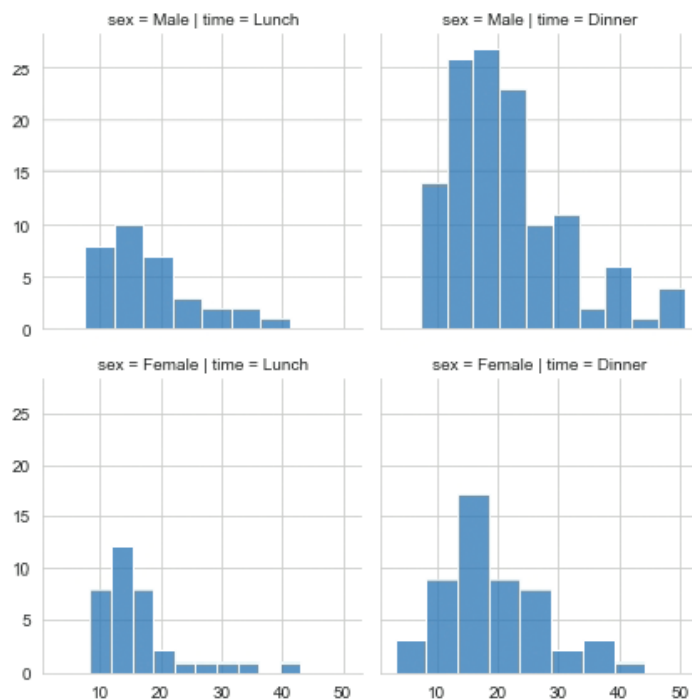
```
In [11]: g = sns.FacetGrid(tips,  
                             col="time",  
                             row="sex")  
  
g.map(sns.scatterplot,  
      "total_bill", "tip")
```

Out [11]:



```
In [12]: g = sns.FacetGrid(tips, col="time",  
                             row="sex")  
  
g.map_dataframe(sns.histplot,  
                x="total_bill")
```

Out [12]:



03

플롯리

1. 플롯리의 특징

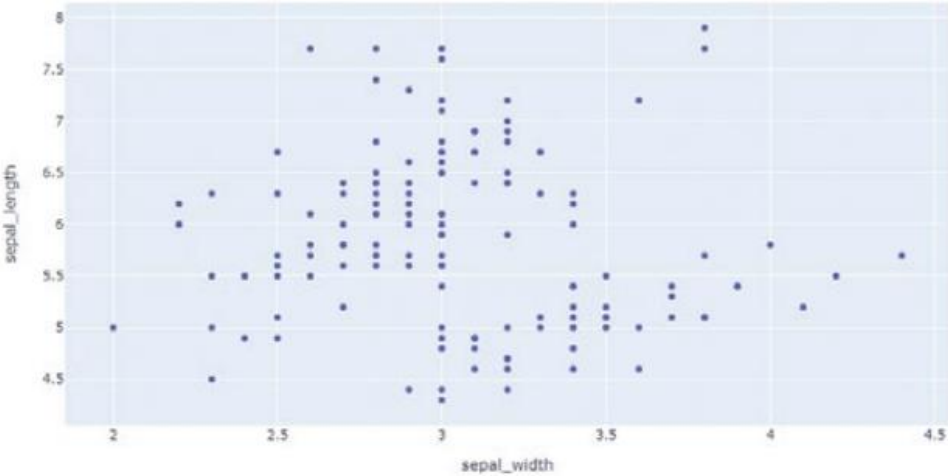
- 플롯리(plotly) : 비즈니스 인텔리전스(Business Intelligence) 대시보드로 개발된 도구
 - 비즈니스 인텔리전스 : BI 도구라고도 불림
사내 여러 데이터들을 정리하여 의사결정을 도움
- 애플리케이션으로, 사용자에게 그래프를 제공
 - 맷플롯립이나 시본은 데이터 분석가들이 데이터의 형태나 분포를 살피기 위해 코드로 사용하는 도구
- 인터랙션 그래프를 지원
 - 인터랙션 그래프 : 그래프 생성 이후 사용자가 인터페이스를 통해 조절 가능

2. 플롯리 사용하기

- 플롯리 설치
 - 터미널에 명령어 입력

```
(ml) C:\...>conda install -c plotly plotly
```

- 문법은 맷플롯립이나 시본과 유사

In [1]:	<pre>import plotly.express as px df = px.data.iris() # iris는 판다스 데이터프레임 fig = px.scatter(df, x="sepal_width", y="sepal_length") fig.show()</pre>
Out [1]:	

- iris 데이터셋을 호출하여 간단한 그래프를 생성
- 래퍼 모듈인 express를 호출한 뒤 산점도를 호출

- 생성된 그래프에 마우스 커서를 올리면 데이터를 볼 수 있음 (인터랙션 그래프)

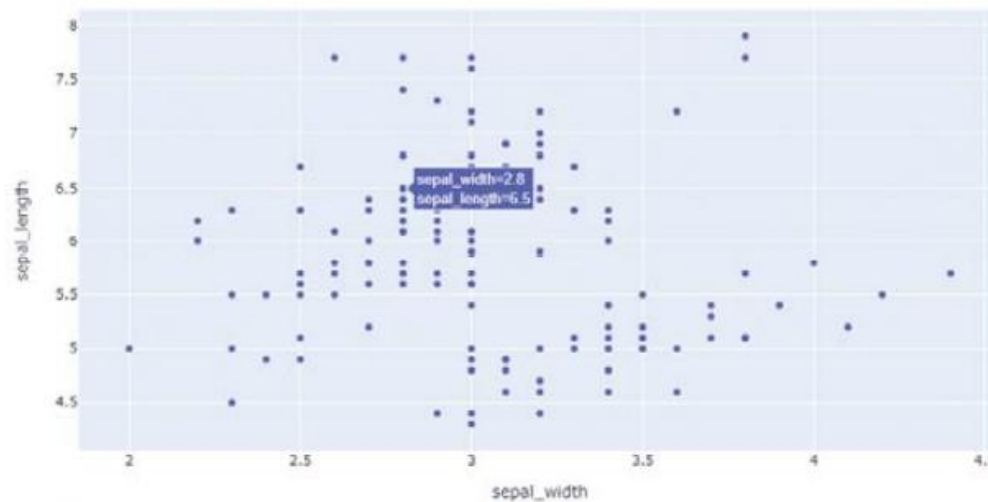
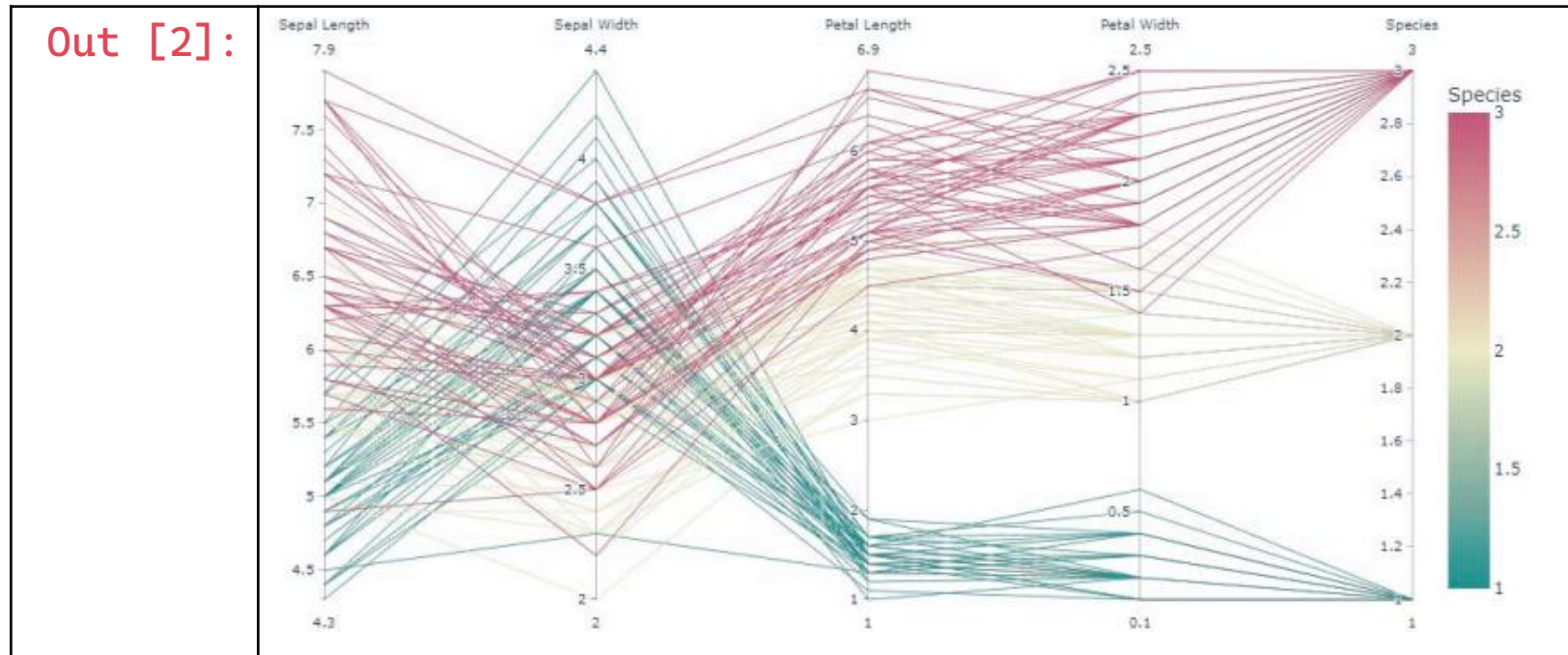


그림 5-6 플롯리로 호출한 그래프의 인터랙션 반응

- 좌표 그래프(coordinates plot) : 데이터 간 관계를 표현
 - 시본은 제공하지 않지만 플롯리에서 제공하는 기능

```
In [2]: fig = px.parallel_coordinates(df, color="species_id",  
                                     labels={"species_id": "Species",  
                                              "sepal_width": "Sepal Width",  
                                              "sepal_length": "Sepal Length",  
                                              "petal_width": "Petal Width",  
                                              "petal_length": "Petal Length", },  
                                     color_continuous_scale=  
                                         px.colors.diverging.Tealrose,  
                                     color_continuous_midpoint=2)  
  
fig.show()
```



[TIP] 다양한 그래프들을 맷플롯립처럼 바닥부터 작성할 수도 있다.
대표적으로 `graph_objects`를 사용하면 다양한 그래프를 만들어 낼 수 있다.