파이썬 프로그래밍

21. 딥러닝 파이썬 패키지(3)

00

Contents

❖ 수업 목표

- seaborn 패키지 내 모듈의 기본 함수를 사용할 수 있다.
- opencv 모듈의 함수를 사용하여 이미지를 로드 및 표시할 수 있다.
- opencv 모듈의 다양한 함수를 사용할 수 있다.

❖ 세부 목표

1. seaborn 패키지(1)

❖ Seaborn 소개 및 기본 설정

▪ Seaborn이란?

- Python의 데이터 시각화 라이브러리
- 통계적 그래프를 간단하고 아름답게 생성 가능
- Matplotlib 위에서 동작
- 간결한 문법과 고급 시각화 기능 제공

▪ 설치 및 불러오기

- 설치
  - pip install seaborn

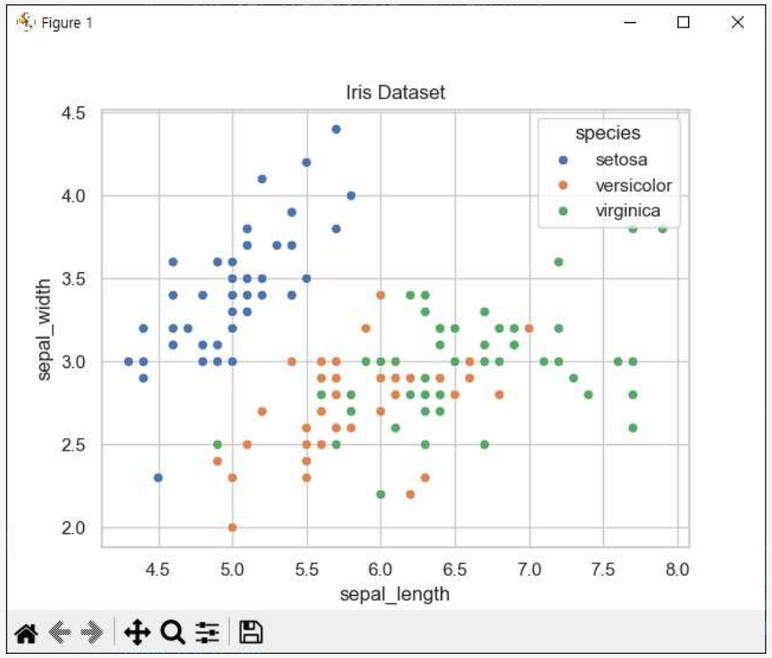
- 라이브러리 임포트
  - import seaborn as sns
  - import matplotlib.pyplot as plt
  - import pandas as pd

1. seaborn 패키지(1)

❖ Seaborn 소개 및 기본 설정

▪ 기본 설정

- 샘플 데이터셋 로드 : 붓꽃 데이터
- 기본 스타일 설정
- 그래프 표시 : 산점도
  - x 축(꽃받침 길이), y 축(꽃받침 너비)
  - hue : 색상을 이용하여 범주 표현



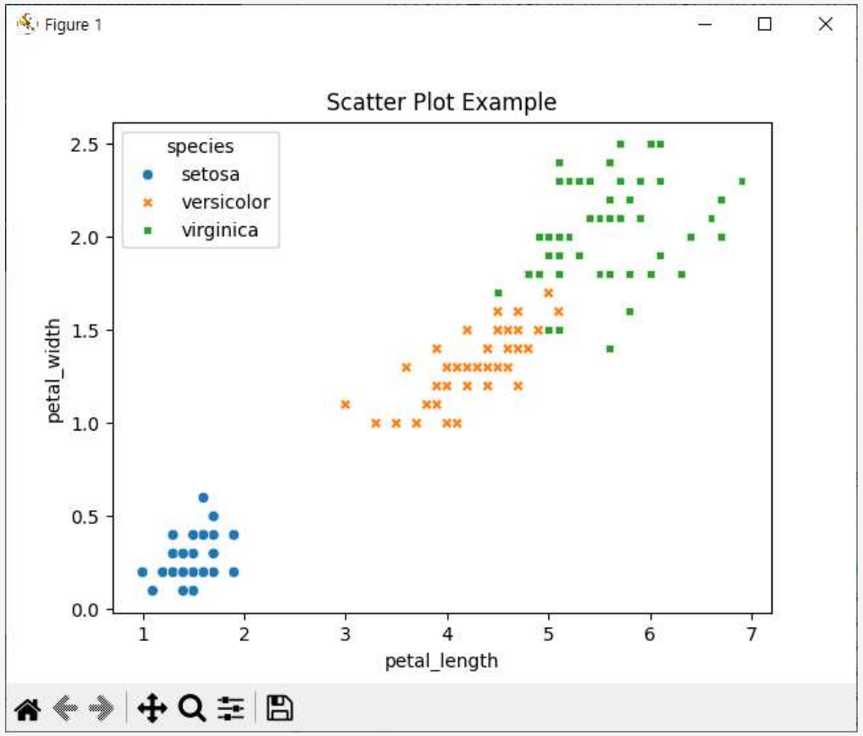

```python
import seaborn as sns
import matplotlib.pyplot as plt

# 샘플 데이터셋 로드
iris = sns.load_dataset("iris")

# 기본 스타일 설정
sns.set_theme(style="whitegrid")

# 그래프 표시
sns.scatterplot(data=iris,
                x="sepal_length",
                y="sepal_width",
                hue="species")
plt.title("Iris Dataset")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

▪ 산점도 (Scatter Plot)

- 두 변수 간의 관계를 시각적으로 표현
  - x 축 : 꽃잎 길이
  - y 축 : 꽃잎 너비
  - style : scatter 스타일 설정



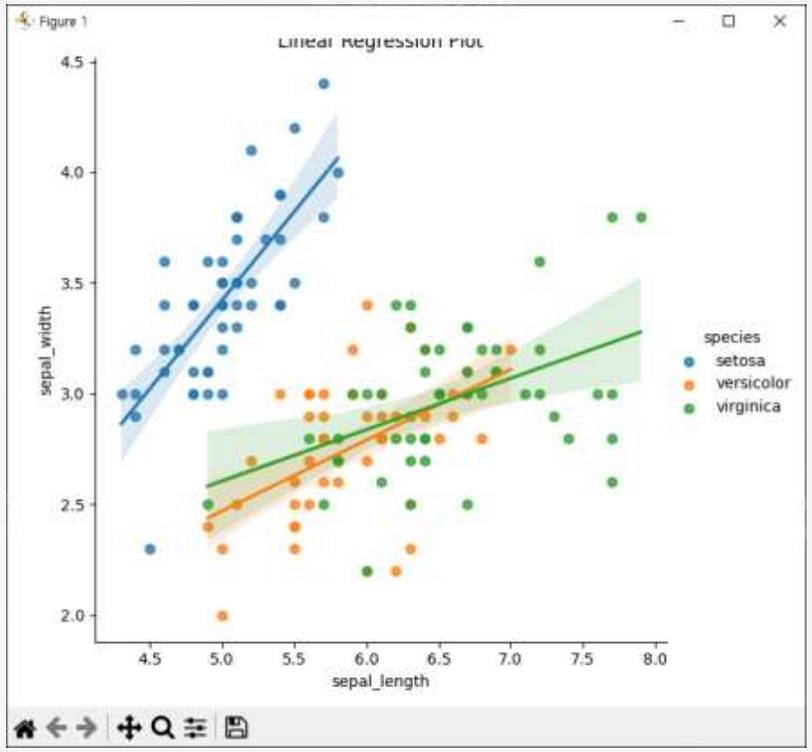

```python
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")

sns.scatterplot(data=iris,
                x="petal_length",
                y="petal_width",
                hue="species",
                style="species")
plt.title("Scatter Plot Example")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

▪ 선형 회귀선 (lmplot, linear model)

- 데이터의 선형적 관계를 시각적으로 표현
  - x 축 : 꽃잎 길이
  - y 축 : 꽃잎 너비
  - height : 높이



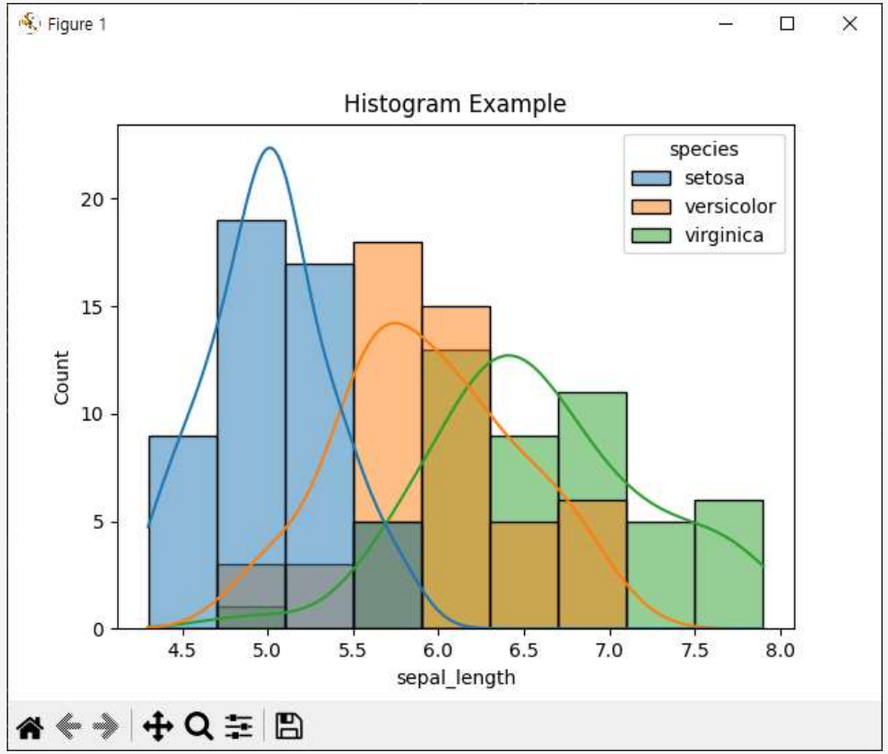

```python
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")

sns.lmplot(data=iris,
           x="sepal_length",
           y="sepal_width",
           hue="species",
           height=6)
plt.title("Linear Regression Plot")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

▪ 히스토그램 (Histogram)

- 데이터의 분포를 확인하는 데 유용
  - x 축 : 꽃받침 길이
  - kde(커널밀도추정) : 분포 곡선 표시



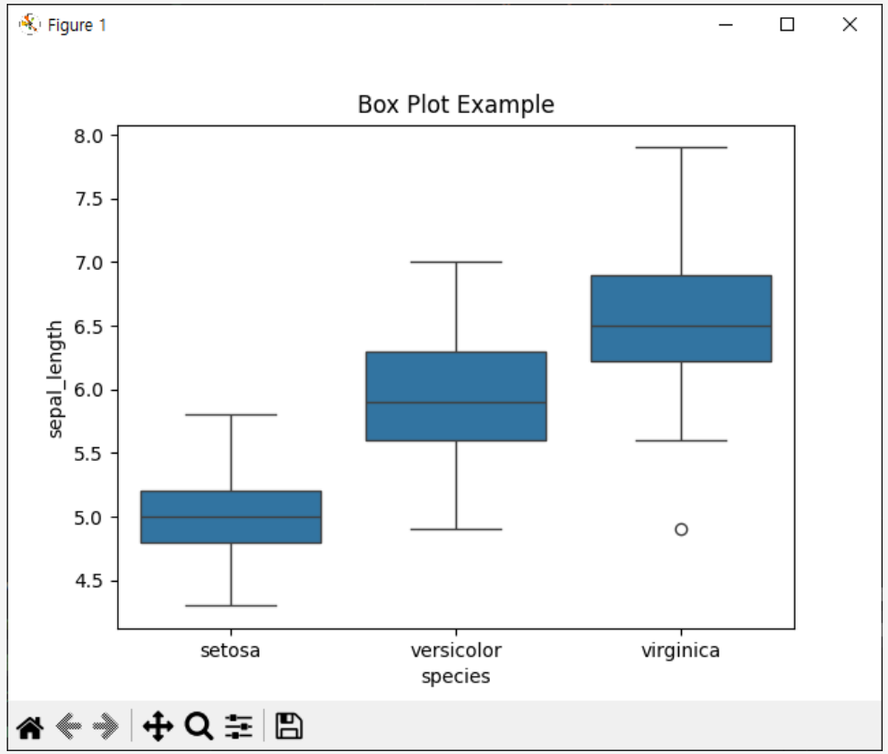

```python
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")

sns.histplot(data=iris,
             x="sepal_length",
             hue="species",
             kde=True)
plt.title("Histogram Example")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

▪ 상자 그림 (Box Plot)

- 데이터 분포와 이상치를 시각적으로 분석
  - x 축 : 붓꽃 종
  - y 축 : 꽃받침 길이



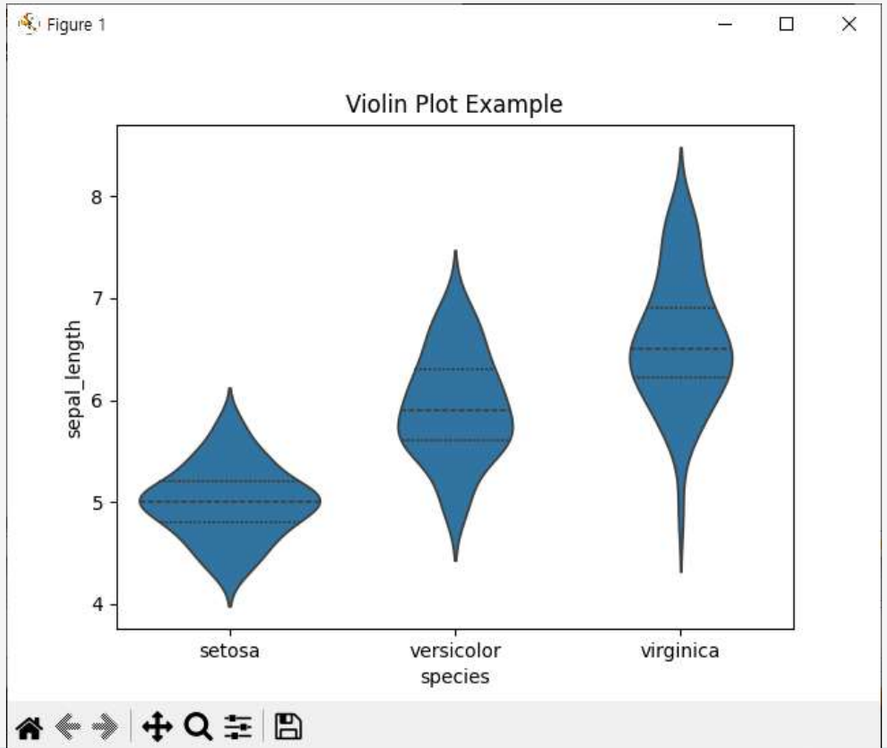

```python
import seaborn as sns
import matplotlib.pyplot as plt

iris = sns.load_dataset("iris")

sns.boxplot(data=iris,
            x="species",
            y="sepal_length")
plt.title("Box Plot Example")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

▪ 바이올린 플롯(Violin Plot)

- 상자 그림과 커널밀도추정이 결합된 형태
  - x 축 : 붓꽃 종
  - y 축 : 꽃받침 길이
  - inner="quart" : 내부에 사분위수 표현




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

iris = sns.load_dataset("iris")

sns.violinplot(data=iris,
               x="species",
               y="sepal_length",
               inner="quart")
plt.title("Violin Plot Example")
plt.show()
```

1. seaborn 패키지(1)

❖ 주요 플롯 유형 및 사용법

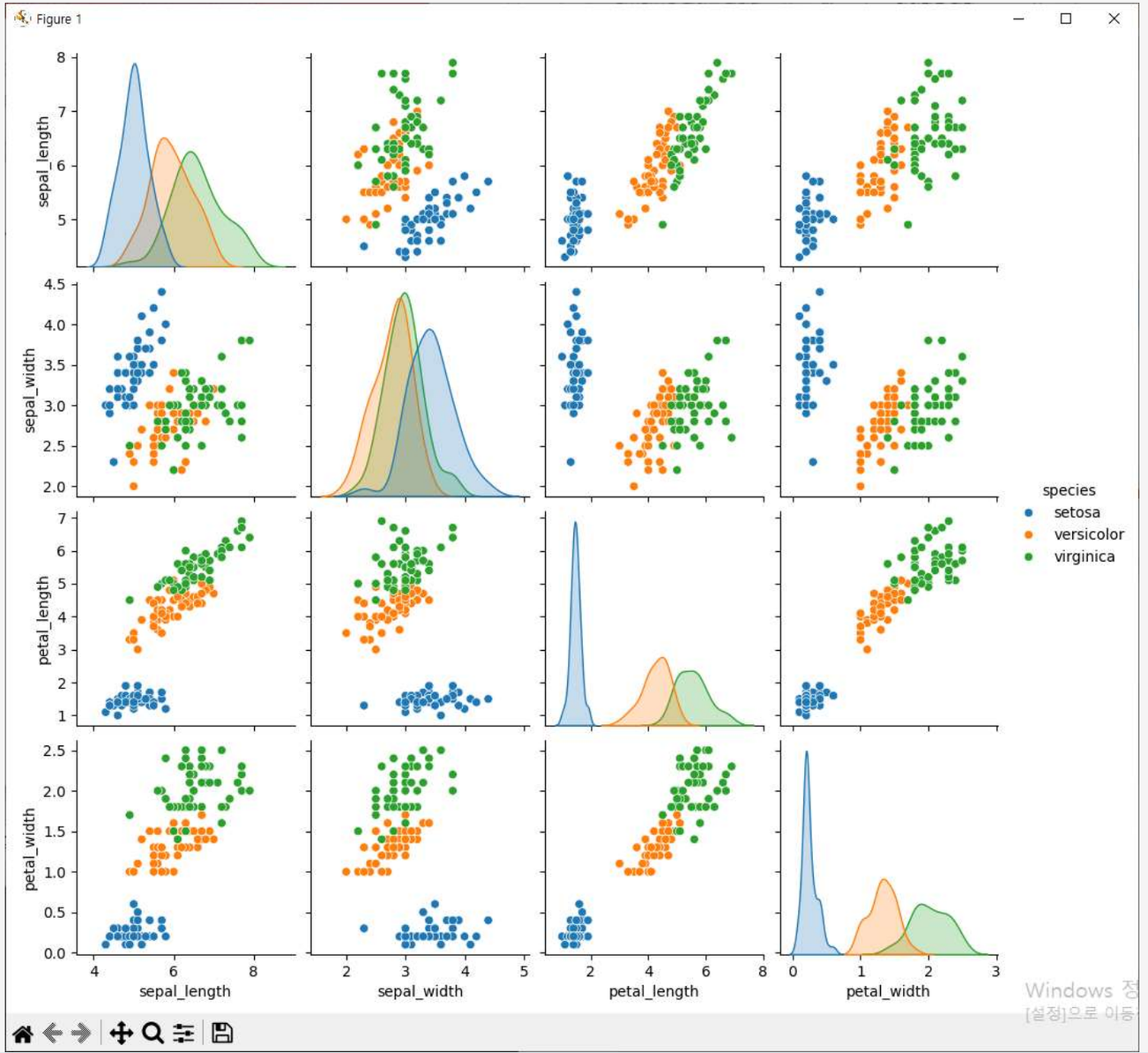
- 페어 플롯(Pair Plot)
  - 여러 변수 간의 관계를 한 번에 확인

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

iris = sns.load_dataset("iris")

sns.pairplot(iris, hue="species")

plt.show()
```

00

1. seaborn 패키지(1)

- 페어 플롯



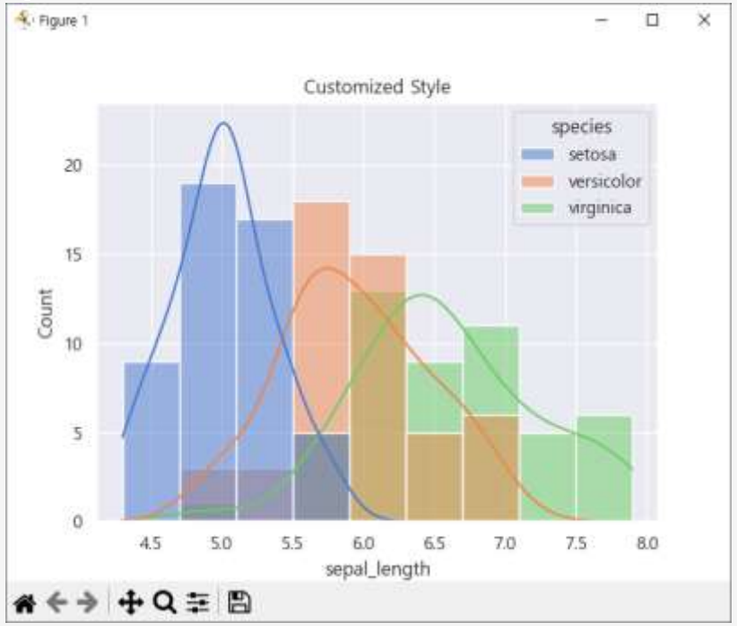
2. seaborn 패키지(2)

❖ 데이터 시각화 심화 기법

▪ 스타일 커스터마이징

- set_theme() 함수를 사용하여 스타일, 팔레트 등 테마를 설정 가능
  - darkgrid, whitegrid, dark, white, ticks 등의 스타일 옵션 제공
  - palette 를 통한 색상 설정 : deep, muted, bright, dark, colorblind

- 히스토그램 추가 코드
  - sns.set_theme(style="darkgrid", palette="muted")



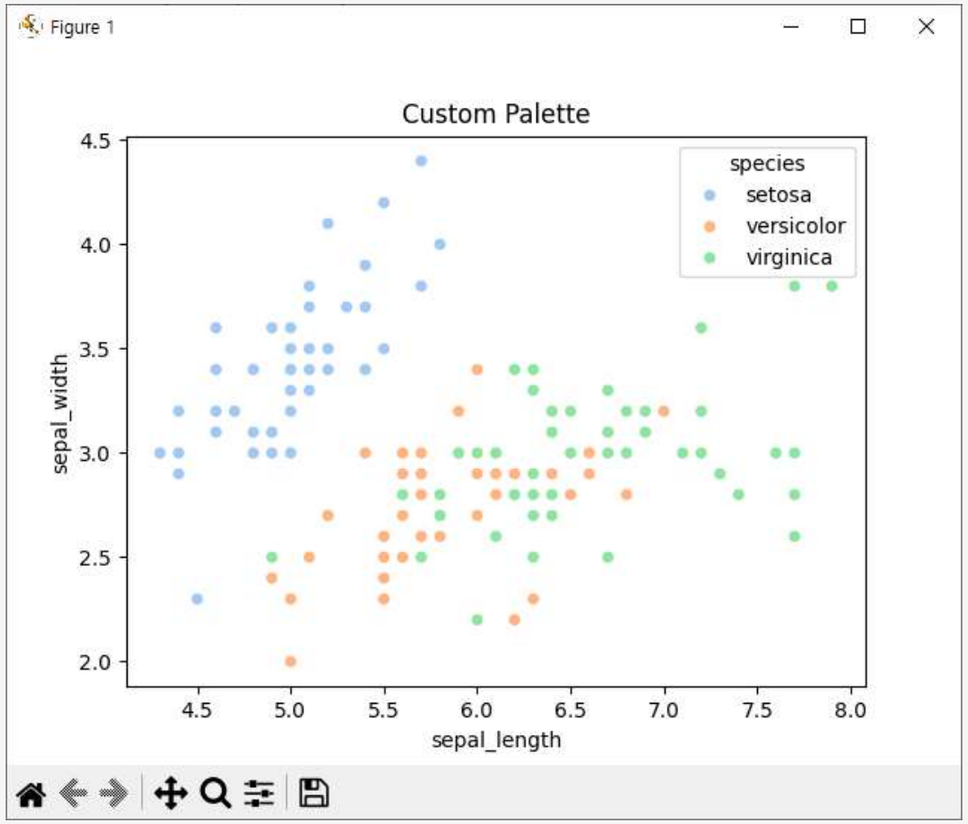
2. seaborn 패키지(2)

❖ 데이터 시각화 심화 기법

▪ 색상 팔레트 변경

- set_palette()를 사용하여 다양한 색상 테마를 적용 가능.

- 산점도 추가 코드
  - sns.set_palette("pastel")



2. seaborn 패키지(2)

❖ 데이터 시각화 심화 기법

▪ 복합 그래프 작성

- 여러 개의 그래프를 한 화면에 배치하여 비교 분석 가능
- g = sns.FacetGrid(data, row, col, height, aspect)
  - » sns.FacetGrid 클래스를 사용, 특정 기준에 따라 데이터셋을 나누고 여러 개의 서브플롯 생성
  - data : DataFrame
  - row : 특정 기준에 따라 행 분할
  - col : 특정 기준에 따라 열 분할
  - height : 각 면의 높이(인치) (width는 없음)
  - aspect : 가로, 세로 비율을 조절. 예를 들어, 1.60이라면 가로:세로=1.6:1

- g.map_dataframe(func, x, kde)
  - » map_dataframe()을 활용하여 원하는 그래프를 적용 가능
  - x : x축 설정
  - kde : 밀도 곡선 추가

2. seaborn 패키지(2)

❖ 데이터 시각화 심화 기법

▪ 복합 그래프 작성

- **set_titles(row_template, col_template)**
  - » set_titles()를 사용해 각 플롯에 제목을 추가 가능
  - row_template : 행 제목
  - col_template : 열 제목

- **예) 복합 그래프 작성**
  - » FacetGrid를 사용하여 'species'별 서브플롯 생성
  - » height: 각 서브플롯의 높이 4, aspect: 가로 세로 비율(1:1) 조정
  - » 각 서브플롯에 히스토그램 적용
  - » x축: 'sepal_length', kde=True를 설정하여 밀도 곡선 추가
  - » 각 플롯의 제목을 종(species) 이름으로 설정

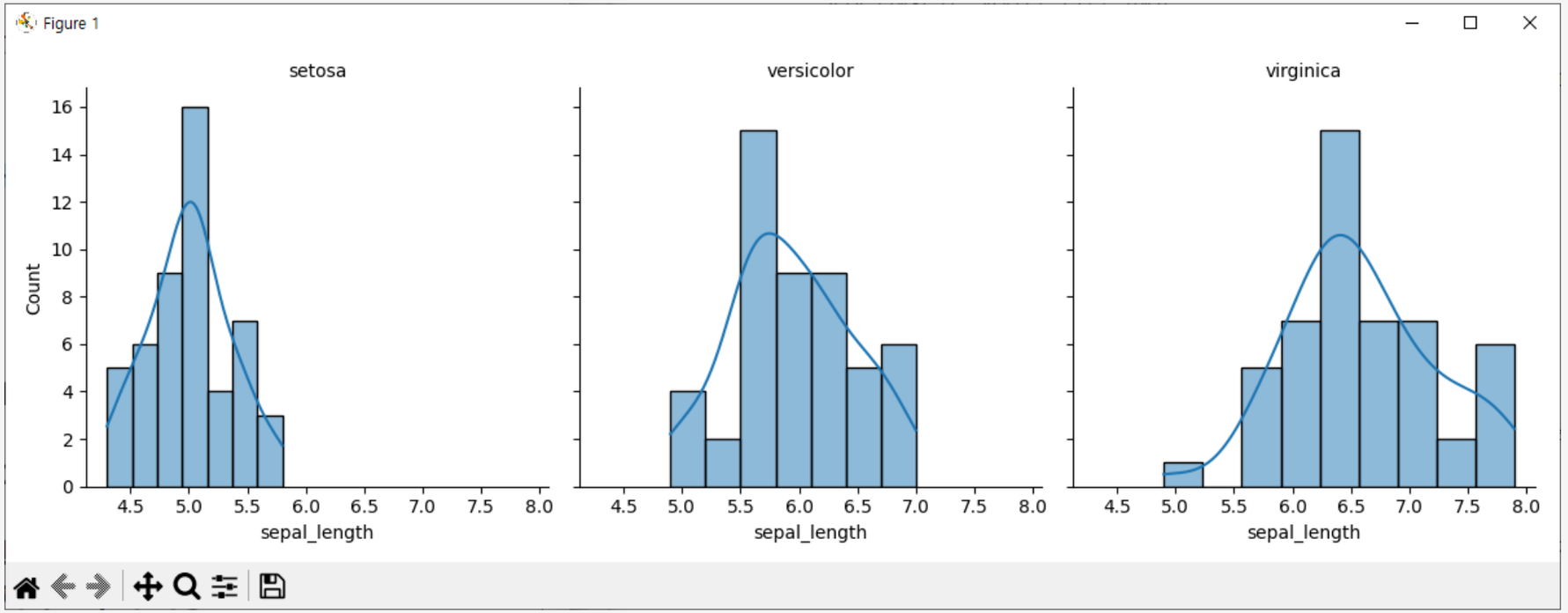
```
g = sns.FacetGrid(iris, col="species", height=4, aspect=1)
g.map_dataframe(sns.histplot, x="sepal_length", kde=True)
g.set_titles(col_template="{col_name}")
```

2. seaborn 패키지(2)

❖ 데이터 시각화 심화 기법

■ 복합 그래프 작성

- 실행 결과



3. opencv(1)

❖ OpenCV 소개 및 기본 설정

▪ OpenCV(Open Source Computer Vision) 소개

- OpenCV는 실시간 컴퓨터 비전을 위한 라이브러리
- Python을 포함한 여러 프로그래밍 언어에서 사용 가능
- 다양한 이미지 및 영상 처리 기능 제공

▪ 설치 및 불러오기

- # 설치
- pip install opencv-python

- # 라이브러리 임포트
- import cv2

3. opencv(1)

❖ OpenCV 소개 및 기본 설정

▪ 기본 이미지 로드 및 표시

- cv2.imread() : 이미지 로드
- cv2.imshow() : 이미지를 OpenCV 창에서 표시
- cv2.waitKey(0) : 키 입력을 기다린 후 창 닫기

```python
import cv2

# 이미지 로드
image = cv2.imread('./ch21/opencv/sample.jpg')

# 이미지 표시
cv2.imshow('Loaded Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. opencv(1)

❖ 이미지 처리 기초

▪ 이미지 크기 조정

- **cv2.resize(src, dsize) -> dst**
  - » 이미지 크기를 너비×높이로 조정
  - src: 원본 이미지
  - dsize: 새로운 이미지의 크기. (width, height) 형식의 튜플로 지정
  - dst: 새로운 이미지 반환

- **예) cv2.resize() 함수를 사용하여 이미지를 300x300 크기로 조정**
  - resized = cv2.resize(image, (300, 300))
  - cv2.imshow('Resized Image', resized)

3. opencv(1)

❖ 이미지 처리 기초

▪ 색상/강도 변환

- **cv2.cvtColor(src, code) -> dst**
  - » 이미지를 다양한 색상 공간으로 변환. 색 공간은 색상(hue), 명도(lightness), 채도(chroma)
  - **src: 입력 이미지**
  - **code: 변환하려는 색상 공간을 지정하는 플래그**
    - » 일반적으로 cv2.COLOR_BGR2GRAY, cv2.COLOR_BGR2RGB, cv2.COLOR_BGR2HSV 등이 사용
    - » 입력 이미지의 색상 공간과 목표 색상 공간 사이의 변환 방법을 결정
  - **dst: 출력 이미지**
    - » 입력 이미지와 동일한 크기와 타입

- **예) cv2.cvtColor()를 이용해 이미지를 그레이스케일로 변환**
  - **gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)**
  - **cv2.imshow('Grayscale Image', gray)**

3. opencv(1)

❖ 이미지 처리 기초

▪ 이미지 회전

- **cv2.getRotationMatrix2D(center, angle, scale) -> retval**
  - » 회전 변환 행렬을 생성 (임의의 행렬을 원점을 중심으로 회전)
  - center: 회전 중심 좌표. (x, y) 튜플
  - angle: (반시계 방향) 회전 각도(degree). 음수는 시계 방향
  - scale: 추가적인 확대 비율
  - retval: 2x3 어파인 변환 행렬. 실수형

- **warpAffine(src, M, dsize) -> dst**
  - » 이미지를 변환
  - src: 입력 영상
  - M: 2x3 어파인 변환 행렬. 실수형
  - dsize: 결과 영상 크기. (w, h) 튜플. (0, 0)이면 src와 같은 크기로 설정
  - dst: 출력 영상

3. opencv(1)

❖ 이미지 처리 기초

▪ 이미지 회전

- 예) 이미지 중심을 기준으로 45도 회전하도록 이미지를 변환
  - (h, w) = image.shape[:2]
  - center = (w // 2, h // 2)
  - M = cv2.getRotationMatrix2D(center, 45, 1.0)
  - rotated = cv2.warpAffine(image, M, (w, h))
  - cv2.imshow('Rotated Image', rotated)

4. opencv(2)

❖ 고급 이미지 처리 기법

▪ 엣지 검출 (Canny Edge Detection)

- 이미지에서 엣지를 검출
- edges = cv2.Canny(image, threshold1, threshold2)
  - image: 입력 이미지. 보통 그레이스케일 이미지를 사용
  - threshold1: 엣지 검출에서 사용되는 최소 임계값
    - » 너무 낮게 설정하면 원치 않은 edge까지 검출
    - » 이 값은 원하는 edge 검출 결과를 얻기 위해서 실험을 통해 찾아야 함
    - » 일반적으로 threshold1과 threshold2를 1:2 또는 1:3 비율로 지정
  - threshold2: 엣지 검출에서 사용되는 최대 임계값
    - » 너무 높게 설정하면 edge가 잘 검출되지 않음
    - » 이 값은 원하는 edge 검출 결과를 얻기 위해서 실험을 통해 찾아야 함
  - edges: 엣지 이미지 반환

- 예) cv2.Canny()를 이용해 엣지 검출을 수행
  - edges = cv2.Canny(image, 100, 200)
  - cv2.imshow('Canny Edge Detection', edges)

4. opencv(2)

❖ 고급 이미지 처리 기법

▪ 블러 처리 (Gaussian Blur)

- cv2.GaussianBlur(src, ksize, sigmaX) -> dst
  - » 가우시안 블러 필터를 적용하여 이미지의 노이즈를 제거 및 부드럽게 만들기
  - src: 입력 이미지
  - ksize: 가우시안 커널 크기
    - » 커널 크기가 클수록 이미지가 더 부드러워 짐
    - » (0, 0)을 지정하면 sigma 값에 의해 자동 결정됨
  - sigmaX : x방향 sigma
  - sigmaY : y방향 sigma
    - » X와 Y 방향의 가우시안 표준 편차(sigmaX, sigmaY).
    - » 0으로 설정하면 OpenCV가 자동으로 값을 계산

- 예) cv2.GaussianBlur()를 사용하여 블러 효과를 적용
  - blurred = cv2.GaussianBlur(image, (15, 15), 0)
  - cv2.imshow('Gaussian Blur', blurred)

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 캐스케이드 분류기(객체/얼굴 검출)

- **cv2.CascadeClassifier(filename)**
  - » Haar Cascade 분류기를 로드하는 OpenCV의 클래스
  - » 얼굴, 눈, 차량 번호판 등의 객체 검출에 사용
  - » 학습된 XML 파일을 입력으로 받아 객체를 생성
- **filename : 객체 검출을 위해 사전 학습된 Haar Cascade 모델 파일**

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 객체 검출 (얼굴 검출)

- **detectMultiScale(image, scaleFactor, minNeighbors, minSize)-> Sequence[Rect]**
  - » 객체(얼굴)를 검출하는 함수
  - **image : 입력 영상. CV_8U 깊이의 행렬**
  - **scaleFactor : 검색 윈도우 확대 비율 (1보다 커야 함)**
  - **minNeighbors : 검출 영역으로 선택하기 위한 최소 (중첩)검출 횟수**
  - **minSize : 검출할 객체의 최소 크기**
  - **maxSize : 검출할 객체의 최대 크기**
  - **Sequence[Rect] : (출력) 검출된 객체의 사각형 좌표 정보**

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 객체 검출 (얼굴 검출)

- **cv2.rectangle(img, pt1, pt2, color, thickness)**
  - » 이미지 상에 직사각형을 그리는 데 사용
  - **img: 사각형을 그릴 이미지**
  - **pt1: 사각형의 왼쪽 상단 꼭지점 좌표. (x, y) 형식의 튜플**
  - **pt2: 사각형의 오른쪽 하단 꼭지점 좌표. (x, y) 형식의 튜플**
  - **color: 사각형의 색상. (B, G, R) 형식의 튜플이나 스칼라 값으로 지정 가능**
  - **thickness: 선택적으로 사각형의 선 두께를 지정. 기본값은 1.**
    - » 음수 값을 전달하면 내부를 채움

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 객체 검출 (얼굴 검출)

◦ 예) 얼굴 검출

- CascadeClassifier를 이용해 사전 학습된 얼굴 검출기를 로딩
- cv2.data.haarcascades
  - » 사전 학습된 Haar Cascade 모델 파일들이 저장된 디렉터리 경로
  - » OS에 따라 OpenCV가 설치된 경로에 차이가 있는 경우, 자동 경로 참조
- 'haarcascade_frontalface_default.xml'
  - » 정면 얼굴 검출을 위한 사전 학습된 Haar Cascade 모델 파일
  - » OpenCV에서 제공하는 XML 파일 중 하나로, 다양한 데이터셋을 기반으로 학습됨
  - » 정면 얼굴 검출에 적합하지만, 측면 얼굴이나 회전된 얼굴에 대해서는 정확도가 낮을 수 있음
- detectMultiScale()을 이용해 얼굴을 찾기
  - » scaleFactor=1.1 : 이미지 크기를 줄여가면서 검출 (1.1배씩 줄여감)
  - » minNeighbors=5 : 검출된 객체 주변의 최소 이웃 개수 (값이 클수록 오탐색 감소)
  - » minSize=(30, 30) : 검출할 객체의 최소 크기 (너무 작은 얼굴을 무시함)
- cv2.rectangle()을 이용해 사각형 그리기
  - » pt1: 사각형 왼쪽 상단 꼭지점 좌표. (x, y)
  - » pt2: 사각형 오른쪽 하단 꼭지점 좌표. (x + w, y + h)
  - » color: 사각형 색상. 파란색(255, 0, 0)
  - » thickness: 사각형 선 두께. 2

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 객체 검출 (얼굴 검출)

- 예) 얼굴 검출

```
import cv2
import matplotlib.pyplot as plt
# 이미지 로드
image = cv2.imread('./ch21/opencv/people.jpg')
face_path = cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
face_cascade = cv2.CascadeClassifier(face_path)
# 이미지를 그레이스케일로 변환
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# 얼굴 검출 수행
faces = face_cascade.detectMultiScale(gray,
                                      scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
# 검출된 얼굴 주위에 사각형 그리기
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
# 결과 출력
cv2.imshow('Face Detection', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. opencv(3)

❖ 고급 이미지 처리 기법

▪ 객체 검출 (얼굴 검출)

- 실행 결과

6. opencv(4)

❖ 다양한 이미지 처리 응용

▪ 실시간 비디오(웹캠) 영상 처리

- **cv2.VideoCapture(index) -> retval**
  - index : 영상 데이터 경로 (또는 실시간 카메라 영상 데이터 가져오기 설정)
    - » camera_id + domain_offset_id 시스템 카메라를 기본 방법으로 열려면 index에 0을 전달
    - » 컴퓨터에 하나의 카메라만 연결되어 있다면 camera_id = 0
    - » 두대 이상의 카메라 연결 시, camera_id 값은 0보다 같거나 큰 정수 (두 대일 경우 0 또는 1)
    - » domain_offset_id는 카메라 장치를 사용하는 방식을 표현하는 정수값을 의미
    - » 대부분 domain_offset_id는 자동 선택을 의미하는 0(CAP_ANY)을 사용하기 때문에 index 값은 camera_id와 같은 값으로 설정
    - » 장치관리자에 등록되어 있는 카메라 순서대로 인덱스 설정
  - retval : cv2.VideoCapture 객체를 반환

- **read() -> tuple[bool, MatLike]**
  - 카메라 또는 동영상 파일로부터 다음 프레임을 받아와 Mat 클래스 형식의 변수에 저장하는 함수 (읽은 프레임을 Matrix 형태로 반환)
  - 정상적으로 프레임을 읽으면 반환 값이 True, 실패하면 False

6. opencv(4)

❖ 다양한 이미지 처리 응용

▪ 실시간 비디오(웹캠) 영상 처리

- **cv2.waitKey(delay)**
  - 키 입력을 기다리는 대기 함수
  - delay : ms(밀리세컨) 단위의 시간을 입력하면 해당 시간만큼 대기
    - » (1000ms = 1초) / (0 = 무한 대기)
  - 리턴 값은 키보드로 입력한 키와 동일한 아스키코드 값

- **video.release()**
  - 동영상을 종료하면 사용했던 메모리 자원을 반환

- **cv2.destroyAllWindows()**
  - 화면에 나타난 윈도우를 종료

6. opencv(4)

❖ 다양한 이미지 처리 응용

▪ 실시간 비디오(웹캠) 영상 처리

- 예) 비디오 영상 처리
  - cv2.VideoCapture(0)를 사용하여 웹캠에서 영상을 가져오기
  - cv2.Canny()를 이용해 실시간으로 엣지 검출을 수행
  - cv2.imshow()를 사용하여 결과를 화면에 표시

```
import cv2
# cap = cv2.VideoCapture(0)      # 웹캠 사용
cap = cv2.VideoCapture("./ch21/opencv/sample.mp4")
while True:
    ret, frame = cap.read()
    if not ret:
        break
    edges = cv2.Canny(frame, 100, 200)
    cv2.imshow('Edge Detection', edges)
    if cv2.waitKey(1) == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

6. opencv(4)

❖ 다양한 이미지 처리 응용

▪ 특정 색상 필터링

- **cv2.inRange(src, lowerb, upperb, dst=None) -> dst**
  - » 특정 색상 영역 추출
  - src: 입력 행렬
  - lowerb: 하한 값 행렬 또는 스칼라
  - upperb: 상한 값 행렬 또는 스칼라
  - dst: 입력 영상과 같은 크기의 마스크 영상. (numpy.uint8)
    - » 범위 안에 들어가는 픽셀은 255, 나머지는 0으로 설정

- **cv2.bitwise_and(src1, src2, mask) -> dst**
  - » 이미지에서 특정 영역을 추출하기 위해 이미지 비트 연산 수행
  - src1: 이미지 파일
  - src2: 이미지 파일
  - mask: 적용 영역 지정
  - dst: 결과 파일

6. opencv(4)

❖ 다양한 이미지 처리 응용

▪ 특정 색상 필터링

- 예) 녹색 필터링 하기
  - cv2.inRange()를 이용해 특정 색상을 필터링
  - bitwise_and()를 이용하여 마스크를 적용한 결과를 표시

```
import cv2
import numpy as np

# 이미지 로드
image = cv2.imread('./ch21/opencv/candies.png')

green_lower = np.array([35, 100, 100])
green_upper = np.array([85, 255, 255])

hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, green_lower, green_upper)
result = cv2.bitwise_and(image, image, mask=mask)

cv2.imshow('Green Color Filtering', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Notice

❖ 과제

- 1. seaborn 모듈의 함수 사용하여 코드 작성하기
- 2. opencv 모듈의 함수를 사용하여 코드 작성하기

❖ 다음 수업 내용

- Python 활용 심화 (1)
  - 파일관리 기본(os, shutil 활용)
  - 파일 정리 프로그램 제작(pathlib 활용)
  - Excel 읽기/쓰기 기초(openpyxl)
  - Excel 데이터 정리 자동화(openpyxl, pandas 활용)
  - 웹 데이터 가져오기(requests, BeautifulSoup)
  - 간단한 데이터 수집 및 저장 프로그램