데이터 시각화 - **matplotlib**

시각화(Visualization)

- 시각화(Visualization)는 데이터분석 결과를 Plot이나 Graph등을 통해서
  시각적으로 전달할 수 있는 방법입니다

- 통계수치를 사용한 정량적 분석이 정확한 분석 내용을 전달하는데 강점이 있다면,
  시각화는 데이터 분석 내용을 한눈에 볼 수 있게 내용을 효과적으로 전달할 수 있습니다
  또한, 통계수치상으로는 파악하기 쉽지 않은 내용 또한 분석이 가능한 경우도 존재합니다
  (e.g. 데이터 분포를 시각화로 나타낼 때)

데이터시각화 - **matplotlib**

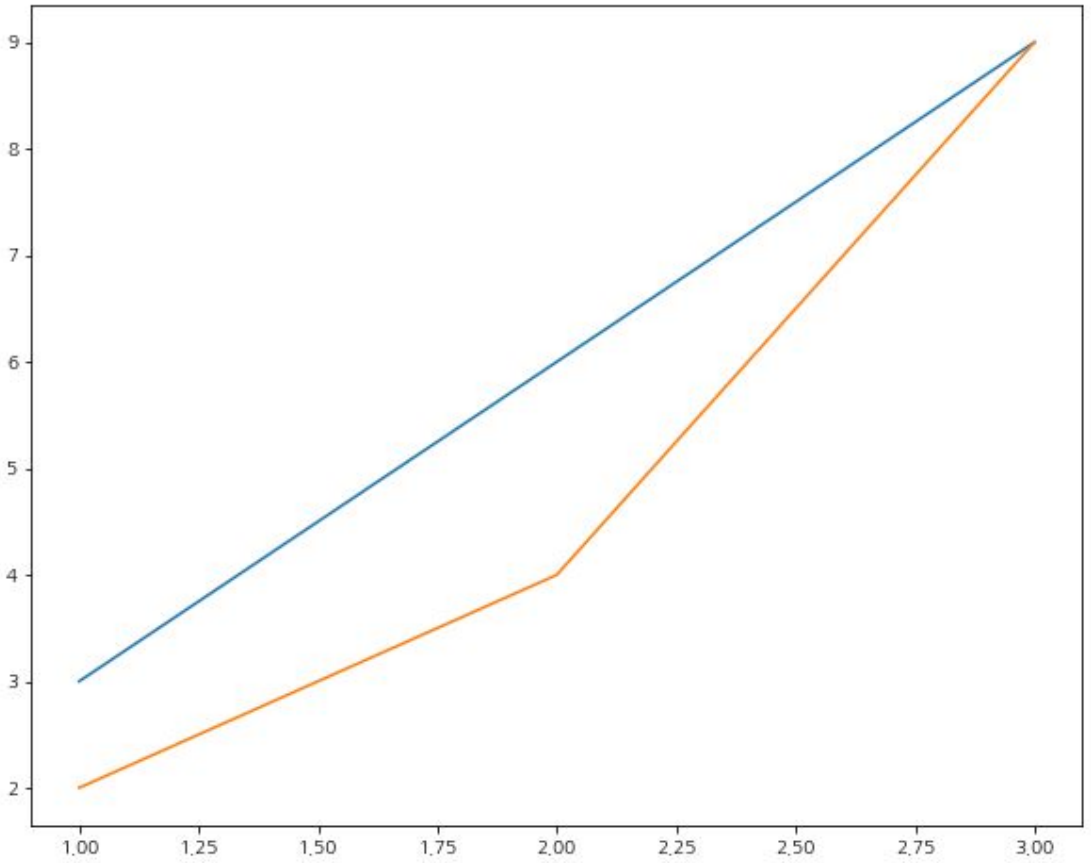
- matplotlib은 다양한 데이터를 많은 방법으로 도식화 할 수 있도록 하는 파이썬 라이브러리로써, 우리는 matplotlib의 pyplot을 이용하게 됩니다

- matplotlib을 이용하면 numpy나 pandas에서 사용되는 자료구조를 쉽게 시각화 할 수 있습니다

Matplotlib Visualization - 기본 문법

- Colab이나 Jupyter notebook 상에서는 plt.plot() 등의 명령문을 작성한 뒤, plt.show() 를 통해서 바로 플롯을 보거나 plt.savefig() 를 통해서 플롯을 파일로 저장합니다

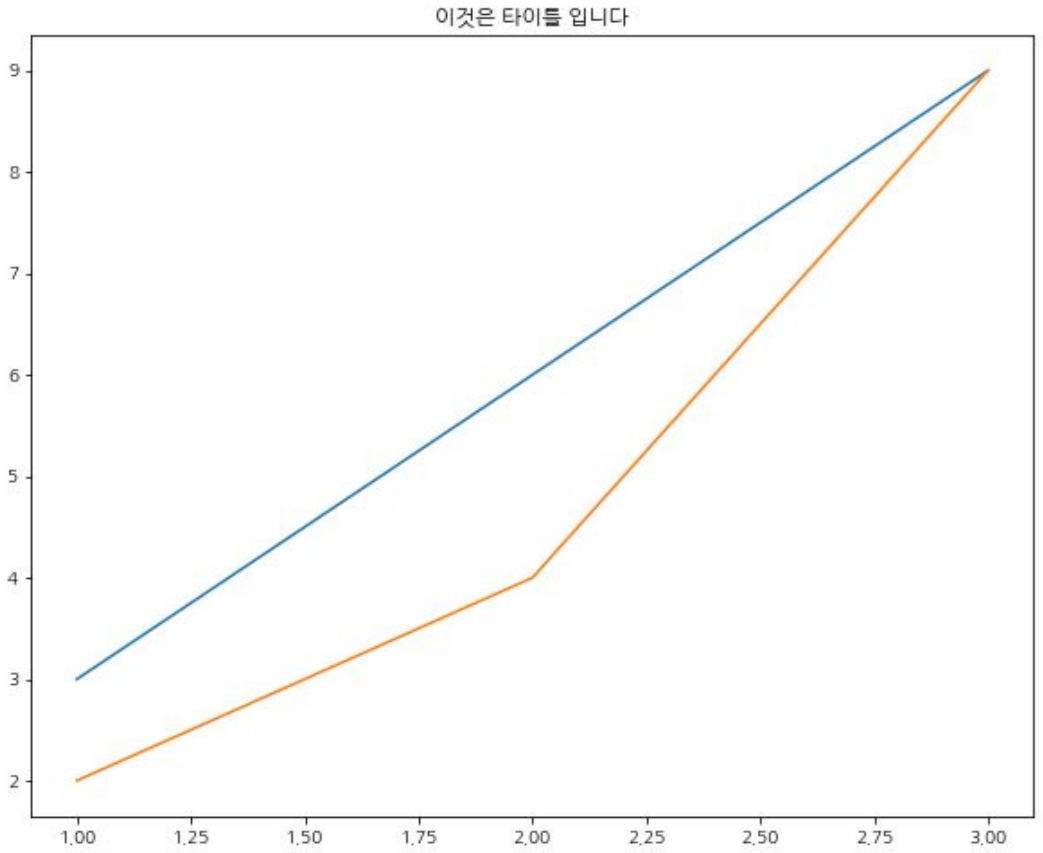
```
import matplotlib.pyplot as plt
# 기본 문법
plt.plot([1, 2, 3], [3, 6, 9])
plt.plot([1, 2, 3], [2, 4, 9])

plt.show()
```



Matplotlib Visualization - 기본 문법

- plt.title(''): 플롯의 타이틀을 작성해줍니다

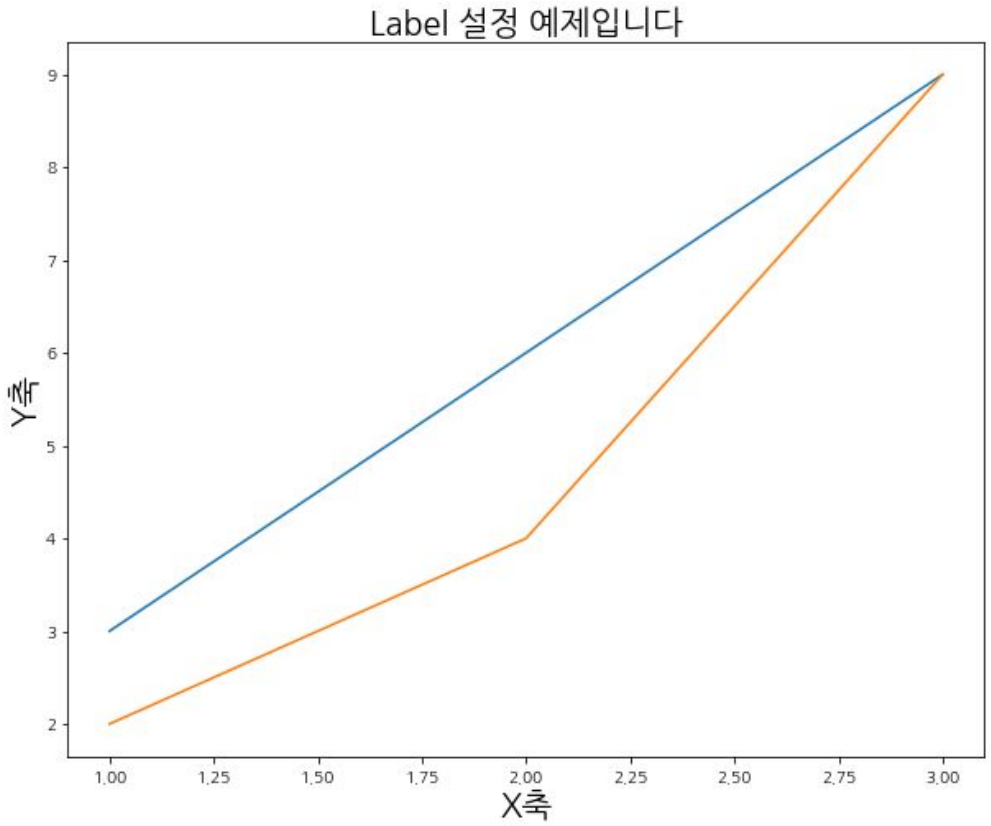
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [3, 6, 9])
plt.plot([1, 2, 3], [2, 4, 9])

# 타이틀 & font 설정
plt.title('이것은 타이틀 입니다')
plt.show()
```



Matplotlib Visualization - 기본 문법

- plt.xlabel(‘’, fontsize=), plt.ylabel(‘’, fontsize=): 플롯의 X축, Y축 label을 설정합니다

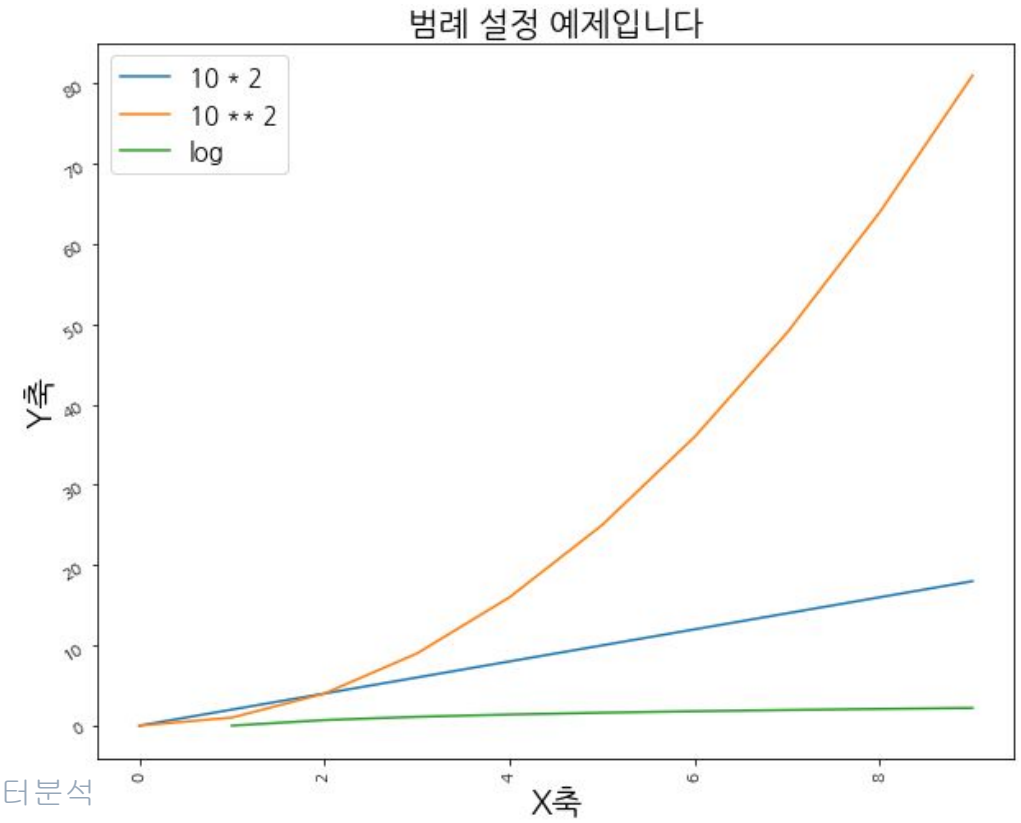
```
plt.plot([1, 2, 3], [3, 6, 9])
plt.plot([1, 2, 3], [2, 4, 9])
plt.title('Label 설정 예제입니다', fontsize=20)
# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

plt.show()
```



Matplotlib Visualization - 기본 문법

- plt.xticks(rotation=), plt.yticks:
  플롯의 X축, Y축에 눈금을 그어줍니다

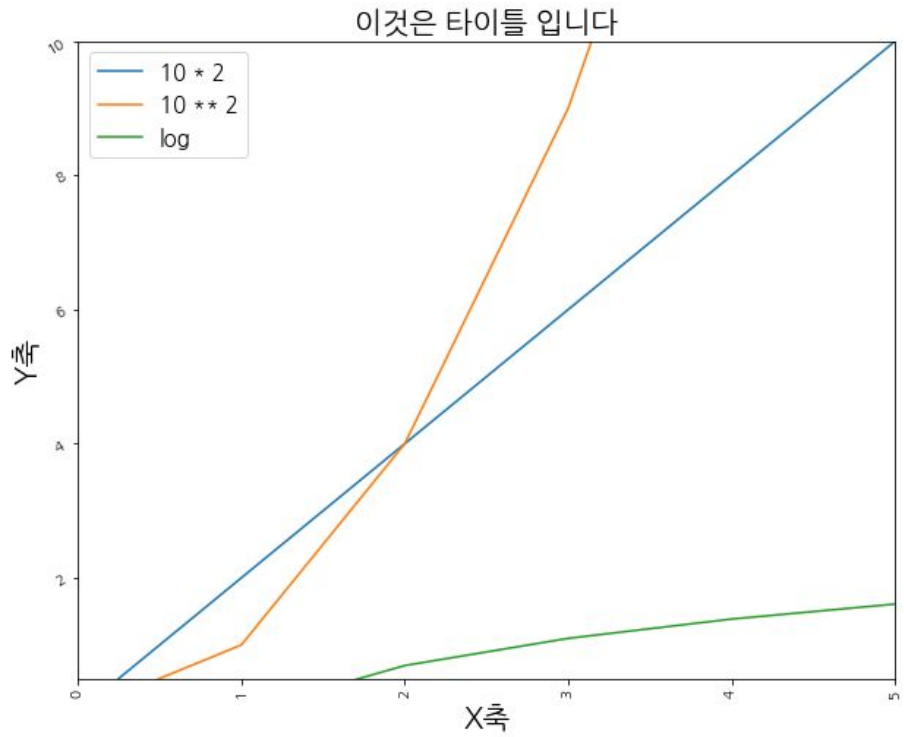
- plt.legend([“,”,...], fontsize=):
  앞에서 정의된 각 플롯에 대해서 범례를 설정합니다




```
# 범례(Legend) 설정
plt.plot(np.arange(10), np.arange(10)*2)
plt.plot(np.arange(10), np.arange(10)**2)
plt.plot(np.arange(10), np.log(np.arange(10)))

# 타이틀 & font 설정
plt.title('범례 설정 예제입니다', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

# legend 설정
plt.legend(['10 * 2', '10 ** 2', 'log'], fontsize=15)

plt.show()
```

Matplotlib Visualization - 기본 문법

- plt.xlim(), plt.ylim():
  플롯의 X축, Y축의 끝값들을 설정해줍니다

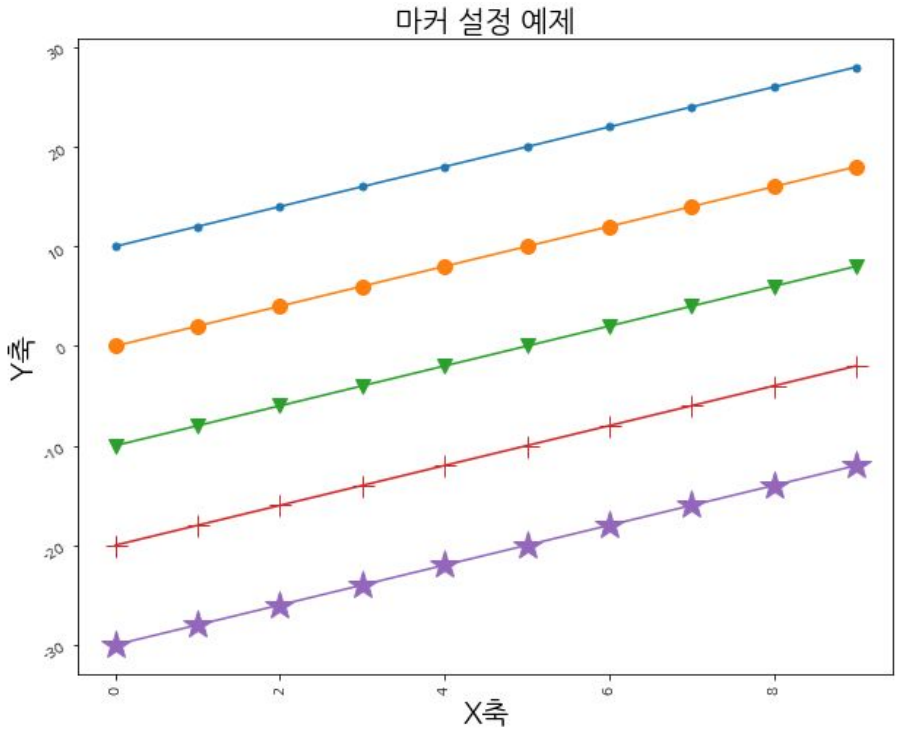



```
# X와 Y의 한계점(Limit) 설정: xlim(), ylim()

plt.plot(np.arange(10), np.arange(10)*2)
plt.plot(np.arange(10), np.arange(10)**2)
plt.plot(np.arange(10), np.log(np.arange(10)))

# 타이틀 & font 설정
plt.title('이것은 타이틀 입니다', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

# legend 설정
plt.legend(['10 * 2', '10 ** 2', 'log'], fontsize=15)

# x, y limit 설정
plt.xlim(0, 5)
plt.ylim(0.5, 10)

plt.show()
```

Matplotlib Visualization - 기본 문법

- 선형 그래프나 산점도의 경우 marker나 markersize, linestyle, color, alpha 등을 설정해 각각 점 모양, 점 크기, 선 모양, 색깔, 투명도 등을 조절할 수 있습니다

- 특히, 산점도에서 color를 label로 주어 label에 따른 분포를 파악하는 것도 가능합니다

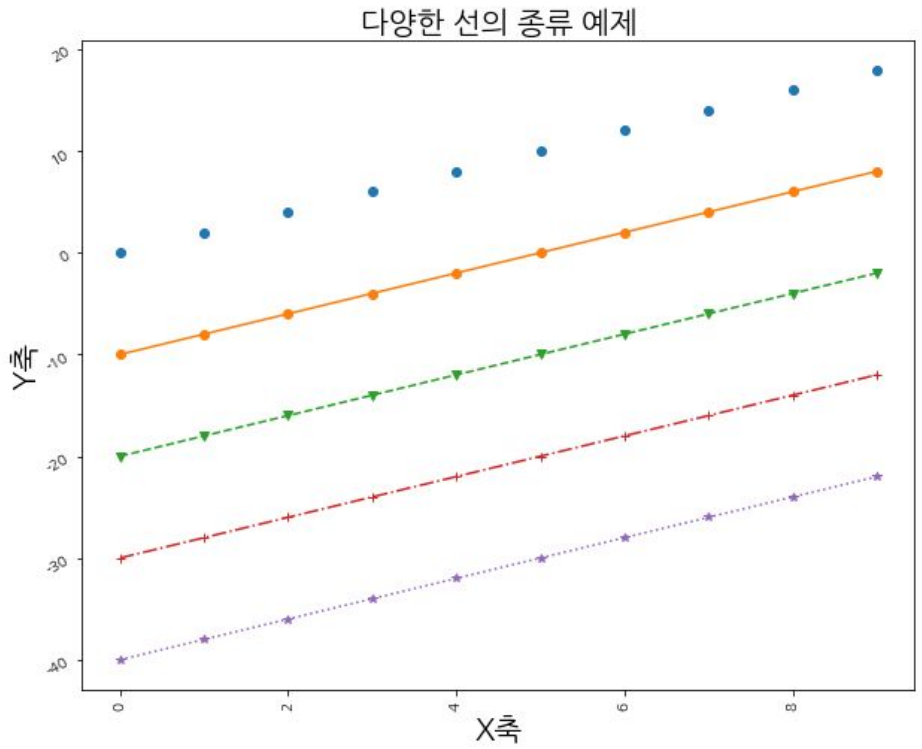



```
# 마커(marker)

# '.': point marker
# 'o': circle marker
# 'v': triangle_down marker
# '+': plus marker
# '*': star marker

plt.plot(np.arange(10), np.arange(10)*2 + 10, marker='.',
markersize=10)
plt.plot(np.arange(10), np.arange(10)*2, marker='o', markersize=10)
plt.plot(np.arange(10), np.arange(10)*2 - 10, marker='v',
markersize=10)
plt.plot(np.arange(10), np.arange(10)*2 - 20, marker='+',
markersize=15)
plt.plot(np.arange(10), np.arange(10)*2 - 30, marker='*',
markersize=20)

# 타이틀 & font 설정
plt.title('마커 설정 예제', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

plt.show()
```

Matplotlib Visualization - 기본 문법

- 선형 그래프나 산점도의 경우 marker나 markersize, linestyle, color, alpha 등을 설정해 각각 점 모양, 점 크기, 선 모양, 색깔, 투명도 등을 조절할 수 있습니다

- 특히, 산점도에서 color를 label로 주어 label에 따른 분포를 파악하는 것도 가능합니다

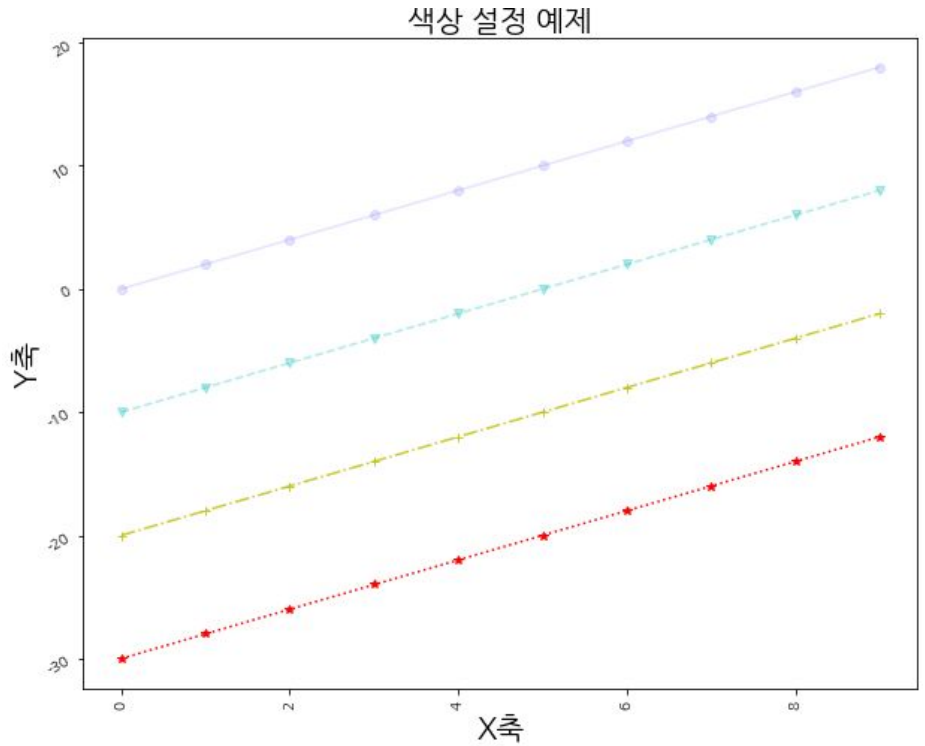



```
# 라인(line)

# '-': solid line style
# '--': dashed line style
# '-.': dash-dot line style
# ':': dotted line style

plt.plot(np.arange(10), np.arange(10)*2, marker='o', linestyle='')
plt.plot(np.arange(10), np.arange(10)*2 - 10, marker='o',
linestyle='-')
plt.plot(np.arange(10), np.arange(10)*2 - 20, marker='v',
linestyle='--')
plt.plot(np.arange(10), np.arange(10)*2 - 30, marker='+',
linestyle='-.')
plt.plot(np.arange(10), np.arange(10)*2 - 40, marker='*',
linestyle=':')

# 타이틀 & font 설정
plt.title('다양한 선의 종류 예제', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

plt.show()
```

Matplotlib Visualization - 기본 문법

- 선형 그래프나 산점도의 경우 marker나 markersize, linestyle, color, alpha 등을 설정해 각각 점 모양, 점 크기, 선 모양, 색깔, 투명도 등을 조절할 수 있습니다

- 특히, 산점도에서 color를 label로 주어 label에 따른 분포를 파악하는 것도 가능합니다




```
# 컬러(color)

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

# alpha를 통해 투명도를 조절할 수 있습니다.

plt.plot(np.arange(10), np.arange(10)*2, marker='o', linestyle='-',
color='b', alpha=.1)
plt.plot(np.arange(10), np.arange(10)*2 - 10, marker='v', linestyle='--',
color='c', alpha=.3)
plt.plot(np.arange(10), np.arange(10)*2 - 20, marker='+', linestyle='-.',
color='y', alpha=.8)
plt.plot(np.arange(10), np.arange(10)*2 - 30, marker='*', linestyle=':',
color='r', alpha=1.)

# 타이틀 & font 설정
plt.title('색상 설정 예제', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

plt.show()
```

Matplotlib Visualization - 기본 문법

- plt.show()와 마찬가지로, 모든 명령 이후 plt.savefig('filename', dpi=)를 이용해 플롯을 저장할 수 있습니다

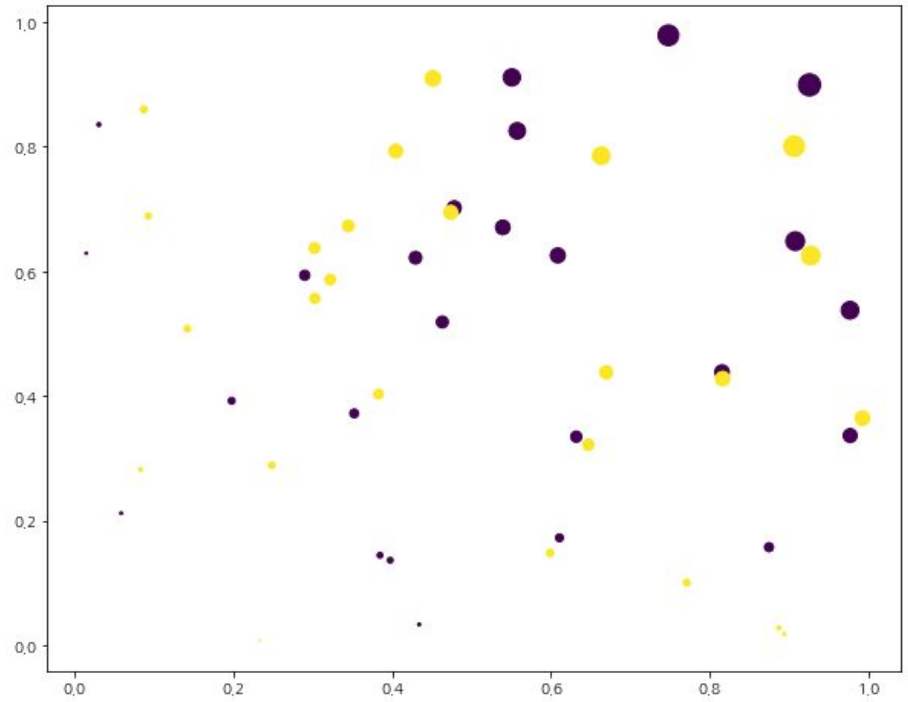
```
#  Plot 저장하기

plt.plot(np.arange(10), np.arange(10)*2, marker='o',
linestyle='-', color='b', alpha=.1)
plt.plot(np.arange(10), np.arange(10)*2 - 10, marker='v',
linestyle='--', color='c', alpha=.3)
plt.plot(np.arange(10), np.arange(10)*2 - 20, marker='+',
linestyle='-.', color='y', alpha=.8)
plt.plot(np.arange(10), np.arange(10)*2 - 30, marker='*',
linestyle=':', color='r', alpha=1.)

# 타이틀 & font 설정
plt.title('색상 설정 예제', fontsize=20)

# X축 & Y축 Label 설정
plt.xlabel('X축', fontsize=20)
plt.ylabel('Y축', fontsize=20)

# X tick, Y tick 설정
plt.xticks(rotation=90)
plt.yticks(rotation=30)

# plt.show()
plt.savefig('savefig_200dpi.png', dpi=200)
```

Matplotlib Visualization - scatter plot(산점도)

- plt.scatter(s=, c=):
  x, y 좌표로 이뤄진 데이터들을 점으로 표현하는 플롯입니다

- s는 점 크기, c는 점 색깔을 조절하며
  label이나 class에 따른 데이터 분포를 확인할 때 활용이 가능합니다

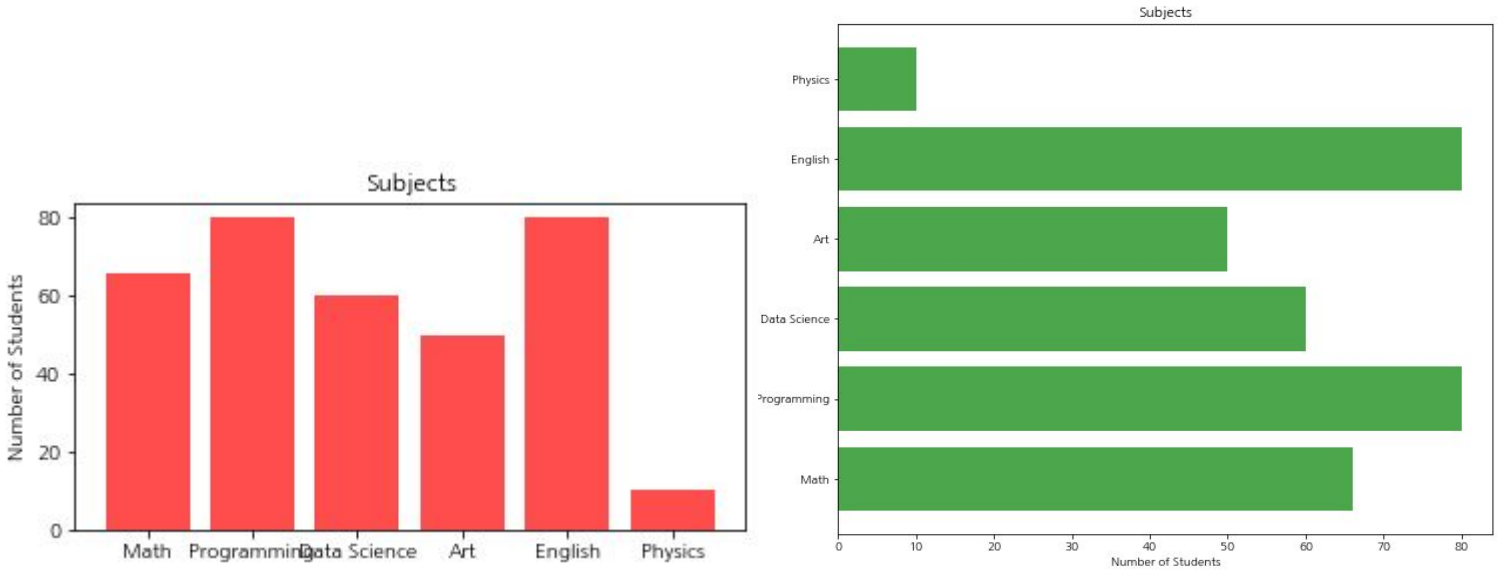
```
x = np.random.rand(50)
y = np.random.rand(50)
# 각 데이터점마다 class를 가지고 있는 경우, class를 color로도 표현할 수 있습니다.
colors = [0]*25 + [1]*25
area = x * y * 250
plt.scatter(x, y, s=area, c=colors)
plt.show()
```



Matplotlib Visualization - Barplot, Barhplot

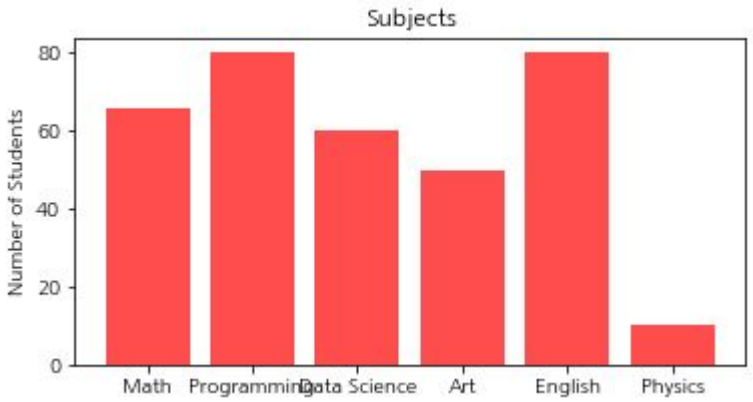
- plt.bar(), plt.barh():
  범주가 있는 데이터 값을 직사각형의 막대로
  표현하는 그래프입니다

- width, align 등의 옵션으로
  막대의 위치를 조정할 수 있으며,
  color와 alpha 옵션을 이용해 각 막대의
  색깔 또한 조정이 가능합니다

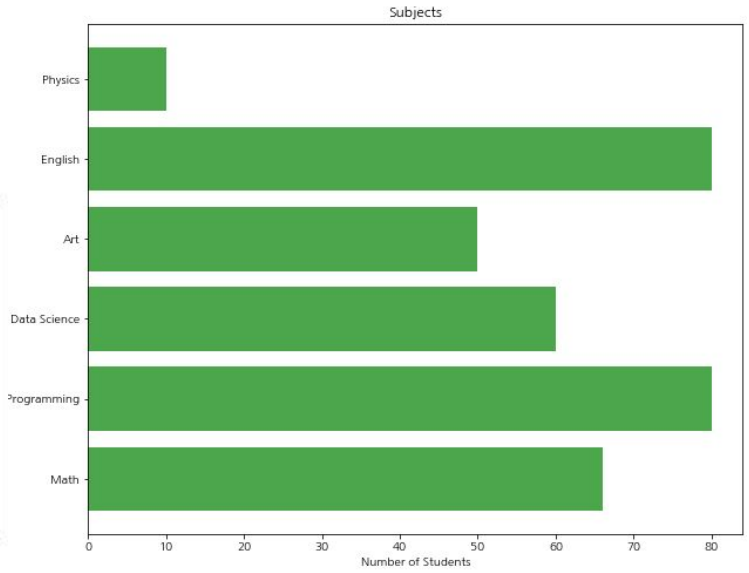


Matplotlib Visualization - Barplot, Barhplot

```
# Barplot: plt.bar()

x = ['Math', 'Programming', 'Data Science', 'Art', 'English',
'Physics']
y = [66, 80, 60, 50, 80, 10]

plt.figure(figsize=(6, 3))
plt.bar(x, y, align='center', alpha=0.7, color='red')
plt.xticks(x)
plt.ylabel('Number of Students')
plt.title('Subjects')

plt.show()
```

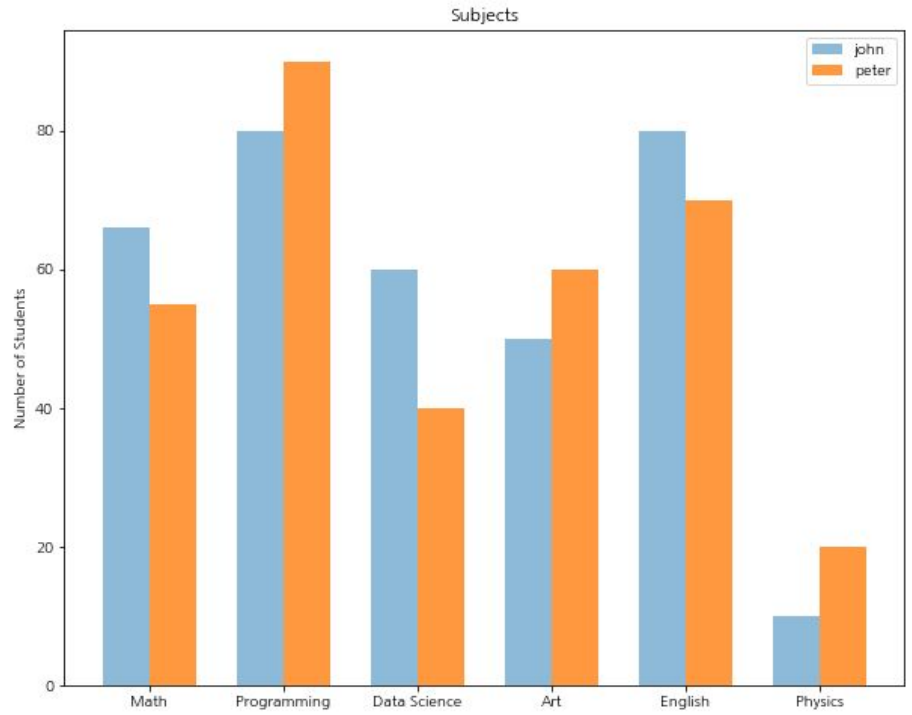



```
# Barhplot (축 변환): plt.barh()
# barh 함수에서는 xticks로 설정했던 부분을 yticks로 변경합니다.

x = ['Math', 'Programming', 'Data Science', 'Art', 'English',
'Physics']
y = [66, 80, 60, 50, 80, 10]

plt.barh(x, y, align='center', alpha=0.7, color='green')
plt.yticks(x)
plt.xlabel('Number of Students')
plt.title('Subjects')

plt.show()
```



Matplotlib Visualization - Barplot, Barhplot

- plt.subplots()과 plt.bar나 plt.barh를 함께 사용하면 비교 그래프 또한 그릴 수 있습니다

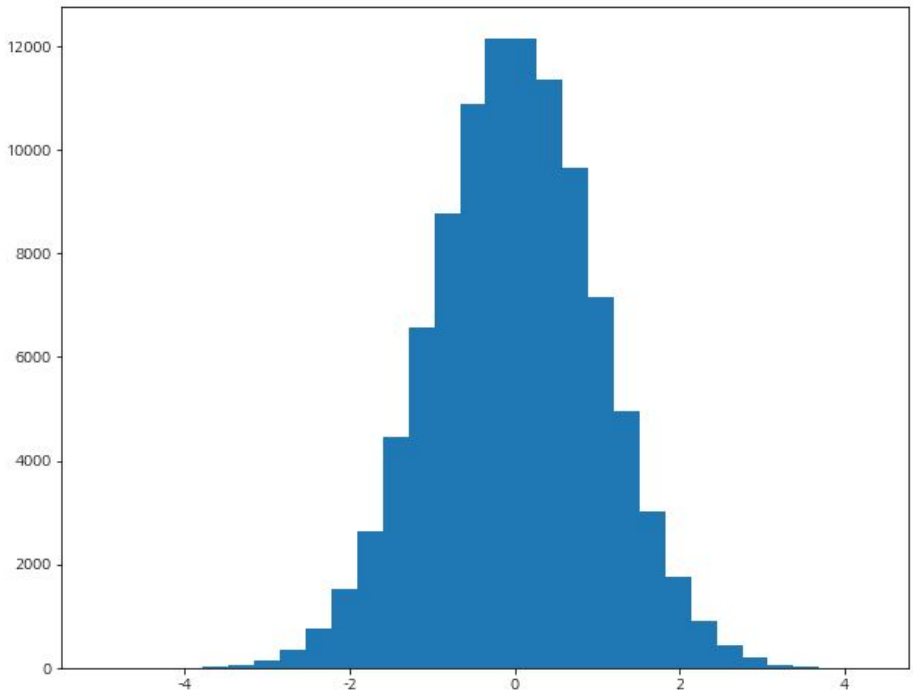



```
x_label = ['Math', 'Programming', 'Data Science', 'Art',
'English', 'Physics']
x = np.arange(len(x_label))
y_1 = [66, 80, 60, 50, 80, 10]
y_2 = [55, 90, 40, 60, 70, 20]

# 넓이 지정
width = 0.35

# subplots 생성
fig, axes = plt.subplots()

# 넓이 설정
axes.bar(x - width/2, y_1, width, align='center', alpha=0.5)
axes.bar(x + width/2, y_2, width, align='center', alpha=0.8)

# xtick 설정
plt.xticks(x)
axes.set_xticklabels(x_label)
plt.ylabel('Number of Students')
plt.title('Subjects')

plt.legend(['john', 'peter'])

plt.show()
```

Matplotlib Visualization - Histogram

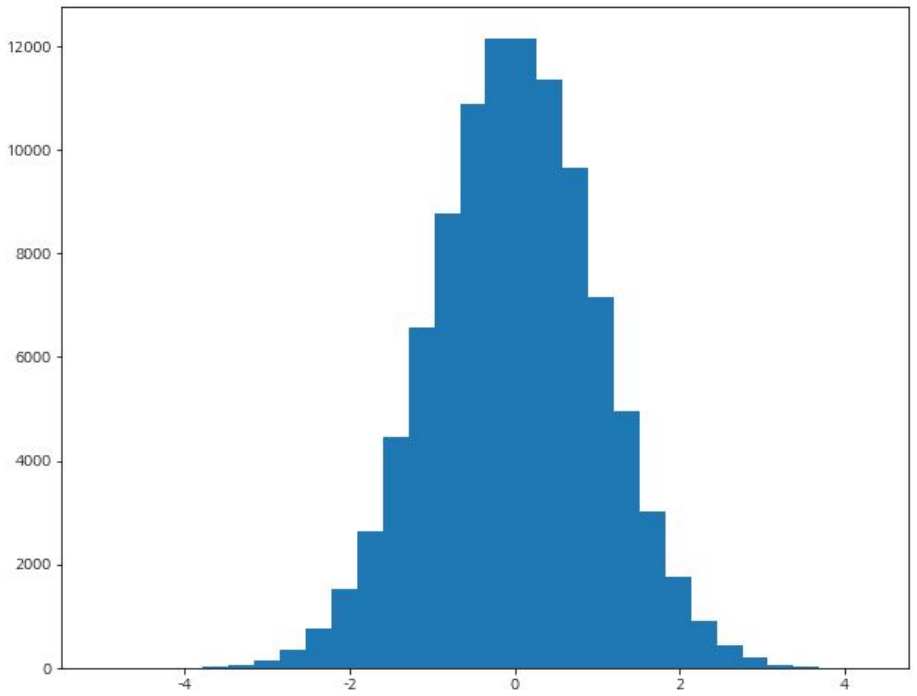
- plt.hist():
  히스토그램 (Histogram)은 도수분포표를
  그래프로 나타낸 것으로서,
  가로축은 변수 값,
  세로축은 빈도나 비율을 나타냅니다

- bins 옵션을 통해 히스토그램에서의 막대 개수를
  조절 할 수 있으며,
  density=True인 경우에는
  frequency 대신 density를 y축에 나타냅니다

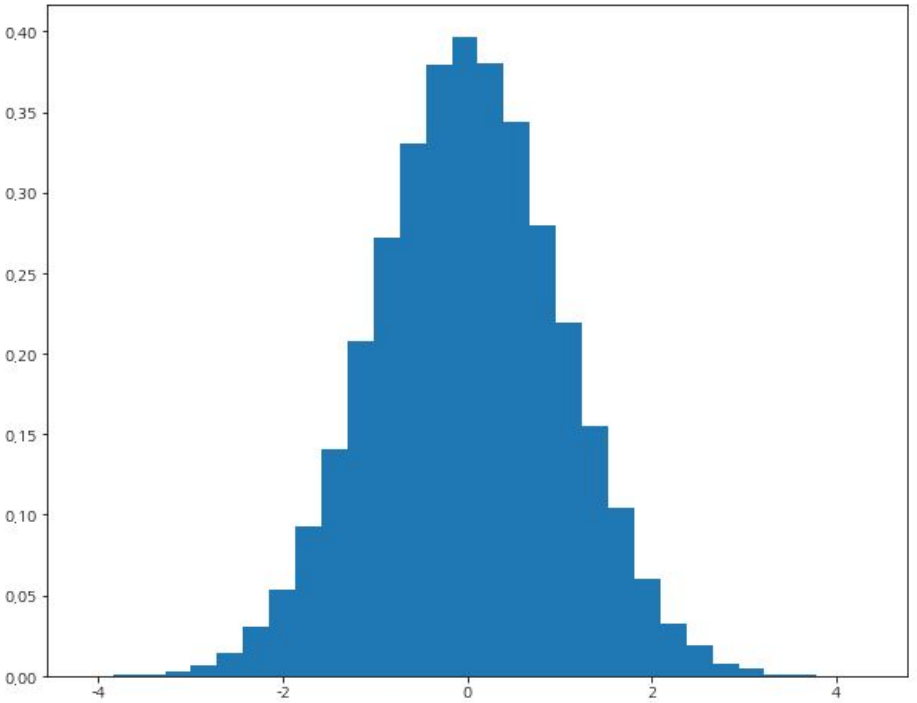


Matplotlib Visualization - Histogram

```
N = 100000
bins = 30

x = np.random.randn(N)

plt.hist(x, bins=bins)

plt.show()
```

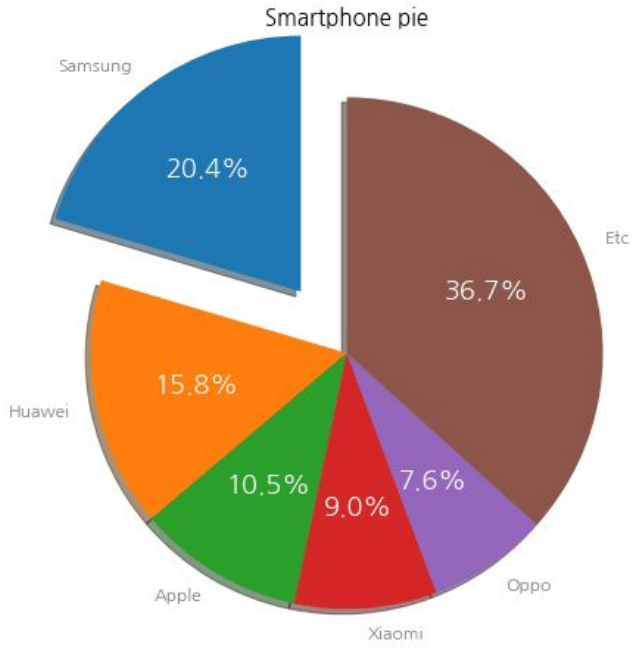



```
# density=True 값을 통하여 Y축에 frequency 대신
# density를 표기할 수 있습니다.

N = 100000
bins = 30

x = np.random.randn(N)

plt.hist(x, bins=bins, density=True)

plt.show()
```



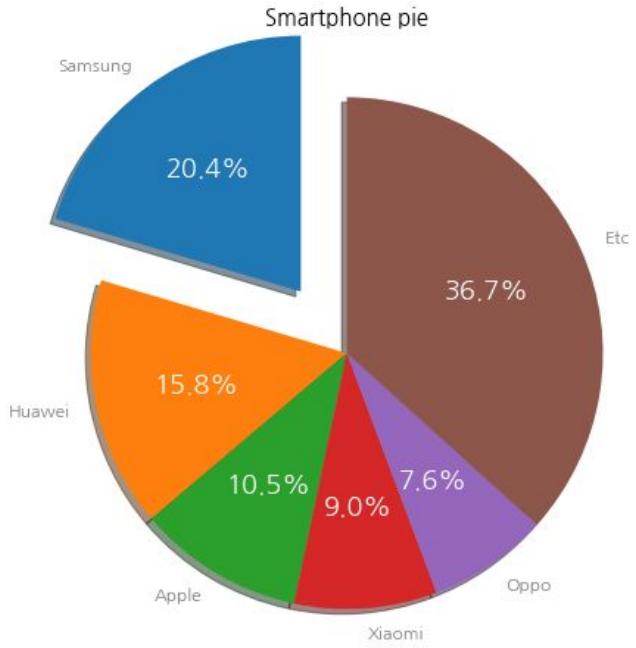
Matplotlib Visualization - Pie chart

- plt.pie():
  파이 차트는 범주별 구성 비율을 원형으로 표현한 그래프이며, 부채꼴의 중심각을 구성 비율에 비례하도록 표현합니다.



Matplotlib Visualization - Pie chart

- explode: 파이에서 툭 튀어져 나온 비율
- autopct: 퍼센트 자동으로 표기
  → '%.1f%%': 소수점 1자리까지 표기
- shadow: 그림자 표시
- startangle: 파이를 그리기 시작할 각도
- texts, autotexts 인자를 리턴 받습니다
  → texts는 label에 대한 텍스트 효과를,
  autotexts는 파이 위에 그려지는 텍스트 효과를 다룰 때 활용합니다

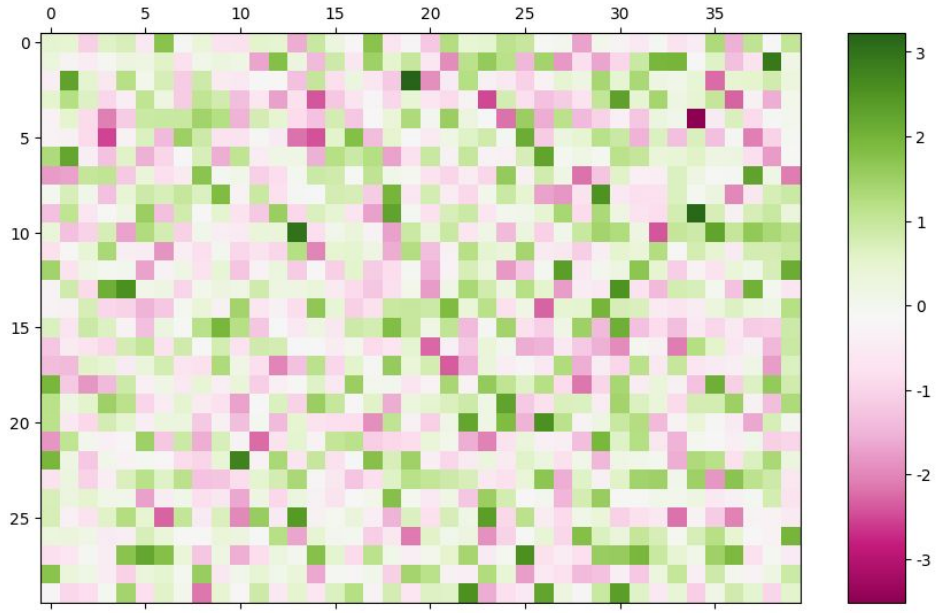



```
labels = ['Samsung', 'Huawei', 'Apple', 'Xiaomi', 'Oppo',
'Etc']
sizes = [20.4, 15.8, 10.5, 9, 7.6, 36.7]
explode = (0.3, 0, 0, 0, 0, 0)

# texts, autotexts 인자를 활용하여 텍스트 스타일링을 적용합니다
patches, texts, autotexts = plt.pie(sizes,
                                    explode=explode,
                                    labels=labels,
                                    # autopct='%.2f%%',
                                    autopct='%.1f%%',
                                    shadow=True,
                                    startangle=90)


plt.title('Smartphone pie', fontsize=15)

# label 텍스트에 대한 스타일 적용
for t in texts:
    t.set_fontsize(12)
    t.set_color('gray')

# pie 위의 텍스트에 대한 스타일 적용
for t in autotexts:
    t.set_color("white")
    t.set_fontsize(18)


plt.show()
```

Matplotlib Visualization - Heatmap

- plt.matshow():
  히트맵 (Heatmap)은 다양한 값을 갖는 숫자 데이터를 열분포 형태와 같이 색상을 이용해서 시각화한 것입니다

- 지도 이미지 위에 인구의 분포와 같이 2차원(x,y) 형태로
  표현되는 데이터의 빈도 분포를 보여주기에 적합합니다

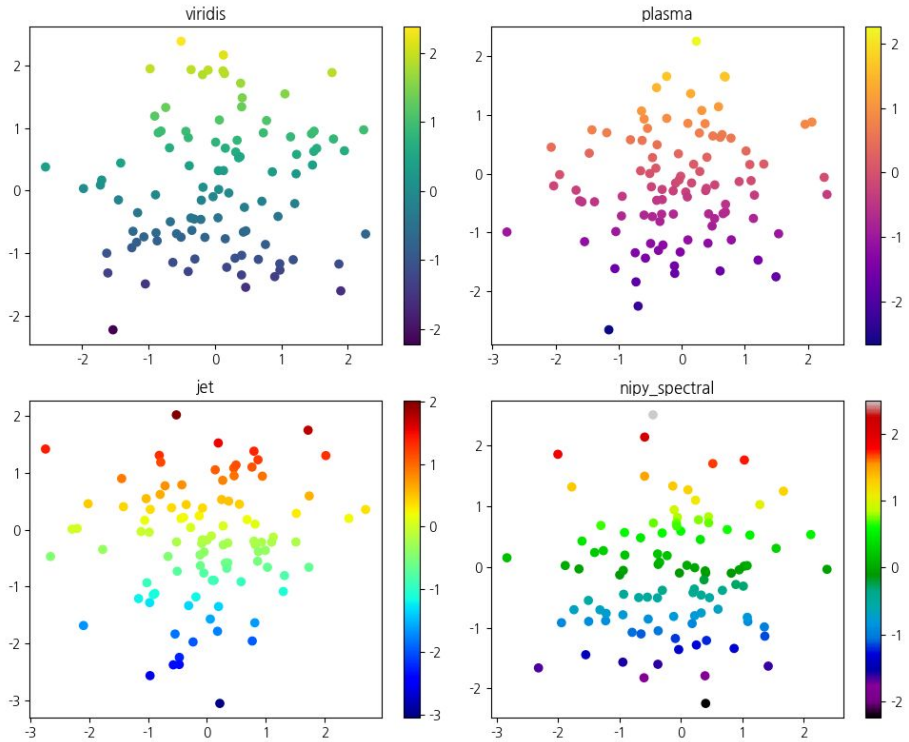



```
# Heatmap

import matplotlib.pyplot as plt
import numpy as np

arr = np.random.standard_normal((30, 40))

# 컬러맵 종류 지정
cmap = plt.get_cmap('PiYG')
# cmap = plt.get_cmap('BuGn')
# cmap = plt.get_cmap('Greys')
# cmap = plt.get_cmap('bwr')

plt.matshow(arr, cmap=cmap)
plt.colorbar(shrink=0.8, aspect=10)

plt.show()
```

Matplotlib Visualization - Colormaps

- matplotlib.pyplot 모듈은 컬러맵을 간편하게 설정하기 위한 여러 함수를 제공합니다

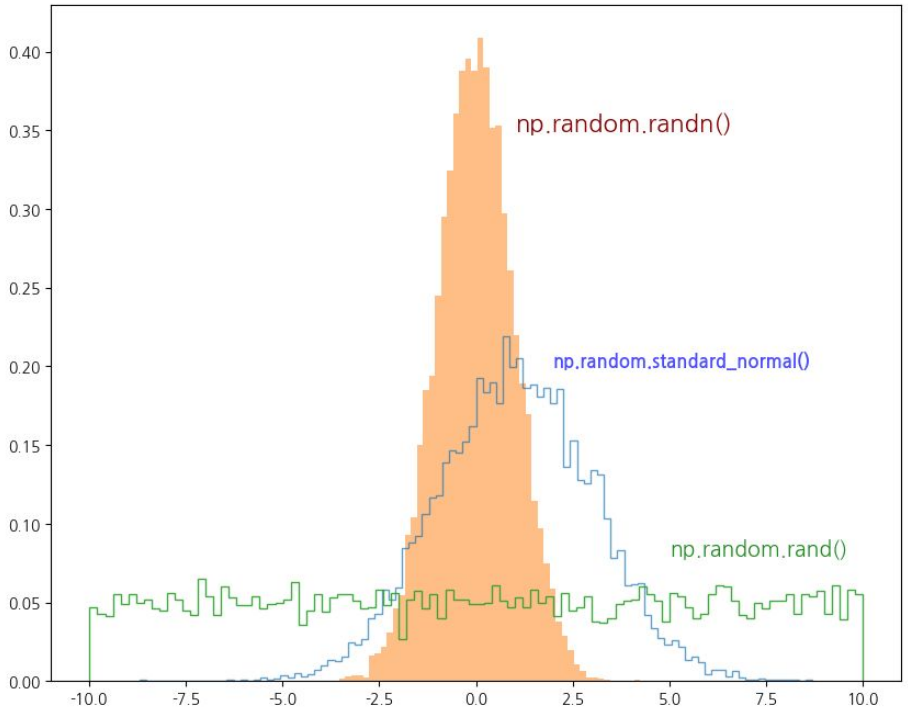
- plt.plasma(), plt.jet()와 같은 함수를 이용해서 플롯에서 사용되는 색깔들을 다양하게 만들 수 있습니다




```
# Colormaps
arr = np.random.standard_normal((8, 100))

plt.subplot(2, 2, 1)
plt.scatter(arr[0], arr[1], c=arr[1])
plt.viridis()
plt.title('viridis')

# colorbar() 함수를 사용하면 그래프 영역에 컬러바를 포함할 수 있습니다.
plt.colorbar()

plt.subplot(2, 2, 2)
plt.scatter(arr[2], arr[3], c=arr[3])
plt.plasma()
plt.title('plasma')
plt.colorbar()

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

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

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

Matplotlib Visualization - 텍스트 삽입하기

- plt.text(x위치, y위치, 텍스트, fontdict):
  그래프의 적절한 위치에 텍스트를 삽입하도록 합니다



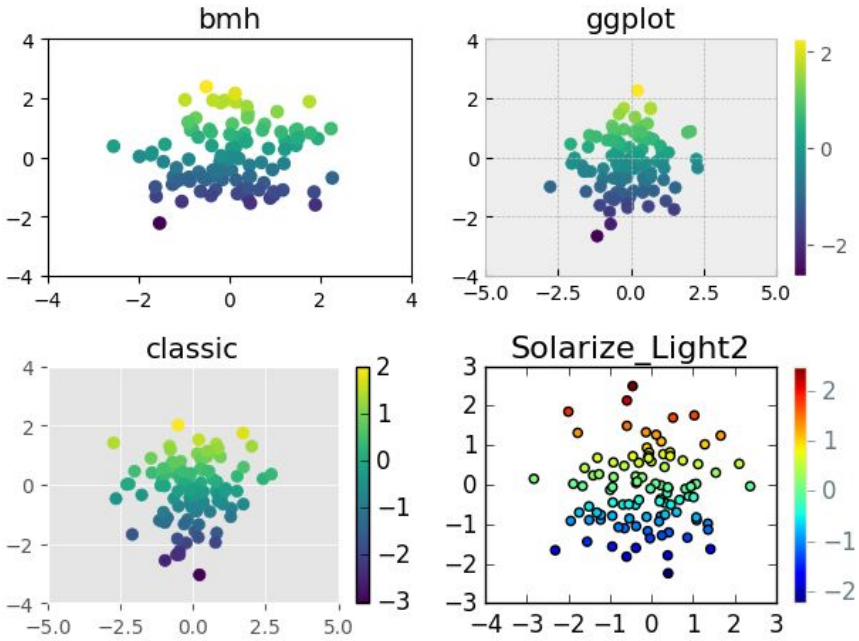

```
# 텍스트 삽입하기

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 = {'color':  'darkred',
     'weight': 'normal',
     'size': 16}

font2 = {'color':  'blue',
     'weight': 'bold',
     'size': 12,
     'alpha': 0.7}

font3 = {'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()
```

Matplotlib Visualization - 그래프 스타일 설정

- plt.style.use():
  미리 만들어놓은 Matplotlib 그래프 스타일을 사용해서 다양한 스타일을 지정할 수 있습니다

- 기본 스타일로 돌아가기 위해서는 plt.style.use('default')를 호출하면 됩니다

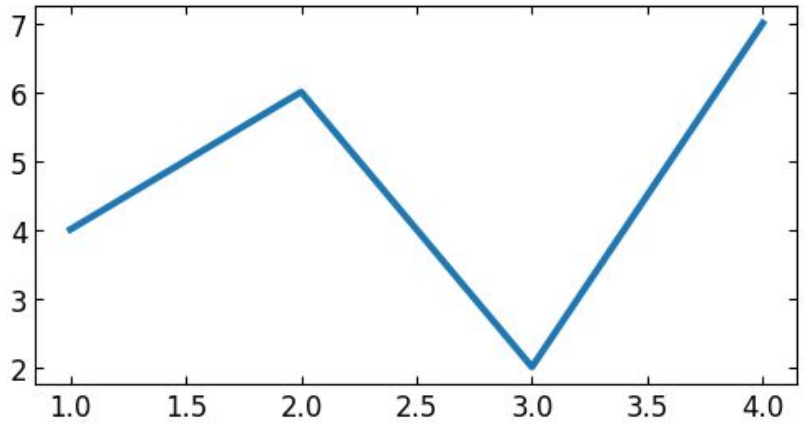
- plt.style.available를 통해 사용 가능한 스타일을 조회할 수 있습니다




```
# 그래프 스타일 설정

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])
plt.style.use('bmh')
plt.title('bmh')

plt.subplot(2, 2, 2)
plt.scatter(arr[2], arr[3], c=arr[3])
plt.style.use('ggplot')
plt.title('ggplot')
plt.colorbar()

plt.subplot(2, 2, 3)
plt.scatter(arr[4], arr[5], c=arr[5])
plt.style.use('classic')
plt.title('classic')
plt.colorbar()

plt.subplot(2, 2, 4)
plt.scatter(arr[6], arr[7], c=arr[7])
plt.style.use('Solarize_Light2')
plt.title('Solarize_Light2')
plt.colorbar()

plt.tight_layout()
plt.show()
plt.style.use('default')
```

Matplotlib Visualization - 그래프 스타일 설정

- 미리 지정해놓은 스타일을 사용하지 않고, 각각의 스타일 관련 파라미터 (rcParams)를 지정할 수 있습니다

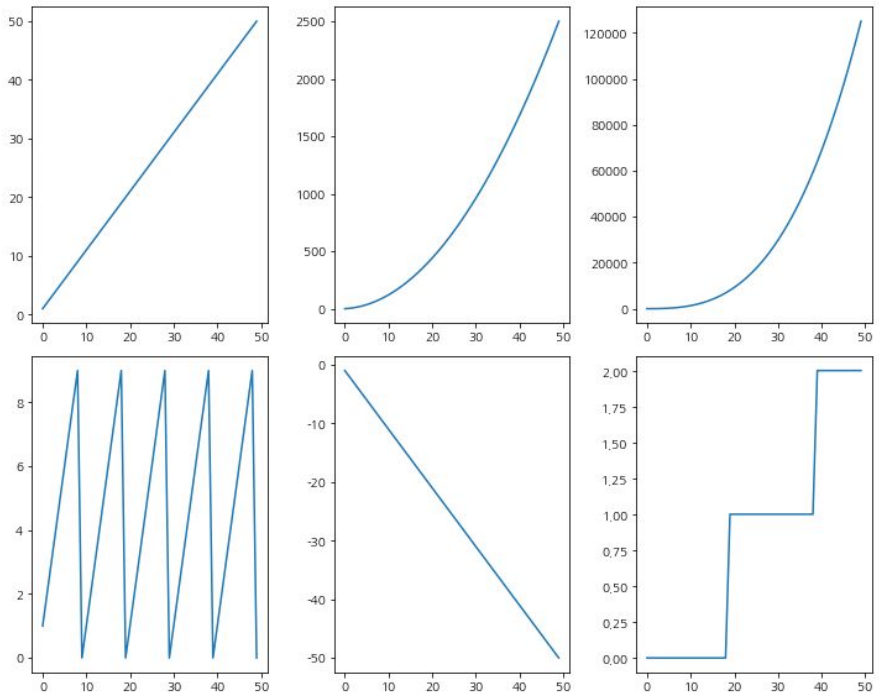



```
# rcParams를 이용해서 커스텀 스타일 설정
import matplotlib.pyplot as plt

plt.style.use('default')
plt.rcParams['figure.figsize'] = (6, 3)
plt.rcParams['font.size'] = 12
# plt.rcParams['figure.figsize'] = (4, 3)
# plt.rcParams['font.size'] = 14

plt.rcParams['lines.linewidth'] = 3
plt.rcParams['lines.linestyle'] = '-'

plt.rcParams['xtick.top'] = True
plt.rcParams['ytick.right'] = True
plt.rcParams['xtick.direction'] = 'in'
plt.rcParams['ytick.direction'] = 'in'

plt.plot([1, 2, 3, 4], [4, 6, 2, 7])
plt.show()

plt.style.use('default')
```

Matplotlib Visualization - Subplots

- plt.subplots(행 개수, 열 개수):
  하나의 화면 안에 여러 플롯을 넣을 수 있습니다

- fig, axes 를 plt.subplots()의 인자로 넣은 뒤
  axes[행,열].plot명령어 를 입력해 각 subplot에
  원하는 플롯을 넣을 수 있습니다

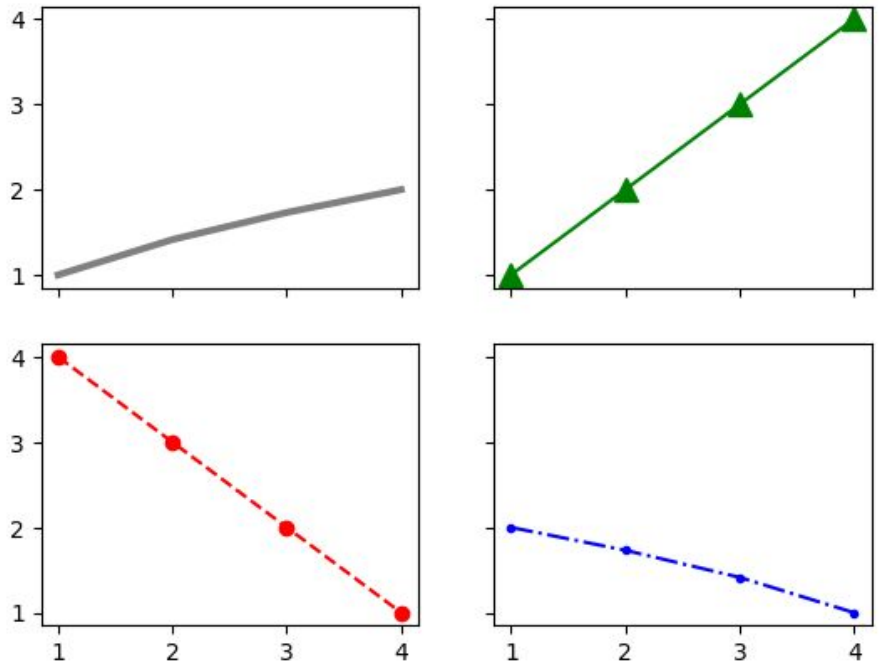


```
# data 생성
data = np.arange(1, 51)

# 밑 그림: 2행 3열
fig, axes = plt.subplots(2, 3)

axes[0, 0].plot(data)
axes[0, 1].plot(data * data)
axes[0, 2].plot(data ** 3)
axes[1, 0].plot(data % 10)
axes[1, 1].plot(-data)
axes[1, 2].plot(data // 20)

# axes

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

Matplotlib Visualization - Subplots

- 각 axes[행,열]에 대해서 plt에서 사용하는 함수들을 이용해서 다양한 스타일 조정이 가능합니다

- sharex=True, sharey=True로 설정함으로써 아래와 같이 중복된 축을 한번만 표시할 수 있습니다

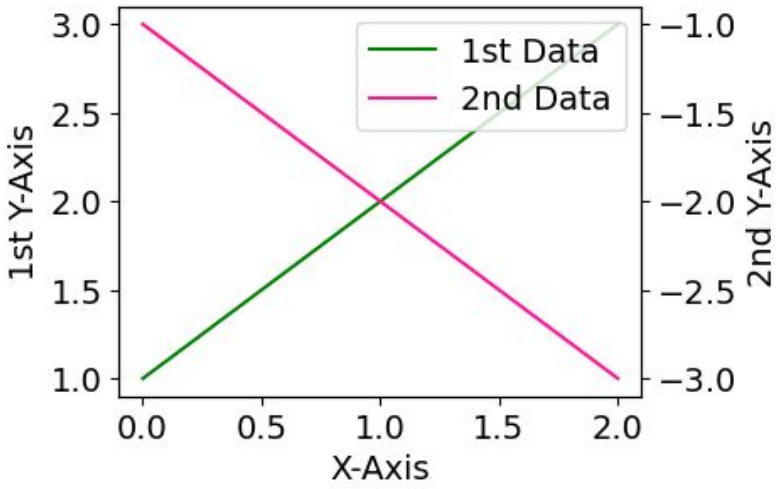



```
import matplotlib.pyplot as plt
import numpy as np

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()
```

Matplotlib Visualization - 이중 y축 표시하기

- 두 종류의 데이터를 동시에 하나의 그래프에 표시하기 위해 이중 축을 표시할 수 있습니다

- ax1.twinx()을 이용해서 ax1과 x축을 공유하는 새로운 Axes 객체인 ax2를 만듭니다



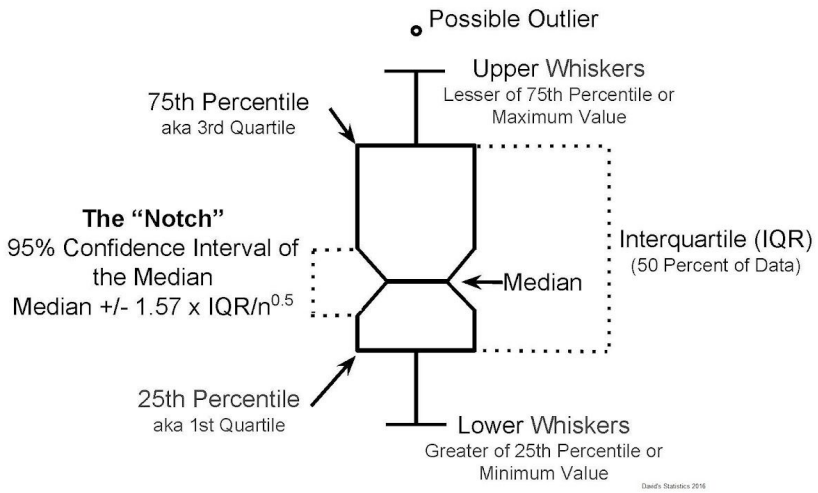

```
# 이중 y축 표시하기
import matplotlib.pyplot as plt
import numpy as np

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()
```

Matplotlib Visualization - Boxplot

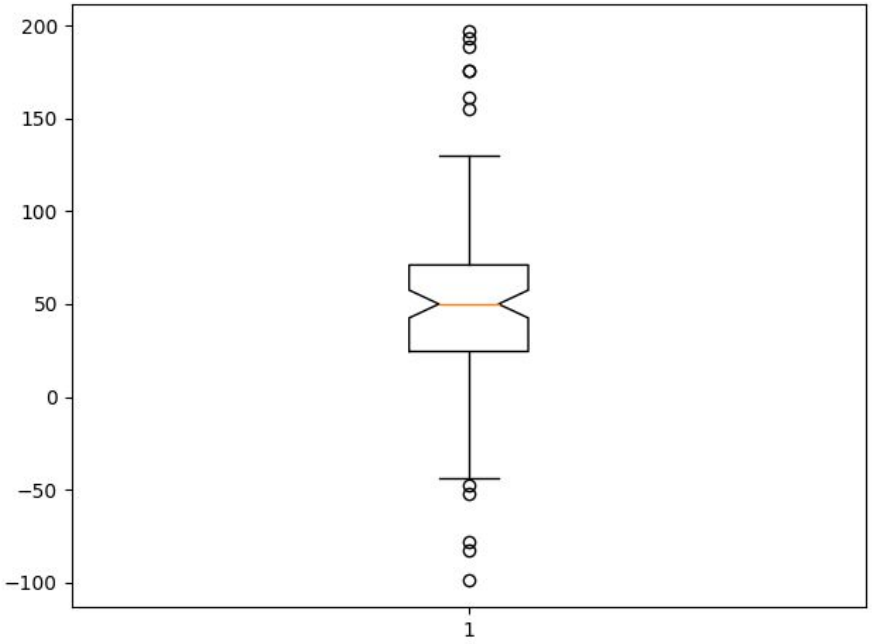
- plt.boxplot():
  박스 플롯 (Box plot) 은 수치 데이터의 분포를 표현합니다

- Maximum / minimum value:
  Q1, Q3로부터 1.5*IQR 내에서 최대/최소값

- notch=True: 중앙값 (Median)의 95% 신뢰 구간을 노치 형태로 표시합니다



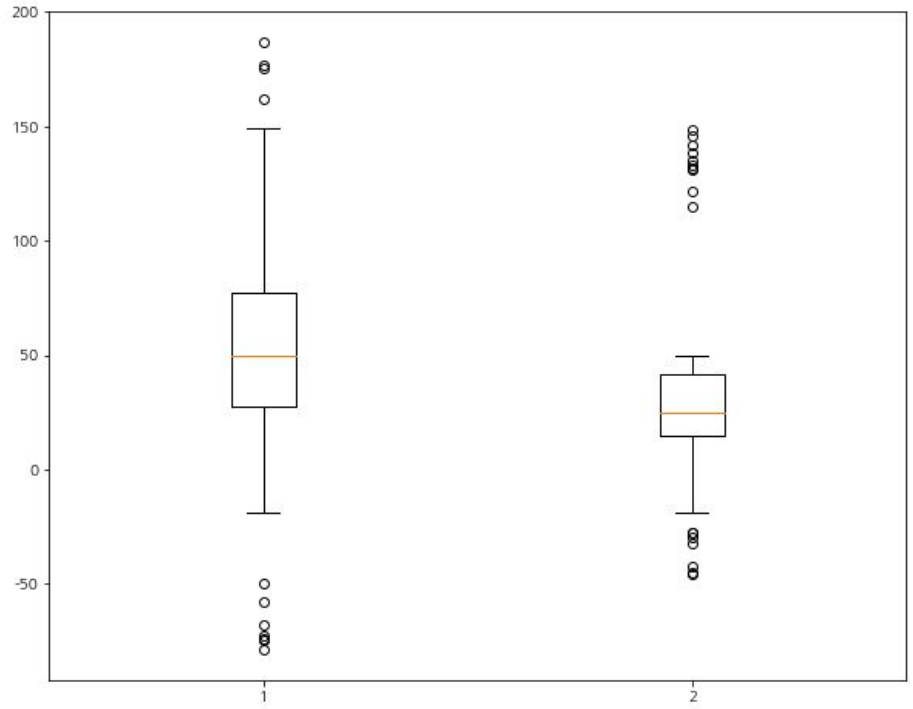

```
# box plot

# 샘플 데이터 생성
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))
# 기본 박스플롯 생성
plt.boxplot(data, notch=True)
plt.tight_layout()
plt.show()
```



Matplotlib Visualization - Boxplot

- plt.subplots()과 plt.boxplot()을 함께 사용하면 한 그림 안에 여러 boxplot을 그릴 수 있습니다

- box['whiskers'], box['median'] 등의 key 호출 후 item.get_ydata()를 통해 각 box의 whisker 범위, median, outlier값들을 조회할 수 있습니다

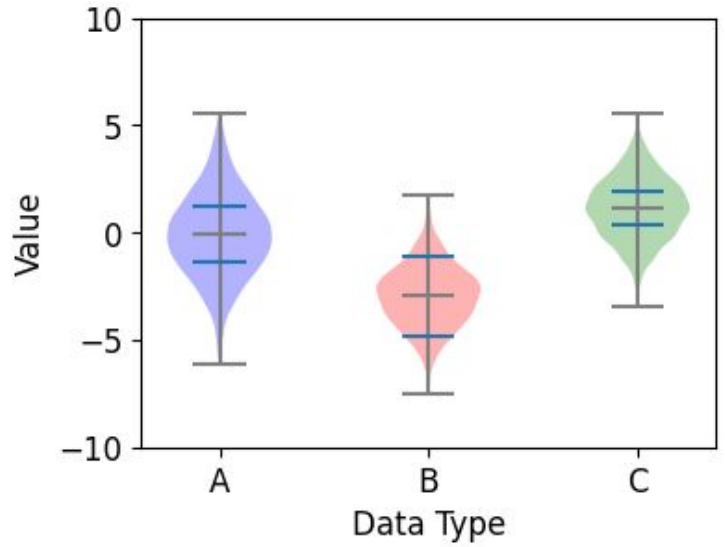



```
# 샘플 데이터 생성
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data_a = np.concatenate((spread, center, flier_high,
flier_low))

spread = np.random.rand(50) * 50
center = np.ones(25) * 25
flier_high = np.random.rand(10) * 50 + 100
flier_low = np.random.rand(10) * -50
data_b = np.concatenate((spread, center, flier_high,
flier_low))

fig, ax = plt.subplots()
box = ax.boxplot([data_a, data_b])
plt.show()

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)
```

Matplotlib Visualization - Violin plot

- plt.violinplot():
  boxplot에 kde 곡선을 같이 그린 형태로
  좀 더 세밀한 분포 확인이 가능합니다

- quantiles 옵션을 이용해 표시할 분위수를
  조절할 수 있습니다




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

violin = ax.violinplot([data_a, data_b, data_c],
                       showmeans=True,
                       showextrema=True,
                      #  showmedians=True,
                       quantiles=[[0.25, 0.75], [0.1, 0.9],
[0.3, 0.7]])
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')

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()
```

데이터 시각화 - **seaborn**

programmers

Visualization - seaborn

- Seaborn은 Python에서 통계 그래픽을 만들기 위한 라이브러리입니다.
  seaborn은 matplotlib을 기반으로 하며 pandas 데이터 구조와 밀접하게 통합됩니다.

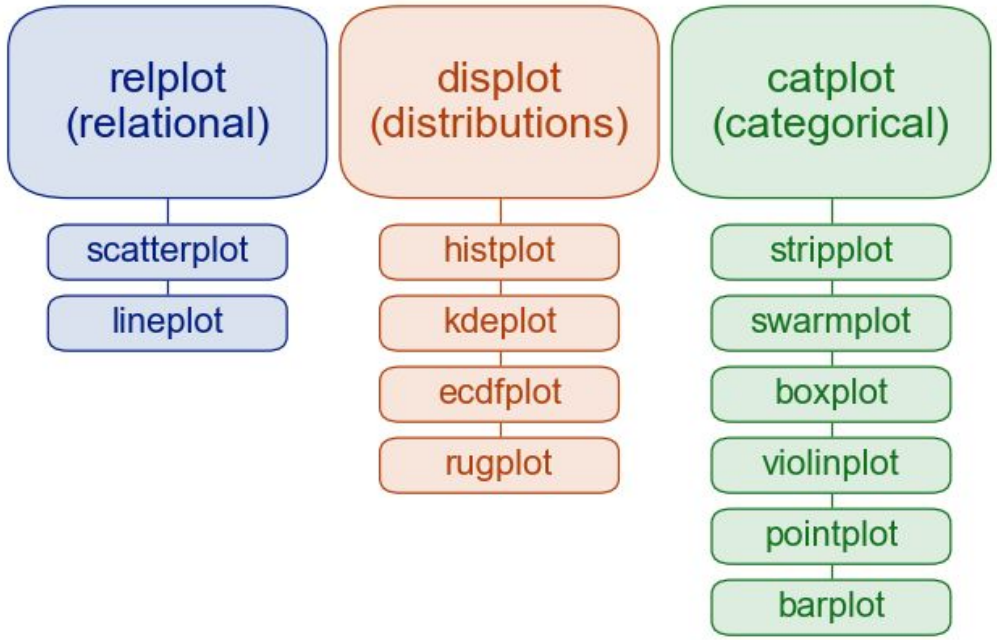
- Pandas DataFrame과 직관적으로 연계되기 때문에
  쉽고 빠르게 데이터 시각화가 가능합니다

- Seaborn에서 사용할 수 있는 플롯들은 크게
  다음 3가지 종류의 플롯군으로 나뉩니다.

  Relplot: 2개 이상의 변수간의 관계를 효과적으로
  나타내는 플롯

  Displot: 1개 이상의 변수 값의 분포를 효과적으로
  나타내는 플롯

  Catplot: 범주형 데이터의 분포를 효과적으로
  나타내는 플롯



Visualization - relplot

- Relplot은 데이터 세트의 변수가 서로 어떻게 연관되어 있는지,
  그리고 이러한 관계가 다른 변수에 어떻게 의존하는지
  이해하는데 도움을 주는 시각화를 가능케 합니다

- Relplot에서 사용 가능한 플롯은
  scatter plot과 line plot이 있습니다

- 색상, 크기 및 스타일을 사용하여
  최대 3개의 추가 변수를 매핑이 가능합니다

```
# scatter plot
sns.relplot(data=tips, x="total_bill", y="tip")
sns.scatterplot(data=tips, x="total_bill", y="tip")

# line plot
sns.relplot(data=dowjones, x="Date", y="Price", kind="line")
sns.lineplot(data=dowjones, x="Date", y="Price")
```

Visualization - scatter plot

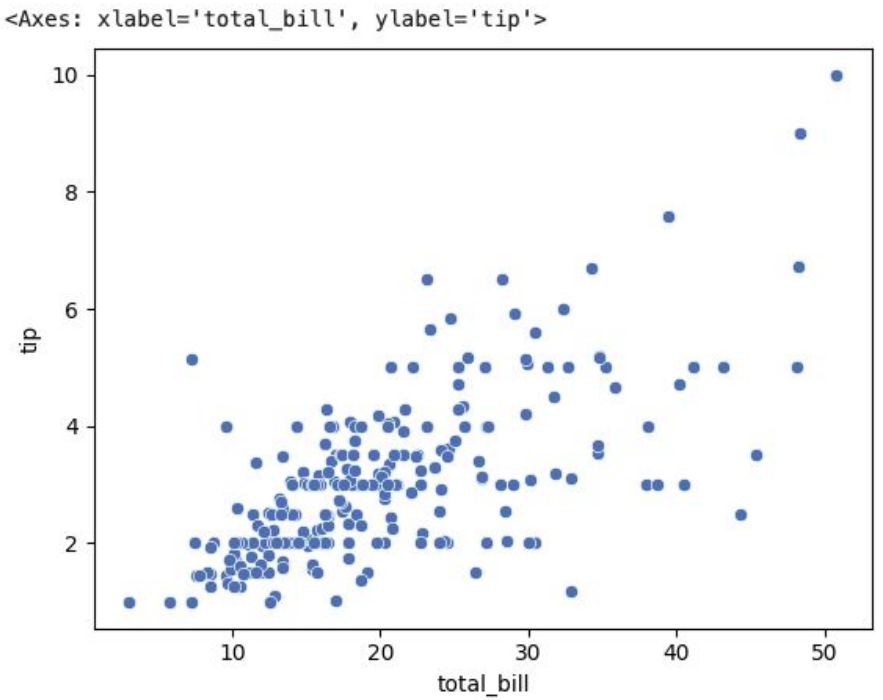
- 두 개의 연속형 변수 total_bill과 tip간의 관계를 scatter plot으로 표현할수 있습니다

```
sns.relplot(data=tips, x="total_bill", y="tip")
sns.scatterplot(data=tips, x="total_bill", y="tip")
```

```
tips = sns.load_dataset("tips")
tips
```

	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

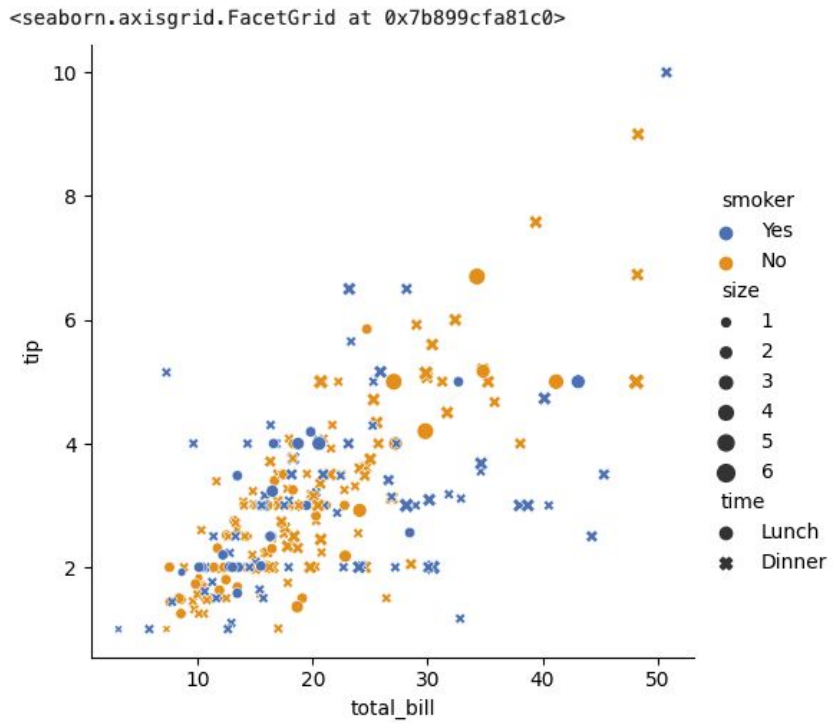
244 rows × 7 columns



Visualization - scatter plot

- hue, size, style을 통해서 데이터포인트가 가지는 categorical 변수값을 표현할 수 있습니다

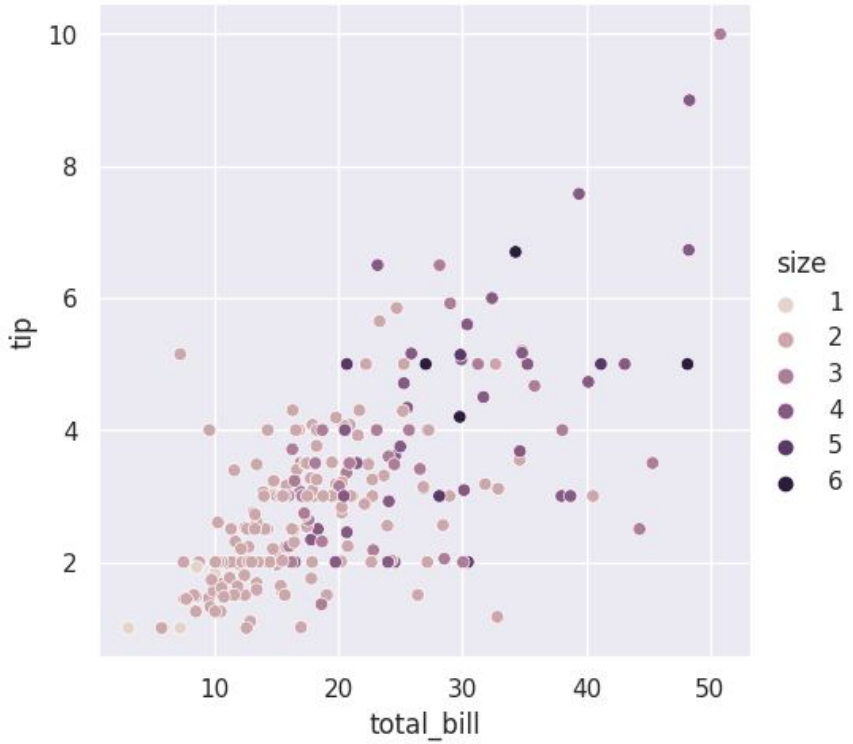
```
sns.relplot(
    data=tips,
    x="total_bill", y="tip", hue="smoker",
style="time",size="size"
)
```



Visualization - scatter plot

- hue가 가지는 값이 연속형 변수인 경우 색이 palette형태로 표현됩니다

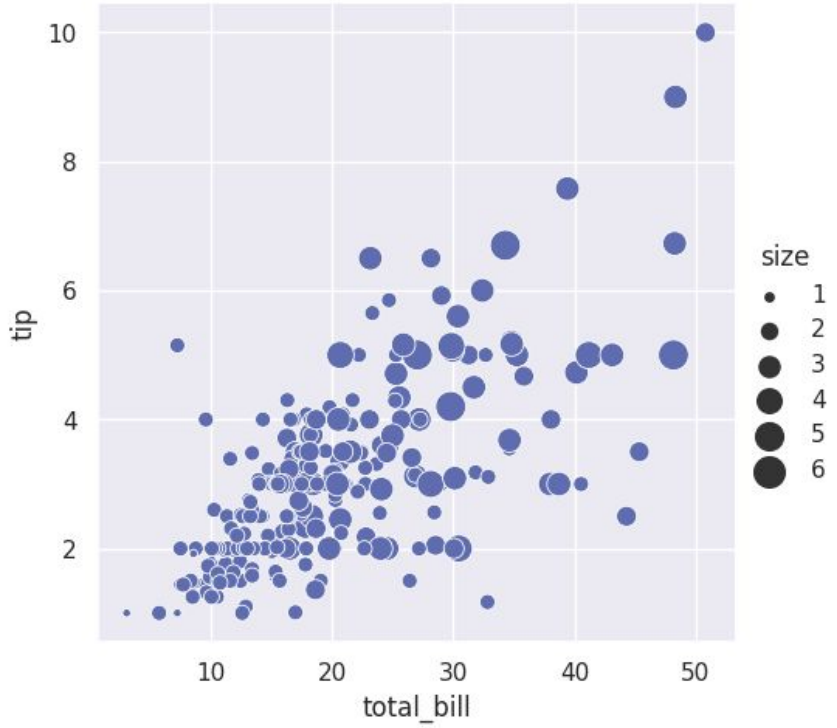
```
sns.relplot(
   data=tips, x="total_bill", y="tip", hue="size",
)
```



- size를 사용하는 경우 sizes=(a,b) 옵션을 통해 점들의 크기를 조정할 수 있습니다

```
sns.relplot(
   data=tips, x="total_bill", y="tip", size="size",
sizes=(15,200)
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7c923ed5fc10>
```



Visualization - line plot

- 시간과 같이 지속성을 가진 변수과 다른 변수의 관계를
  나타낼 때는 lineplot이 효과적입니다

```
sns.relplot(data=dowjones, x="Date", y="Price", kind="line")
sns.lineplot(data=dowjones, x="Date", y="Price")
```

```
dowjones = sns.load_dataset("dowjones")
dowjones
```

	Date	Price
0	1914-12-01	55.00
1	1915-01-01	56.55
2	1915-02-01	56.00
3	1915-03-01	58.30
4	1915-04-01	66.45
...	...	...
644	1968-08-01	883.72
645	1968-09-01	922.80
646	1968-10-01	955.47
647	1968-11-01	964.12
648	1968-12-01	965.39

649 rows × 2 columns

Visualization - line plot

- x축 값 하나에 따른 y값이 하나가 아닌 다양한 값을 갖고 있을 수도 있습니다.
- 이 때 기본적으로, lineplot은 여러 값들을 모아서 평균을 선으로, 95% 신뢰구간을 색으로 표현합니다.

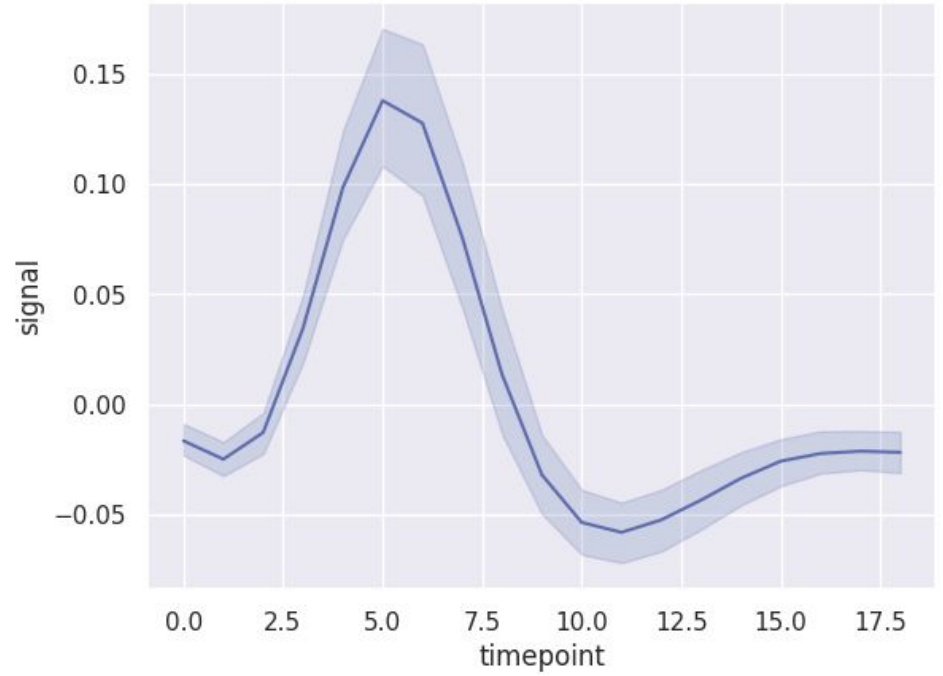
```
sns.lineplot(data=fmri, x="timepoint", y="signal")
```

```
fmri = sns.load_dataset("fmri")
fmri
```

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970
...	...	...	...	...	...
1059	s0	8	cue	frontal	0.018165
1060	s13	7	cue	frontal	-0.029130
1061	s12	7	cue	frontal	-0.004939
1062	s11	7	cue	frontal	-0.025367
1063	s0	0	cue	parietal	-0.006899

1064 rows × 5 columns

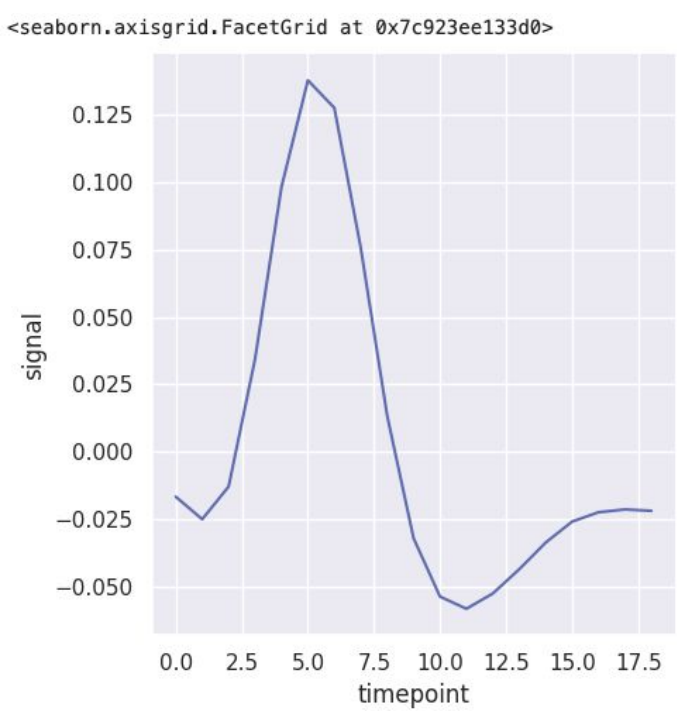
```
<Axes: xlabel='timepoint', ylabel='signal'>
```



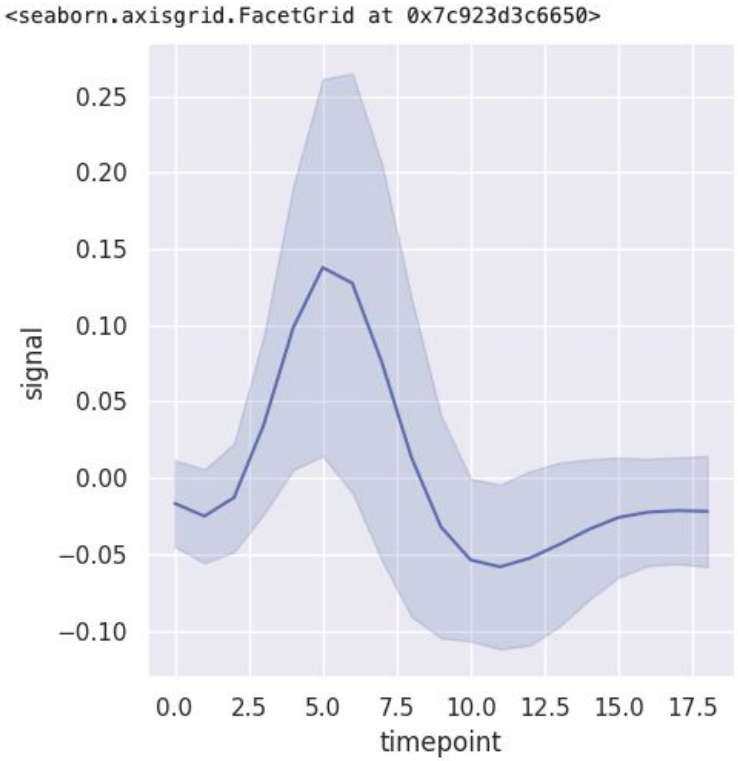
Visualization - line plot

- CI를 표시하지 않으려면 errorbar=None으로, 혹은 표준편차의 범위로 표시하고싶으면 errorbar='sd'로 옵션을 줍니다

```
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", errorbar=None,
)
```




```
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", errorbar='sd',
)
```



Visualization - line plot

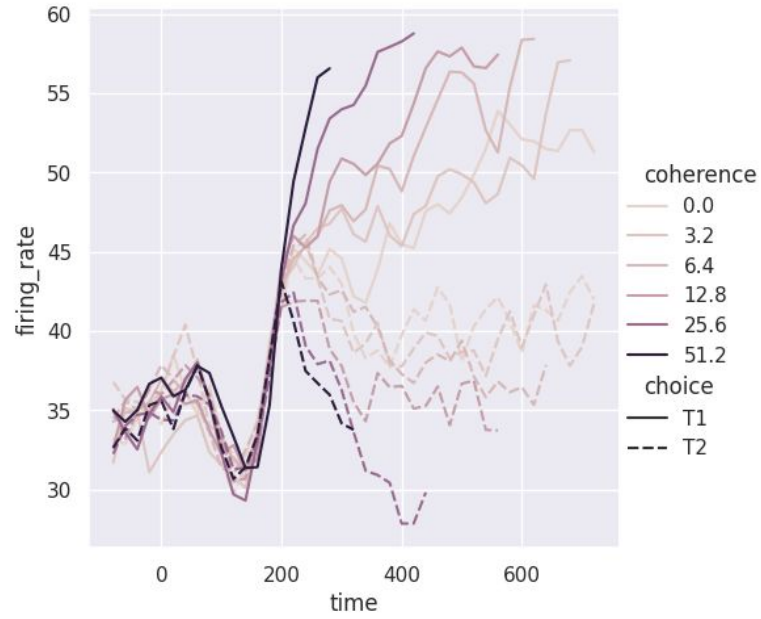
- hue를 통해서 연속형 변수 또한 구분하여 lineplot 을 그릴 수 있습니다

```
dots = sns.load_dataset("dots").query("align == 'dots'")
dots
```

	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487

```
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate",
    hue="coherence", style="choice",
)
```

```
<seaborn.axisgrid.FacetGrid at 0x7c923d0db730>
```

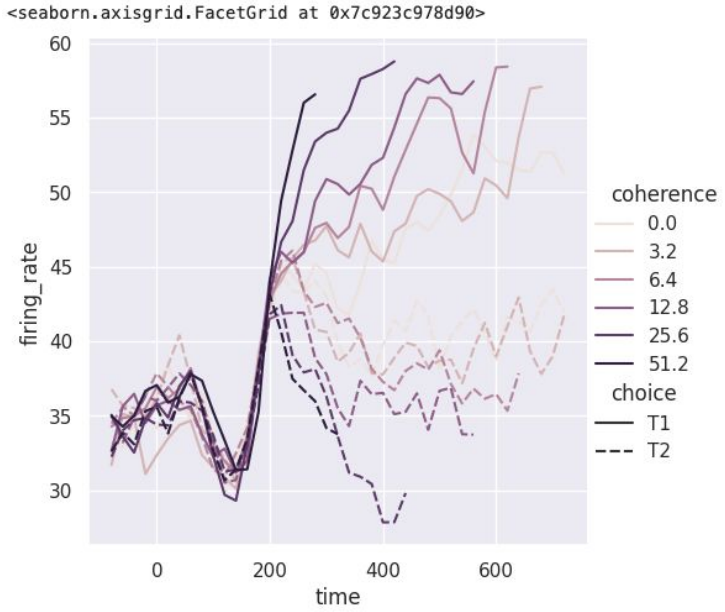


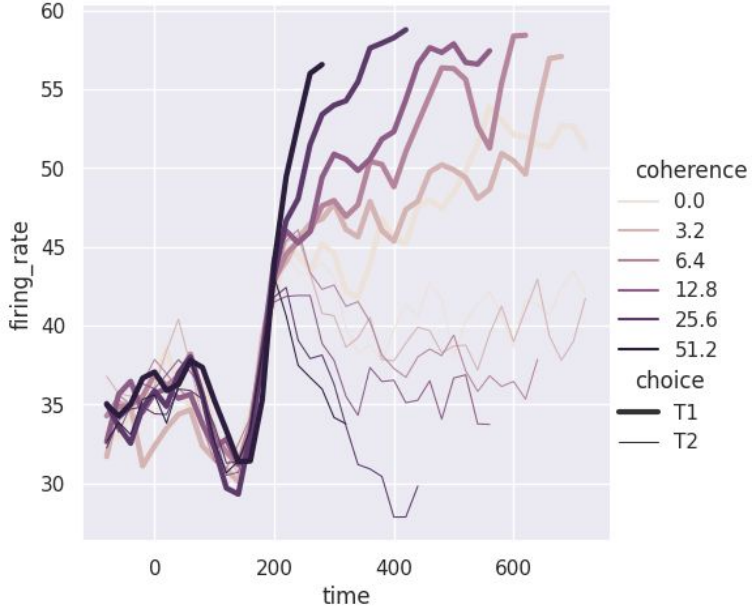
Visualization - line plot

- Palette를 이용하여 선의 색 차이를 조정할 수도 있습니다
- light 값이 커질수록 가장 밝은 색의 명도가 높아져 명도 차이가 커집니다
  n_colors에는 가질 수 있는 카테고리 개수를 넣어줍니다
  size를 이용해서 선의 굵기를 조절해줄 수도 있습니다

```
palette=sns.cubehelix_palette(light=.9, n_colors=6)
sns.relplot(x="time", y="firing_rate",
            hue="coherence", style="choice",
            palette=palette,
            kind="line", data=dots)
```

```
sns.relplot(x="time", y="firing_rate",
            hue="coherence", size="choice",
            palette=palette,
            kind="line", data=dots)
```



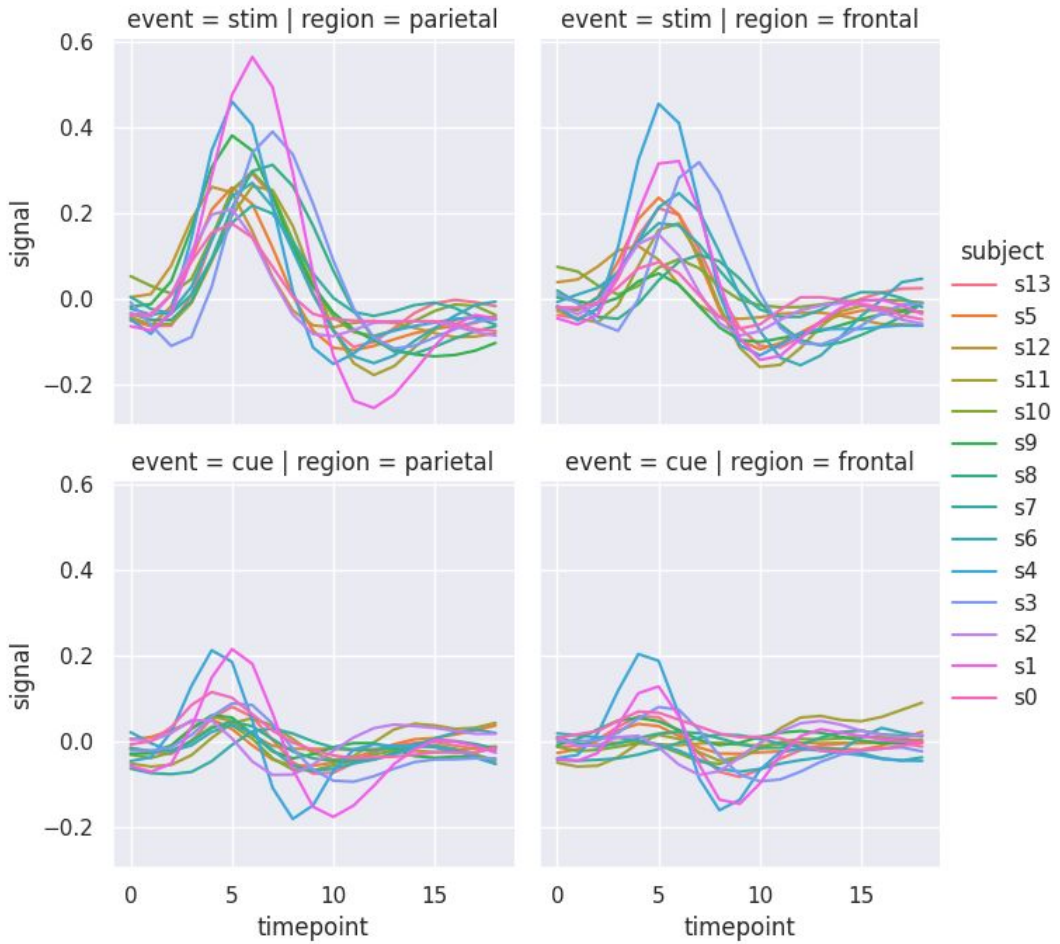


Visualization - facet

- 변수가 많아질수록 한 플롯 안에서 표현하다 보면 가시성이 떨어지고 효과적인 분석이 어려울 수 있습니다
- facet, 즉 여러 플롯으로 나눠 분석을 하는 것이 효과적일 수 있습니다
- col이나 row를 이용해 플롯들을 나누고 height을 통해서 플롯의 크기를 조절할 수 있습니다

```
sns.relplot(x="total_bill", y="tip", hue="smoker",
            col="time", data=tips)
```


```
sns.relplot(x="timepoint", y="signal", hue="subject",
            col="region", row="event", height=3.5,
            kind="line", estimator=None, data=fmri)
```

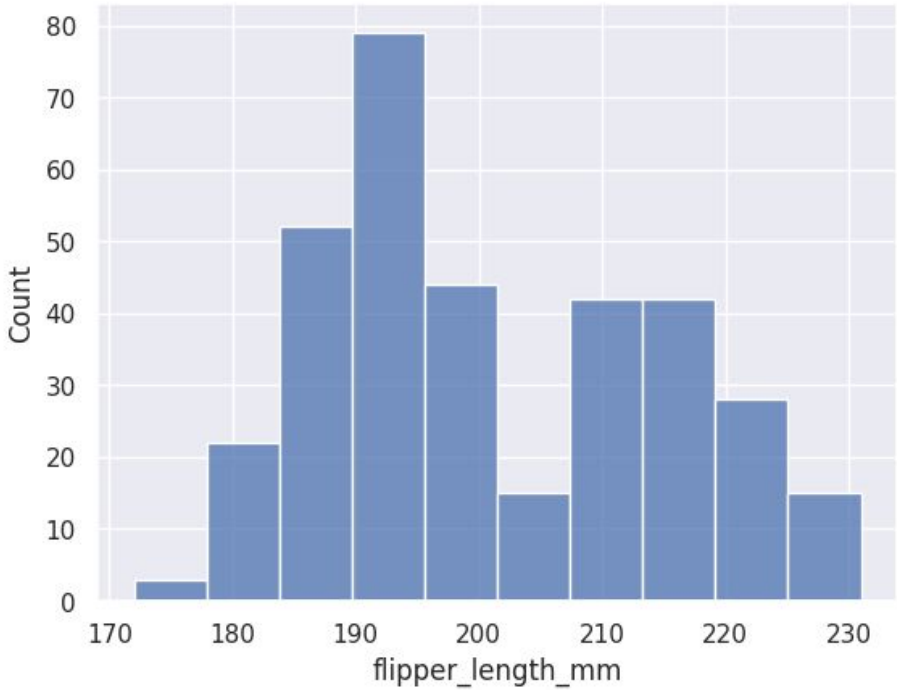


Visualization - displot

- 변수 하나 혹은 두 개의 분포를 나타낼 때 distplot에 포함된 여러 플롯들을 사용하면 해당 변수 값이 가지는 분포 및 범위를 효과적으로 확인할 수 있습니다

- 기본적으로 displot을 사용하면 히스토그램을 그려줍니다

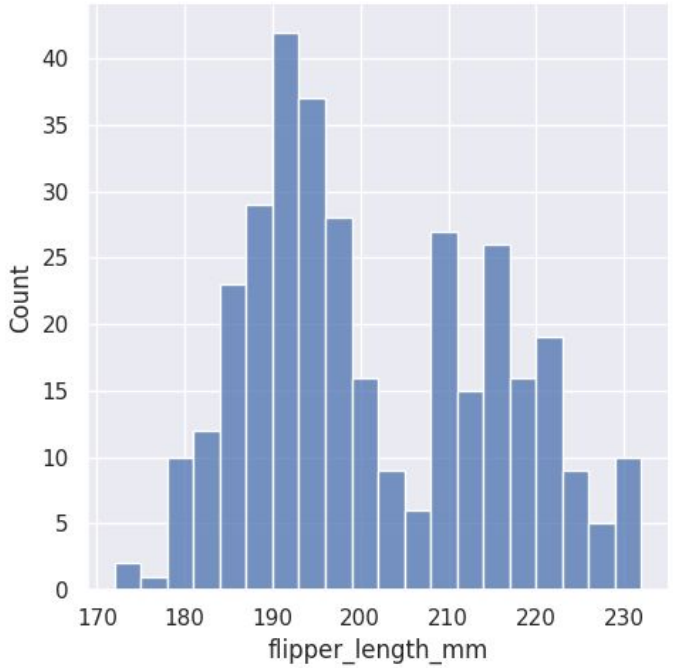
```
sns.histplot(penguins, x="flipper_length_mm")
sns.displot(penguins, x="flipper_length_mm")
```



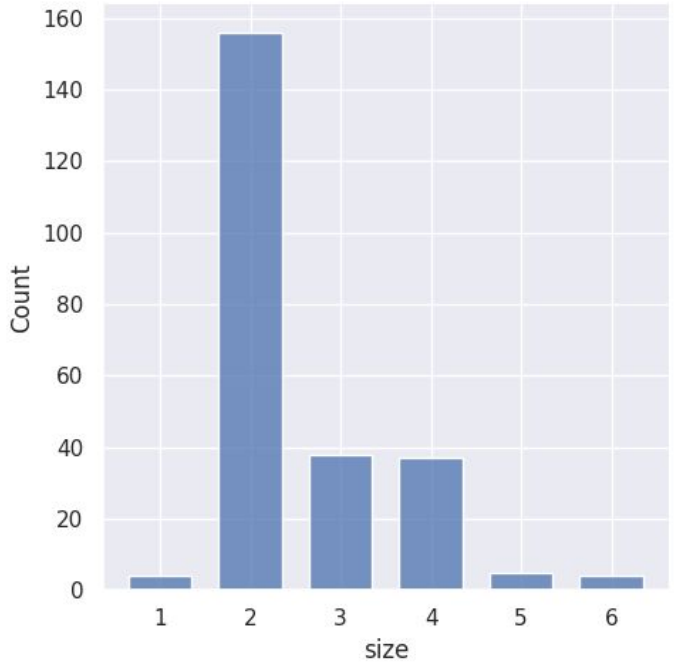
Visualization - histogram

- histogram의 막대 너비는 binwidth 혹은 bins를 사용해 조절해줄 수 있습니다
- 변수가 이산형 변수인 경우, discrete=True 옵션을 통해
  x축 값 중앙에 막대가 위치하도록 할 수 있습니다
- 또한, shrink 옵션을 통해 막대 간 공간 조절이 가능합니다

```
sns.displot(penguins, x="flipper_length_mm", binwidth=3)
sns.displot(penguins, x="flipper_length_mm", bins=20)
```



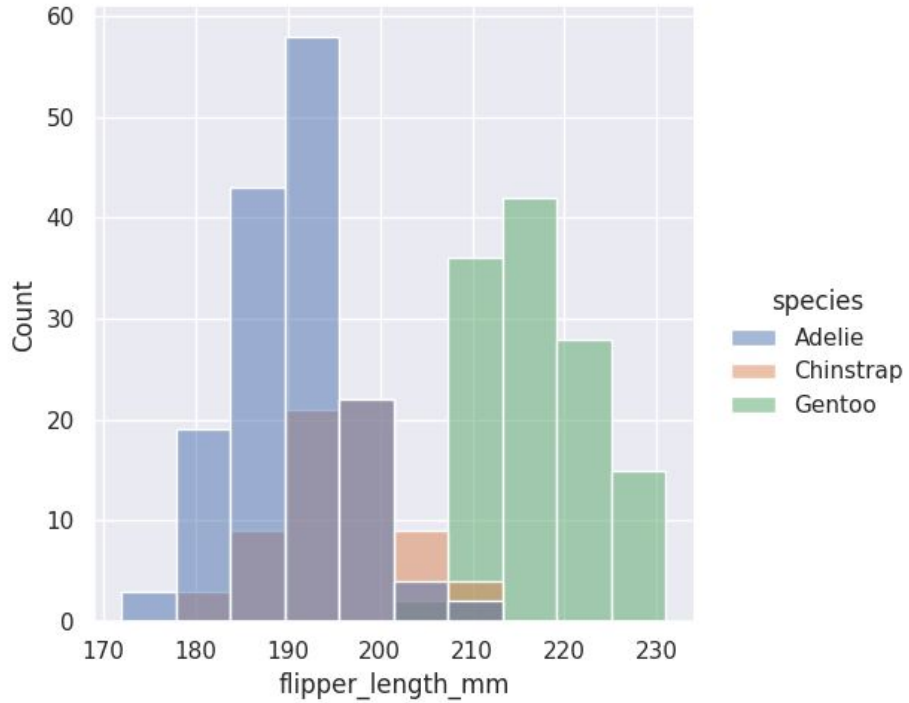

```
sns.displot(tips, x="size", discrete=True, shrink=.7)
```



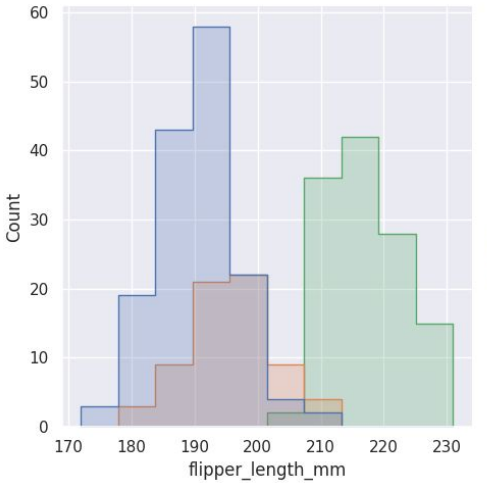
Visualization - histogram

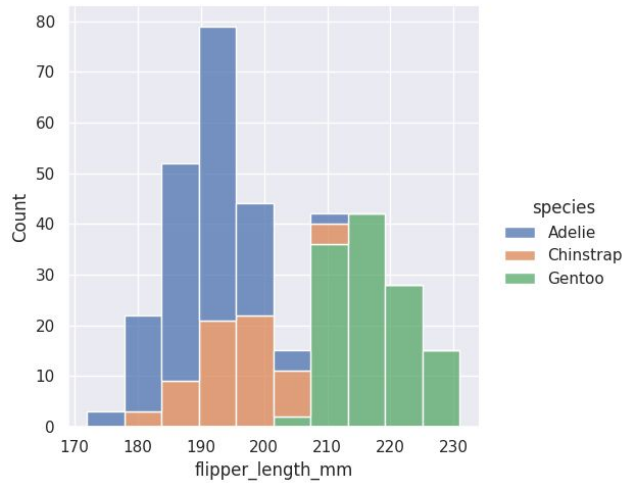
- displot의 경우 hue 옵션을 통해 추가 변수를 조건으로 한 분포를 시각화 할 수 있습니다
- 막대 간 겹쳐지는 부분의 경우 element=step 옵션을 사용하거나 multiple=stack 옵션을 사용 가능합니다 단, 변수가 카테고리형인 경우 multiple=dodge 또한 사용 가능합니다

```
sns.displot(penguins, x="flipper_length_mm", hue="species")
```

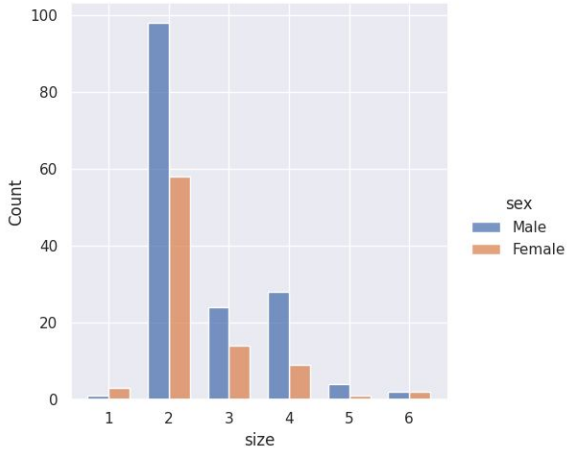



```
sns.displot(penguins, x="flipper_length_mm", hue="species",
element="step")
sns.displot(penguins, x="flipper_length_mm", hue="species",
multiple="stack")
```





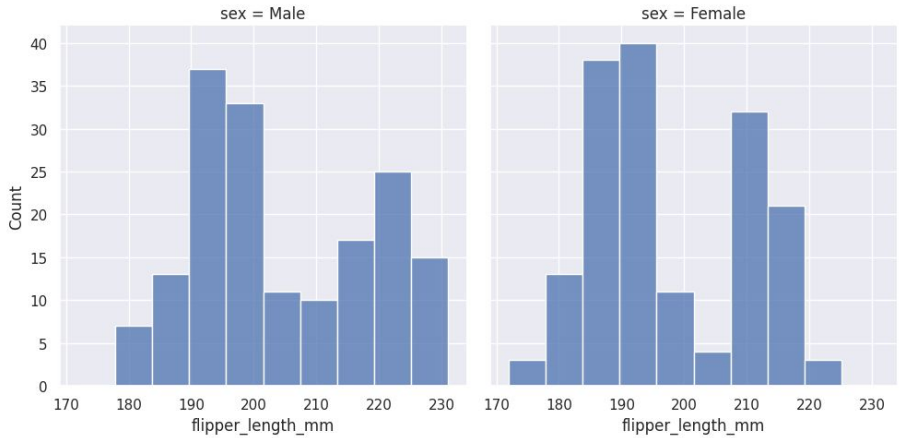

```
sns.displot(tips, x="size", discrete=True, shrink=.7,
hue='sex', multiple='dodge')
```



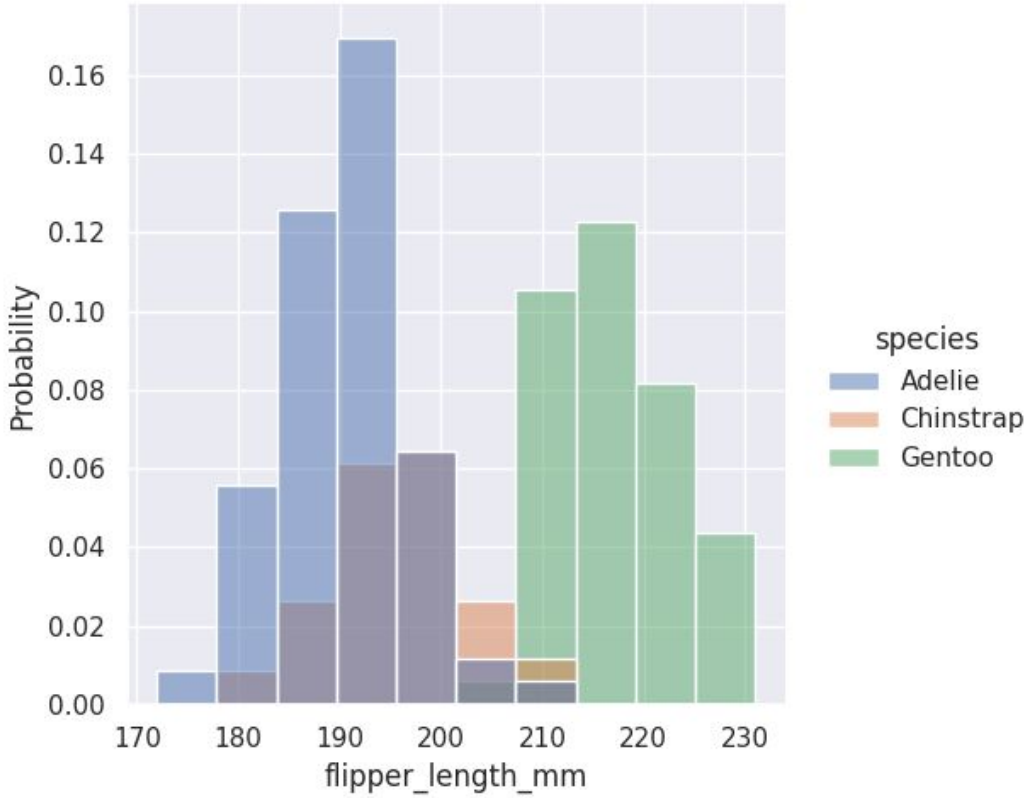
Visualization - histogram

- displot 또한 col이나 row를 이용해 변수에 따른 여러 플롯으로 나눠 그릴 수 있습니다
- stat='probability' 옵션을 통해 모든 막대의 높이의 합이 1이 되게 만들 수 있으며, 이를 빈도의 비율값으로 해석할 수 있습니다.

```
sns.displot(penguins, x="flipper_length_mm", col="sex")
```




```
sns.displot(penguins, x="flipper_length_mm", hue="species",
stat="probability")
```

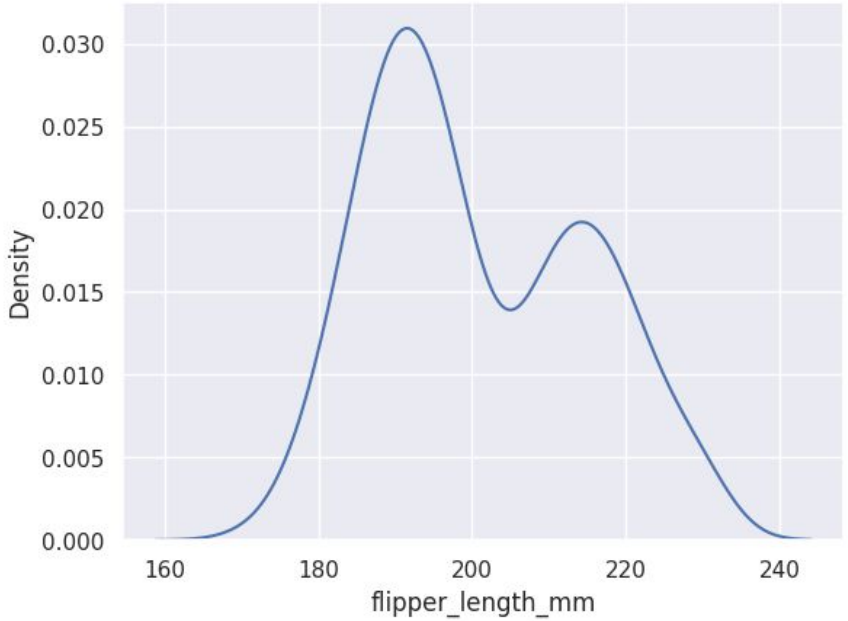


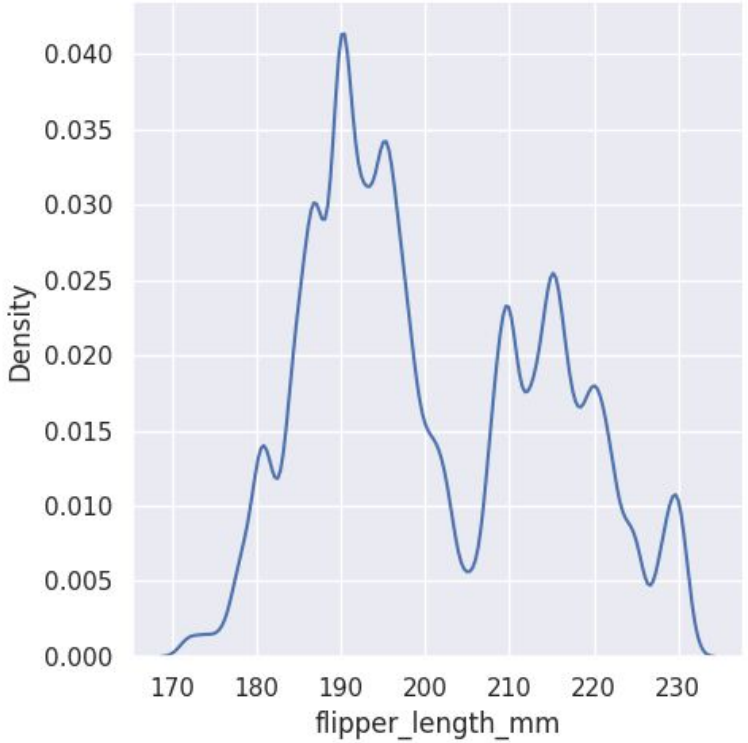
Visualization - kdeplot

- kde plot을 이용해 histogram과 다른 부드러운 곡선으로 이어진 분포 시각화가 가능합니다
- bw_adjust를 이용해 곡선을 세밀하게 혹은 더 부드럽게 표현이 가능합니다

```
sns.displot(penguins, x="flipper_length_mm", kind="kde",
bw_adjust=.25)
```

```
sns.kdeplot(penguins, x="flipper_length_mm")
```



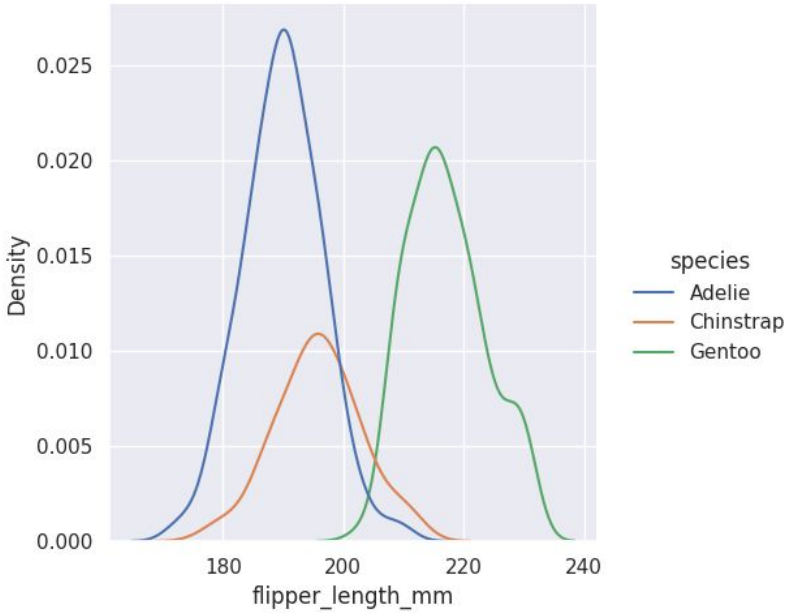


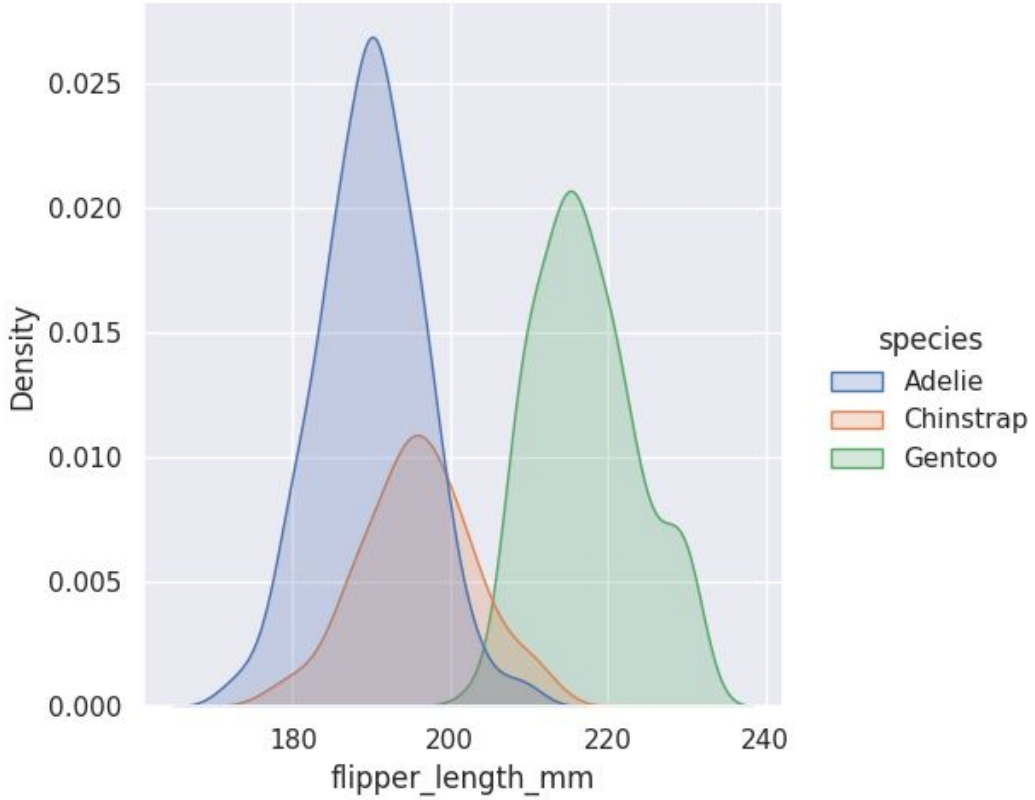
Visualization - kdeplot

- kde plot 또한 hue를 이용해서 변수간 분포를 더 세밀하게 시각화할 수 있습니다
- 또한, fill=True 옵션을 이용해 곡선 아래 색칠된 형태로 표현이 가능합니다

```
sns.displot(penguins, x="flipper_length_mm", hue="species",
kind="kde", fill=True)
```

```
sns.displot(penguins, x="flipper_length_mm", hue="species",
kind="kde")
```

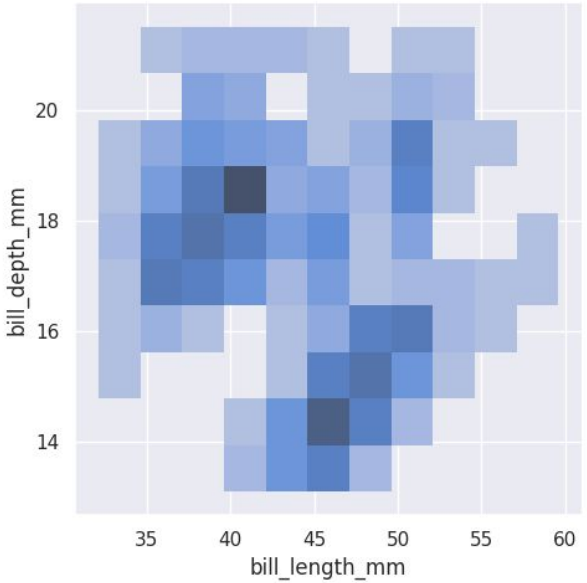




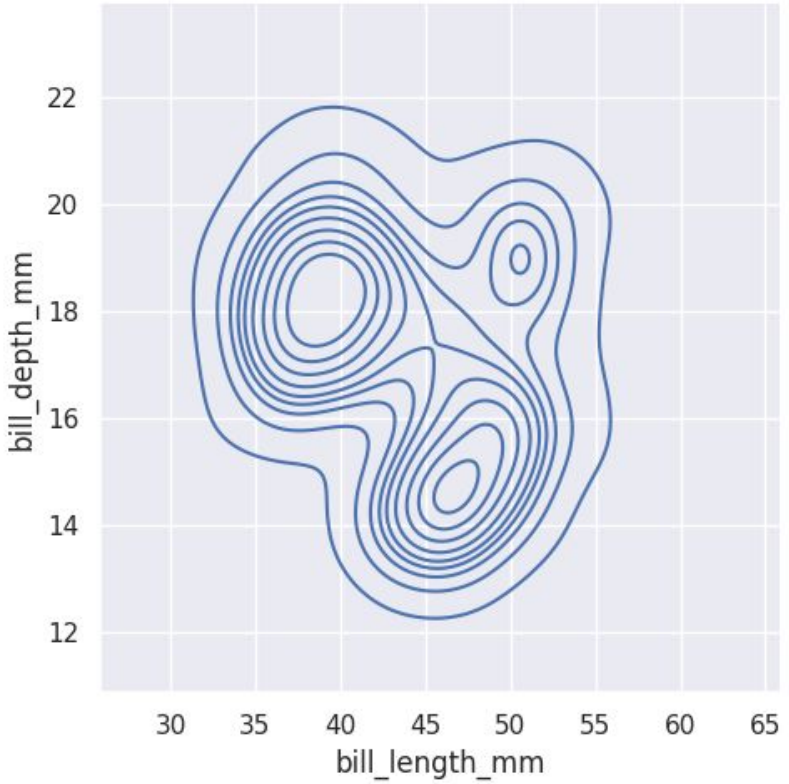
Visualization - heatmap / contour

- hue 옵션을 사용하는 대신, y에 2번째 변수를 넣어 여러 변수의 분포를 표현하는 시각화 또한 가능합니다
- 기본적으로 displot은 heatmap을 그려줍니다
- kde plot에서 y 변수를 추가해주면 contour plot을 그려줍니다

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm")
```



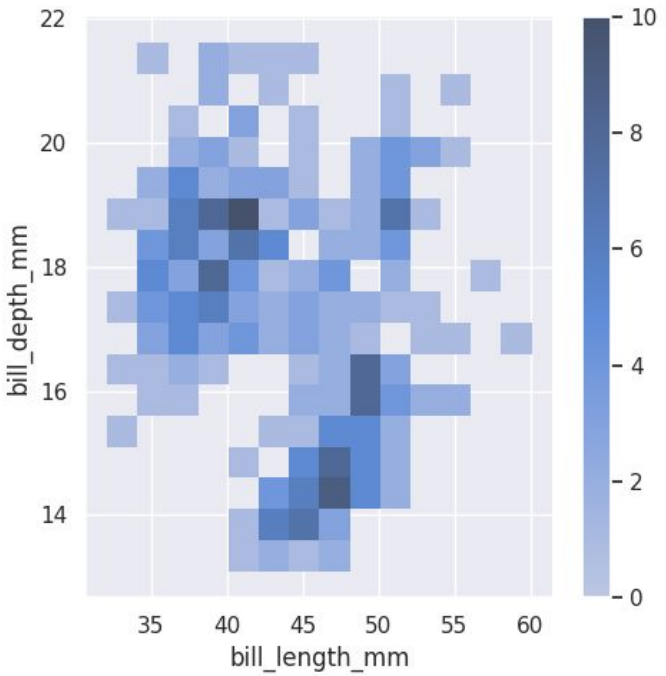

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
kind="kde")
```



Visualization - heatmap / contour

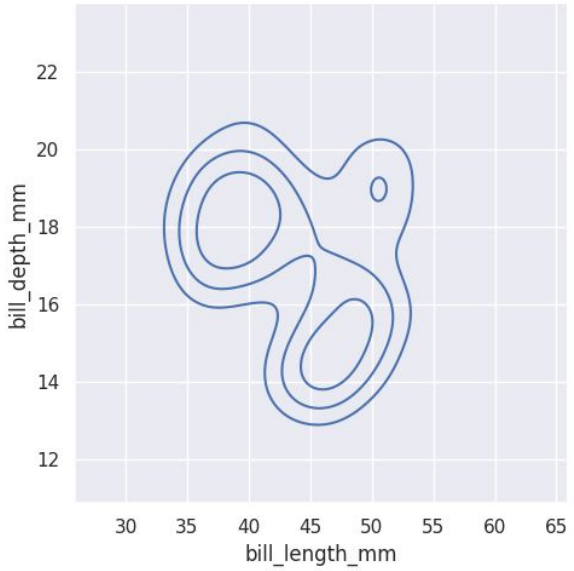
- heatmap의 경우 (x축,y축)의 binwidth를 조절하여
  각 칸의 크기를 조절해줄 수 있습니다.
- 또한, cbar=True 로 설정시 색깔에 따라 갖는 빈도 값을 확인할 수 있습니다
- contour plot의 경우 thresh와 levels를 이용해 선의 간격을 조절할 수 있습니다

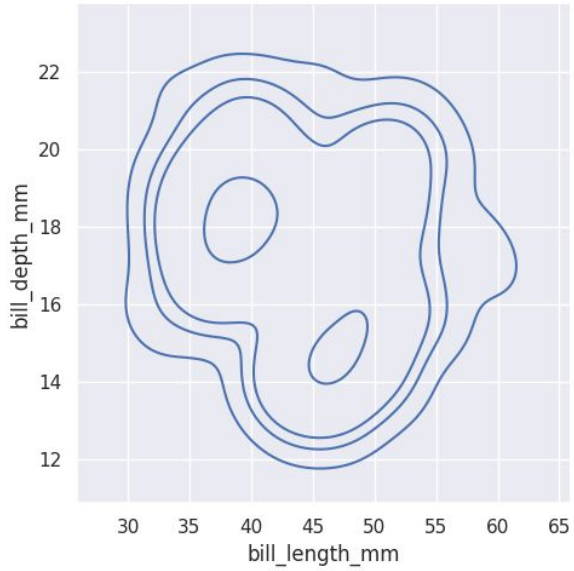
```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
binwidth=(2, .5), cbar=True)
```




```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
kind="kde", thresh=.2, levels=4)
```

```
sns.displot(penguins, x="bill_length_mm", y="bill_depth_mm",
kind="kde", levels=[.01, .05, .1, .8])
```

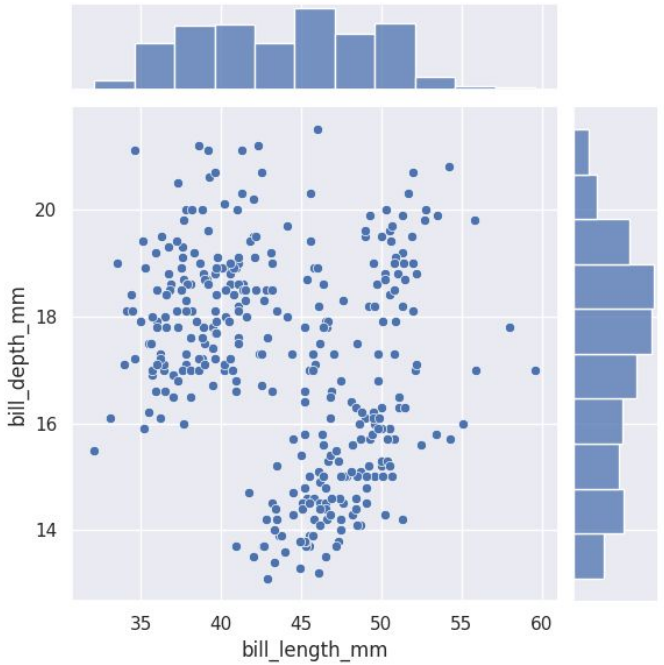




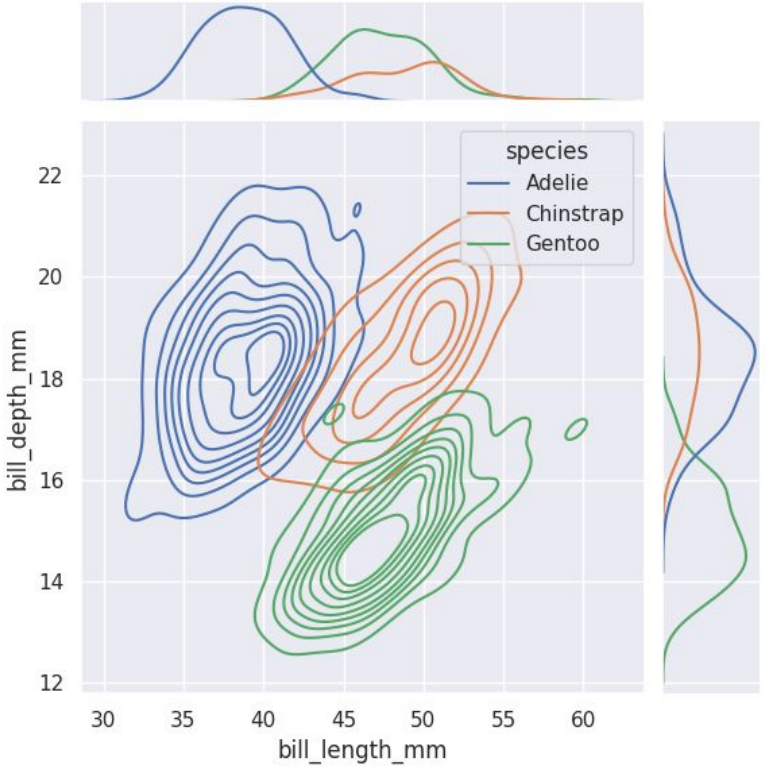
Visualization - joint plot

- joint plot의 경우 2개의 변수 간 분포와 각 변수의 분포를 동시에 보여줍니다
- 기본적으로는 scatter plot과 histogram, kind='kde' 인 경우 contour plot과 kde plot을 그려줍니다

```
sns.jointplot(data=penguins, x="bill_length_mm",
y="bill_depth_mm")
```




```
sns.jointplot(
    data=penguins,
    x="bill_length_mm", y="bill_depth_mm", hue="species",
    kind="kde"
)
```

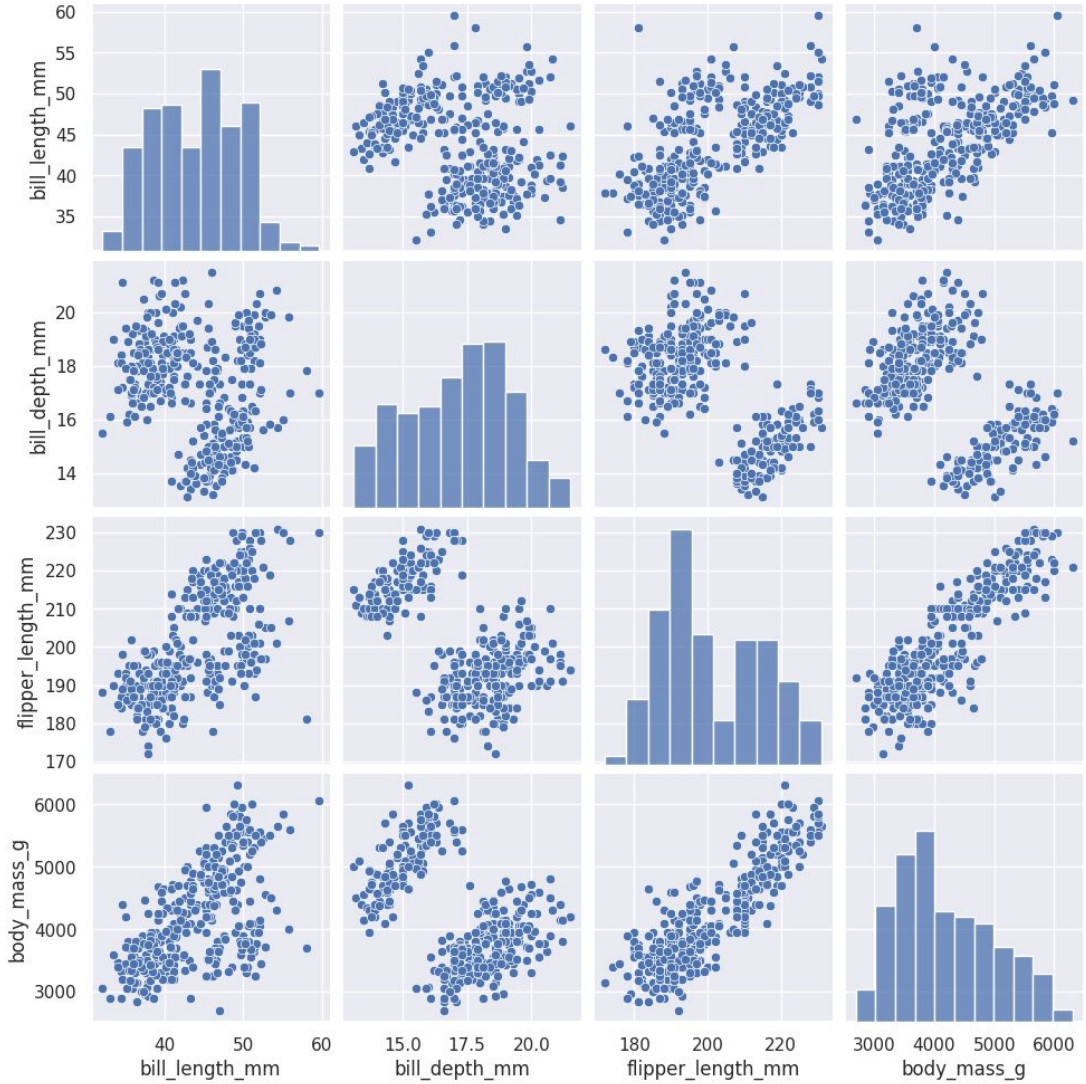


Visualization - pair plot

- Pair plot을 이용해 각 변수들간의 pairwise 관계를 시각화할 수 있습니다

- 특히, 고려해야할 변수들이 많은데 변수간 상관관계 파악이 필요한 경우 유용합니다

```
sns.pairplot(penguins)
```

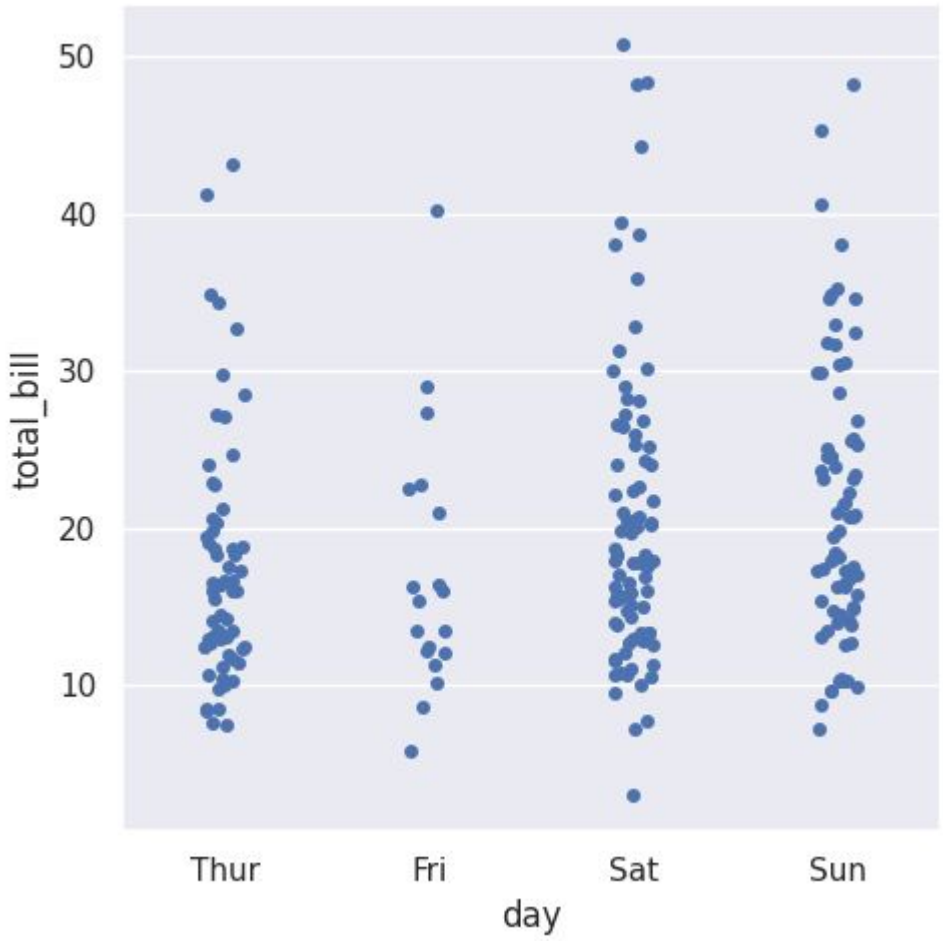


Visualization - catplot

- Relplot에서 다루고자 하는 변수가 categorical(혹은 이산형) 데이터인 경우 scatter plot이나 line plot 대신 catplot을 사용하여
  좀 더 효과적인 시각화가 가능합니다

- catplot에서 사용 가능한 플롯들은 크게 다음 3가지로 나뉩니다:
  - 범주형 산점도: strip plot, swarm plot
  - 범주형 분포도: box plot, violin plot
  - 범주형 추정도표: bar plot, point plot

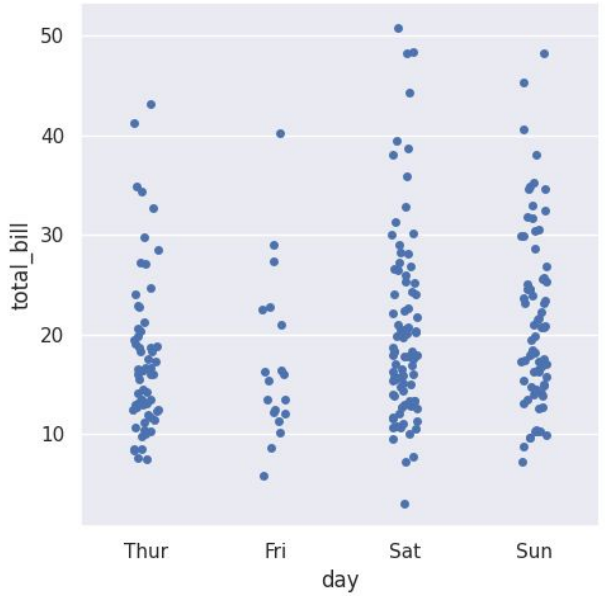
```
sns.catplot(data=tips, x="day", y="total_bill")
```



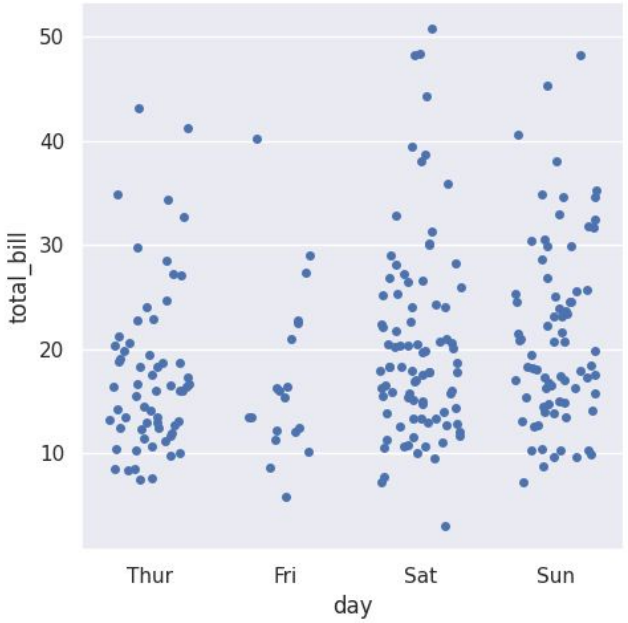
Visualization - strip plot

- catplot의 기본 플롯은 strip plot이며,
  한쪽 변수가 categorical 데이터인 산점도라고
  생각하면 됩니다
- 한 범주에 속하는 모든 점이 범주형 변수에
  해당하는
  축을 따라 동일한 위치에 있게 됩니다
- jitter 옵션을 통해서 좌우로 퍼진 정도를
  조절하거나 비활성화합니다

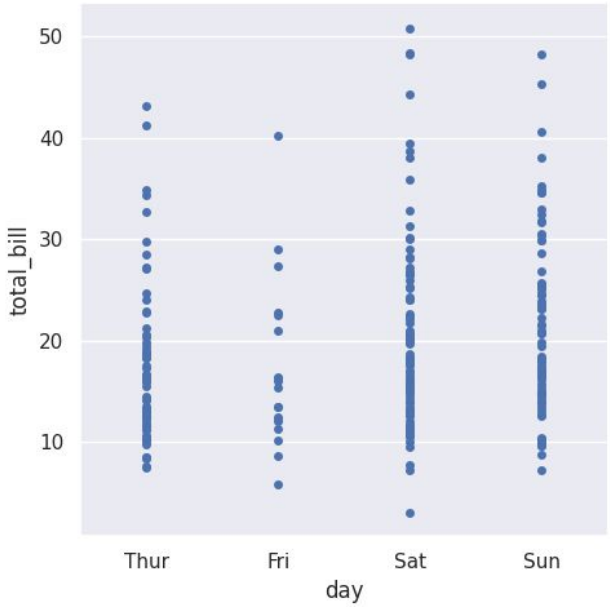
```
sns.catplot(data=tips, x="day", y="total_bill")
```




```
sns.catplot(data=tips, x="day", y="total_bill", jitter=.3)
```



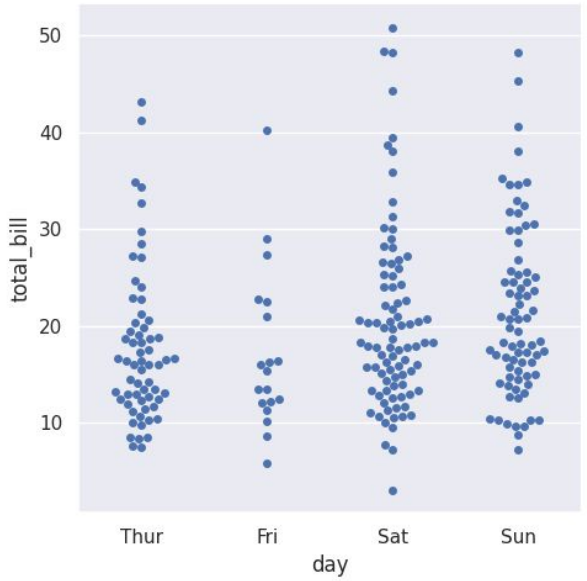

```
sns.catplot(data=tips, x="day", y="total_bill", jitter=False)
```



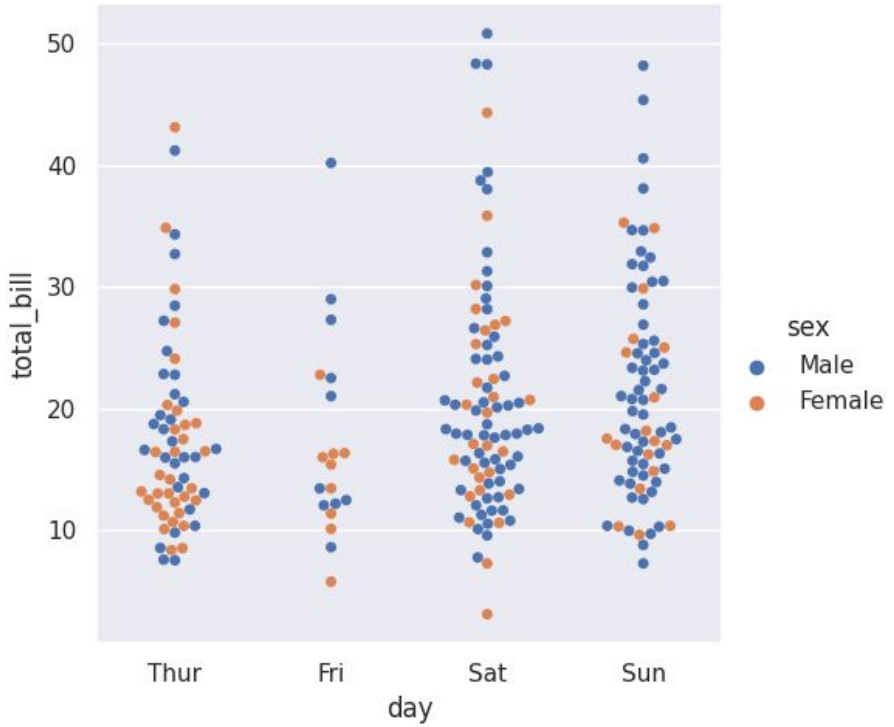
Visualization - swarm plot

- swarm plot을 활용하면 점들의 중첩을 방지하는 알고리즘을 사용하여 시각화가 가능합니다
- 상대적으로 데이터 수가 적은 경우에만 잘 작동하지만 관측치 분포를 더 잘 표현할 수 있습니다
- 관계형 도표와 마찬가지로 hue를 사용하여 범주형 도표에 다른 차원을 추가할 수 있습니다 (size나 style은 지원되지 않습니다)

```
sns.swarmplot(data=tips, x="day", y="total_bill")
sns.catplot(data=tips, x="day", y="total_bill", kind="swarm")
```




```
sns.catplot(data=tips, x="day", y="total_bill", hue="sex",
kind="swarm")
```



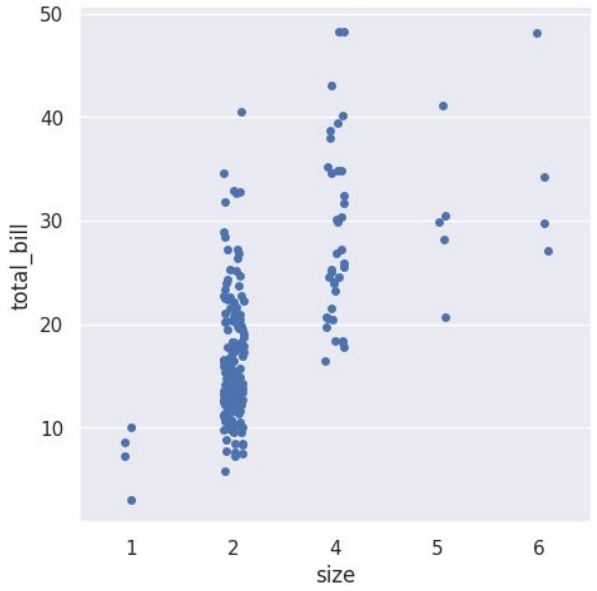
Visualization - strip plot

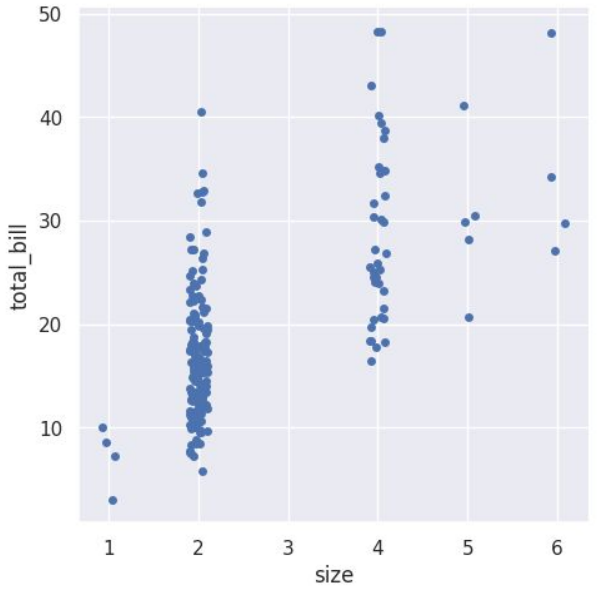
- 데이터에 pandas Categorical 데이터 유형이 있는 경우 범주의 기본 순서를 여기에서 설정할 수 있습니다.
- 데이터 유형이 Categorical이 아닌 경우에도 적용 가능하지만 Categorical 데이터로 취급되어 문제가 발생하는 경우가 있습니다.
- 이 때, native_scale을 사용하면 x축 데이터의 분포를 보존할 수 있습니다

```
tips['day'].dtype
```

```
CategoricalDtype(categories=['Thur', 'Fri', 'Sat', 'Sun'], ordered=False)
```

```
sns.catplot(data=tips.loc[tips['size']!=3], x="size",
y="total_bill")
sns.catplot(data=tips.loc[tips['size']!=3], x="size",
y="total_bill", native_scale=True)
```





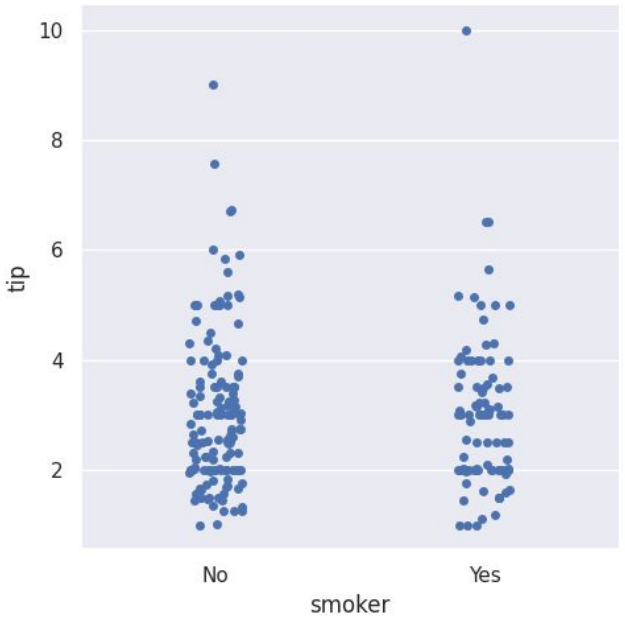
Visualization - strip plot

- order 옵션을 사용해서 Categorical 변수의 순서를 바꿔줄 수도 있습니다
- x와 y에 들어가는 변수 이름에 따라 축을 가로로 변경 가능합니다

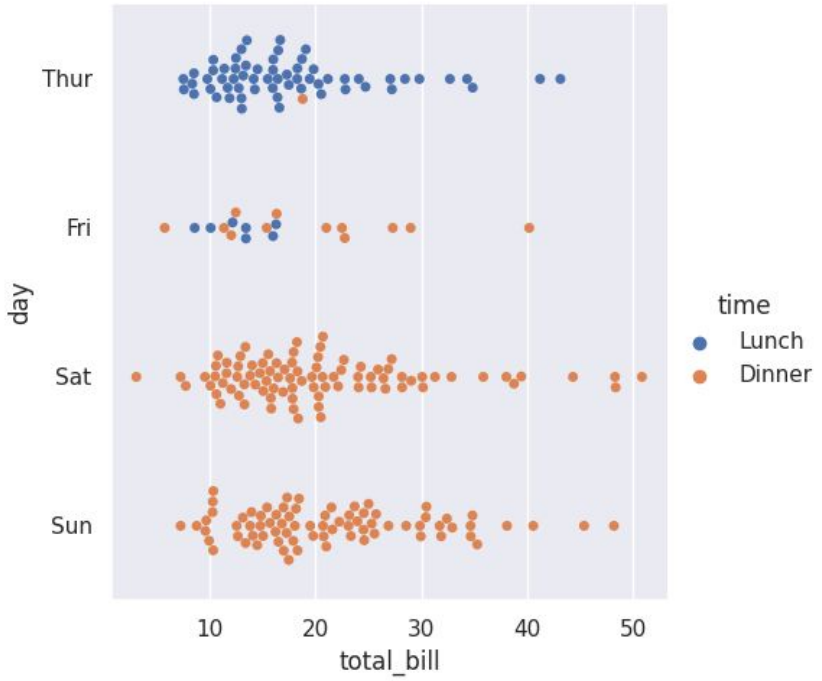
```
tips['smoker'].dtypes
```

```
CategoricalDtype(categories=['Yes', 'No'], ordered=False)
```

```
sns.catplot(data=tips, x="smoker", y="tip", order=["No",
"Yes"])
```



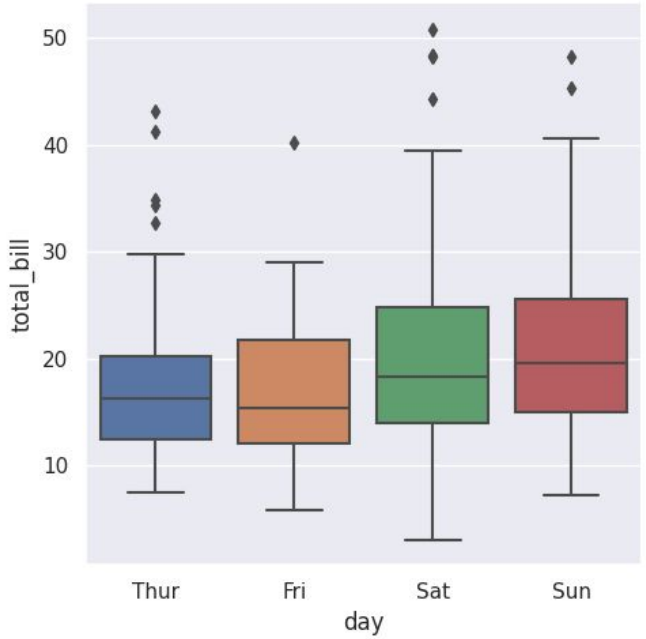

```
sns.catplot(data=tips, x="total_bill", y="day", hue="time",
kind="swarm")
```



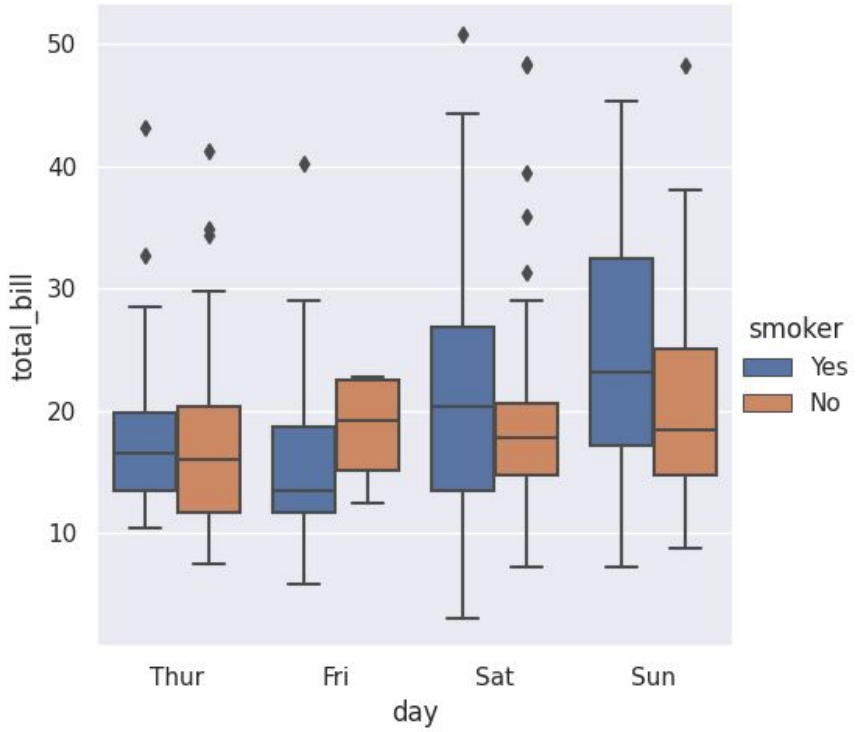
Visualization - boxplot

- 데이터 세트의 크기가 커짐에 따라 범주형 산점도는 각 범주 내의 값 분포에 대해 제공할 수 있는 정보가 제한됩니다
- 이 때 boxplot을 사용할 수 있습니다
- hue를 추가 가능하며 각 수준에 대한 상자는 더 좁아지고 범주형 축을 따라 이동됩니다

```
sns.boxplot(data=tips, x="day", y="total_bill")
sns.catplot(data=tips, x="day", y="total_bill", kind="box")
```




```
sns.catplot(data=tips, x="day", y="total_bill", hue="smoker",
kind="box")
```



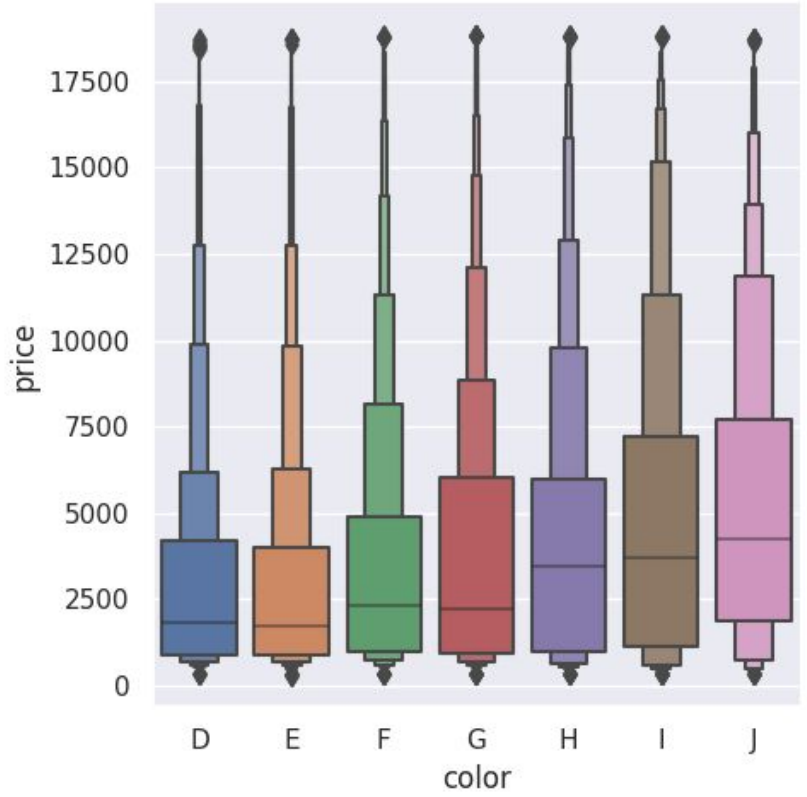
Visualization - boxenplot

- boxen plot을 이용하면 분위수를 좀 더 자세하게 표시할 수 있습니다

```
sns.boxenplot(data=diamonds.sort_values("color"), x="color",
y="price")
sns.catplot(
    data=diamonds.sort_values("color"),
    x="color", y="price", kind="boxen",
    )
```

```
diamonds = sns.load_dataset("diamonds")
```

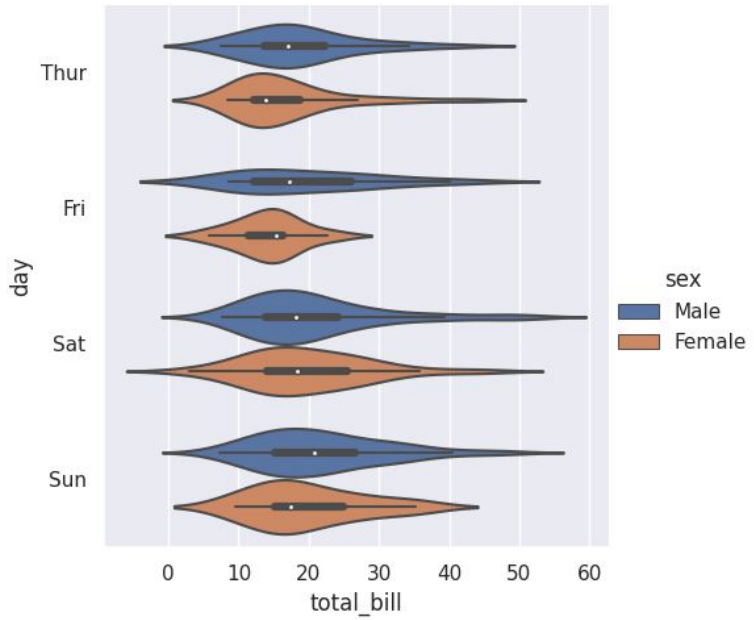
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75



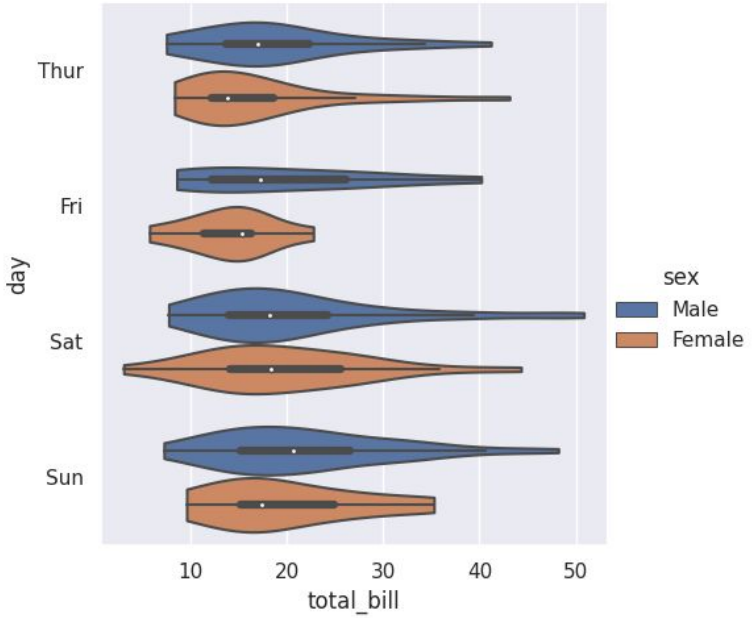
Visualization - violinplot

- violin plot은 box plot과 kde 곡선을 결합한 형태입니다
- Boxplot에서 쓰인 IQR과 whisker또한 표현됩니다
- bw와 cut을 이용해서 kde 곡선을 조절할 수 있습니다

```
sns.violinplot(
   data=tips, x="total_bill", y="day", hue="sex",
)
sns.catplot(
   data=tips, x="total_bill", y="day", hue="sex",
kind="violin",
)
```




```
sns.catplot(
   data=tips, x="total_bill", y="day", hue="sex",
   kind="violin", bw_adjust=.5, cut=0,
)
```

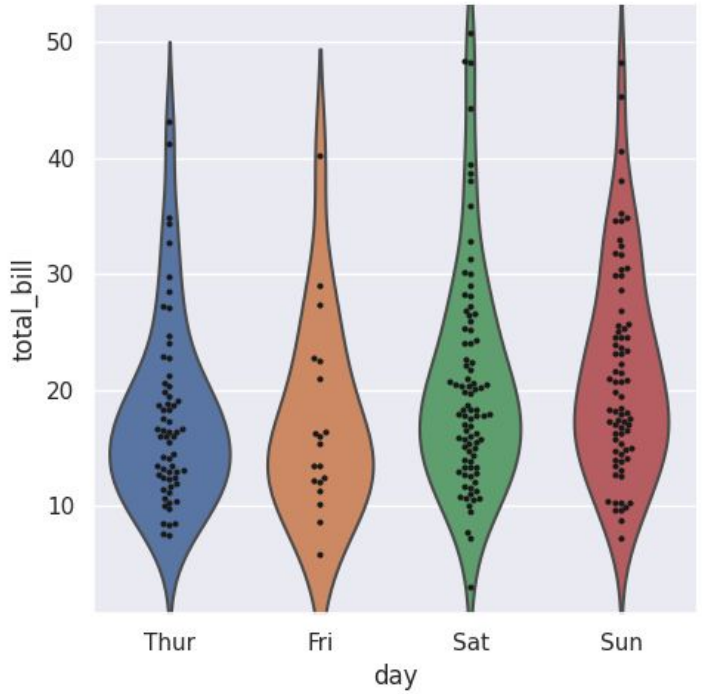


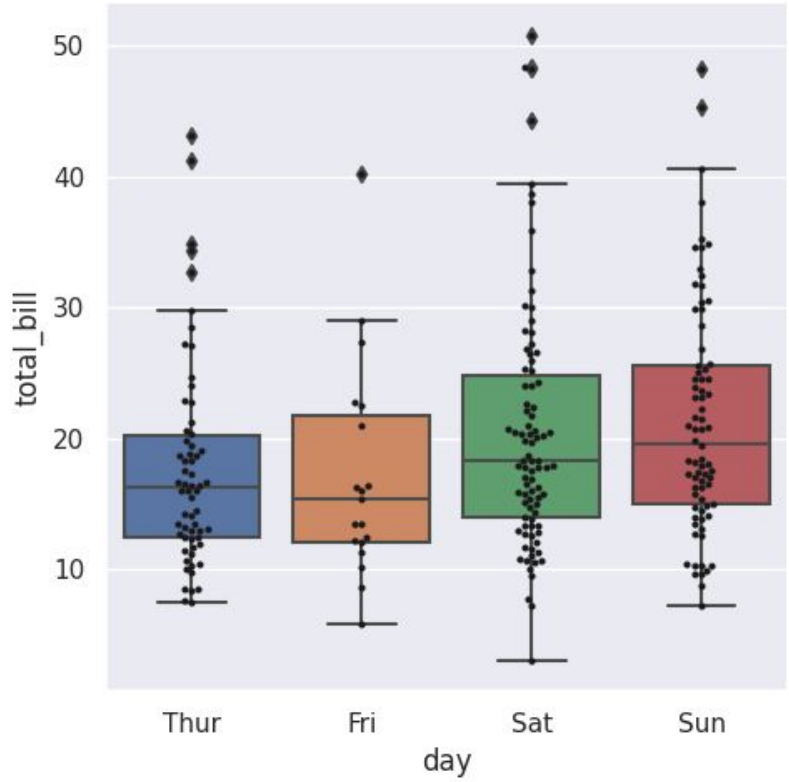
Visualization - violinplot

- box plot이나 violin plot을 swarm plot과 결합하는 것도 가능합니다
- swarmplot의 ax에 box plot이나 violin plot을 넣어줍니다
- 이 때, violin plot에서 inner=None을 통해 내부의 boxplot을 제거해줍니다

```
g = sns.catplot(data=tips, x="day", y="total_bill",
kind="violin", inner=None)
sns.swarmplot(data=tips, x="day", y="total_bill", color="k",
size=3, ax=g.ax)
```

```
g = sns.catplot(data=tips, x="day", y="total_bill", kind="box")
sns.swarmplot(data=tips, x="day", y="total_bill", color="k",
size=3, ax=g.ax)
```

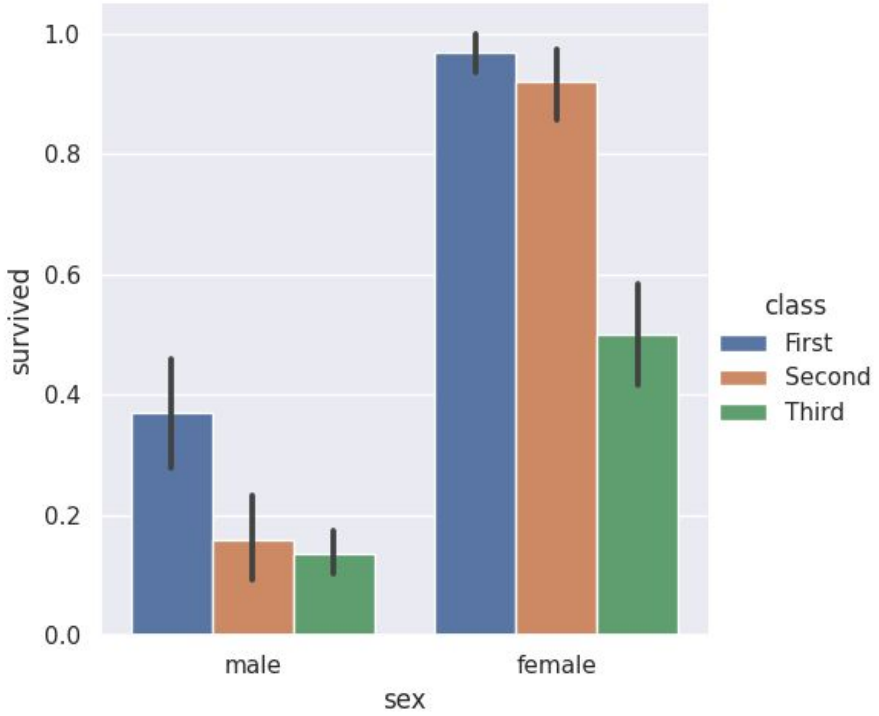




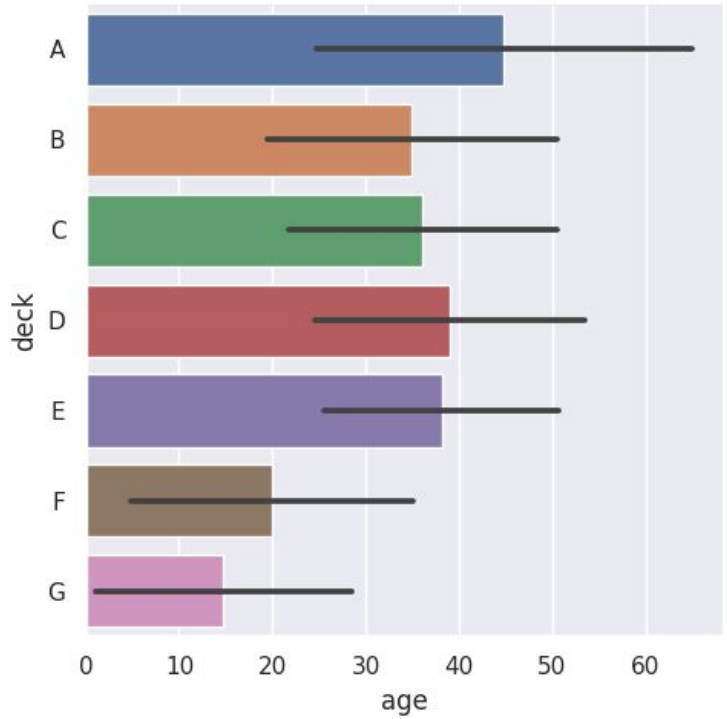
Visualization - barplot

- catplot을 통해서도 barplot을 그릴 수 있습니다
- 이 때, 추가적으로 95% 신뢰구간을 볼 수 있습니다
- errorbar 옵션을 이용해서 다른 방식으로 분산을 표현할 수 있습니다

```
sns.catplot(data=titanic, x="sex", y="survived", hue="class",
kind="bar")
```



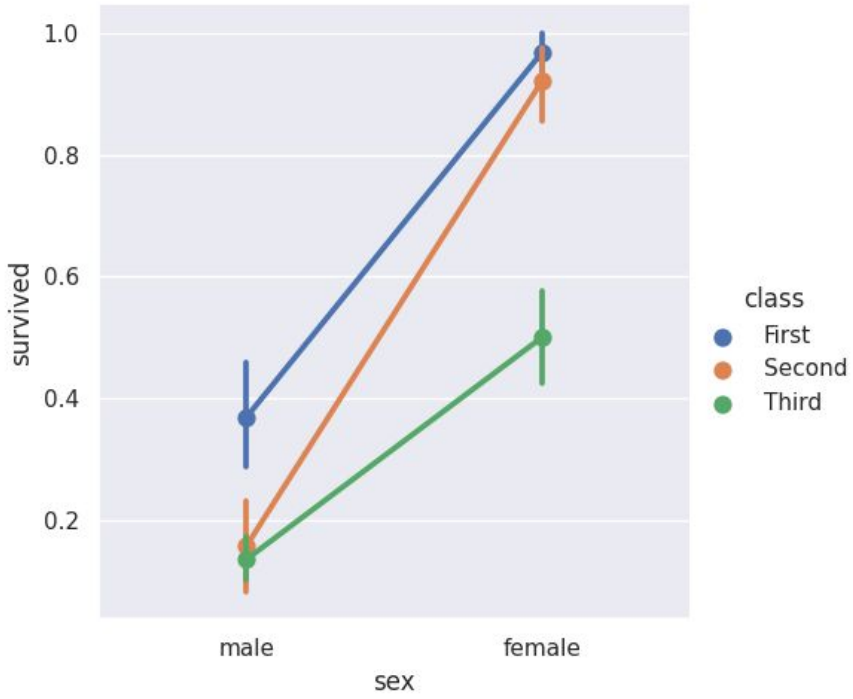

```
sns.catplot(data=titanic, x="age", y="deck", errorbar="sd",
kind="bar")
```



Visualization - Pointplot

- Pointplot은 Categorical 변수에 따라 다른 변수가 어떻게 변화하는지를 효과적으로 나타냅니다
- 이 때 점 추정치 중심으로 95% 신뢰구간이 표현됩니다
- marker, linestyles 옵션을 이용해 선과 점 추정치의 형태를 바꿔줄 수 있습니다

```
sns.catplot(data=titanic, x="sex", y="survived", hue="class",
kind="point")
```




```
sns.catplot(
    data=titanic, x="class", y="survived", hue="sex",
    palette={"male": "g", "female": "m"},
    markers=["^", "o"], linestyles=["-", "--"],
    kind="point"
)
```