

데이터프로그래밍 기초 9일차

2026-1 DS Bootcamp

부산대학교
데이터사이언스전문대학원
석사과정 박민서

CONTENTS

1 Matplotlib & Pyplot

2 Seaborn & Plotly

3 Practice (Data EDA)

■ 맷플롯립 (Matplotlib)

- 데이터 시각화 + 2D 그래프를 그릴 때 사용하는 Python 라이브러리
- <https://matplotlib.org/stable/gallery/index.html>
- `pip install matplotlib`



■ 파이플롯 (Pyplot)

- MATLAB 스타일의 함수 호출 방식으로 Python에서 그래프를 그릴 수 있게 하는 모듈

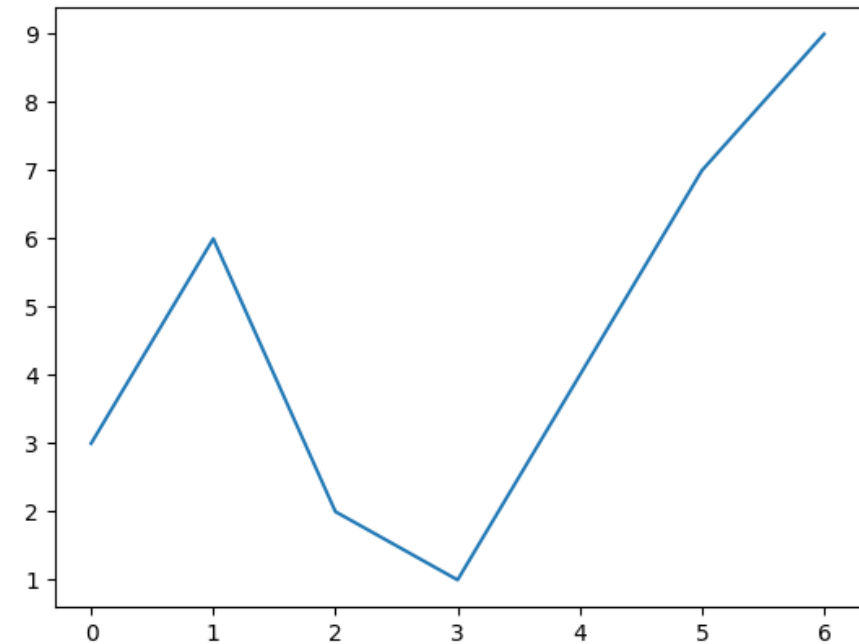
■ 기초 시각화

- .plot(데이터): 데이터를 2차원 line plot으로 표현
- 데이터의 종류: (요소가 전부 숫자인) list, array (numpy), DataFrame의 열

```
# 기본 시각화
import matplotlib.pyplot as plt

my_list = [3, 6, 2, 1, 4, 7, 9]
plt.plot(my_list)
plt.show() # 그래프 시각화
```

✓ 0.6s

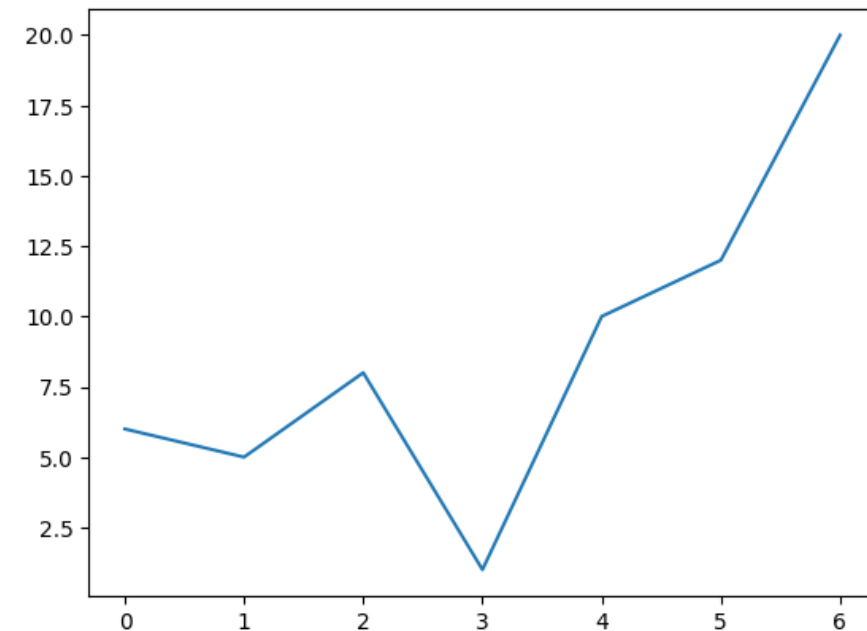


■ 기초 시각화

- `.plot(데이터)`: 데이터를 2차원 line plot으로 표현
- 데이터의 종류: (요소가 전부 숫자인) list, array (numpy), DataFrame의 열

```
my_arr = np.array([6, 5, 8, 1, 10, 12, 20])  
plt.plot(my_arr)  
plt.show()
```

✓ 0.0s

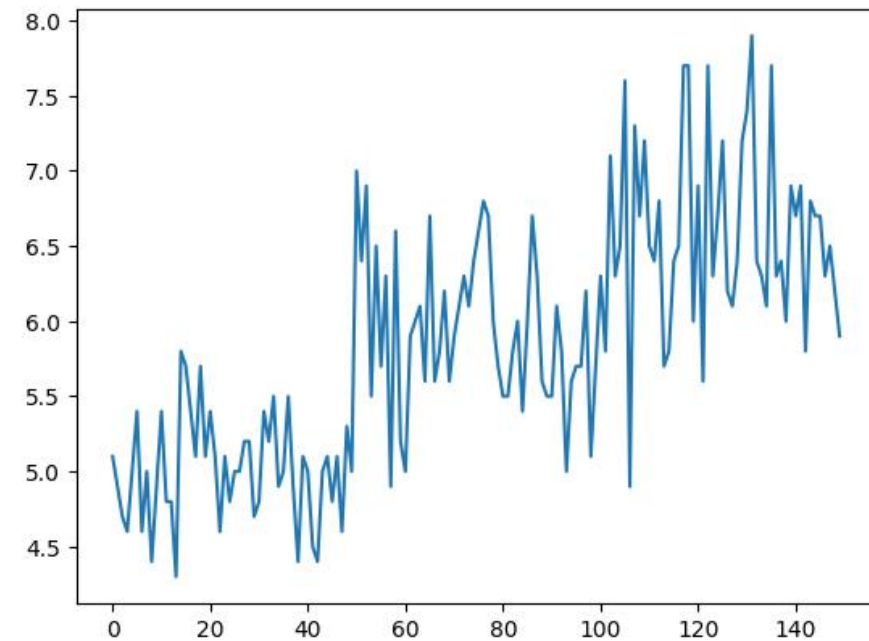


■ 기초 시각화

- .plot(데이터): 데이터를 2차원 line plot으로 표현
- 데이터의 종류: (요소가 전부 숫자인) list, array (numpy), DataFrame의 열

```
df = pd.read_csv("Iris.csv")  
  
plt.plot(df["SepalLengthCm"])  
plt.show()
```

✓ 0.1s

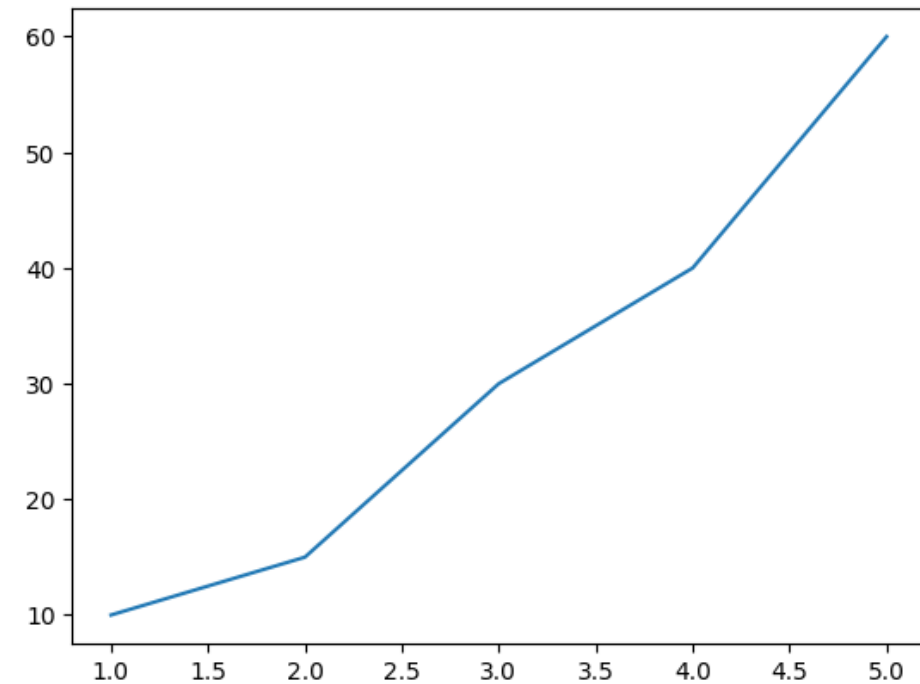


■ 기초 시각화

- `.plot(x값, y값)`: x, y값을 동시에 plot

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

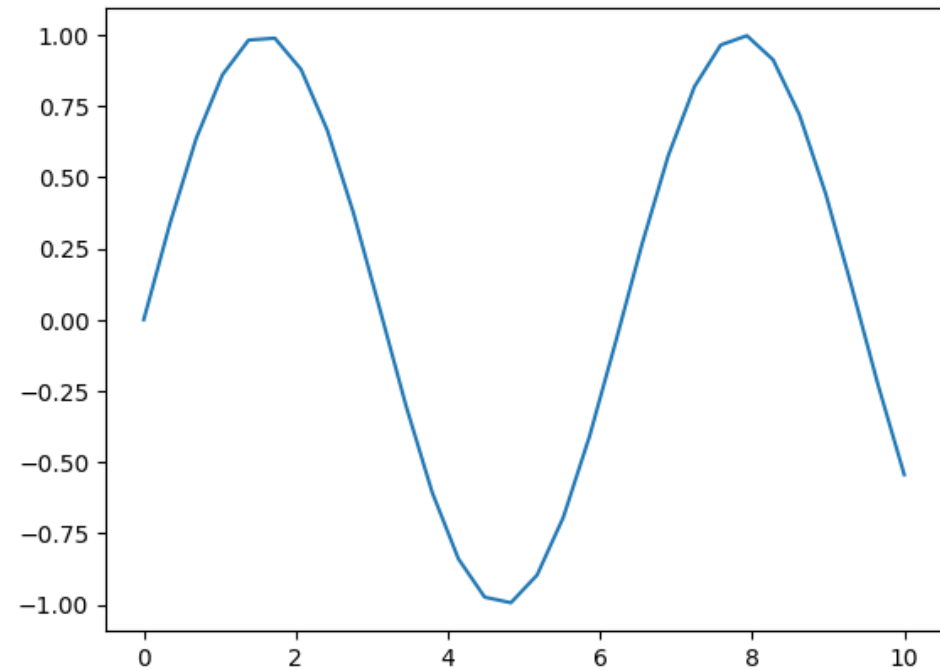
plt.plot(x, y)
plt.show()
```



■ 기초 시각화

- `.plot(x값, y값)`: x, y값을 동시에 plot

```
x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

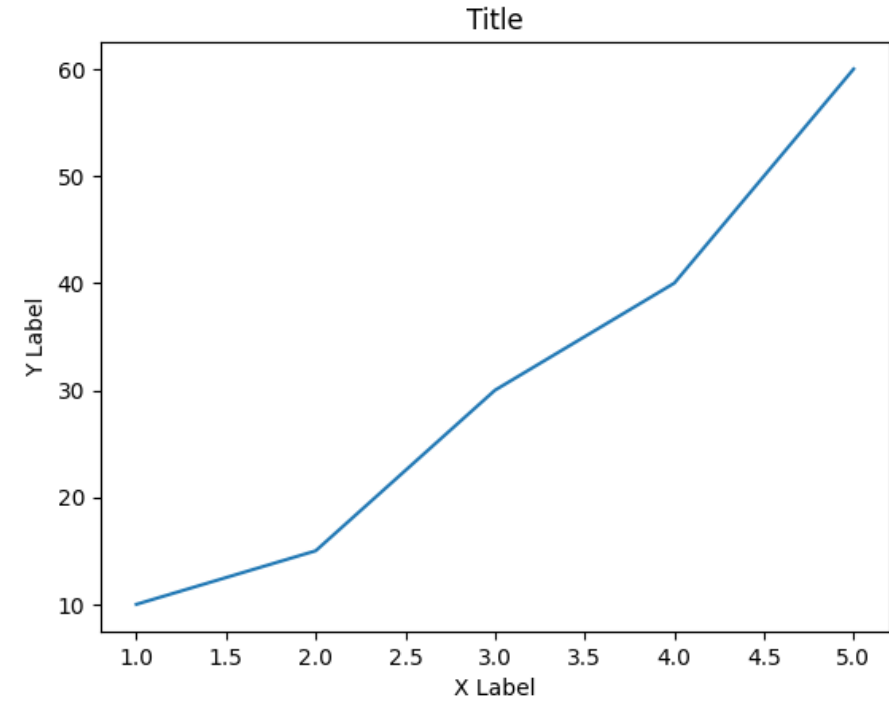


■ 기초 시각화

- `.xlabel("라벨 입력")`, `.ylabel("라벨 입력")`: x축, y축에 쓸 축 이름 생성
- `.title("제목 입력")`: 그래프 제목 생성

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y)
plt.xlabel("X Label")
plt.ylabel("Y Label") # 세로로 회전된 글
plt.title("Title")
plt.show()
```

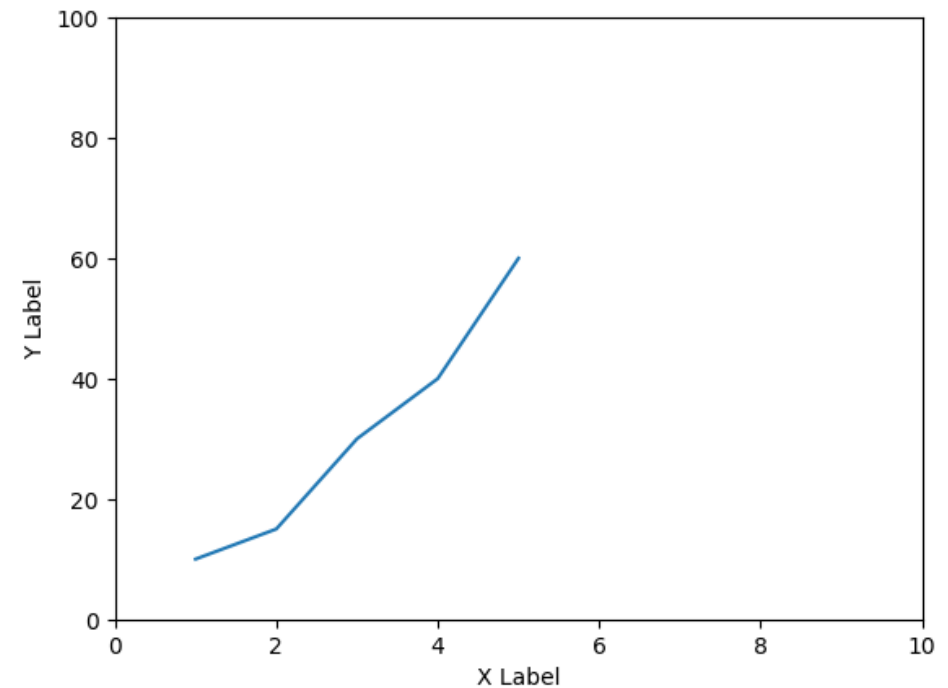


■ 기초 시각화

- `.xlim([시작, 끝])`, `.ylim([시작, 끝])`: x값과 y값의 범위를 제한

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y)
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.xlim([0, 10]) # 원본 x값의 범위 무시 -> 사용자 지정
plt.ylim([0, 100]) # 원본 y값의 범위 무시 -> 사용자 지정
plt.show()
```

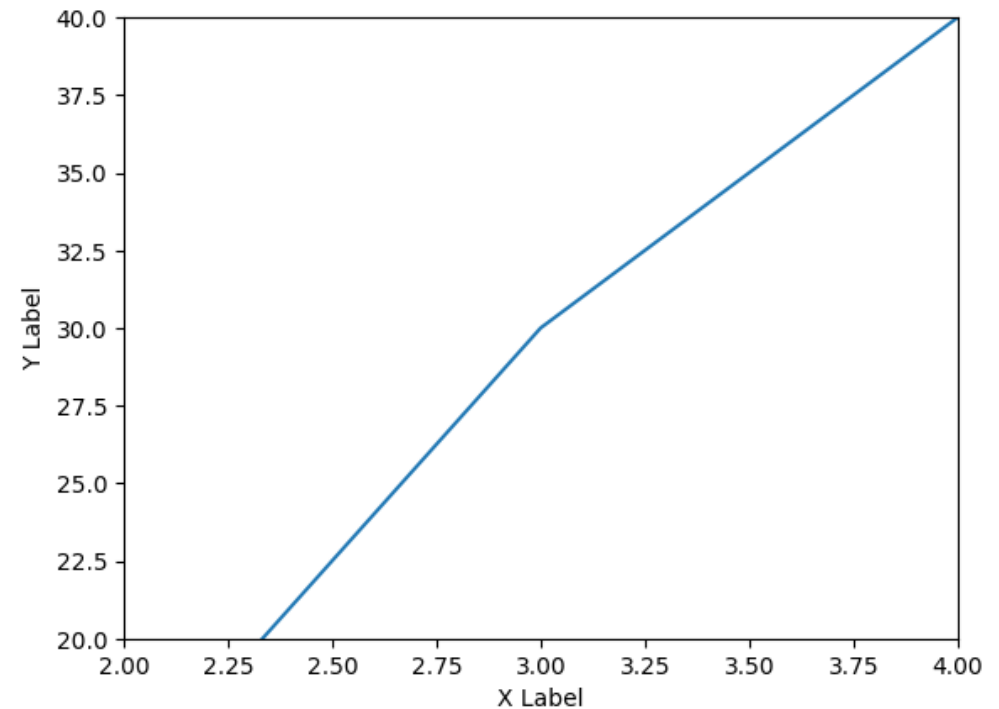


■ 기초 시각화

- `.xlim([시작, 끝])`, `.ylim([시작, 끝])`: x값과 y값의 범위를 제한

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y)
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.xlim([2, 4]) # 원본 x값의 범위 무시 -> 사용자 지정
plt.ylim([20, 40]) # 원본 y값의 범위 무시 -> 사용자 지정
plt.show()
```

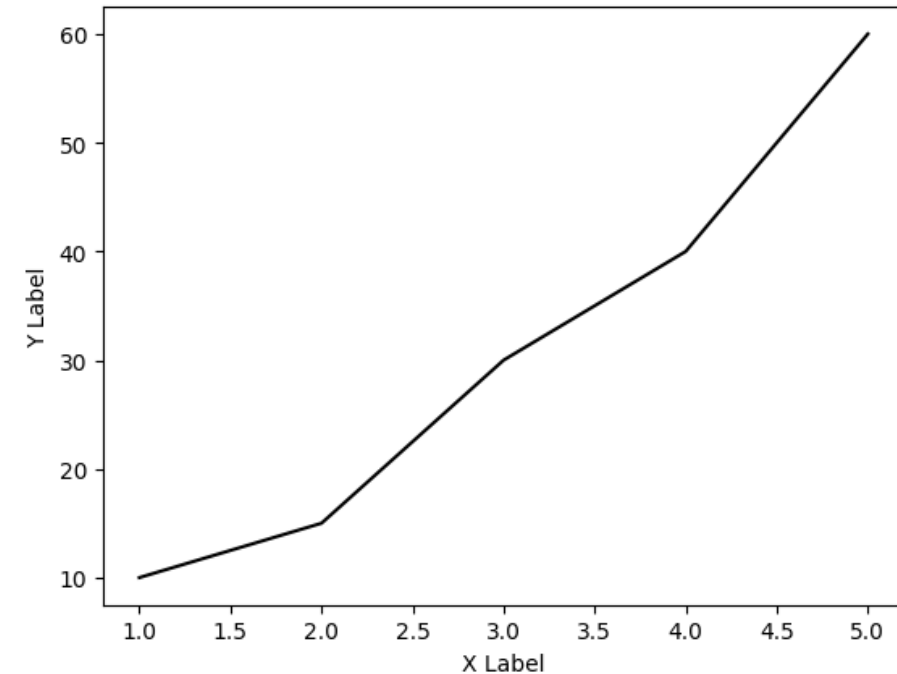


■ 기초 시각화

- `.plt(color = “색깔”)`: 그래프의 색을 지정
- ‘b’: 파랑, ‘g’: 초록, ‘r’: 빨강, ‘c’: 청록 (cyan),
- ‘m’: 마젠타 (magenta), ‘y’: 노랑, ‘k’:검정(?), ‘w’:흰색, <https://wikidocs.net/92085>

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, color="k") # 색깔 지정
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.show()
```

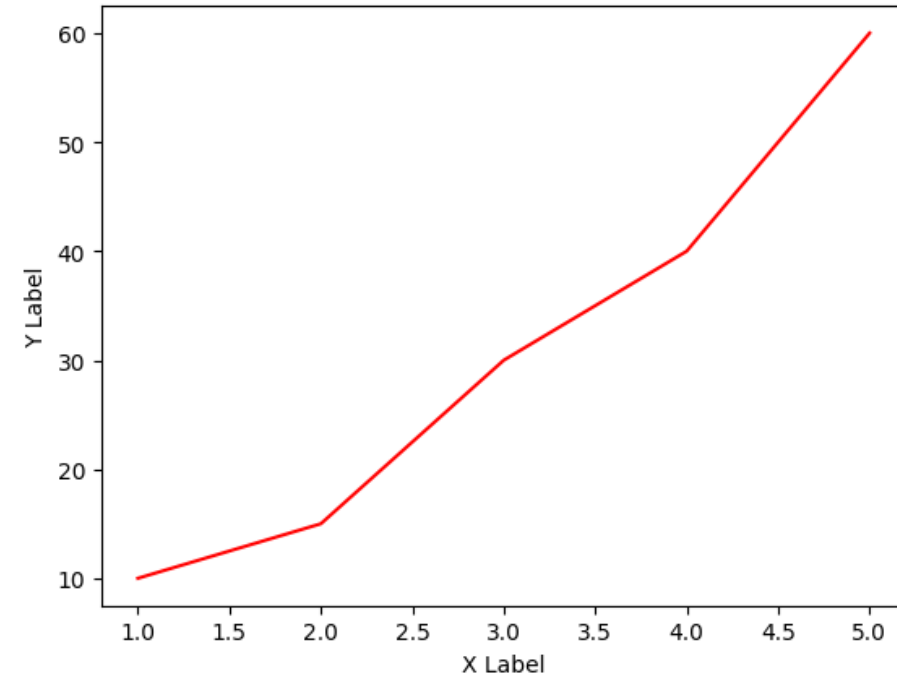


■ 기초 시각화

- `.plt(color = “색깔”)`: 그래프의 색을 지정
- ‘b’: 파랑, ‘g’: 초록, ‘r’: 빨강, ‘c’: 청록 (cyan),
- ‘m’: 마젠타 (magenta), ‘y’: 노랑, ‘k’:검정(?), ‘w’:흰색, <https://wikidocs.net/92085>

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, color="red") # 색깔의 이름도 가능
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.show()
```

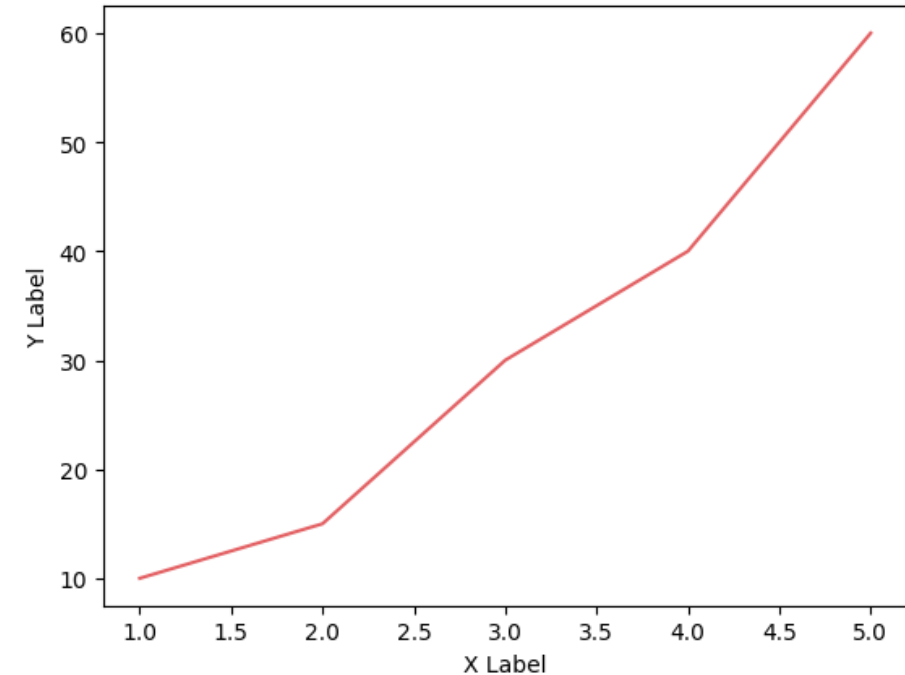


■ 기초 시각화

- `.plt(color = “색깔”)`: 그래프의 색을 지정
- ‘b’: 파랑, ‘g’: 초록, ‘r’: 빨강, ‘c’: 청록 (cyan),
- ‘m’: 마젠타 (magenta), ‘y’: 노랑, ‘k’:검정(?), ‘w’:흰색, <https://wikidocs.net/92085>

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, color='e35f62') # 16진수 색 코드도 사용 가능 (기본값 #1f77b4)
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.show()
```

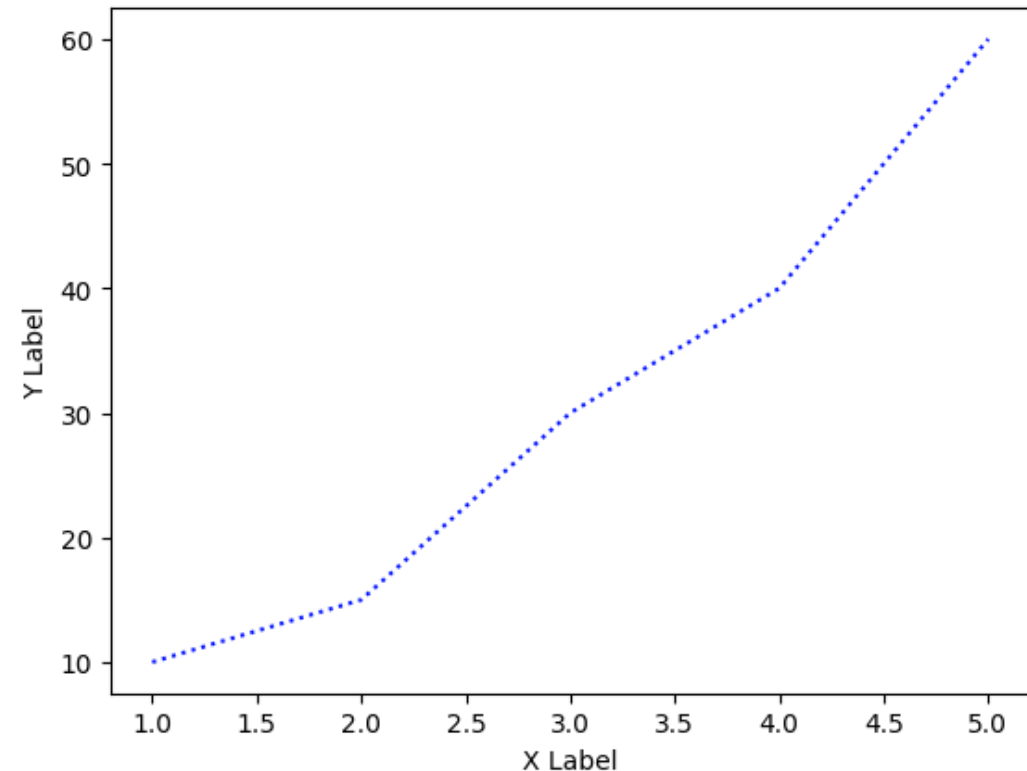


■ 기초 시각화

- `.plt(linestyle = “스타일”)`: 선 그래프의 스타일을 지정
- “-”: solid (실선, 기본값), “--”: dashed (파선), “-.”: dashdot (일점쇄선), “.”: dotted(점선)

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, color="#0011FF", linestyle = ".") # linestyle 지정(점선)
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.show()
```

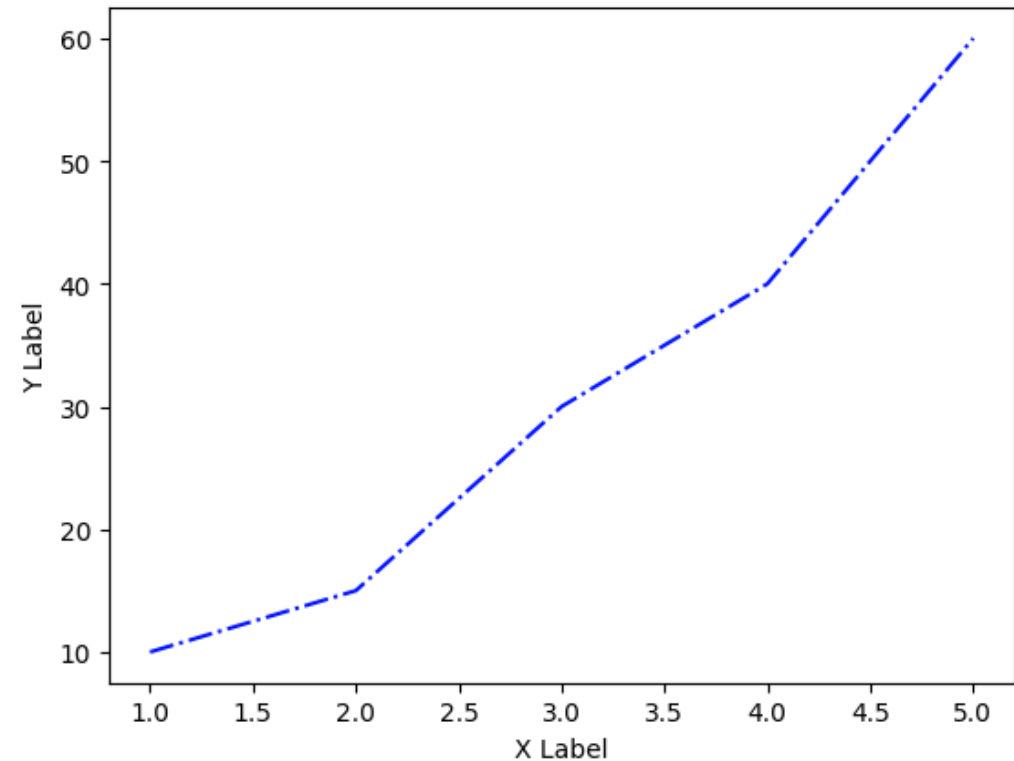


■ 기초 시각화

- `.plt(linestyle = "스타일")`: 선 그래프의 스타일을 지정
- `"-"`: solid (실선, 기본값), `--`: dashed (파선), `"-."`: dashdot (일점쇄선), `"."`: dotted(점선)

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, color="#0011FF", linestyle = "dashdot") # 이름으로도 가능
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.show()
```

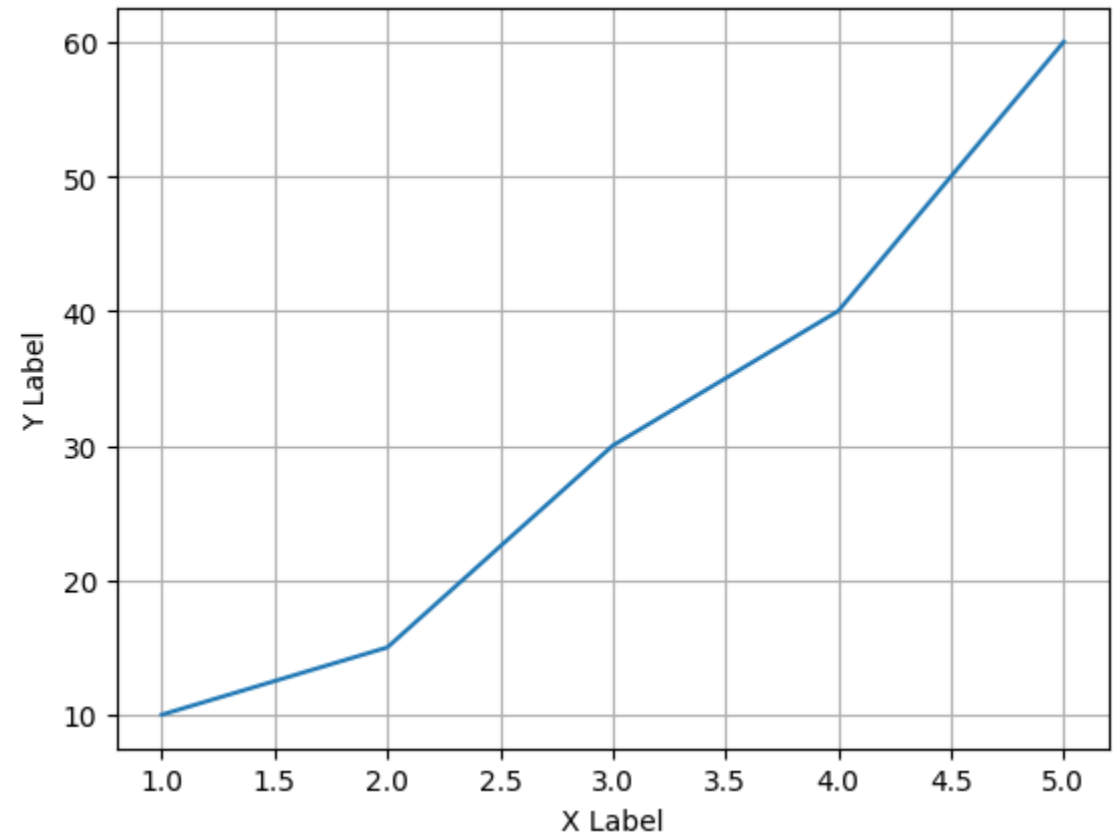


■ 기초 시각화

- `.grid()`: 그래프의 격자 그리는 설정 (기본값 True)

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y)
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.grid() # 격자 설정 (해제는 False)
plt.show()
```

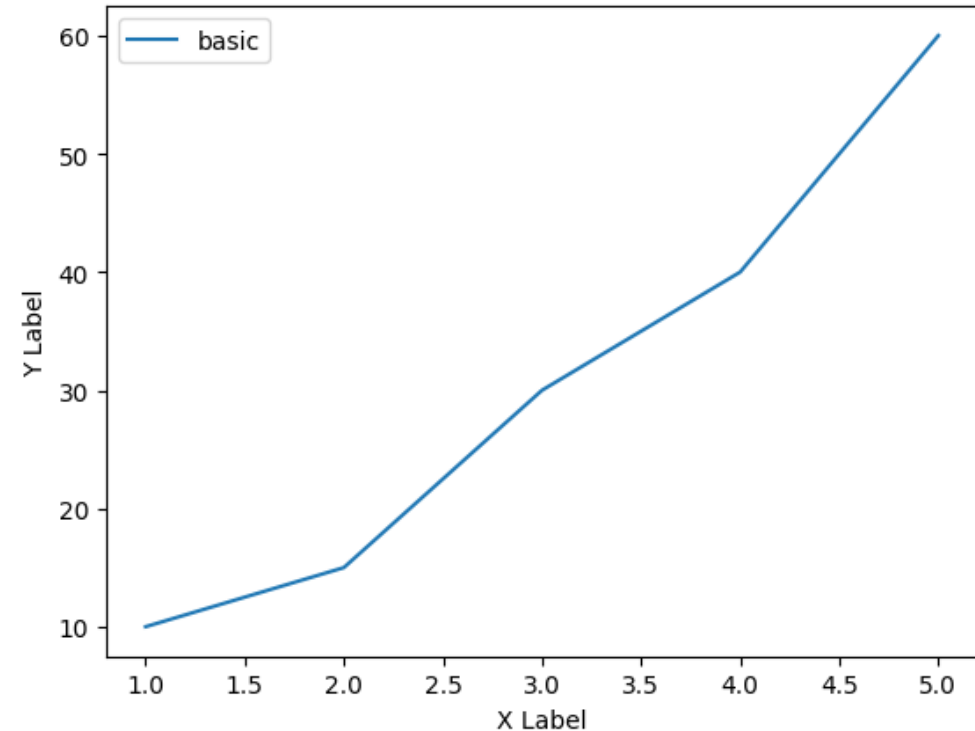


■ 기초 시각화

- `.legend()`: 그래프의 격자 그리는 설정
- `loc = [옵션]`: 범례 위치 설정 (기본은 "best")

```
x = [1, 2, 3, 4, 5]
y = [10, 15, 30, 40, 60]

plt.plot(x, y, label = "basic") # label = "원하는 라벨" 설정 후
plt.xlabel("X Label")
plt.ylabel("Y Label")
plt.legend() # 범례 설정 (loc = "best" 기본)
plt.show()
```



■ 기초 시각화

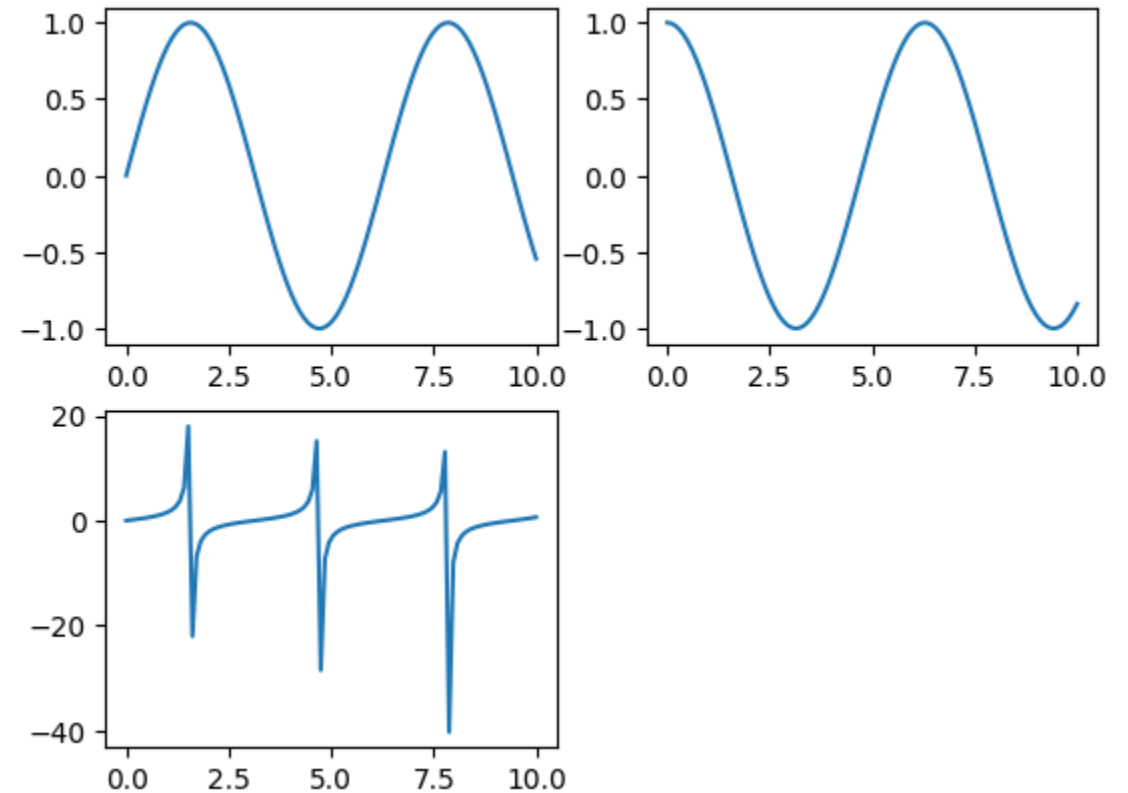
- `.subplot(행 개수, 열 개수, index[1~])`: 여러 개의 그래프 동시에 그리는 설정

```
x = np.linspace(0, 10, 100)
plt.subplot(2, 2, 1)    # 2행 2열의 위치 중 1번째 위치 (좌측 상단)
y1 = np.sin(x)
plt.plot(x, y1)

plt.subplot(2, 2, 2)    # 2행 2열의 위치 중 2번째 위치 (우측 상단)
y2 = np.cos(x)
plt.plot(x, y2)

plt.subplot(2, 2, 3)    # 2행 2열의 위치 중 3번째 위치 (우측 하단)
y3 = np.tan(x)
plt.plot(x, y3)

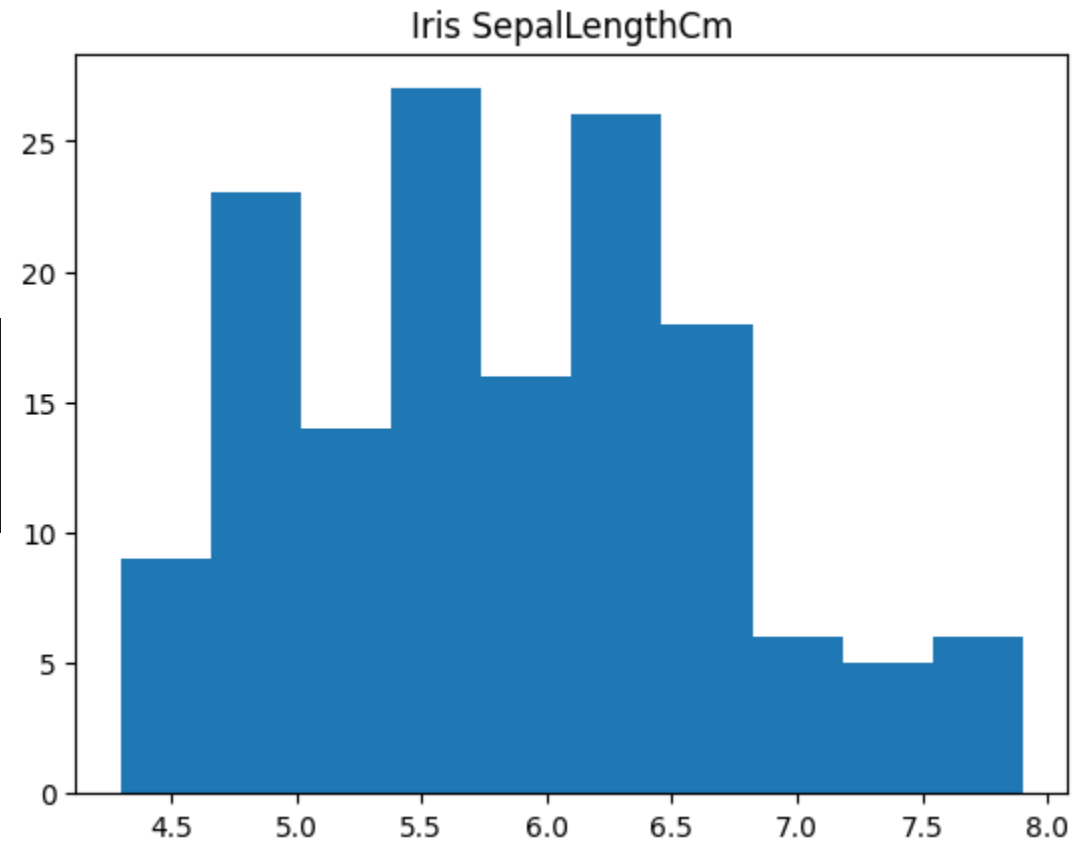
plt.show()
```



■ 기초 시각화

- .hist(데이터): 히스토그램 생성

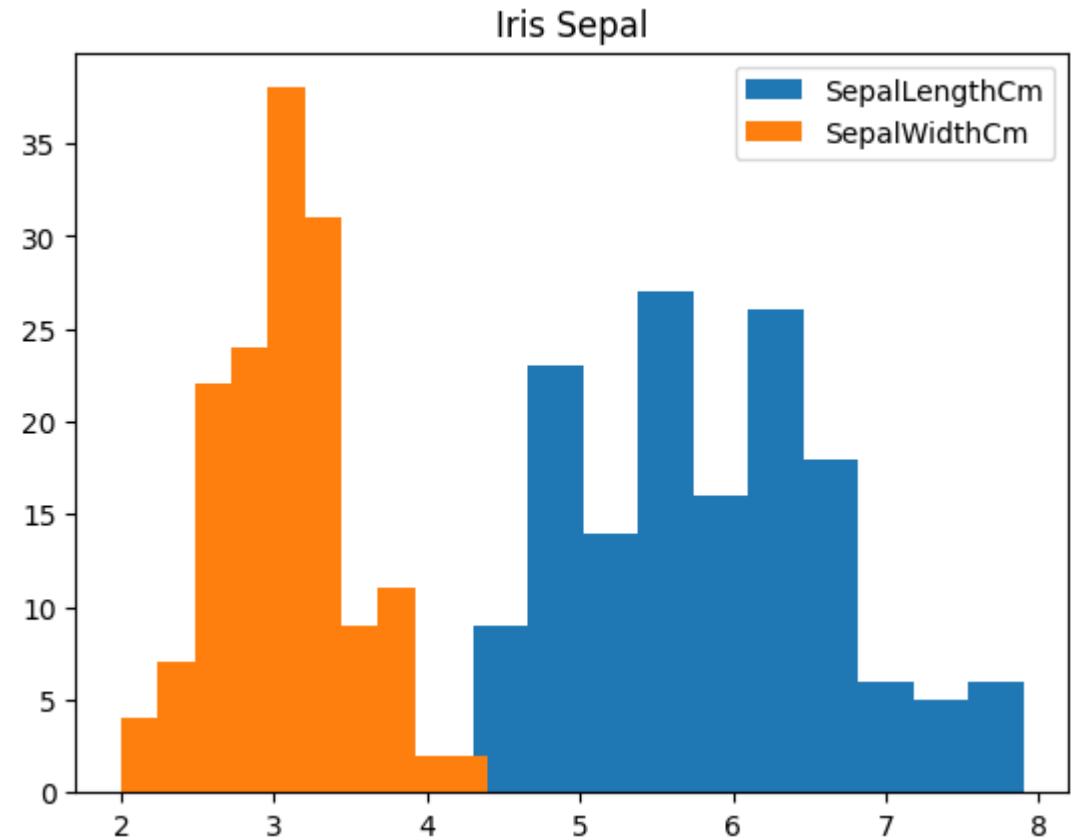
```
plt.hist(df["SepalLengthCm"], label = "SepalLengthCm")  
plt.title("Iris SepalLengthCm")  
plt.legend()  
plt.show()
```



■ 기초 시각화

- .hist(데이터): 히스토그램 생성

```
plt.hist(df["SepalLengthCm"], label = "SepalLengthCm")  
plt.hist(df["SepalWidthCm"], label = "SepalWidthCm")  
plt.title("Iris Sepal")  
plt.legend()  
plt.show()
```



■ 기초 시각화

- .hist(데이터): 히스토그램 생성
- histtype 옵션 지정:
 - “bar” – 막대그래프, “barstacked” – 막대 + 위로 쌓기, “step” – 계단식 (윤곽선만), “stepfilled” – 계단식 (색)

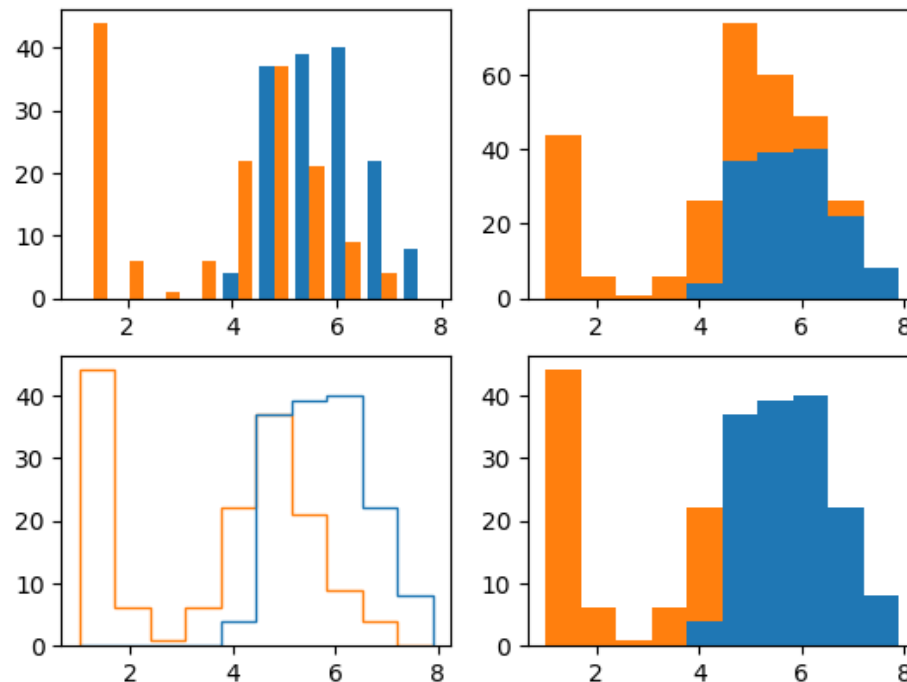
```
plt.subplot(2, 2, 1)    # 2행 2열의 위치 중 1번째 위치 (좌측 상단)
plt.hist((df["SepalLengthCm"], df["PetalLengthCm"]), histtype='bar')

plt.subplot(2, 2, 2)    # 2행 2열의 위치 중 2번째 위치 (우측 상단)
plt.hist((df["SepalLengthCm"], df["PetalLengthCm"]), histtype='barstacked')

plt.subplot(2, 2, 3)    # 2행 2열의 위치 중 3번째 위치 (우측 하단)
plt.hist((df["SepalLengthCm"], df["PetalLengthCm"]), histtype='step')

plt.subplot(2, 2, 4)    # 2행 2열의 위치 중 3번째 위치 (우측 하단)
plt.hist((df["SepalLengthCm"], df["PetalLengthCm"]), histtype='stepfilled')

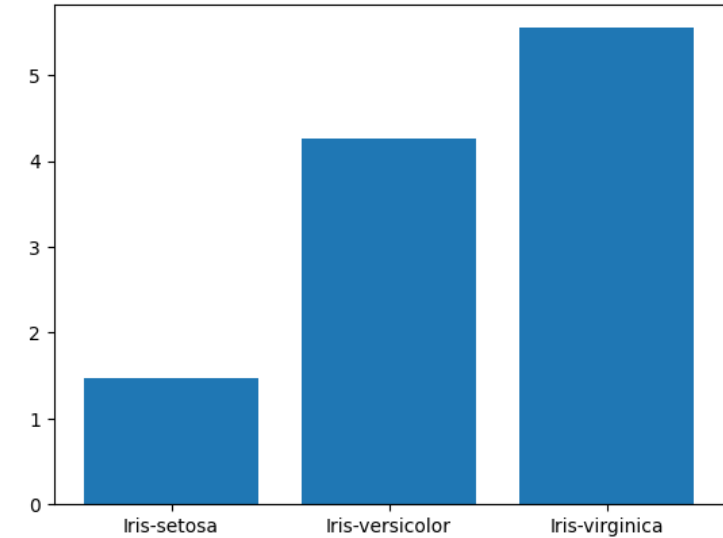
plt.show()
```



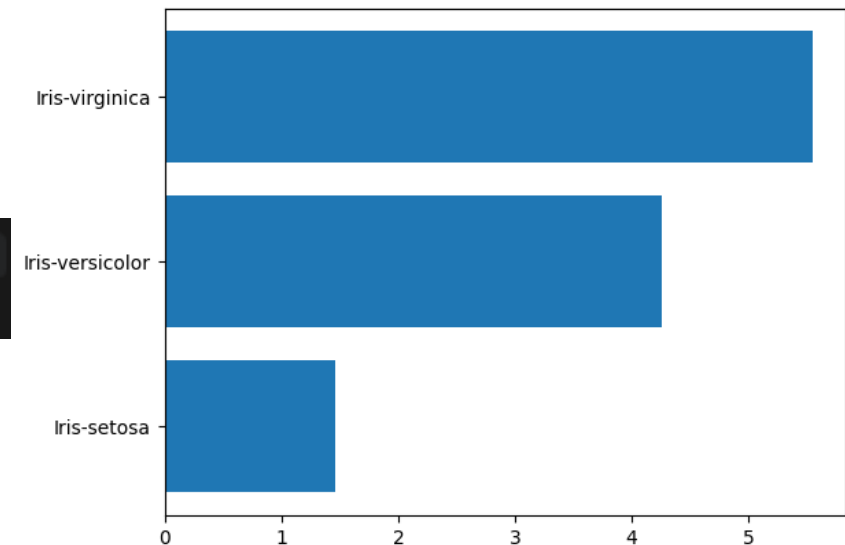
■ 기초 시각화

- `.bar(x, y)`: 막대그래프 생성
- `.barh(x, y)`: 수평 방향의 막대그래프 생성

```
plt.bar(df["Species"].unique(), df.groupby(["Species"])["PetalLengthCm"].mean())  
plt.show()
```



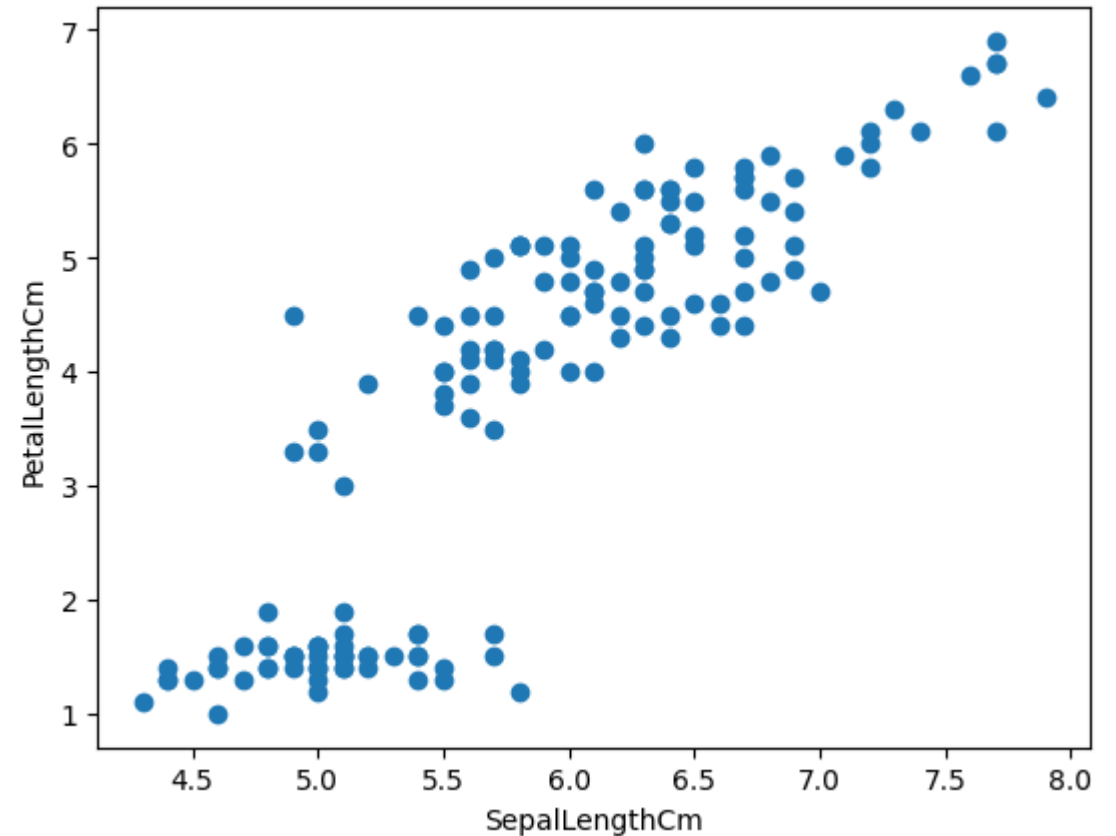
```
plt.barh(df["Species"].unique(), df.groupby(["Species"])["PetalLengthCm"].mean())  
plt.show()
```



■ 기초 시각화

- `.scatter(x, y)`: 산점도 생성

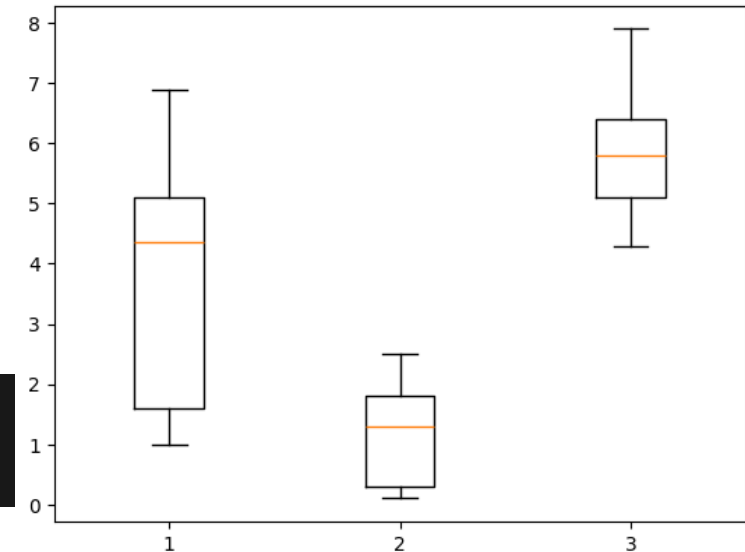
```
plt.scatter(df["SepalLengthCm"], df["PetalLengthCm"])  
plt.xlabel("SepalLengthCm")  
plt.ylabel("PetalLengthCm")  
plt.show()
```



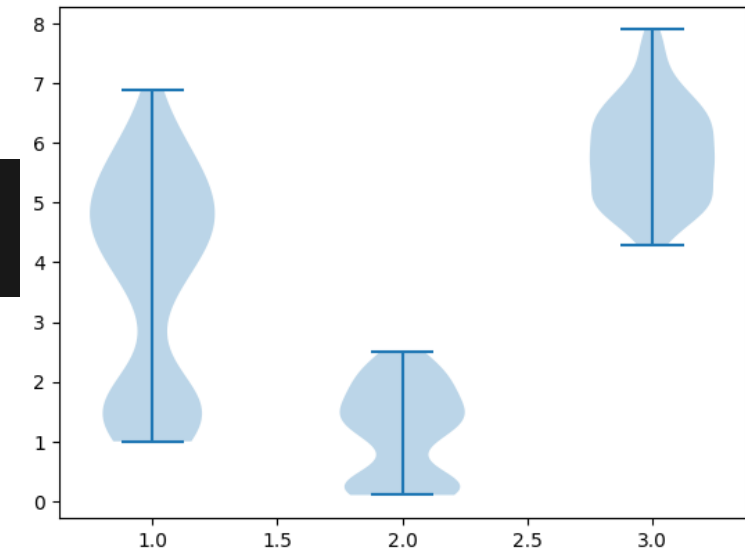
■ 기초 시각화

- .boxplot(데이터): 박스플롯 생성
- .violinplot(데이터): 바이올린 플롯 생성

```
plt.boxplot([df["PetalLengthCm"], df["PetalWidthCm"], df["SepalLengthCm"]])  
plt.show()
```



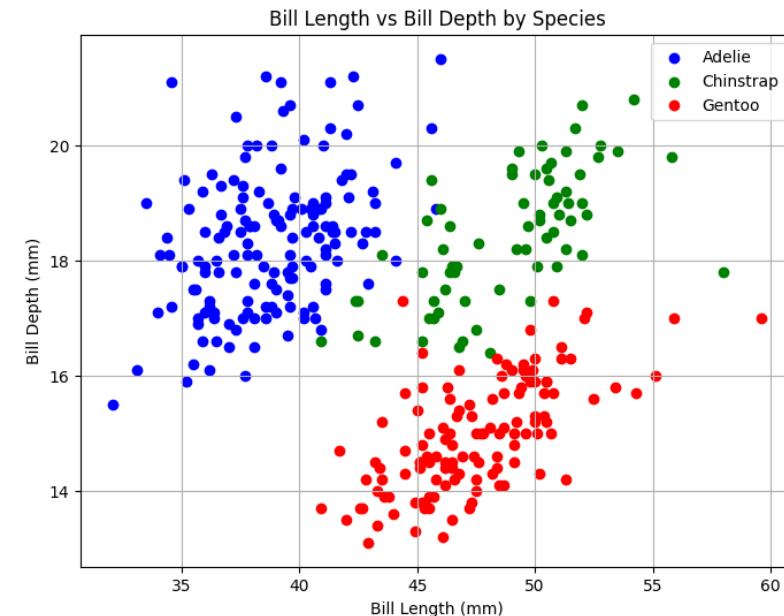
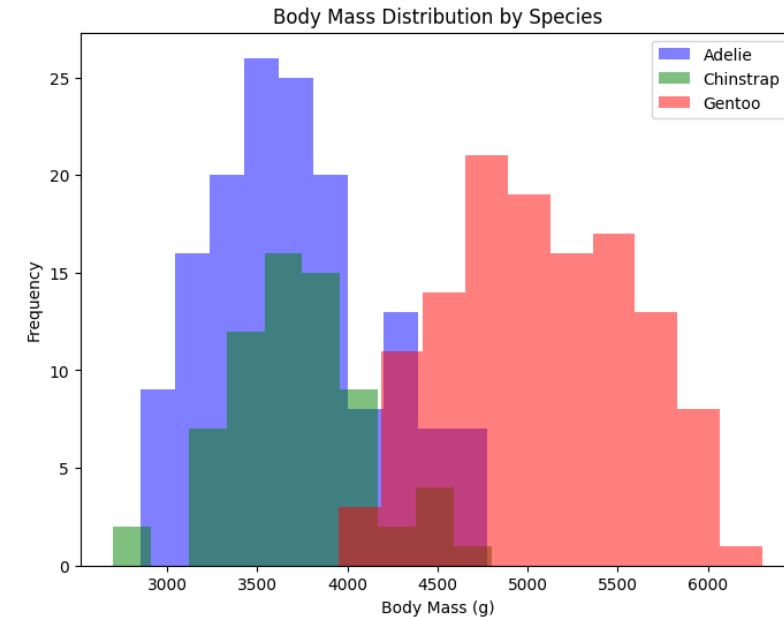
```
plt.violinplot([df["PetalLengthCm"], df["PetalWidthCm"], df["SepalLengthCm"]])  
plt.show()
```



Q&A

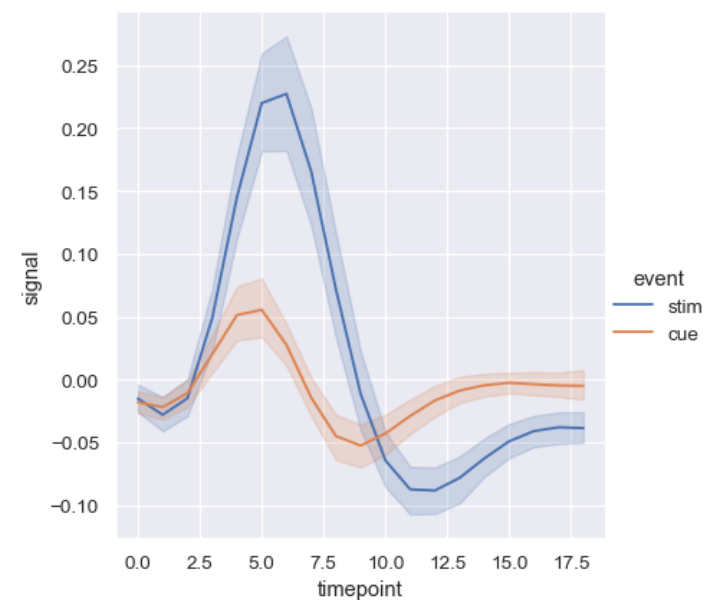
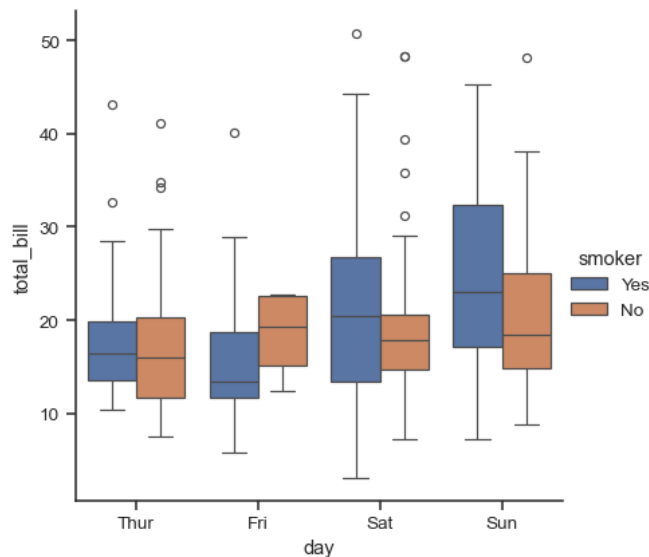
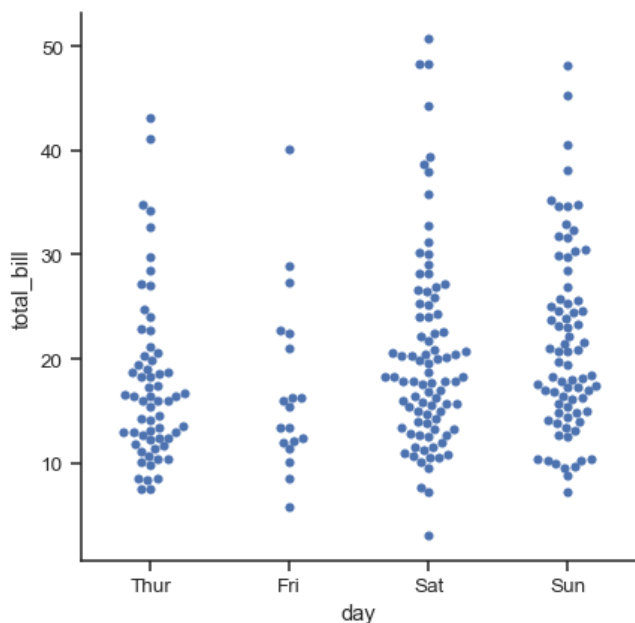
■ 실습 1

- “penguins_size.csv” 파일에 대해 다음 작업을 수행하세요.
 - species별 ‘body_mass_g’를 한 그래프에 겹쳐서 히스토그램으로 그리시오.
 - alpha (투명도) = 0.5로, 서로 다른 색상으로 설정하세요.
 - 범례, 제목, x, y 라벨 설정 (범례의 label은 각 species 이름으로 설정)
 - ‘culmen_length_mm’와 ‘culmen_depth_mm’에 대한 산점도를 그리시오.
 - 각 species 별로 색을 다르게 해서 표현
 - 범례, 제목, x, y 라벨 설정 (범례의 label은 각 species 이름으로 설정)



■ 시본 (Seaborn)

- Matplotlib 기반의 데이터 시각화 라이브러리
- pyplot보다 좀 더 고차원의 인터페이스 + 간결한 코드
- Python 3.8 이상 지원
- <https://seaborn.pydata.org/api.html>

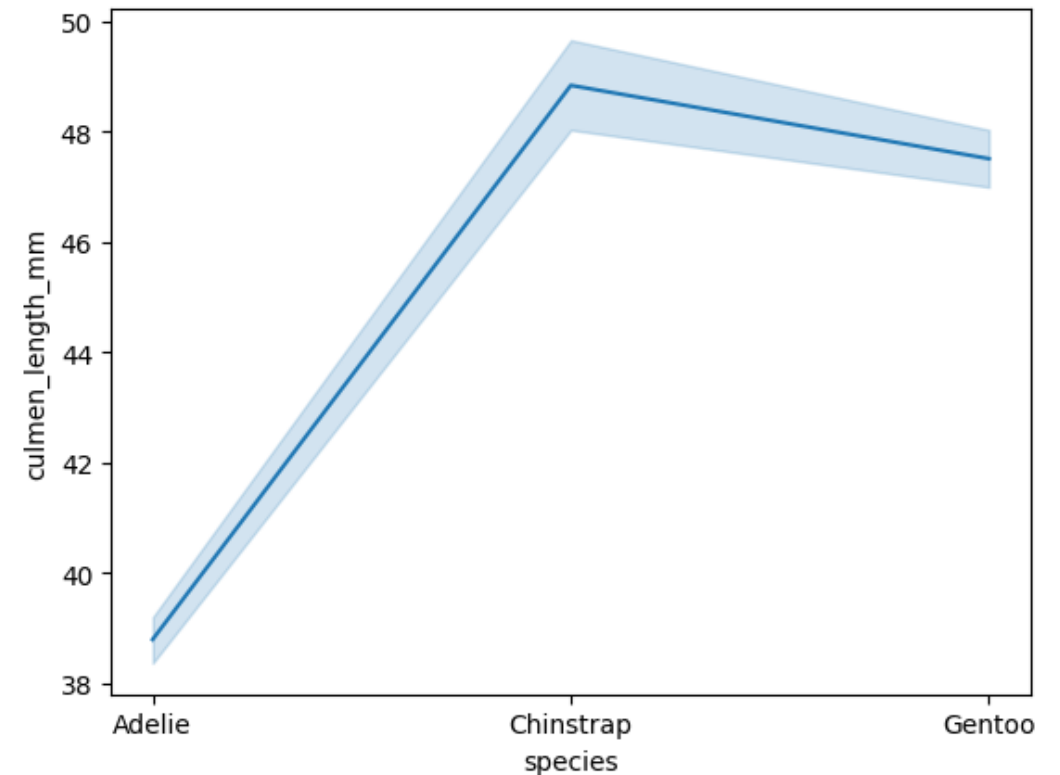


■ Seaborn 시각화

- `.lineplot(data, x, y)`: 선 그래프 생성
- 같은 x값들에 대한 y값이 집계되어 평균 계산 + 95% 신뢰구간 표시

```
import seaborn as sns

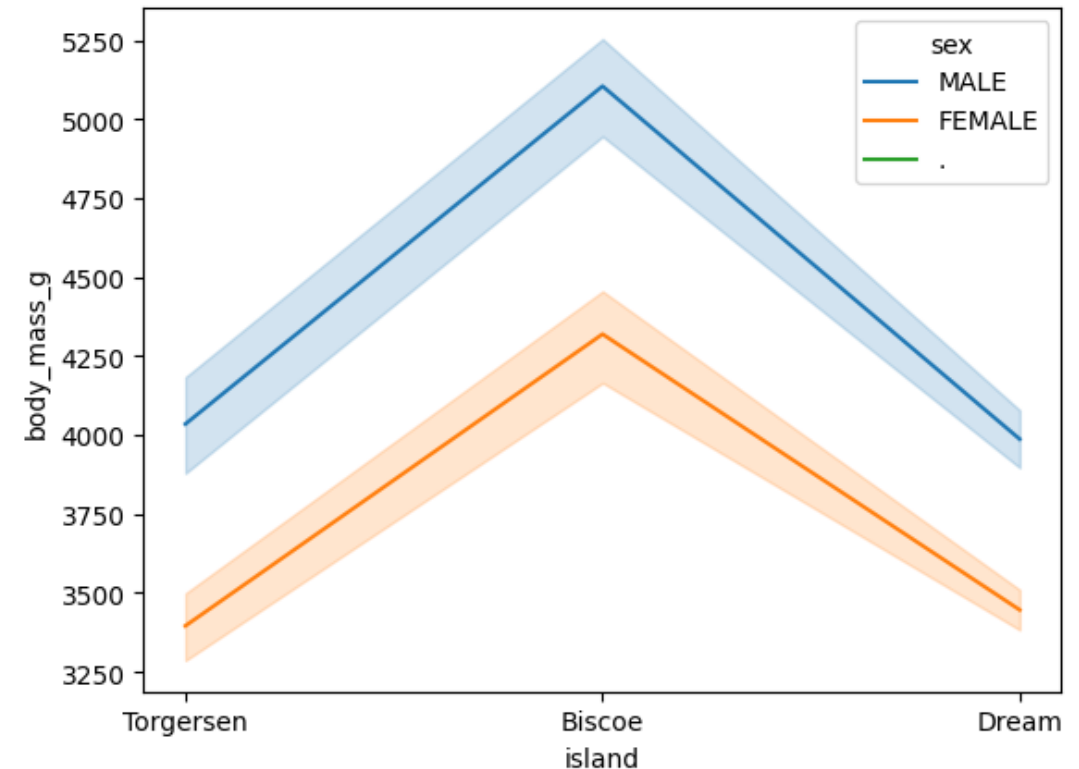
sns.lineplot(data = df, x = df["species"], y = df["culmen_length_mm"])
plt.show()
```



Seaborn 시각화

- `.lineplot(data, x, y)`: 선 그래프 생성
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정

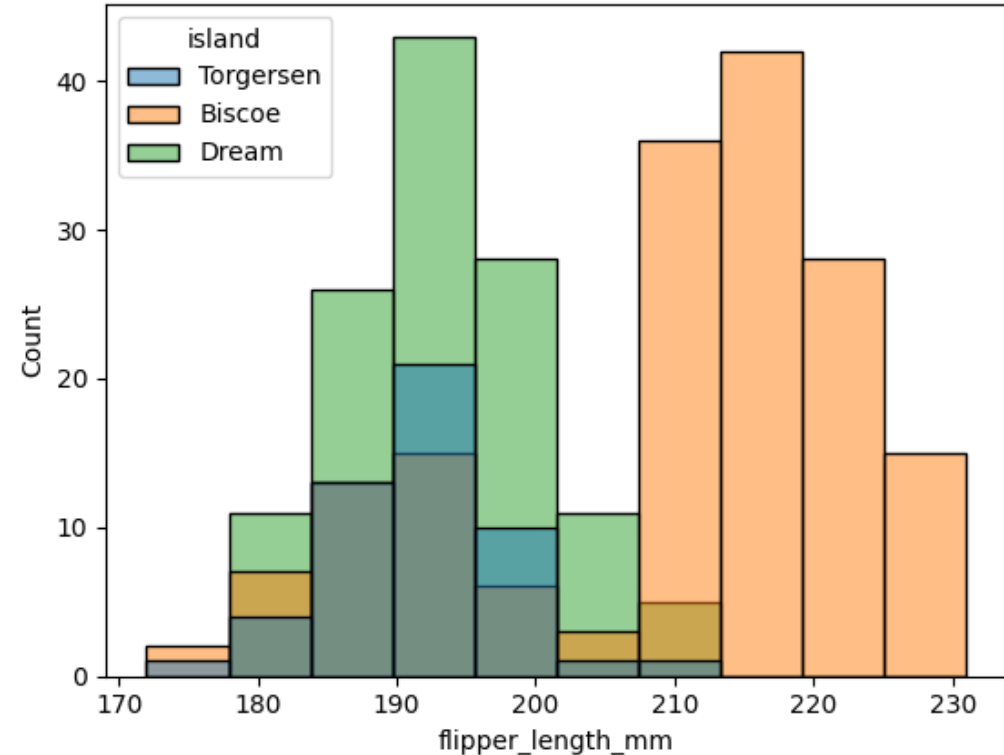
```
sns.lineplot(data = df,  
             x = df["island"],  
             y = df["body_mass_g"],  
             hue = df["sex"])  
plt.show()
```



Seaborn 시각화

- `.histplot(data, x, [bins]):` 히스토그램 생성
- `hue = "범주형 변수":` 범주에 따른 색 자동 지정

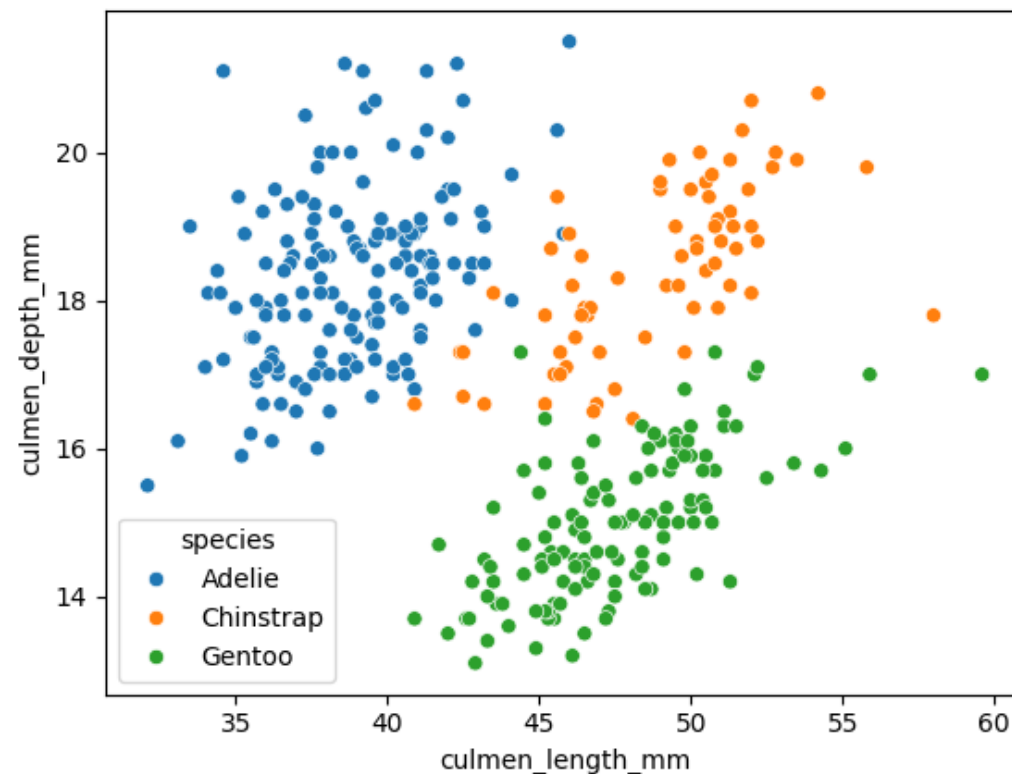
```
sns.histplot(data = df,  
             x = df["flipper_length_mm"],  
             hue = df["island"])  
plt.show()
```



Seaborn 시각화

- `.scatter(data, x, y)`: 산점도 그래프 생성
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정

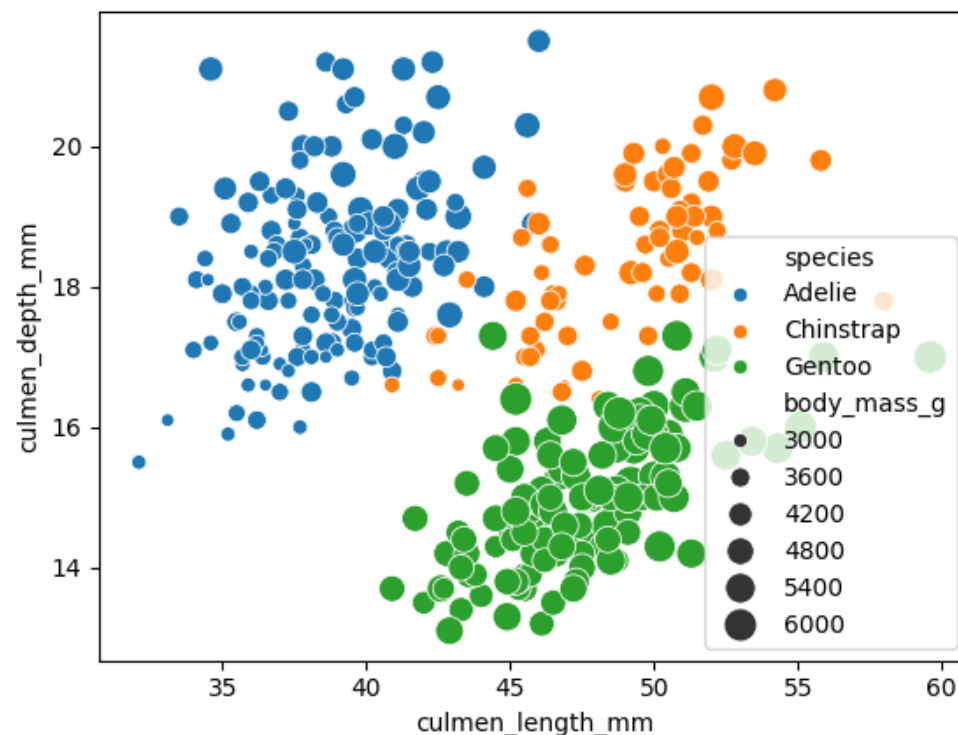
```
sns.scatterplot(data = df,  
                x = df["culmen_length_mm"],  
                y = df["culmen_depth_mm"],  
                hue = df["species"])  
plt.show()
```



Seaborn 시각화

- `.scatter(data, x, y)`: 산점도 그래프 생성
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정
- `size = "수치형 변수"`: 수치에 따른 사이즈 조정, `sizes = (최소, 최대)`

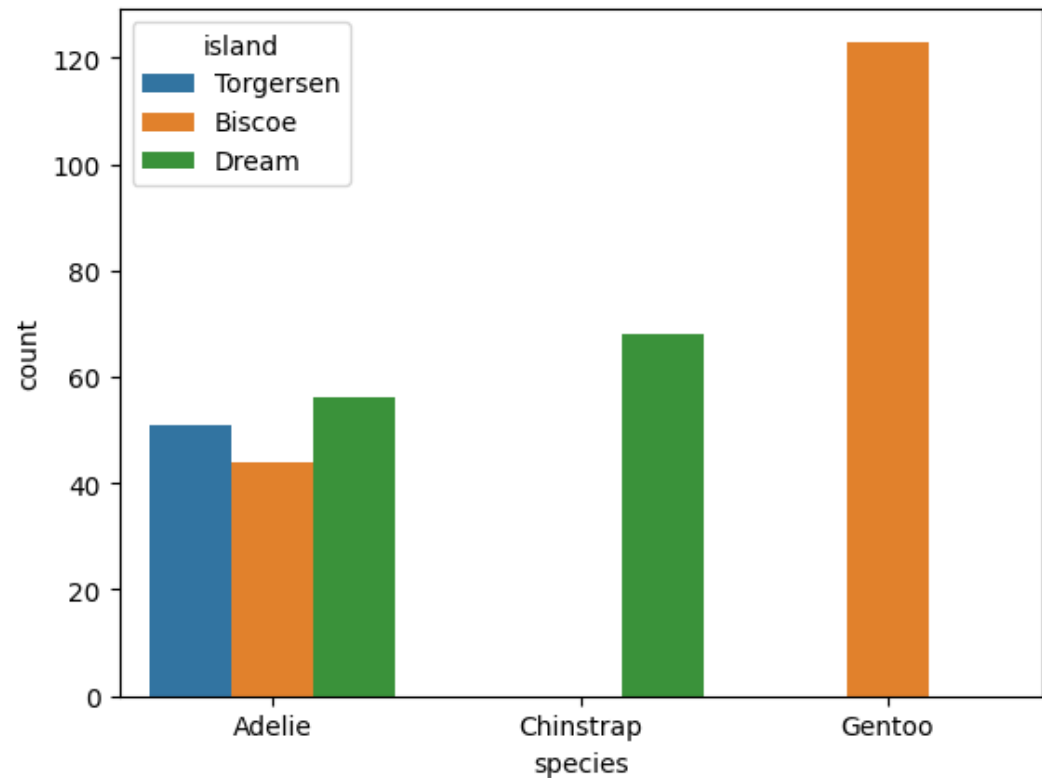
```
sns.scatterplot(data = df,  
                x = df["culmen_length_mm"],  
                y = df["culmen_depth_mm"],  
                hue = df["species"],  
                size = df["body_mass_g"],  
                sizes = (20, 200))  
plt.show()
```



Seaborn 시각화

- `.countplot(data, x)`: 범주형 자료의 개수 확인
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정

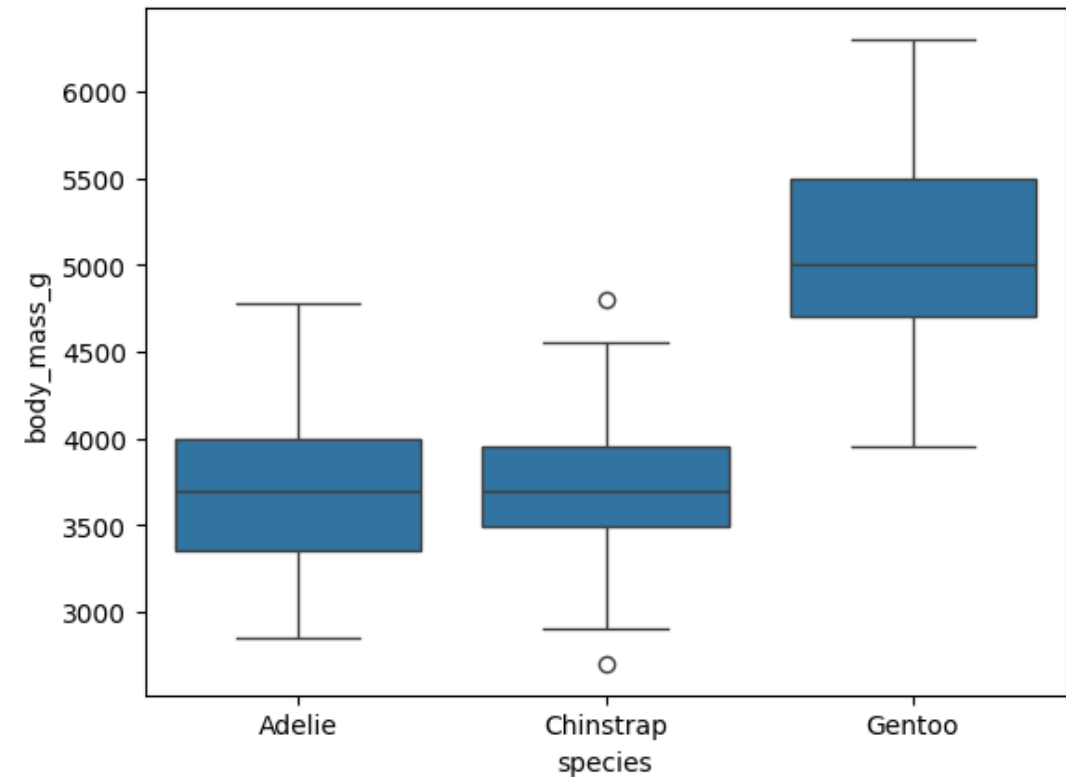
```
sns.countplot(data = df,  
               x = df["species"],  
               hue = df["island"])  
plt.show()
```



Seaborn 시각화

- `.boxplot(data, x, y)`: 박스플롯 생성
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정

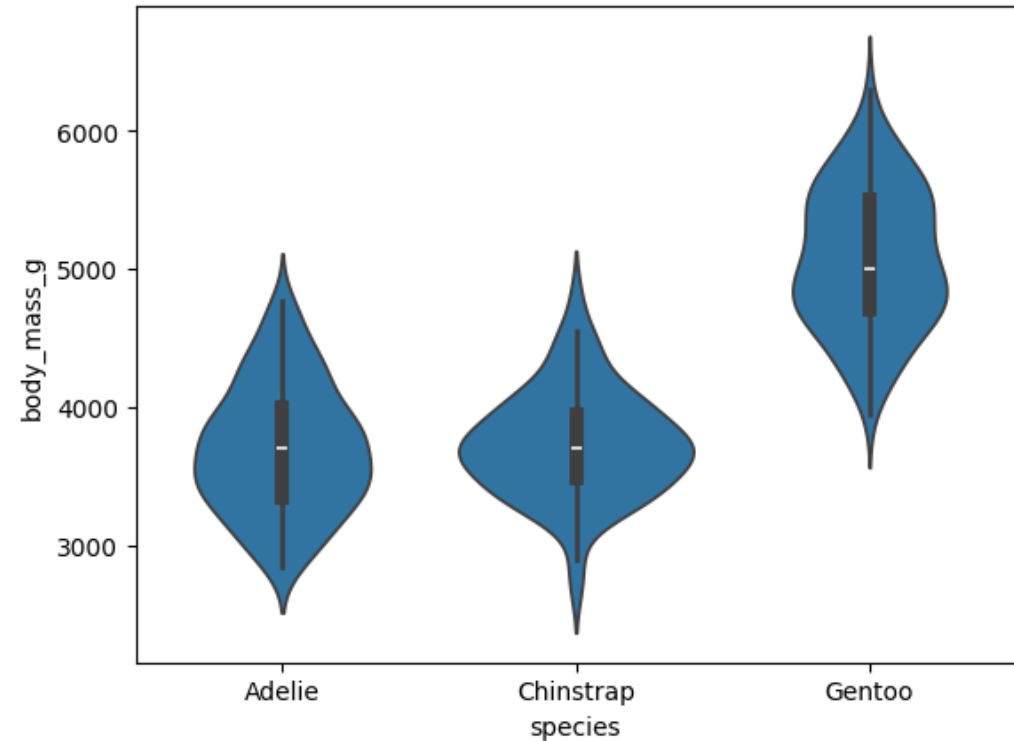
```
sns.boxplot(data = df,  
            x = df["species"],  
            y = df["body_mass_g"])  
plt.show()
```



Seaborn 시각화

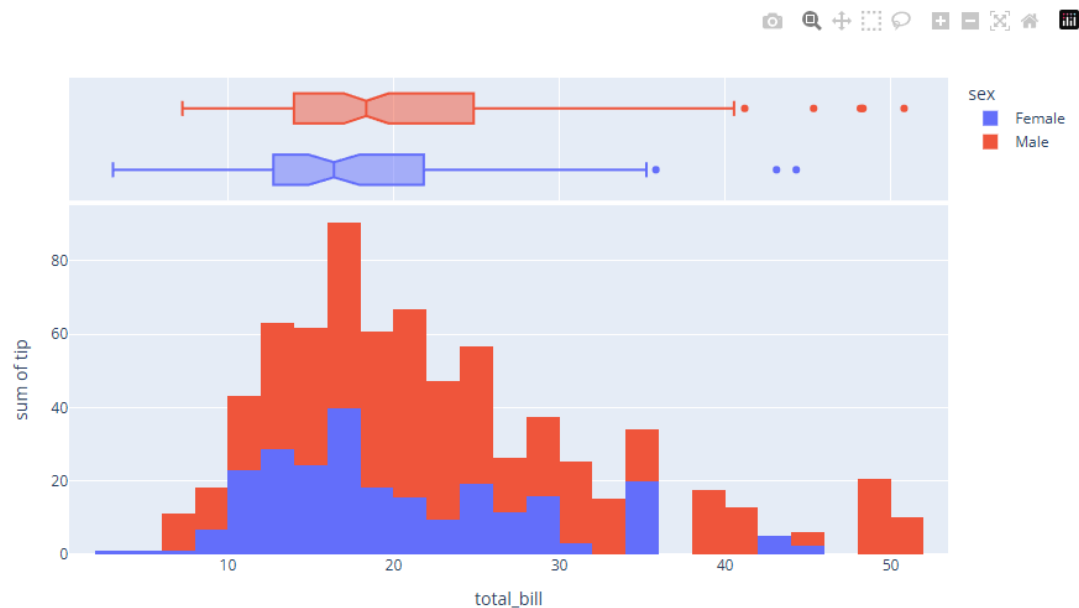
- `.violinplot(data, x, y)`: 바이올린플롯 생성
- `hue = "범주형 변수"`: 범주에 따른 색 자동 지정

```
sns.violinplot(data = df,  
               x = df["species"],  
               y = df["body_mass_g"])  
plt.show()
```



■ 플롯트리 (Plotly) 소개

- Matplotlib / Seaborn과 다르게 Interactive한 시각화 가능
- Matplotlib 대비 간결한 코드
- 연습 코드: <https://wikidocs.net/185024>
- <https://plotly.com/python/>



Q&A
