Data Science

Data Visualization

홍익대학교 소프트웨어융합학과

노기섭 교수

(kafa46@hongik.ac.kr)

Lecture Goals

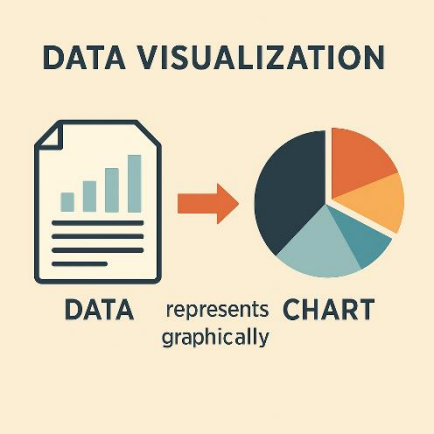
- 데이터 시각화 개요
- Matplotlib 기본 학습
- Seaborn 기본 학습
- 데이터의 분포(Distribution), 관계(Relationship), 시계열(Time Series) 시각화 이해
- 실제 데이터를 다양한 차트로 표현하며 데이터 패턴 파악 능력 배양

데이터 시각화 개요

데이터 시각화의 의미와 필요성

데이터 시각화란 무엇인가?

- 수치를 그래프·차트로 표현하여 쉽게 이해할 수 있도록 하는 과정
- 단순히 숫자를 그림으로 바꾸는 것 → X



- 데이터의 본질적 의미를 직관적으로 드러내는 핵심 도구
- 방대한·복잡한 데이터를 직관적 형태로 변환 → 추세, 패턴, 이상치 한눈에 파악 가능

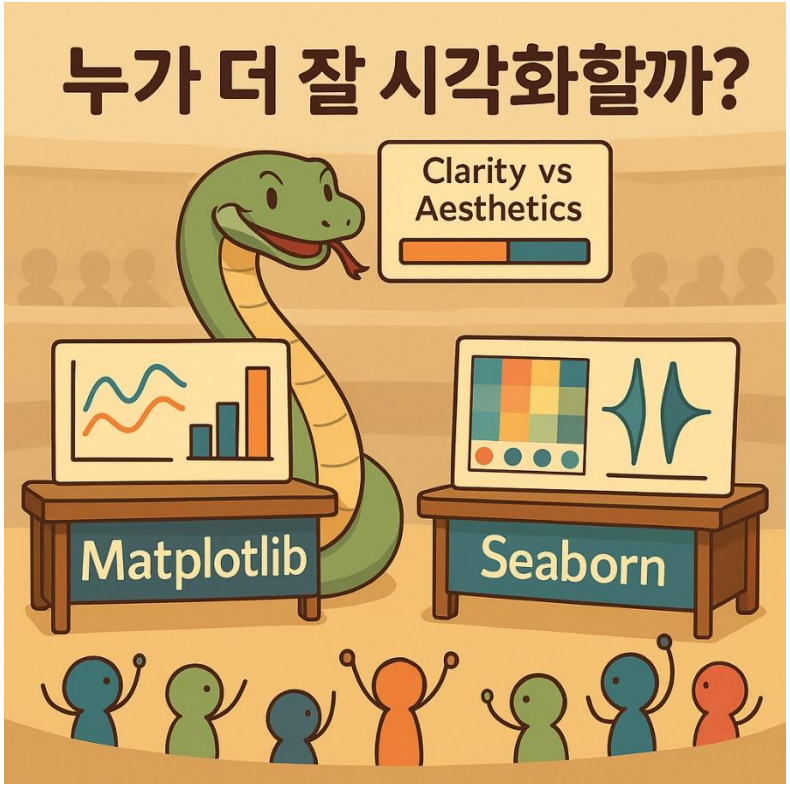
데이터 시각화의 다면적 역할

- **통찰 제공 (Insight)**
  - 경영자: 매출 추세 → 전략 조정
  - 의사: 검사 차트 → 치료 방향 설정
- **데이터 품질 확인**
  - 이상치·결측치 확인 → 박스플롯, 히스토그램 활용
- **소통과 설득 도구**
  - 복잡한 수식 대신 그래프 한 장으로 메시지 전달
  - 다양한 이해관계자가 합의할 때 효과적

Python의 대표 시각화 도구

대표 Python 시각화 라이브러리

- Matplotlib
  - 기본 시각화 도구, 세밀한 제어 가능
- Seaborn
  - 고수준 API 및 미려한 통계 시각화에 강점



Matplotlib Basics

Matplotlib 개요

Matplotlib 개요

- 파이썬에서 가장 널리 사용되는 시각화 라이브러리
- 2000년대 초반 John D. Hunter 개발 → MATLAB 시각화를 파이썬에서 구현하려는 요구에서 출발
- 출판 품질(publication quality)의 그래프 생성 가능데이터 과학, 인공지능, 산업 전반에서 기본 도구로 활용

주요 특징

- 두 가지 사용 방식
  - pyplot: 상태 기반 인터페이스 → 간단하지만 복잡해지면 가독성 저하
  - 객체 지향 API: Figure, Axes를 명시적으로 다룸 → 복잡한 그래프 제어에 유리
- 강력한 커스터마이징
  - 축, 눈금, 서체, 색상, 마커 등 세밀 제어PNG, PDF, SVG, EPS 등 다양한 출력 포맷 지원

생태계, 활용분야, 한계

생태계와 확장성

- 파이썬 과학 라이브러리와 긴밀한 연계: NumPy, pandas, SciPy
- DataFrame.plot()도 내부적으로 Matplotlib 기반
- 확장 라이브러리
  - Seaborn: 통계적 시각화
  - Cartopy: 지도 기반 시각화
  - mplfinance: 금융 데이터 시각화
- 풍부한 문서·예제, 큰 사용자 커뮤니티슬라이드

활용 분야 & 한계

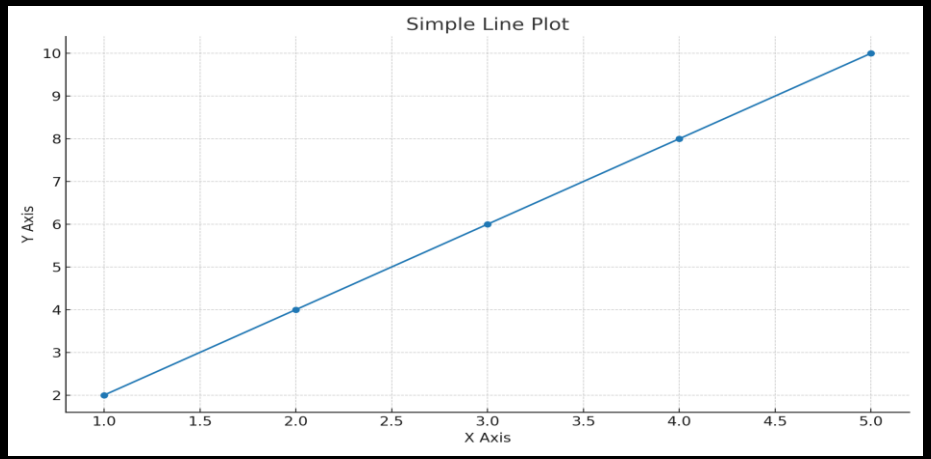
- 활용 분야: 과학 실험 분석, 공학 시뮬레이션, 재무 보고서, 교육용 시각화, 대시보드 초기 설계
- 장점: 사실상 표준, 세밀 제어, 출판 품질 그림 생성 가능
- 단점: 학습 곡선이 가파름, 코드 장황, 기본 스타일은 미려함 부족
- 보완책: plt.style.use(), rcParams, Seaborn 활용

무작정 시작하기 예제 1

목표: 선형 관계를 가진 직선 그래프가 표시

```
# 설치
$ pip install matplotlib
```

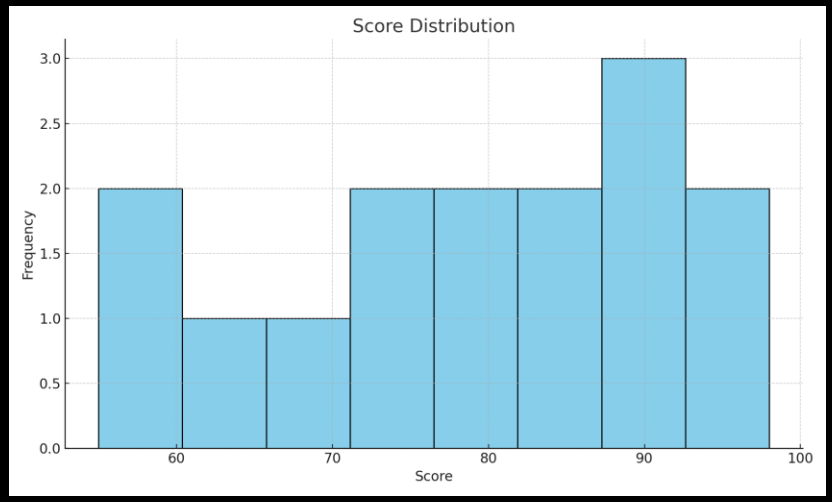
```
import matplotlib.pyplot as plt


x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# 선 그래프, 데이터 모양은 원형
plt.plot(x, y, marker="o")
plt.title("Simple Line Plot")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```



무작정 시작하기 예제 2

목표: 학생들의 시험 점수를 히스토그램으로 시각화

```
import matplotlib.pyplot as plt

scores = [55, 60, 65, 70, 72, 75, 78, 80, 82, 85, 88, 90, 92, 95, 98]

plt.hist(scores, bins=8, color="skyblue", edgecolor="black")
plt.title("Score Distribution")
plt.xlabel("Score")
plt.ylabel("Frequency")
plt.show()
```



최소 소스 코드로 시각화

기본 흐름

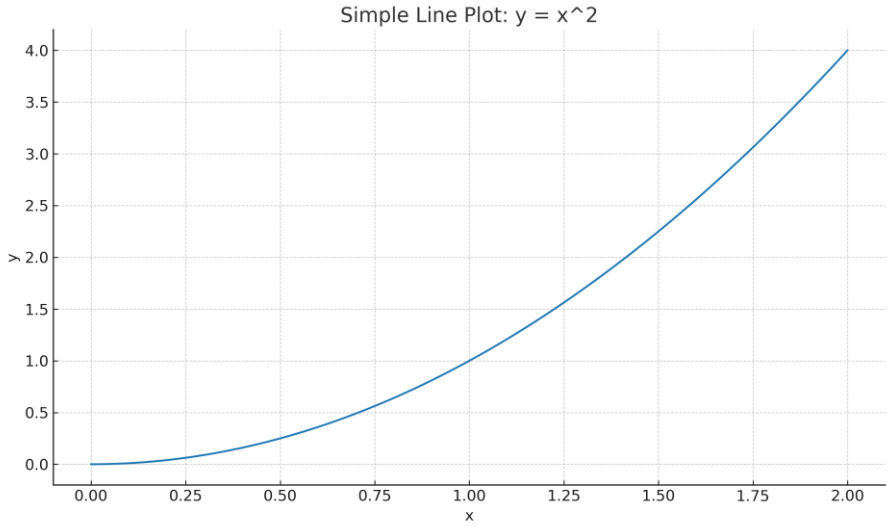
- plt.subplots( ) → Figure와 Axes 생성
- Axes.plot( ) → 데이터 시각화
- plt.show( ) → 그래프 출력

구조 이해

- Figure: 전체 캔버스 (그래프를 담는 큰 틀)
- Axes: 실제 데이터가 그려지는 영역

"무엇을 어디에 그리는가?"라는 질문이 발생함.

```python
'''간단한 2차 함수 곡선 그래프'''

import matplotlib.pyplot as plt
import numpy as np

# 데이터 준비
x = np.linspace(0, 2, 100)
y = x**2

# Figure와 Axes 생성
plt.figure()
plt.plot(x, y)
plt.title("Simple Line Plot: y = x^2")
plt.xlabel("x")
plt.ylabel("y")

# 화면 출력
plt.show()
```



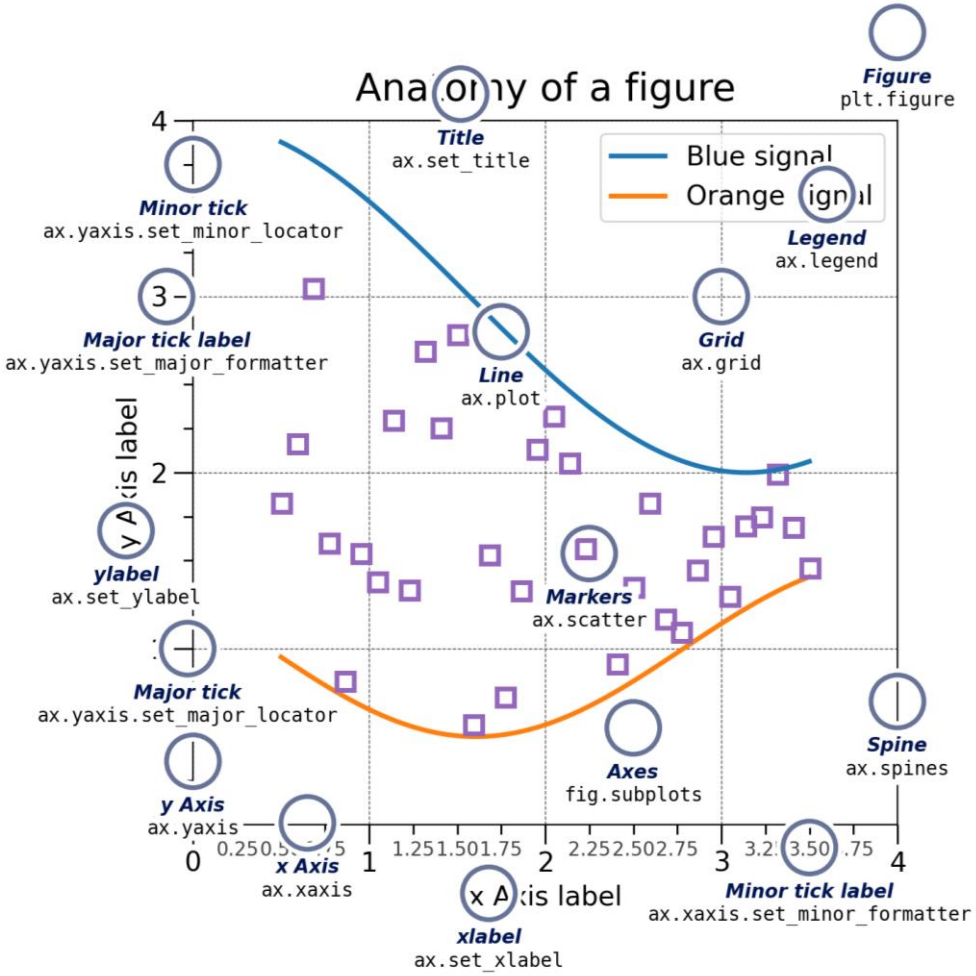
시각화 3대 요소

핵심 질문

- 무엇을 어디에 그리는가?

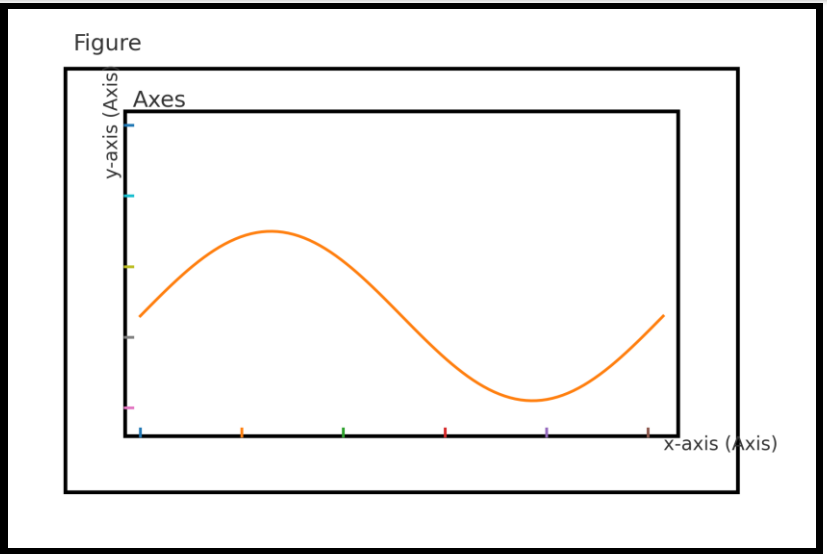
구조적 관점

- Figure → 전체 캔버스
- Axes → 데이터가 표현되는 영역
  - x축, y축, 눈금(ticks), 레이블(labels)
- Artist → 선, 점, 글자 등 실제 시각 요소



Figure–Axes–Axis 구조 다이어그램

```python
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
import numpy as np

fig, ax = plt.subplots(figsize=(6, 4))
ax.set_xlim(0, 10)
ax.set_ylim(0, 7)
ax.axis("off")

# Figure 영역
fig_box = Rectangle((0.5, 0.5), 9, 6, fill=False, linewidth=2)
ax.add_patch(fig_box)
ax.text(0.6, 6.7, "Figure", fontsize=12, va="bottom")

# Axes 영역
axes_box = Rectangle((1.3, 1.3), 7.4, 4.6, fill=False, linewidth=2)
ax.add_patch(axes_box)
ax.text(1.4, 5.9, "Axes", fontsize=12, va="bottom")

# 축 레이블 표시
ax.text(8.5, 1.1, "x-axis (Axis)", fontsize=10)
ax.text(1.0, 5.0, "y-axis (Axis)", fontsize=10, rotation=90)

# 눈금 흉내
for t in np.linspace(1.5, 8.3, 6):
    ax.plot([t, t], [1.3, 1.4])
for t in np.linspace(1.7, 5.7, 5):
    ax.plot([1.3, 1.4], [t, t])

# Axes 내부 데이터 예시
xs = np.linspace(1.5, 8.5, 100)
ys = 3 + 1.2*np.sin((xs-1.5)/7.0 * 2*np.pi)
ax.plot(xs, ys)

plt.show()
```



Matplotlib - Figure

- subplots: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html#matplotlib.pyplot.subplots
- subplot_mosaic: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplot_mosaic.html#matplotlib.pyplot.subplot_mosaic

Figure

```
fig = plt.figure()              # an empty figure with no Axes
fig, ax = plt.subplots()        # a figure with a single Axes
fig, axs = plt.subplots(2, 2)   # a figure with a 2x2 grid of Axes
# a figure with one Axes on the left, and two on the right:
fig, axs = plt.subplot_mosaic([['left', 'right_top'],
                               ['left', 'right_bottom']])
```

Figure는 전체 그림을 의미

- Figure는 여러 Axes와 특수한 Artist(제목, 범례, 컬러바 등)를 포함
- 새로운 Figure는 plt.figure(), plt.subplots(), plt.subplots(2,2) 같은 함수로 생성 가능
- subplots()는 Figure와 동시에 Axes를 함께 만들어주는 편리한 함수
- subplot_mosaic은 Figure 안에 Axes 객체를 추가로 생성해주는 편의 함수

Matplotlib – Figure - Axes

Axes

- Axes는 데이터를 실제로 그리는 영역
- 하나의 Figure 안에는 보통 두 개의 Axis(x축, y축)가 들어있고, 3D의 경우 세 개가 들어갈 수 있음
- 각 Axes는 눈금(tick)과 레이블(label)을 가진다.
- set_title(), set_xlabel(), set_ylabel() 등을 통해 제목과 축 이름을 설정 가능
- 데이터 추가, 스케일 설정, 축 범위 지정 등 대부분의 플롯 설정은 Axes 객체 메서드를 통해 조작

axes [æksɪz]: **영어 단어 axis의 복수형으로,**
하나의 좌표 평면이나 공간을 구성하는
여러 개의 축(가로축, 세로축 등)을 통칭하는 용어

axis [æksɪs]: **여러 축 중 하나의 특정 축을 가리키는 단수형**

Matplotlib – Figure - Axis

Axis

- Axis는 x축과 y축 각각을 담당하는 객체
- 축의 스케일, 눈금, 눈금 레이블을 관리
- 눈금 위치는 Locator 객체가 결정하고, 눈금 텍스트는 Formatter가 지정
- Axis는 데이터 범위와 레이블링을 세밀하게 제어하는 용도

Matplotlib – Figure - Artist

Artist (Figure에서 눈에 보이는 모든 요소)

- Figure, Axes, Axis뿐만 아니라,
- Text, Line2D, Patch(사각형·원 등), 컬렉션(collections), 범례, 제목 등이 모두 Artist
- 그림이 렌더링 (rendering)될 때 모든 Artist는 canvas 위에 그려 짐
- 대부분의 Artist는 특정 Axes에 연결되며, 여러 Axes에 동시에 공유되거나 옮겨 다니지 않는다.

- **Matplotlib의 구조는 Figure(전체 캔버스) 안에 Axes(좌표 영역)가 있고,**
- **Axes에는 Axis(x축, y축) 있으며, 이 위에 수많은 Artist(선, 글자, 범례 등) 이 그려지는 방식**
- **Figure → Axes → Axis/Artist 라는 계층을 이해하는 것이 Matplotlib을 활용하는 핵심**

Matplotlib 입력 데이터 구조

입력 데이터 구조

- Matplotlib의 플로팅 함수들은 보통 numpy.array 또는 numpy.ma.masked_array를 입력으로 받음
- 혹은 numpy.asarray를 통해 배열로 변환될 수 있는 객체들도 가능
- 그러나 pandas의 데이터 객체나 numpy.matrix와 같은 배열 유사 객체(array-like)는 의도대로 동작하지 않을 수 있음
- 따라서 일반적으로는 이들을 먼저 numpy.array로 변환한 뒤 사용하는 것이 관례

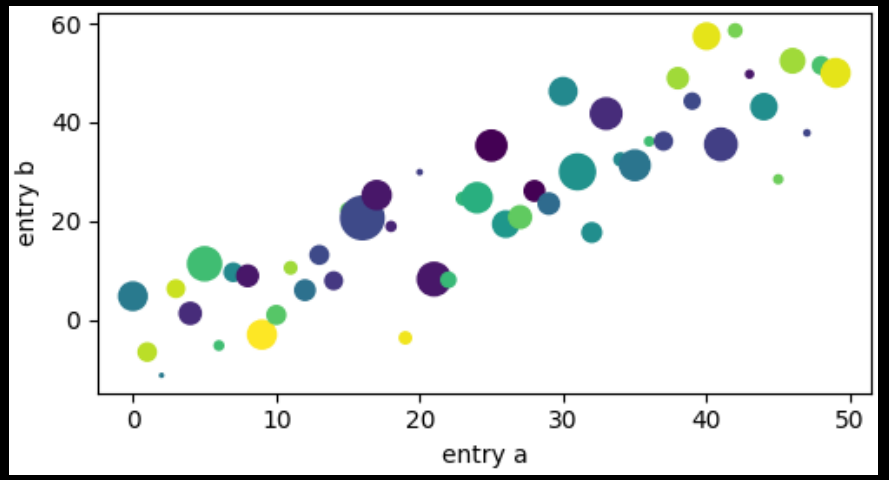
[예시] numpy.matrix를 변환 배열로 변환

```
import numpy as np
b = np.matrix([[1, 2], [3, 4]])
b_asarray = np.asarray(b)
print(b_asarray)
```

```
[[1 2]
 [3 4]]
```

Matplotlib 입력 데이터 – 문자열 인덱싱

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

# 난수 생성기 시드 고정 (재현 가능성을 위해 항상 동일한 난수 생성)
np.random.seed(19680801)

# 데이터 딕셔너리 생성
data = {
    'a': np.arange(50),                 # 0부터 49까지 정수 (x축 데이터)
    'c': np.random.randint(0, 50, 50),  # 0~49 사이 난수 50개 (색상 값으로 활용)
    'd': np.random.randn(50)            # 표준정규분포 난수 50개 (크기 값으로 활용)
}

# y축 데이터 'b' = x축 데이터 'a' + 잡음(정규분포 * 10)
data['b'] = data['a'] + 10 * np.random.randn(50)

# 'd' 값을 양수로 변환 후 100배 확대 (산점도의 점 크기로 사용)
data['d'] = np.abs(data['d']) * 100

# Figure(전체 캔버스), Axes(데이터 영역) 생성
# figsize: 그래프 크기, layout='constrained': 자동으로 여백 최적화
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')

# 산점도(scatter plot) 그리기
# 'a' → x축, 'b' → y축, 'c' → 점 색상, 'd' → 점 크기
ax.scatter('a', 'b', c='c', s='d', data=data)

# x축, y축 라벨 설정
ax.set_xlabel('entry a')
ax.set_ylabel('entry b')

# 그래프 출력
plt.show()
```



Matplotlib 코딩 스타일

객체지향 방식

- 명시적 인터페이스(Object-Oriented, OO 스타일) 방식
- Figure와 Axes를 직접 생성하고, 각 객체의 메서드를 호출하여 그래프를 제어

(코딩 예시)

  - fig, ax = plt.subplots()로 Figure와 Axes를 만든 뒤,
  - ax.plot()이나 ax.set_title() 같은 메서드를 통해 구체적인 설정을 수행

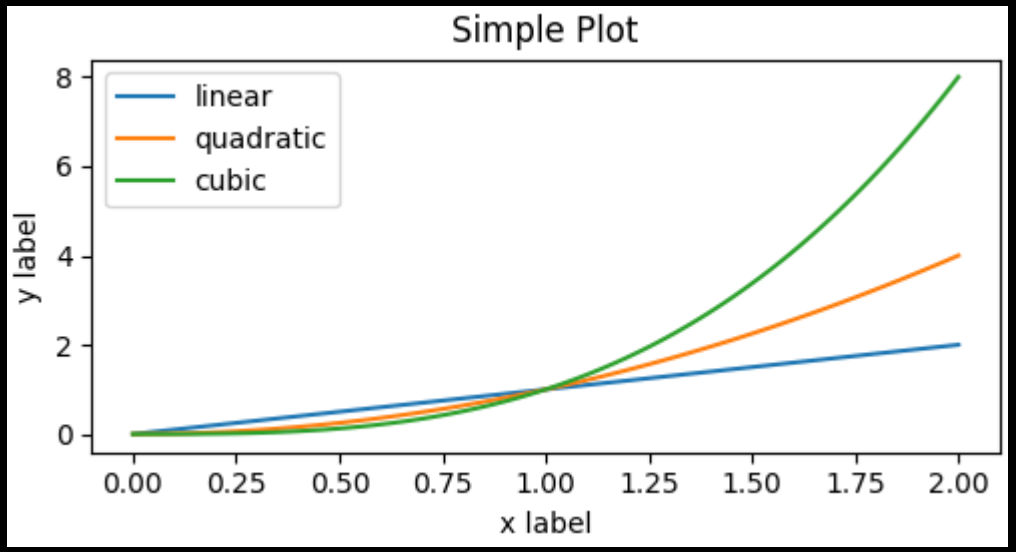
- 코드의 구조가 명확하며, 복잡한 그래프나 다중 축 레이아웃을 다루는 데 유리

암묵적 암묵적 인터페이스(Pyplot 방식)

- 사용자는 단순히 plt.plot() 같은 함수만 호출
- Matplotlib 내부에서 현재 활성화된 Figure와 Axes를 자동으로 관리
- 장점: 단순한 그래프를 빠르게 그리는 데는 편리
- 단점: 여러 개의 그래프를 동시에 다룰 때는 코드 가독성이 떨어지고 관리가 어려워질 수 있음.

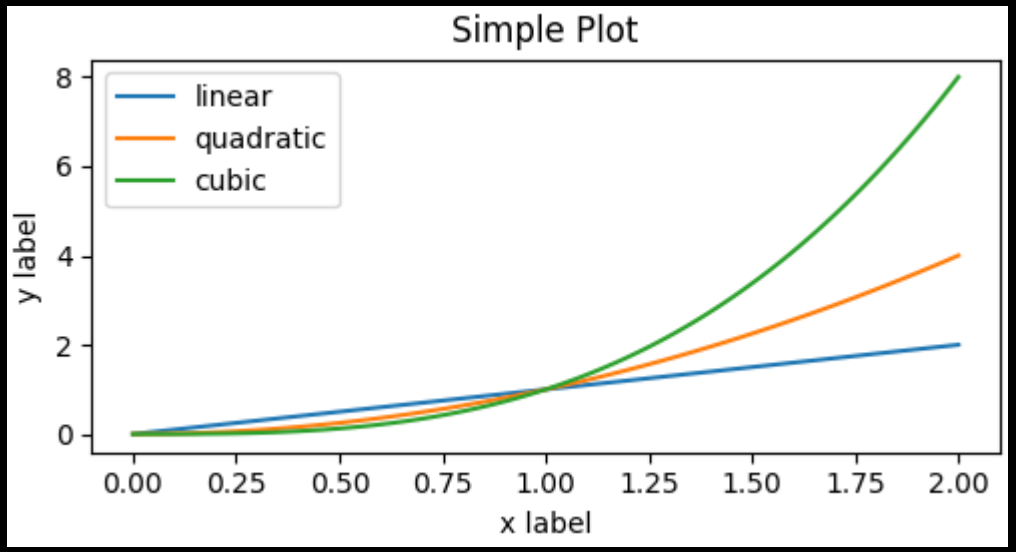
객체지향 (OOP) 방식 예제

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

x = np.linspace(0, 2, 100)  # 샘플 데이터 생성

fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')

# 서로 다른 함수 플로팅
ax.plot(x, x, label='linear')        # y = x
ax.plot(x, x**2, label='quadratic')  # y = x^2
ax.plot(x, x**3, label='cubic')      # y = x^3

# 축과 제목 설정
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")

# 범례 추가
ax.legend()

plt.show()
```



암묵적 인터페이스 방식 예제

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

x = np.linspace(0, 2, 100)  # Sample data.

plt.figure(figsize=(5, 2.7), layout='constrained')
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) Axes.
plt.plot(x, x**2, label='quadratic')  # etc.
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
```



Artist 스타일링

Artist 객체의 스타일을 조정할 수 있는 옵션

- 플롯 메서드(plot)를 호출할 때 인자로 직접 전달할 수도 있고, 혹은 Artist의 setter 메서드를 통해 사후에 수정할 수도 있음.

  (예시)

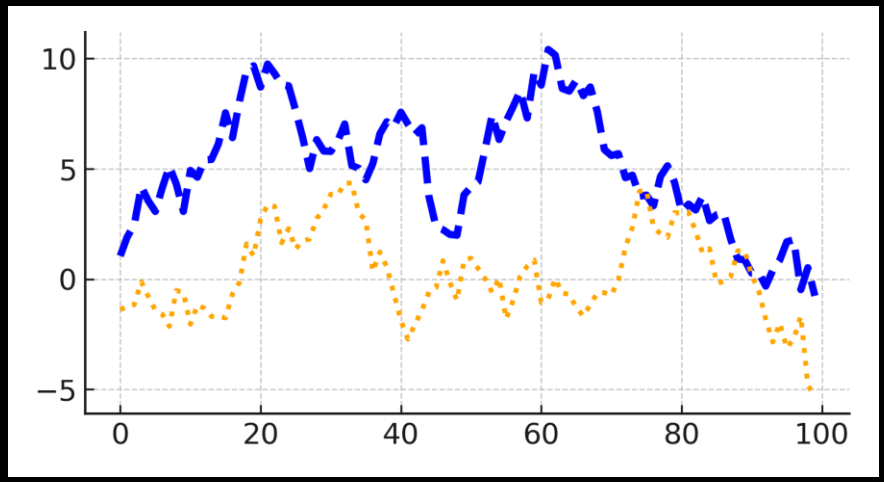
  - plot 함수를 이용해 선 색상(color), 선 굵기(linewidth), 선 스타일(linestyle)을 지정
  - 선에 대해서는 set_linestyle 메서드를 호출하여 스타일을 변경 등

set_linestyle

https://matplotlib.org/stable/api/_as_gen/matplotlib.lines.Line2D.html#matplotlib.lines.Line2D.set_linestyle

Line Style

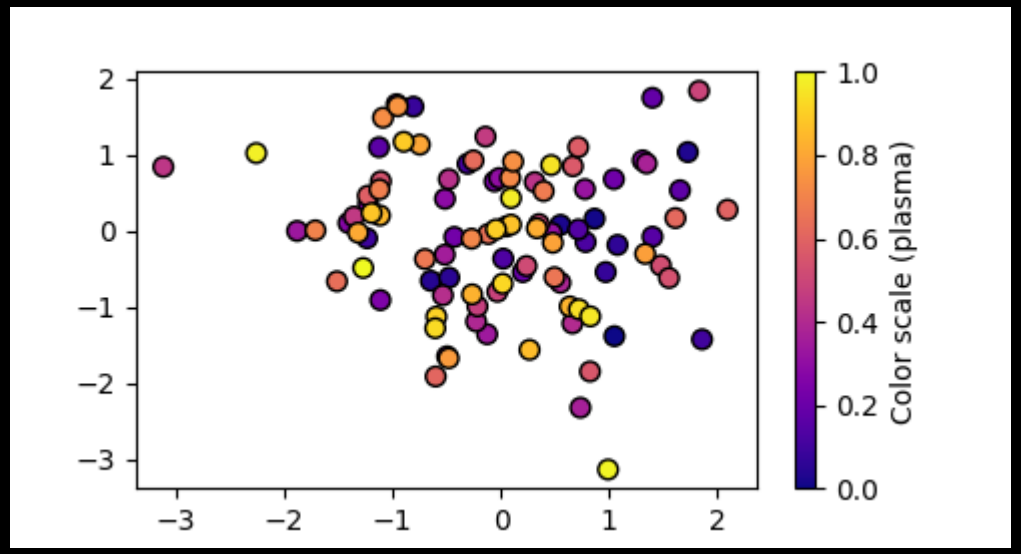
```python
import numpy as np
import matplotlib.pyplot as plt

# 1. 데이터 준비

# 난수 시드를 고정하여 실행할 때마다 동일한 결과가 나오도록 설정
np.random.seed(19680801)

# 정규분포를 따르는 난수 100개를 누적합(cumulative sum)
data1 = np.random.randn(100).cumsum()
data2 = np.random.randn(100).cumsum()

# 2. Figure와 Axes 생성

# 5x2.7 인치 크기의 Figure와 하나의 Axes 생성
fig, ax = plt.subplots(figsize=(5, 2.7))

# x축 값: 0부터 데이터 길이(100)까지 정수 배열
x = np.arange(len(data1))

# 3. 선 그래프 그리기
# 첫 번째 선: data1 값 시각화, 색상: 파란색, 선 굵기: 3, 선 스타일: '--' (점선)
ax.plot(x, data1, color='blue', linewidth=3, linestyle='--')

# 두 번째 선: data2 값 시각화, # 색상: 주황색, 선 굵기: 2
# 반환값 l은 Line2D 객체로, 이후 스타일 변경에 사용됨
l, = ax.plot(x, data2, color='orange', linewidth=2)

# 4. 스타일 사후 변경
l.set_linestyle(':’) # 두 번째 선의 선 스타일을 ':' (점선)으로 변경

plt.show()
```



색상(Colors)

- **Matplotlib은 대부분의 Artist에서 사용할 수 있는 매우 다양한 색상 옵션을 지원**
  - 색상 정의 문서(allowable color definitions)를 참고

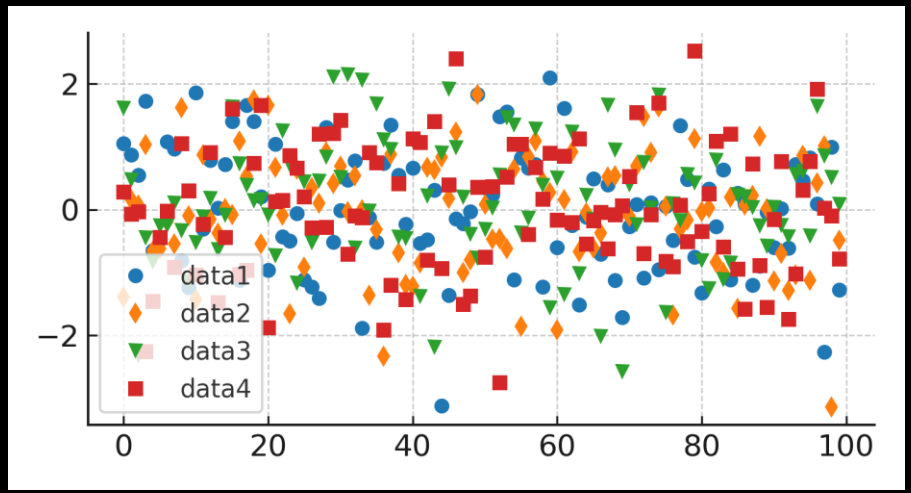
```python
import numpy as np
import matplotlib.pyplot as plt

# 난수 데이터 생성
np.random.seed(19680801)
data1 = np.random.randn(100)
data2 = np.random.randn(100)

# 색상 매핑 기준 (0 ~ 1 범위)
colors = np.linspace(0, 1, 100)

# 산점도 그리기
fig, ax = plt.subplots(figsize=(5, 2.7))
scatter = ax.scatter(
    data1, data2,
    c=colors,           # 색상 값 지정
    cmap='plasma',      # plasma 팔레트 적용
    s=50,               # 마커 크기
    edgecolor='k'       # 테두리 색상 (검정)
)

# 컬러바 추가
plt.colorbar(scatter, ax=ax, label="Color scale (plasma)")

plt.show()
```



선 굵기, 선 스타일, 마커 크기

- 선 스타일(linestyle): https://matplotlib.org/stable/gallery/lines_bars_and_markers/linestyles.html
- 마커 스타일: https://matplotlib.org/stable/gallery/lines_bars_and_markers/marker_reference.html

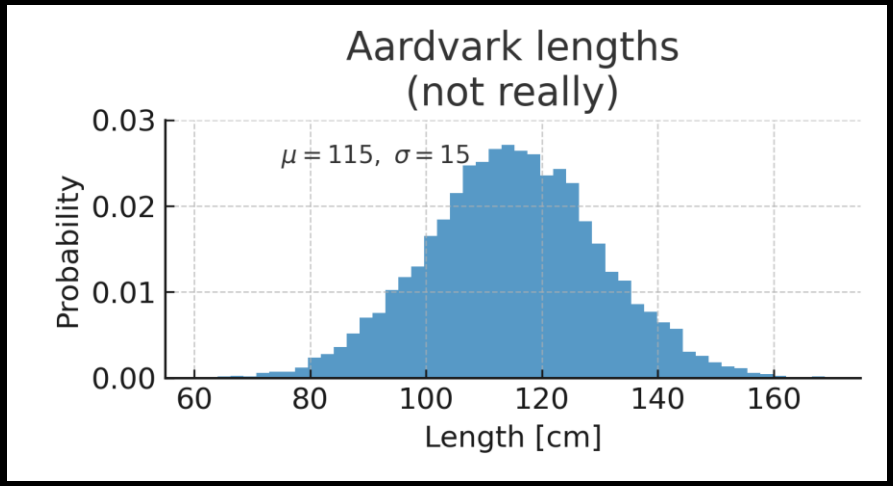
```
import numpy as np
import matplotlib.pyplot as plt

# 난수 데이터 준비
np.random.seed(19680801)
data1 = np.random.randn(100)
data2 = np.random.randn(100)
data3 = np.random.randn(100)
data4 = np.random.randn(100)

fig, ax = plt.subplots(figsize=(5, 2.7))

# 각 데이터셋을 서로 다른 마커 스타일로 플로팅
ax.plot(data1, 'o', label='data1')  # 원형 마커
ax.plot(data2, 'd', label='data2')  # 다이아몬드 마커
ax.plot(data3, 'v', label='data3')  # 역삼각형 마커
ax.plot(data4, 's', label='data4')  # 사각형 마커

# 범례 표시
ax.legend()
plt.show()
```



레이블과 텍스트

- 그래프에 레이블과 텍스트를 추가하여 데이터의 의미를 명확하게 전달

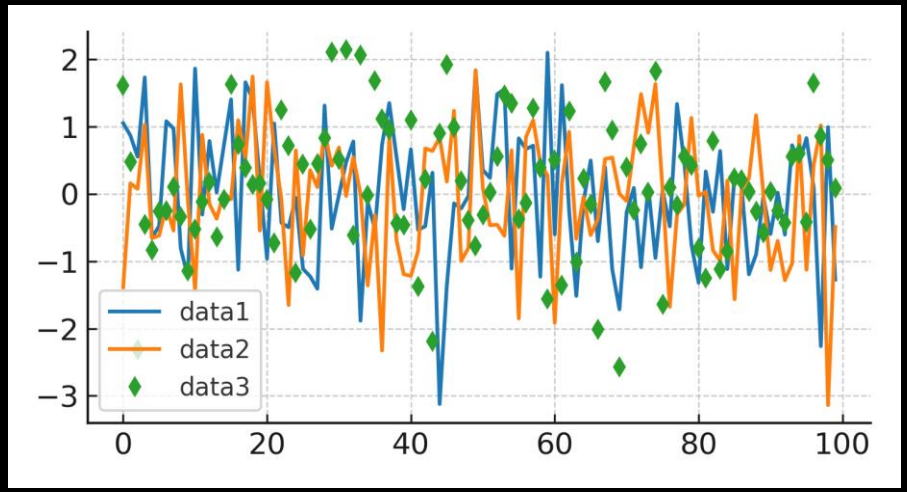
```python
import numpy as np
import matplotlib.pyplot as plt

# 데이터 생성: 평균 115, 표준편차 15를 따르는 정규분포 난수 10000개
mu, sigma = 115, 15
x = mu + sigma * np.random.randn(10000)

fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')

# 히스토그램 그리기
n, bins, patches = ax.hist(x, 50, density=True, facecolor='C0', alpha=0.75)

# 축 레이블, 제목, 텍스트 추가
ax.set_xlabel('Length [cm]')
ax.set_ylabel('Probability')
ax.set_title('Aardvark lengths\n(not really)')

# 좌표 (75, 0.025)에 텍스트 추가
ax.text(75, 0.025, r'$\mu=115,\ \sigma=15$')

# 축 범위와 격자 설정
ax.axis([55, 175, 0, 0.03])
ax.grid(True)

plt.show()
```



범례(Legends)

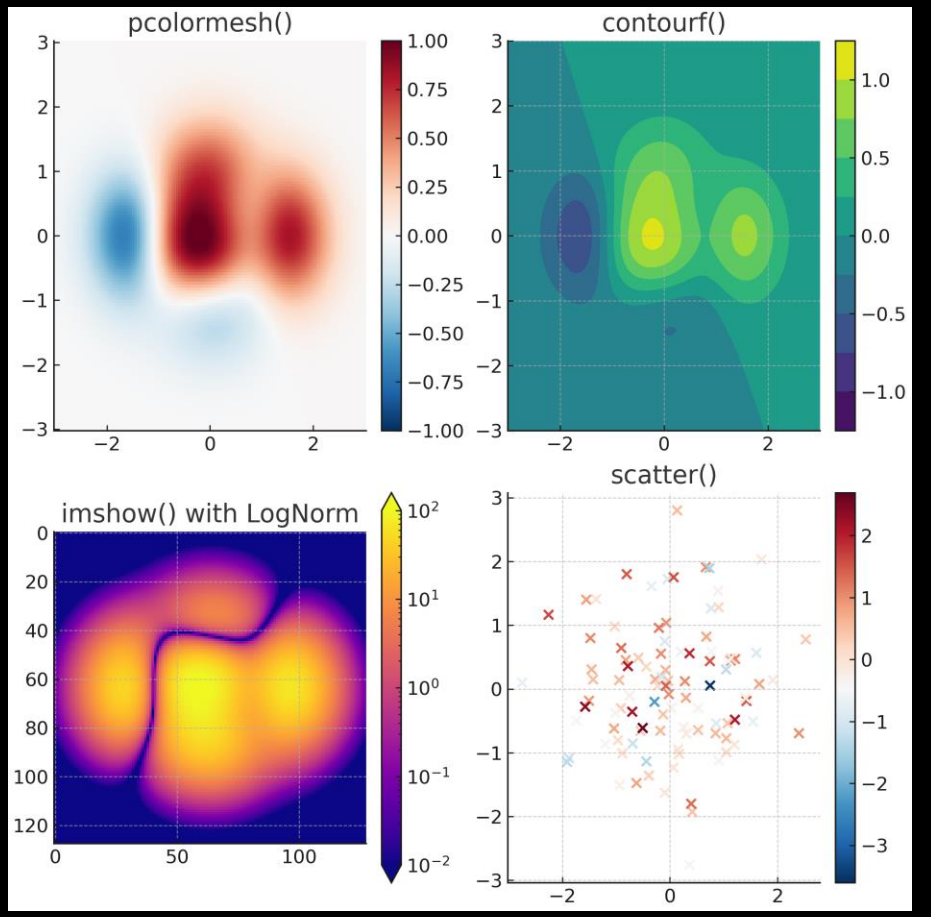
- 그래프에서 여러 데이터 시리즈를 명확히 구분하는 데 중요한 요소

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

# 난수 데이터 생성
np.random.seed(19680801)
data1 = np.random.randn(100)
data2 = np.random.randn(100)
data3 = np.random.randn(100)
fig, ax = plt.subplots(figsize=(5, 2.7))

# 서로 다른 데이터와 레이블 지정
ax.plot(np.arange(len(data1)), data1, label='data1')   # 기본 선 그래프
ax.plot(np.arange(len(data2)), data2, label='data2')   # 기본 선 그래프
ax.plot(np.arange(len(data3)), data3, 'd', label='data3')  # 다이아몬드 마커 사용

ax.legend() # 범례 표시
plt.show()
```



색상 맵 데이터 (Color mapped data)

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LogNorm

# 1. 데이터 생성
# X, Y: -3 ~ 3 구간을 128등분한 격자 좌표 생성
X, Y = np.meshgrid(np.linspace(-3, 3, 128), np.linspace(-3, 3, 128))

# Z: X, Y를 이용해 계산된 함수 값 (곡면 형태)
Z = (1 - X/2 + X**5 + Y**3) * np.exp(-X**2 - Y**2)

# 2. Figure와 2x2 Axes 생성
fig, axs = plt.subplots(2, 2, layout='constrained', figsize=(8, 8))

# 3. pcolormesh()
# 격자 데이터(Z)를 색상으로 채워 넣는 함수
# cmap='RdBu_r': 빨강-파랑 반전 팔레트
pc = axs[0, 0].pcolormesh(X, Y, Z, vmin=-1, vmax=1, cmap='RdBu_r')
fig.colorbar(pc, ax=axs[0, 0])  # 색상바(colorbar) 추가
axs[0, 0].set_title('pcolormesh()')

# 4. contourf()
# 등고선 기반 색상 맵
# levels=np.linspace(-1.25, 1.25, 11): -1.25~1.25 구간을 11단계로 나눔
co = axs[0, 1].contourf(X, Y, Z, levels=np.linspace(-1.25, 1.25, 11))
fig.colorbar(co, ax=axs[0, 1])  # 색상바 추가
axs[0, 1].set_title('contourf()')

# 5. imshow() + LogNorm
# 2차원 배열을 이미지처럼 표시
# LogNorm: 로그 스케일 색상 맵 → 작은 값과 큰 값을 동시에 표현 가능
pc2 = axs[1, 0].imshow(Z**2 * 100, cmap='plasma',
                       norm=LogNorm(vmin=0.01, vmax=100))
fig.colorbar(pc2, ax=axs[1, 0], extend='both')
axs[1, 0].set_title('imshow() with LogNorm')

# 6. scatter()
# 점 단위로 데이터 표시, 각 점은 세 번째 데이터(data3)에 따라 색상이 달라짐
data1, data2, data3 = np.random.randn(3, 100)  # x, y, 색상 값
sc = axs[1, 1].scatter(data1, data2, c=data3, cmap='RdBu_r')
fig.colorbar(sc, ax=axs[1, 1])
axs[1, 1].set_title('scatter()')

plt.show()
```



Seaborn Basics

Seaborn 소개

Seaborn 소개

- Python의 데이터 시각화 라이브러리
- Matplotlib 기반으로 개발단순 보완이 아니라 통계적 시각화를 직관적으로 구현
- 기본적으로 세련된 스타일 제공 → 커스터마이징 최소화
- Pandas DataFrame과 긴밀히 연동

개발 배경

- 개발자: Michael Waskom
- 목적
  - Matplotlib의 복잡한 설정을 단순화
  - 통계적 데이터 시각화(statistical data visualization) 구현
- 기존 Matplotlib의 한계
  - 회귀선, 분포 플롯, 다변량 관계 시각화 구현 시 많은 코드 필요
  - Seaborn: 간결한 코드로 빠른 인사이트 제공

Seaborn 활용 및 특징

개발 생태계 & 활용

- 기반 생태계: Pandas, NumPy, Matplotlib, SciPy
- Python 데이터 분석 생태계와 밀접 연동
- 현재도 활발히 유지보수
- 활용 분야
    - Kaggle, 데이터 사이언스 교육, 논문 및 보고서 작성, 학계와 산업계 전반

주요 특징

- 기본적으로 아름답고 일관성 있는 스타일 제공
- 통계적 플롯 (회귀선, 분포도, 다변량 관계 등) 지원
- 데이터 처리 → 시각화 과정을 자연스럽게 연결
- 학계·산업계 모두에서 표준적인 시각화 도구로 자리매김

Seaborn 공식 홈페이지: https://seaborn.pydata.org/

Seaborn 장단점

장점

- 코드가 간결하여 초보자도 쉽게 사용 가능
- Matplotlib보다 시각적으로 세련된 기본 스타일 제공
- Pandas와 자연스러운 연동 → 데이터 처리와 시각화가 매끄럽게 연결됨
- 통계적 분석 결과를 직관적으로 시각화 가능 (예: 회귀 분석, 분포 비교)
- 다양한 색상 팔레트와 테마 지원 → 보고서, 프레젠테이션에 적합

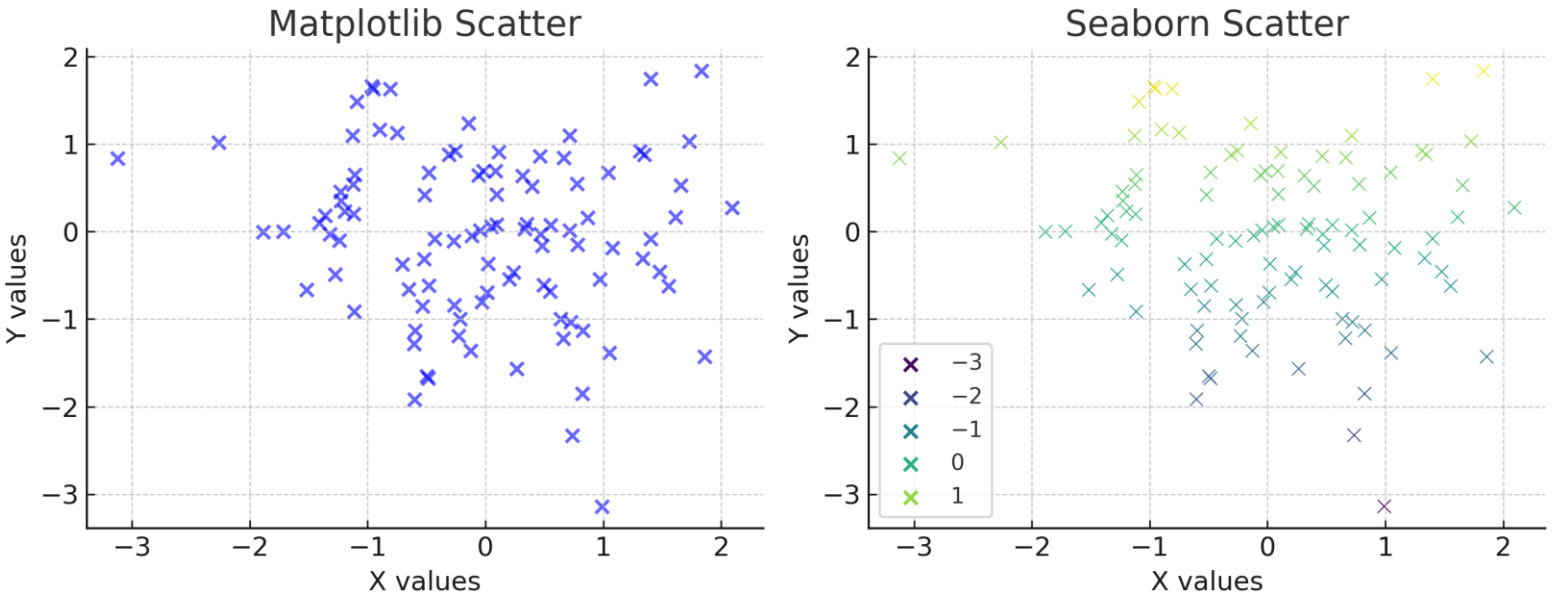
단점

- Matplotlib에 비해 세부적인 커스터마이징은 제한적임
- 그래프가 추상화되어 있어 복잡한 커스터마이징 시에는 Matplotlib을 직접 병행해야 함
- 대규모 데이터 시각화에는 성능 제약이 있을 수 있음
- 매우 특수한 시각화(예: 3D 시각화)는 지원하지 않음

Matplotlib vs Seaborn 비교

구분	Matplotlib	Seaborn
개발 배경	Python 최초의 본격적인 시각화 라이브러리로, 다양한 그래프를 저수준에서 자유롭게 커스터마이징 가능	Matplotlib의 복잡한 설정을 단순화하고, 통계적 데이터 시각화를 직관적으로 구현하기 위해 개발
특징	- 저수준 API 제공 - 세부 조정 가능 - 기본 스타일 단순	- 고수준 API 제공 - 기본 스타일 세련됨 - Pandas DataFrame과 자연스럽게 연동
사용 난이도	비교적 복잡, 많은 코드 필요	간단한 코드로 직관적인 시각화 가능
지원 기능	- 기본 플롯 (선, 막대, 산점도, 히스토그램 등) - 3D 플롯 지원 - 완전한 커스터마이징 가능	- 통계 시각화 (회귀선, 분포도, 카테고리 비교 등) - Heatmap, pairplot, violinplot 등 고급 기능
장점	- 강력한 유연성 - 복잡한 그래프도 구현 가능 - 거의 모든 시각화 가능	- 시각적으로 세련된 기본 결과물 - 코드가 간결 - 통계적 시각화에 최적화
단점	- 기본 그래프가 투박함 - 설정 코드가 많아 초보자에게 어려움	- 커스터마이징 제약 존재 - Matplotlib과 병행해야 할 때가 있음
활용 분야	- 연구 논문용 고급 그래프 - 커스터마이징이 필요한 복잡한 대시보드	- 데이터 분석 수업, Kaggle 분석 - 탐색적 데이터 분석(EDA) - 보고서, 프레젠테이션

Matplotlib vs Seaborn 비교 예제



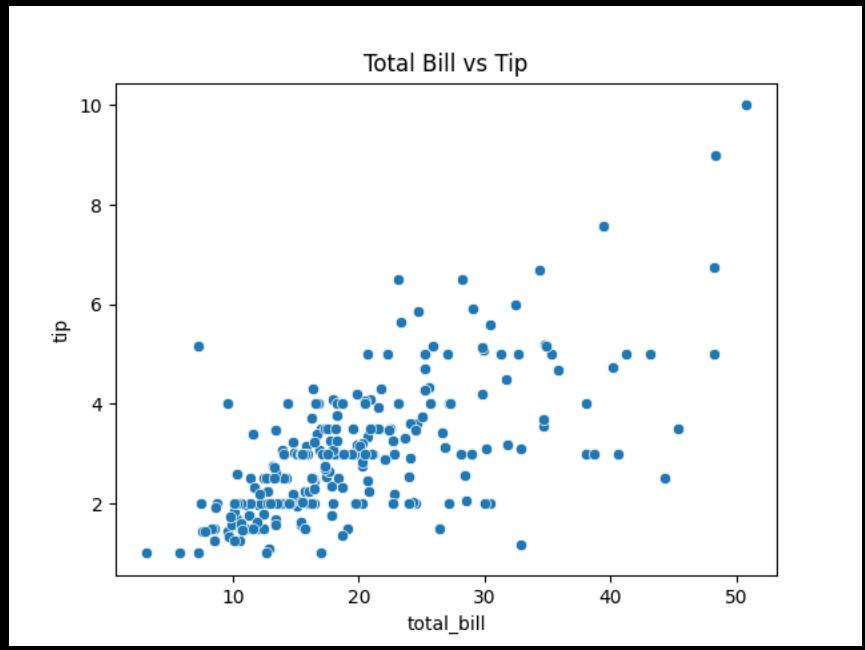
- 동일한 데이터셋을 산점도로 표현했을 때, Matplotlib(왼쪽)는 기본 스타일로 단순하고 직관적인 결과를 제공
- 반면 Seaborn(오른쪽)은 색상 팔레트와 자동 스타일링을 적용하여 시각적으로 더 세련되고 데이터의 분포 차이를 강조

기본 문법

```
# 설치
$ pip install seaborn
```

목표: 식사 금액과 팁의 상관관계 시각화

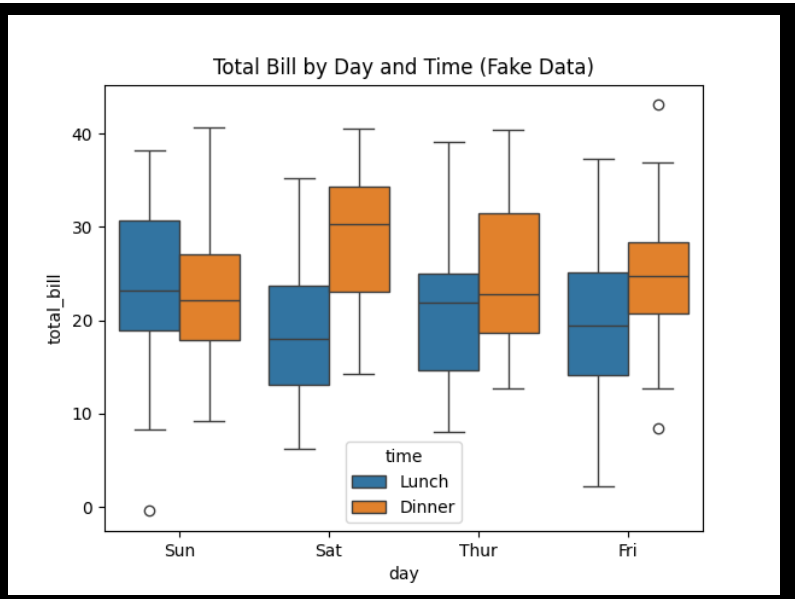
```
import seaborn as sns
import matplotlib.pyplot as plt

tips = sns.load_dataset("tips")
sns.scatterplot(
        data=tips,
        x="total_bill",
        y="tip“
)
plt.title("Total Bill vs Tip")
plt.show()
```



Seaborn 중급 예제

시나리오: 요일별·시간대별 식사 비용 차이를 박스플롯으로 비교

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np


# 가상 데이터 생성: 난수 시드 고정 → 항상 동일한 값 생성
np.random.seed(0)

days = ["Sun", "Sat", "Thur", "Fri"]
times = ["Lunch", "Dinner"]

data = []
for day in days:
  for time in times:
    # 각 조합에 대해 총 30개 샘플 생성
    bills = np.random.normal(
      loc=20 if time=="Lunch" else 25, # 정규분포의 평균
      scale=8, # 정규분포의 표준편차
      size=30  # 샘플 개수
    )
    for b in bills:
      data.append({"day": day, "time": time, "total_bill": b})

df = pd.DataFrame(data)

# Boxplot 그리기
sns.boxplot(
  data=df,
  x="day", # x축: 요일
  y="total_bill", # y축: 총 금액
  hue="time", # 색상 구분: 점심/저녁
  order=days # x축 순서 지정
)

plt.title("Total Bill by Day and Time (Fake Data)")
plt.show()
```



박스플롯 해석

■ **특징**

- 박스플롯의 상자, 수염, 이상치를 통해 분포의 특징을 직관적으로 파악 가능
- 박스플롯은 데이터의 분포와 이상치를 직관적으로 확인할 수 있는 대표적인 통계적 시각화 도구
- 하나의 그래프 안에서 중앙값, 사분위수, 데이터의 범위, 그리고 이상치를 동시에 표현하기 때문에 탐색적 데이터 분석(EDA)에서 자주 사용

■ **중앙값(Median)**

- 박스 안의 가로선이 중앙값을 나타낸다. 데이터의 절반은 이 값보다 크고, 나머지 절반은 이 값보다 작다. 분포의 중심을 파악하는 데 중요한 지표이다.

■ **사분위수(Quartiles)와 박스**

- 박스의 아래쪽 경계는 제1사분위수(Q1, 25% 위치), 위쪽 경계는 제3사분위수(Q3, 75% 위치)를 의미한다.
- 따라서 박스 자체는 데이터의 중간 50% 구간(Interquartile Range, IQR)을 보여준다.
- 박스가 크면 데이터가 넓게 퍼져 있고, 작으면 데이터가 모여 있다는 뜻이다.

■ **수염(Whiskers)**

- 박스 위아래로 뻗은 선을 수염이라고 한다. 일반적으로 Q1 - 1.5×IQR, Q3 + 1.5×IQR 범위 안에 있는 가장 작은 값과 가장 큰 값까지 이어진다.
- 수염은 데이터의 전반적인 분포 범위를 나타낸다.

■ **이상치(Outliers)**

- 수염 밖에 위치한 점들은 이상치로 표시된다. 이상치는 데이터의 변동성, 오류, 또는 특별한 패턴을 나타낼 수 있으므로 분석 과정에서 주의 깊게 해석해야 한다.

■ **분포 해석**

- 박스가 위로 치우쳐 있으면 데이터가 하위 값 쪽에 많이 분포한 것(왼쪽 꼬리가 김).
- 박스가 아래로 치우쳐 있으면 데이터가 상위 값 쪽에 몰려 있는 것(오른쪽 꼬리가 김).
- 중앙값이 박스의 정중앙이 아니라 한쪽으로 치우쳐 있으면 분포가 비대칭임을 의미한다.

다양한 Seaborn 맛보기

Seaborn 공통 헤더 코드

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Theme once
sns.set_theme(style="whitegrid")

# Reproducibility
np.random.seed(42)
n = 150

# 공통적으로 사용할 데이터
df = pd.DataFrame({
    "category": np.random.choice(["A", "B", "C"], size=n, p=[0.4, 0.35, 0.25]),
    "value": np.concatenate([
        np.random.normal(0.0, 1.0, 60),
        np.random.normal(0.5, 1.2, 50),
        np.random.normal(-0.3, 0.8, 40)
    ]),
    "group": np.random.choice(["G1", "G2"], size=n),
    "feature1": np.random.normal(0, 1, n),
    "feature2": np.random.normal(1, 1.2, n),
    "feature3": np.random.normal(-0.5, 0.7, n),
})

# 여기에 Seaborn 플롯 별로 사용할 코드를 여기에 붙이기
# Copy & paste codes here!
```

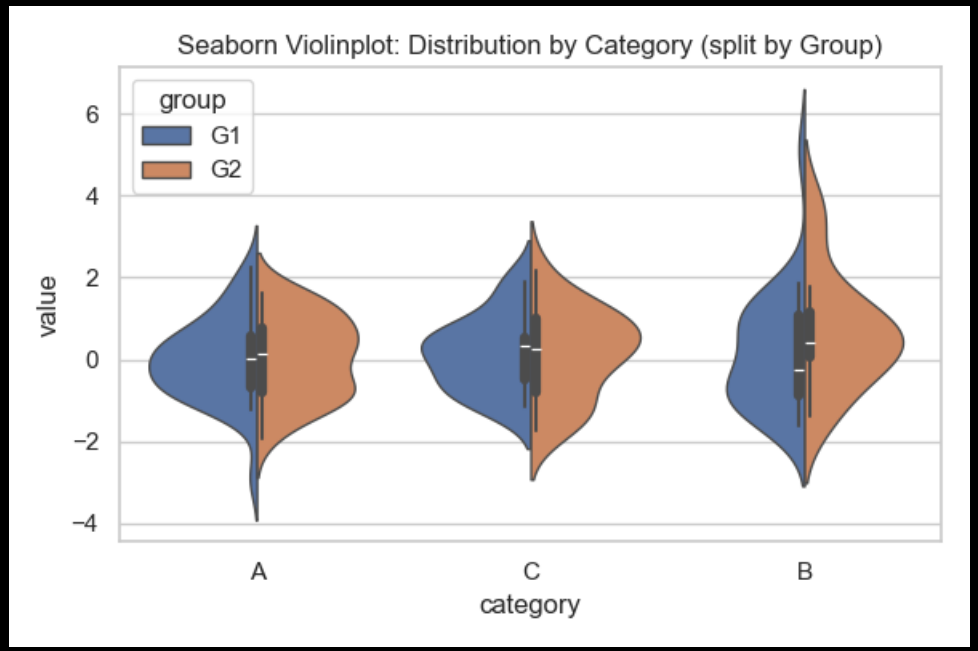
Violinplot

- 🎻 **Violin Plot (바이올린 플롯) 소개**
  - Boxplot + Kernel Density Estimation (KDE) 결합
  - 중앙값과 사분위수 정보 제공 + 분포 형태까지 시각화
  - 데이터의 중심 + 퍼짐 + 분포 모양을 한눈에 확인 가능
- **중앙 구조 (Boxplot 부분)**
  - 중앙 선 → 데이터의 중앙값
  - 박스 → 사분위 구간(Q1 ~ Q3, IQR)
  - 해석: 중앙값 위치 확인 → 분포 중심 파악
- 🎻 **바이올린 외곽 (KDE 부분)**
  - 곡선 두께 = 데이터 밀집도
  - 두꺼운 부분 → 데이터가 많이 모임, 얇은 부분 → 데이터가 드묾
  - Boxplot보다 실제 분포 형태를 더 잘 보여줌
- **그룹 비교 (hue + split=True)**
  - 두 그룹을 하나의 축에서 좌우 분리 표현
  - 비교 포인트:각 그룹의 밀집 구간대칭 여부
  - 한쪽 치우침 여부

seaborn.violinplot:
https://seaborn.pydata.org/generated/seaborn.violinplot.html

Violinplot 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Violinplot: 분포 모양과 중앙값, 사분위수 표현
# split=True → hue가 두 그룹일 때 두 그룹을 한 축에 나누어 그림

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# Violinplot 그리기
sns.violinplot(
    data=df, x="category", y="value", hue="group", split=True,
    linewidth=1.0,  # 테두리 선 두께
    ax=ax
)

# 제목 설정
ax.set_title(
    "Seaborn Violinplot: Distribution \
by Category (split by Group)“
)

# 레이아웃 조정
fig.tight_layout()
plt.show()
```



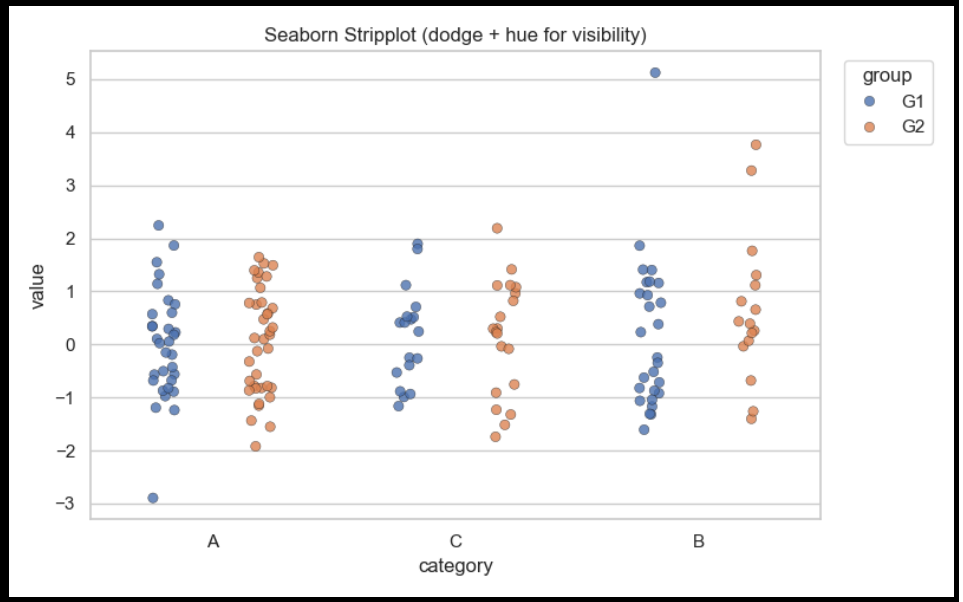
Stripplot

- **Stripplot (스트립 플롯)**
  - 범주형 변수 → 가로축, 수치형 값 → 세로축
  - 개별 데이터 점(point)을 그대로 표시 → 실제 분포 직관적 확인 가능
- **특징**
  - 점들이 세로로 모여 → 데이터 분포 상태 표현
  - 특정 구간에 몰림 → 데이터 집중
  - 고르게 퍼짐 → 균등 분포
- **옵션**
  - jitter: 점들을 가로로 흩뿌려 겹침 해소 (값 자체는 변하지 않음)
  - alpha: 점의 투명도 조절 → 밀집 구간 강조
- **해석 포인트**
  - 세로 위치 → 데이터 범위 파악
  - 점 밀집 정도 → 특정 구간 집중 여부 확인
  - jitter/alpha 적용 여부 → 분포 해석 용이성 향상

seaborn.stripplot:
https://seaborn.pydata.org/generated/seaborn.stripplot.html

Stripplot 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# Stripplot 그리기
sns.stripplot(
    data=df, x="category", y="value", hue="group",
    dodge=True,          # 그룹을 좌우로 분리
    jitter=True,         # 겹침 방지
    size=6,              # 마커 크기 키움
    alpha=0.8,           # 반투명
    edgecolor="k",       # 테두리로 대비 강화
    linewidth=0.3,
    ax=ax
)
ax.set_title("Seaborn Stripplot \
(dodge + hue for visibility)")
ax.legend(
    title="group",                  # 범례 제목을 "group"으로 표시
    bbox_to_anchor=(1.02, 1),       # 범례 위치: 그래프 오른쪽 바깥쪽에 배치
    loc="upper left"                # 범례의 기준점을 왼쪽 위 모서리로 설정
                                    # 좌표축 오른쪽 끝보다 살짝 오른쪽 (1.02),
                                    # 그리고 맨 위 (1) 지점
)

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



Swarmplot

Swarmplot

- Stripplot과 유사하지만, 점이 겹치지 않도록 자동 배치
- 범주(category)별로 데이터의 실제 값을 모두 표시
- 벌집 형태 배치 → 빈도와 분포 패턴을 명확히 관찰 가능

특징 & 활용

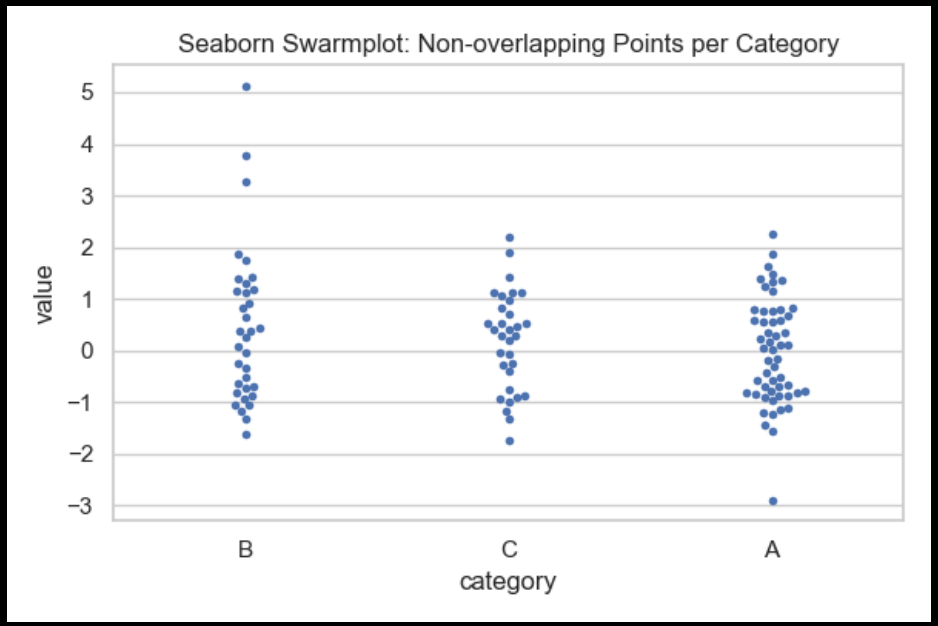
- 소규모 데이터에서 모든 데이터 포인트 확인에 유리
- 그룹 간 분포 비교에 효과적 (hue 옵션 지원)
- Stripplot보다 해석이 더 명확

한계

- 데이터가 많으면 점이 과도하게 확산 → 해석 어려움

Swarmplot 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# Swarmplot 그리기
# Swarmplot: Stripplot과 유사하지만, 점이 겹치지 않게 자동 배치
# sample(120) → 데이터 일부 샘플링 (성능 고려)
sns.swarmplot(
    data=df.sample(120, random_state=0),
    x="category", y="value",
    size=4, linewidth=0,   # 점 크기와 테두리 제거
    ax=ax
)

ax.set_title(
    "Seaborn Swarmplot: \
Non-overlapping Points per Category“
)
fig.tight_layout()
plt.show()
```



Barplot

Barplot

- 범주형 데이터 → 막대(bar)로 집계 통계량 표현
- 기본: 평균(mean) 높이 표시 + 신뢰구간(CI) 함께 시각화
- hue 옵션 → 범주 내 그룹 간 평균 차이 비교 가능

seaborn.barplot:
https://seaborn.pydata.org/generated/seaborn.barplot.html

특징

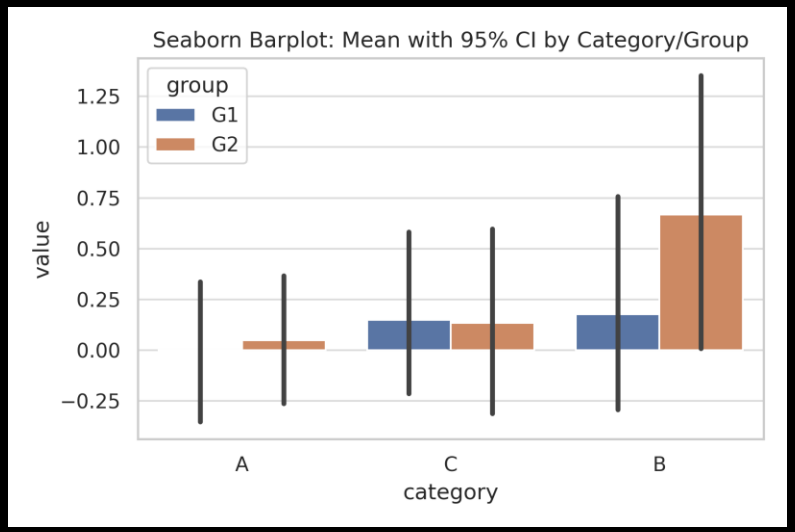
- 막대 높이 → 범주의 대표 값 (평균/중앙값/합계 등 estimator로 변경 가능)
- 신뢰구간(CI) → 검은색 실선으로 표시, 데이터 변동성과 불확실성 표현
  - 신뢰구간 겹침 → 차이가 유의하지 않을 가능성
  - 신뢰구간 분리 → 차이가 뚜렷할 수 있음

해석

- 포인트막대 높이 → 범주별 대표 값 확인
- 신뢰구간 → 데이터 변동성 & 그룹 간 차이 해석
- 그룹 막대 비교(hue) → 범주 내 그룹 관계 분석

Barplot 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# Barplot 그리기
sns.barplot(
    data=df,            # 사용할 데이터프레임 (df)
    x="category",       # x축에 표시할 변수 (범주형 변수)
    y="value",          # y축에 표시할 변수 (수치형 변수)
    hue="group",        # 범주 그룹을 나눌 변수 (막대를 그룹별로 색상 분리)
    estimator=np.mean, # y값에 대해 어떤 통계량을 계산할지 지정 (기본값은 평균 np.mean)
    ci=95,              # 신뢰구간(confidence interval) 설정 (여기서는 95% CI)
    ax=ax               # 그릴 matplotlib Axes 객체 (subplot에 그릴 때 사용)
)

ax.set_title(
  "Seaborn Barplot: \
Mean with 95% CI by Category/Group“
)

plt.show()
```



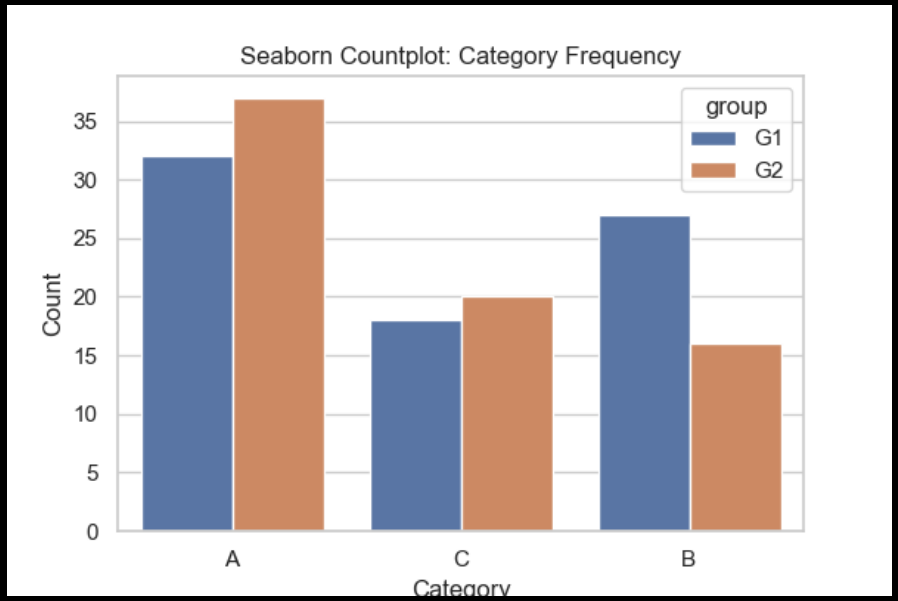
Countplot

- **Countplot**
  - 범주형 데이터의 빈도(횟수)를 막대 높이로 표현
  - Barplot과 달리 평균·통계값이 아닌 단순 빈도를 보여줌
- **특징**
  - 가로축: 범주(category)
  - 세로축: 빈도(count)
  - hue 옵션 → 범주 내 그룹별 분포 비교 가능
- **활용**
  - 범주형 변수 분포 파악
  - 클래스 불균형(class imbalance) 확인
  - 데이터 전처리 및 분류 문제 준비 단계에서 자주 사용
- **해석**
  - 포인트막대 높이 → 각 범주의 데이터 수 비교
  - 그룹 지정 시 → 그룹 간 상대적 비율 파악
  - 전체 데이터셋 내 불균형 여부 확인

seaborn.countplot:
https://seaborn.pydata.org/generated/seaborn.countplot.html

Countplot 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# Countplot 그리기
sns.countplot(
    data=df,         # 사용할 데이터프레임 (df)
    x="category",    # x축에 표시할 변수 (범주형 변수, 카테고리별 빈도를 계산)
    hue="group",     # 그룹 변수 (각 카테고리별로 그룹을 나눠 색상 구분)
    ax=ax            # 그릴 matplotlib Axes 객체 (subplot에 그릴 때 사용)
)

ax.set_title(
  "Seaborn Countplot: \
Category Frequency“
)
ax.set_xlabel("Category")
ax.set_ylabel("Count")
plt.show()
```



Heatmap

Heatmap

- 데이터 행렬을 색상으로 시각화 → 값의 크기를 직관적으로 비교
- 주로 상관관계 행렬(correlation matrix) 시각화에 활용
- 표보다 패턴·경향성을 쉽게 파악 가능

특징

- 가로축·세로축: 변수명
- 셀 색상: 두 변수 간 관계 (예: 상관계수)
  - +1에 가까움 → 강한 양의 상관관계
  - -1에 가까움 → 강한 음의 상관관계
  - 0 근처 → 선형 관계 없음

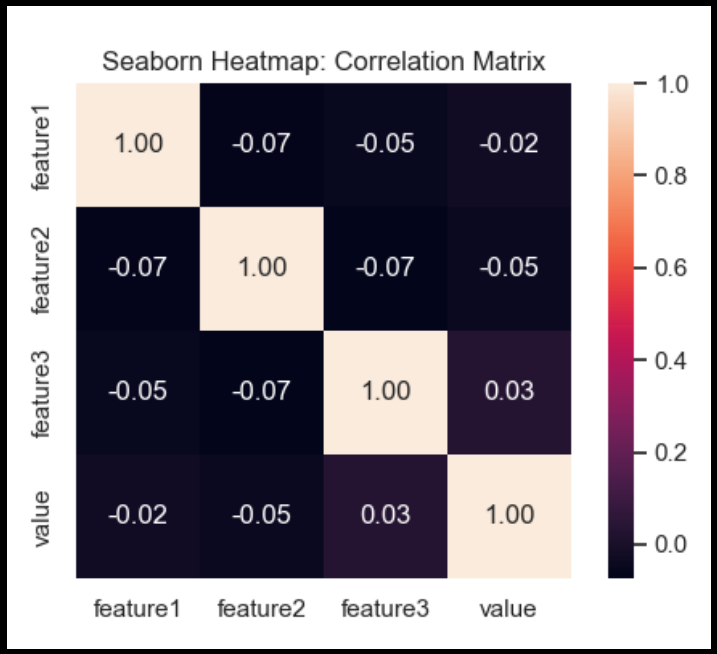
활용

- 변수 선택 및 다중공선성(multicollinearity) 탐지
- 변수 간 관계 이해 및 불필요/중복 변수 제거
- 보고서·프레젠테이션에서 패턴을 직관적으로 전달

seaborn.heatmap:
https://seaborn.pydata.org/generated/seaborn.heatmap.html

Heatmap 예시

```
# Code snippet -> 복사하여 공통 헤더 블록의 플롯 별로 사용할 코드 영역에 붙이기

# Figure와 Axes 객체 생성
fig, ax = plt.subplots(figsize=(6, 4))

# 상관계수 행렬 계산: 각 변수(feature1, feature2, feature3, value) 간의 상관관계
#   상관계수: -1.0 ~ 1.0 사이 실수값
corr = df[["feature1", "feature2", "feature3", "value"]].corr()

# Heatmap 그리기
sns.heatmap(
    corr,         # 입력 데이터 (상관계수 행렬)
    annot=True,   # 각 셀에 숫자(상관계수 값)를 표시
    fmt=".2f",    # 소수점 둘째 자리까지 표시
    square=True, # 각 셀을 정사각형으로 표시
    ax=ax         # matplotlib Axes 객체 (subplot에 그릴 때 사용)
)

ax.set_title("Seaborn Heatmap: Correlation Matrix")
plt.show()
```



Pairplot

Pairplot

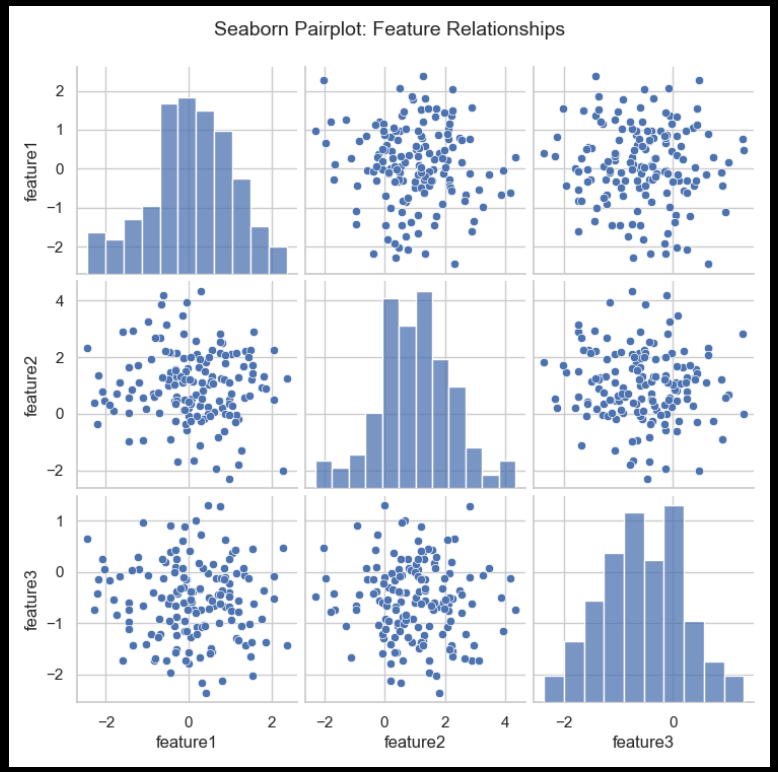
- 변수 간 관계를 산점도(Scatter plot)와 히스토그램으로 시각화
- 셀 = 두 변수 간 관계
  - 점이 직선 형태 → 강한 선형 관계
  - 점이 무작위 분포 → 선형 관계 없음
- 대각선 = 각 변수의 히스토그램(분포 특성)

활용

- 탐색적 데이터 분석(EDA) 단계에서 자주 사용
- 다중공선성(multicollinearity) 가능성 점검
- 변수 간 선형/비선형 관계 확인
- hue 옵션 → 범주별 데이터 분포 차이 시각화 가능

Pairplot 예시

```
sub = df[["feature1", "feature2", "feature3"]]

# Pairplot 그리기
#   - 변수 쌍별로 산점도(scatter plot)를 그림
#   - 대각선(diagonal)에는 히스토그램(histogram)을 표시
#   - corner=True → 하삼각형 부분만 표시 (중복 제거)
#   - corner=False → 상삼각형과 하삼각형 모두 표시 (모든 변수 쌍)
g = sns.pairplot(
    # 시각화할 DataFrame (수치형 변수만 선택)
    sub,
    # 대각선에 표시할 그래프 종류 ("hist" 또는 "kde")
    diag_kind="hist",
    # 변수 쌍을 모두 표시할지 여부
    # (False면 전체, True면 절반만)
    corner=False
)

# 제목 추가
g.figure.suptitle(
  "Seaborn Pairplot: \
Feature Relationships",
   fontsize=14
)

# 여백 조정 (top을 줄여 제목 공간 확보)
g.figure.subplots_adjust(top=0.92)

plt.show()
```



사용자 데이터 분석

Seaborn 내장 데이터를 활용한 시계열 시각화

■ **Flights 데이터셋 시각화**

- Dataset: Seaborn 내장 flights (1949~1960, 월별 항공 여객 수)
    - 장기적 증가 추세 + 계절적 변동성 관찰 가능

■ **시각화 목표**

- 연도별 월별 변화 패턴 확인
- 월별 장기적 추세 파악

■ **접근 방법**

- Long-format
    - X축=월, Y축=승객 수, 색상=연도
    - 👉 연도별 계절 패턴 비교
- Wide-format
    - X축=연도, 각 월을 시계열로 표시
    - 👉 월별 장기 추세 분석

■ ✅ **핵심 인사이트**

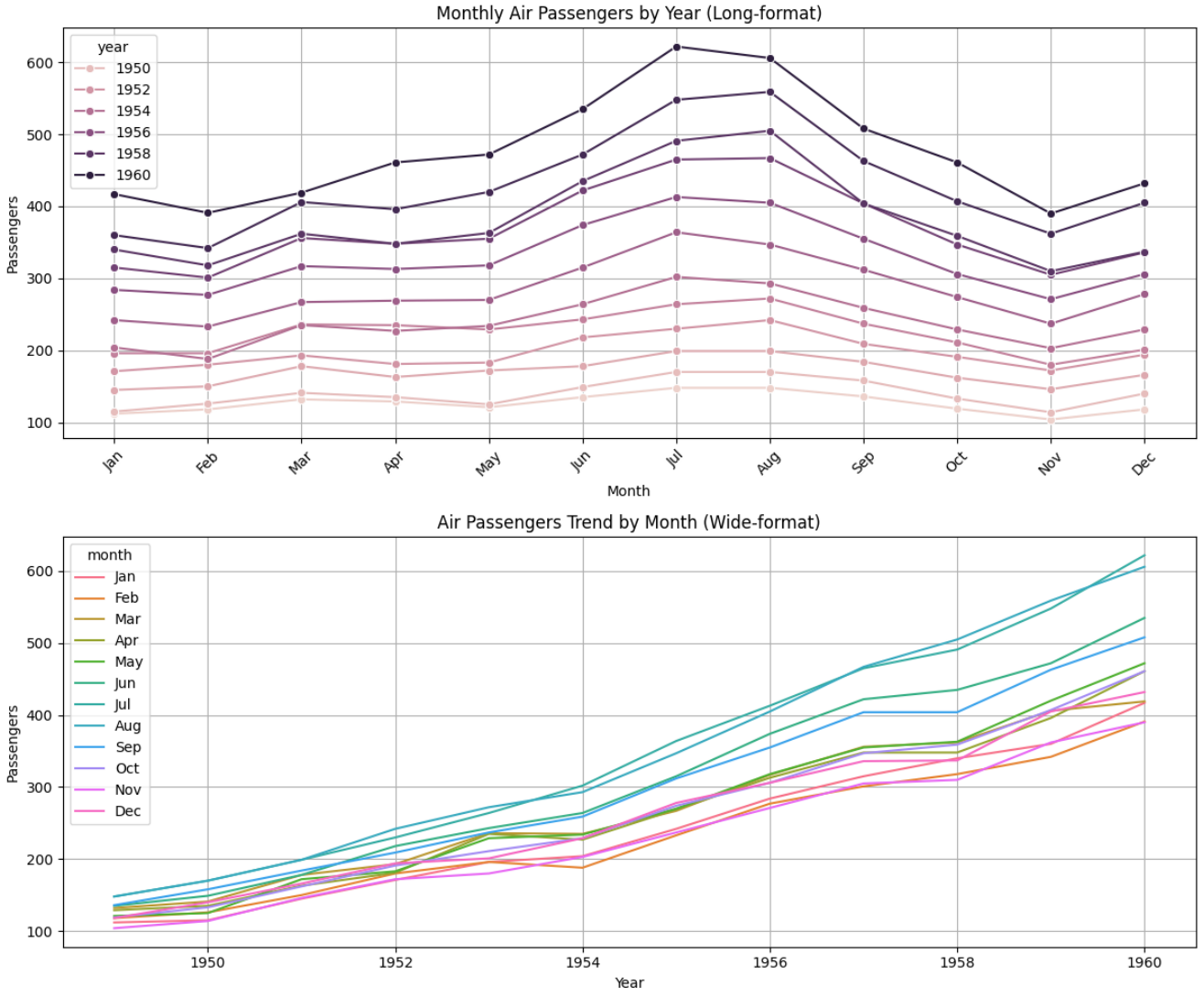
- 동일한 데이터라도 포맷 선택에 따라 해석이 달라짐

Seaborn 내장 데이터를 활용한 시계열 시각화 – 샘플 코드

```python
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Seaborn 내장 데이터 불러오기
flights = sns.load_dataset("flights")   # 항공 승객 수 데이터

# flights 데이터를 wide-format으로 변환
flights_wide = flights.pivot(
    index="year",       # 행 인덱스: 연도
    columns="month",    # 열: 월
    values="passengers" # 값: 승객 수
)

# 변환 결과 예시:
#   DataFrame은 각 열이 한 달을 나타내며, 행은 연도별 승객 수를 기록
#   "월별 데이터의 연도별 흐름"을 동시에 파악

# month   January  February  March  ...  November  December
# year
# 1949        112       118    132  ...       104       118
# 1950        115       126    141  ...       114       140
# ...

# subplot 2행 1열 배치
fig, axes = plt.subplots(2, 1, figsize=(12, 10))
```

```python
# (1) Long-format 데이터 시각화
# 시계열 데이터 시각화
# relplot: "relationship plot"의 줄임말
# 산점도나 선 그래프를 그릴 때 사용되는 Seaborn의 범용 함수
sns.lineplot(
    data=flights,  # 사용할 데이터프레임
                   # (flights: 1949~1960 항공 여객 데이터)

    x="month",     # X축: 월(month) → 1월 ~ 12월
    y="passengers",# Y축: 승객 수(passengers)
    hue="year",    # 색상 구분: 연도별(year)로 색을 다르게 표시
    marker="o",    # 데이터 지점마다 동그란 마커(o)를 추가하여
                   # 값이 더 잘 보이게 함
    ax=axes[0],    # 첫 번째 subplot(axes[0])에 그래프 그리기
)
axes[0].set_title("Monthly Air Passengers by Year (Long-format)")
axes[0].set_xlabel("Month")
axes[0].set_ylabel("Passengers")
axes[0].tick_params(axis="x", rotation=45)
axes[0].grid(True)

# (2) Wide-format 데이터 시각화
sns.lineplot(
    data=flights_wide,
    dashes=False,  # 각 월별 시계열 구분이 명확하게 보이도록 설정
    ax=axes[1]
)
axes[1].set_title("Air Passengers Trend by Month (Wide-format)")
axes[1].set_xlabel("Year")
axes[1].set_ylabel("Passengers")
axes[1].grid(True)
plt.tight_layout()
plt.show()
```

Seaborn 내장 데이터를 활용한 시계열 시각화 – 실행 결과



Flights 데이터 시각화 요약

- Long-format 그래프
  - X축=월, Y축=승객 수, 색상=연도
  - 👉 연도별 월별 계절 패턴 비교에 적합
  - 특징: 여름(7~8월) 성수기, 연말 감소, 전체적으로 꾸준한 증가 추세
- Wide-format 그래프
  - X축=연도, Y축=승객 수, 선=월
  - 👉 월별 장기 추세 파악에 유용
  - 특징: 7, 8월 → 지속적 고수준(성수기), 1, 2월 → 낮은 값(비수기)

결론

- Long-format: 계절성(Seasonality) 분석에 효과적
- Wide-format: 장기 추세(Trend) 분석에 효과적
- 두 시각화를 함께 활용하면 계절성과 추세를 동시에 이해 가능

수고하셨습니다 ..^^..