۱.

$$x[n] = [0, 1, 1, 2, 1, 2, 1, 0, 1, 2, 2, 1, 0, 1, 1, 0]$$

Co-Occurrence Matrices:

$$15 \times P_1 = \begin{bmatrix} 0 & 3 & 0 \\ 3 & 2 & 3 \\ 0 & 3 & 1 \end{bmatrix}, 14 \times P_2 = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 4 & 2 \\ 2 & 1 & 1 \end{bmatrix}, 13 \times P_3 = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 3 & 2 \\ 1 & 3 & 0 \end{bmatrix}$$

Energy:

$$T_{a_{P_1}} = 0 + 9 + 0 + 9 + 4 + 9 + 0 + 9 + 1 = 39$$

$$T_{a_{P_2}} = 0 + 4 + 1 + 1 + 16 + 4 + 4 + 1 + 1 = 32$$

$$T_{a_{P_3}} = 1 + 0 + 4 + 1 + 9 + 4 + 1 + 9 + 0 = 29$$

۲.

در این سوال، با استفاده از پایتون و کتابخانه های آن، تابع KDE را برای دیتا های زیر یافته و آن را رسم می کنیم.

$$[(X,Y)] = [(1,1.02),(1.02,2.009),(2,3.001),(2,3),(3,3.01),(4.2,4.5)]$$

برای یافتن تابع KDE از کرنل گوسی و از فرمول زیر استفاده می کنیم.

$$f(x) = \frac{1}{\sqrt{2\pi}\delta n}\sum_{k=1}^{n} exp\left(\frac{-1}{2}\left(\frac{x-\mu_k}{\delta}\right)^2\right)$$

توضیحات کد محاسبه تابع KDE و رسم آن

```
from matplotlib import pyplot as plt
import numpy as np
import math
```

برای حل این تمرین از کتابخانه matplotlib برای رسم نمودار KDE، از کتابخانه numpy برای محاسبات و از کتابخانه math برای محاسبات ریاضی(رادیکال و عدد پی) استفاده می کنیم.

```
x = np.array([1, 1.02, 2, 2, 3, 4.2])
y = np.array([1.02, 2.009, 3.001, 3, 3.01, 4.5])
```

ابتدا مقادیر داده شده را در آرایه های x و y ذخیره می کنیم.

```
def calculate_KDE(arr):
```

سپس تابع را می سازیم که یک آرایه را به عنوان ورودی دریافت می کند و در انتها x و y را به این تابع می دهیم تا مقادیر محاسبه شده توسط تابع KDE را دریافت کنیم.

```
# Calculate Average
average = 0
for i in range(n):
    average += arr[i]
average /= 6
```

```
# Calculate Standard Deviation
standard_deviation = 0
for i in range(n):
    standard_deviation += pow(arr[i] - average, 2)
standard_deviation /= 6
standard_deviation = math.sqrt(standard_deviation)
```

در ادامه میانگین و انحراف معیار را محاسبه می کنیم.

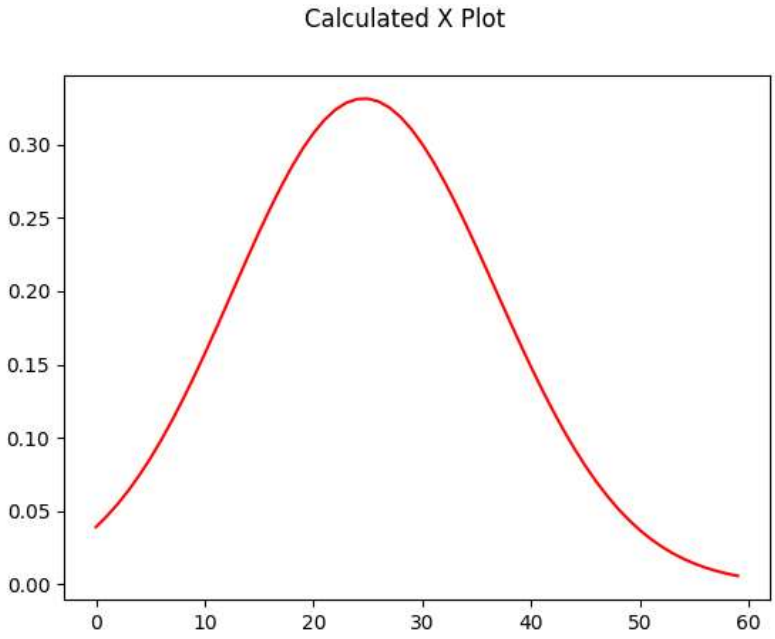
```
# Calculate Function
calculated = np.array([])

pre = 1 / (math.sqrt(2 * math.pi) * standard_deviation * n)

for i in np.arange(-1, 5, 0.1):
    exp_val = 0
    for k in range(n):
        k_average = 0
        for j in range(k +1):
            k_average += arr[j]
        k_average /= (k + 1)

        exp_val += (np.exp((-1/2) * pow((i - k_average) /
standard_deviation, 2)))
    value = pre * exp_val
    calculated = np.append(calculated, value)
```

سپس تابع را با استفاده از فرمولی که در ابتدا بیان شد، محاسبه می کنیم.

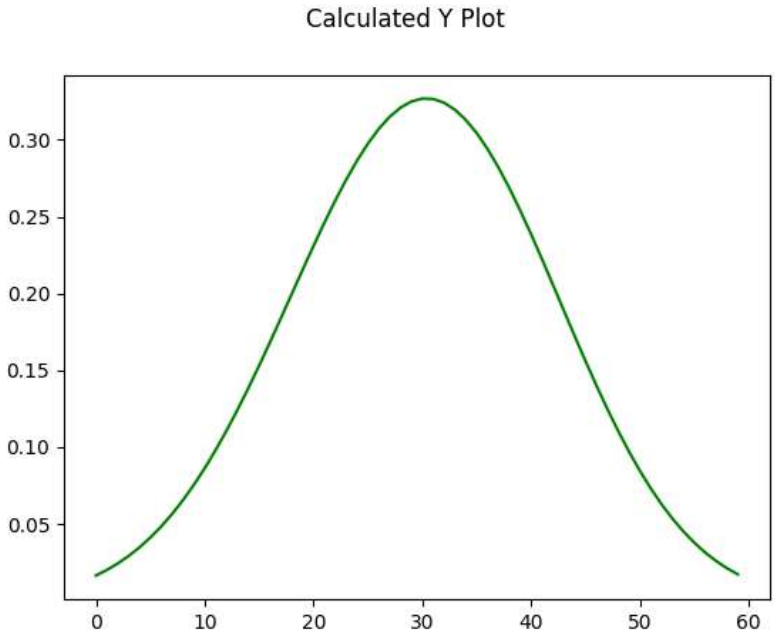
```
# Draw X Plot
x_plot_value = calculate_KDE(x)
plot3 = plt.figure(3)
plot3.suptitle('Calculated X Plot')
plt.plot(x_plot_value, color="red")

# Draw Y Plot
y_plot_value = calculate_KDE(y)
plot4 = plt.figure(4)
plot4.suptitle('Calculated Y Plot')
plt.plot(y_plot_value, color="green")
```

در نهایت تابع KDE را برای x و y رسم می کنیم.

نتیجه محاسبه تابع KDE و رسم آن



رسم تابع KDE برای مقادیر x



رسم تابع KDE برای مقادیر y

توضیحات کد رسم KDE با استفاده از کتابخانه Seaborn

برای مقایسه توابع رسم شده، همین مقادیر را با استفاده از کتابخانه آماده Seaborn رسم می کنیم.

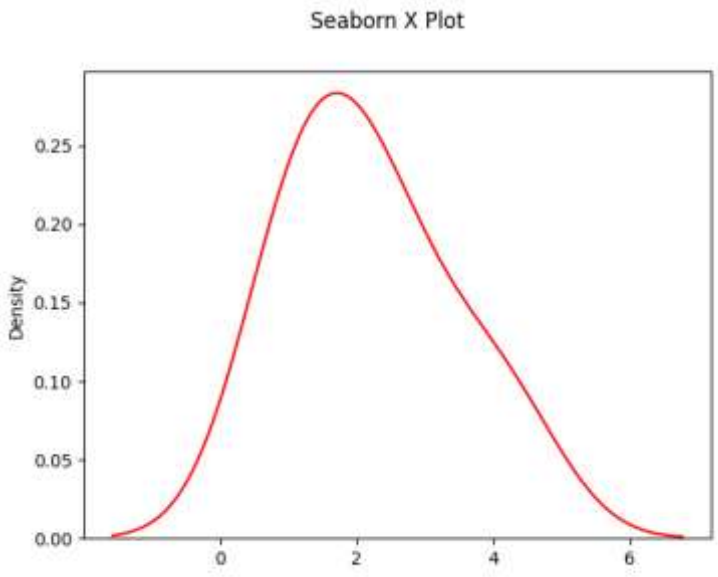
```
import seaborn as sns
```

ابتدا این کتابخانه را import می کنیم.

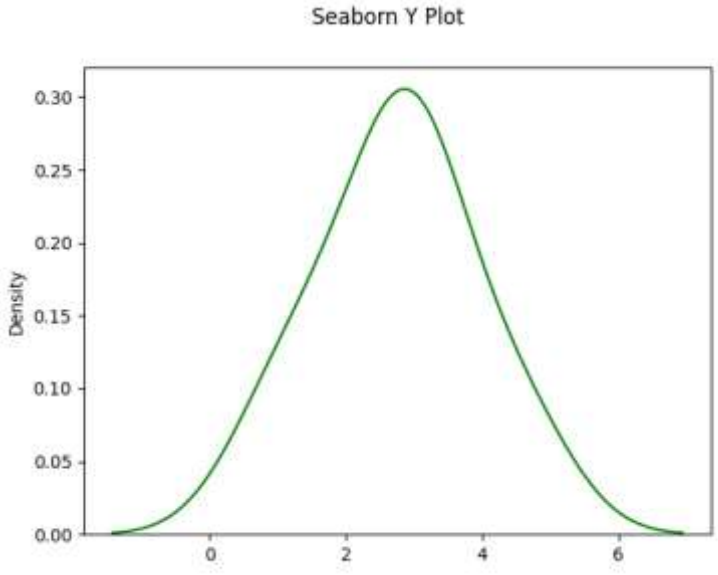
```
plot1 = plt.figure(1)
plot1.suptitle('Seaborn X Plot')
sns.kdeplot(x, color="red")

plot2 = plt.figure(2)
plot2.suptitle('Seaborn Y Plot')
sns.kdeplot(y, color="green")
```

سپس با استفاده از تابع kdeplot مقادیر x و y را رسم می کنیم.

نتیجه رسم KDE با استفاده از کتابخانه Seaborn



رسم تابع KDE برای مقادیر x توسط کتابخانه Seaborn



رسم تابع KDE برای مقادیر y توسط کتابخانه Seaborn

پلات های رسم شده توسط تابع آماده، شباهت زیادی به پلات های محاسبه شده توسط فرمول دارد.

۳.

در این سوال، با استفاده از پایتون و کتابخانه های آن، هیستوگرام تصویر خود را رسم کرده و با استفاده از آستانه محاسبه شده، تصویر را بازسازی می کنیم.

توضیحات کد محاسبه هیستوگرام

```
from skimage import io, img_as_ubyte
import numpy as np
from matplotlib import pyplot as plt
```

برای حل این تمرین از کتابخانه skimage برای خواندن عکس، از numpy برای محاسبات و از کتابخانه matplotlib برای رسم نمودار هیستوگرام استفاده می کنیم.

```
# Read Image as Colored and Gray
image_colored = img_as_ubyte(io.imread("KimiaMahdinejad.jpg",
as_gray=False))
image_gray = img_as_ubyte(io.imread("KimiaMahdinejad.jpg",
as_gray=True))
```

ابتدا تصویر را به دو صورت رنگی و سیاه و سفید می خوانیم.

```
scales_colored = np.zeros(256)

scales_gray = np.zeros(256)
```

سپس دو آرایه به تعداد ۲۵۵ خانه برای ذخیره فراوانی هر کدام از اعداد می سازیم.

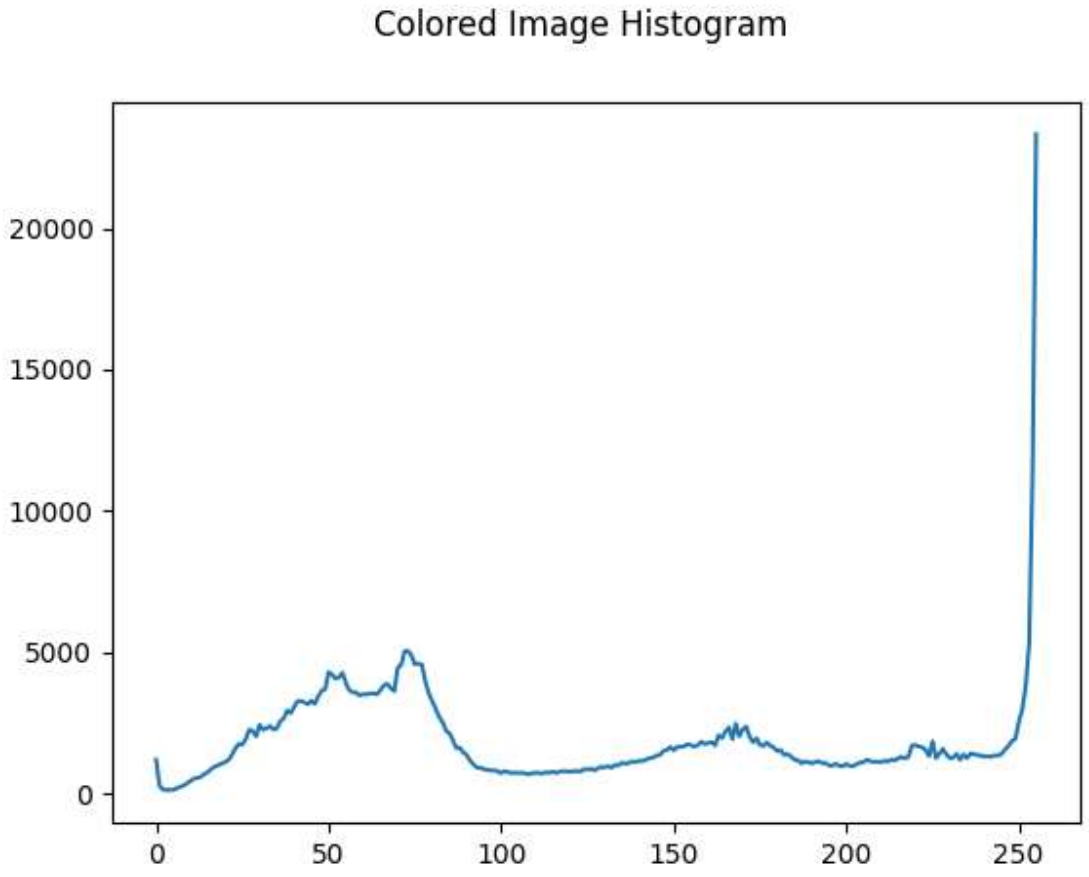
```
# Calculate Colored Image Histogram
for i in range(len(image_colored)):
    for j in range(len(image_colored[0])):
        for k in range(len(image_colored[0][0])):
            scales_colored[image_colored[i][j][k]] += 1

# Calculate Gray Image Histogram
for i in range(len(image_gray)):
    for j in range(len(image_gray[0])):
        scales_gray[image_gray[i][j]] += 1
```

با استفاده از حلقه های تو در تو، فراوانی هر کدام از مقادیر را در آرایه های ساخته شده ذخیره می کنیم.

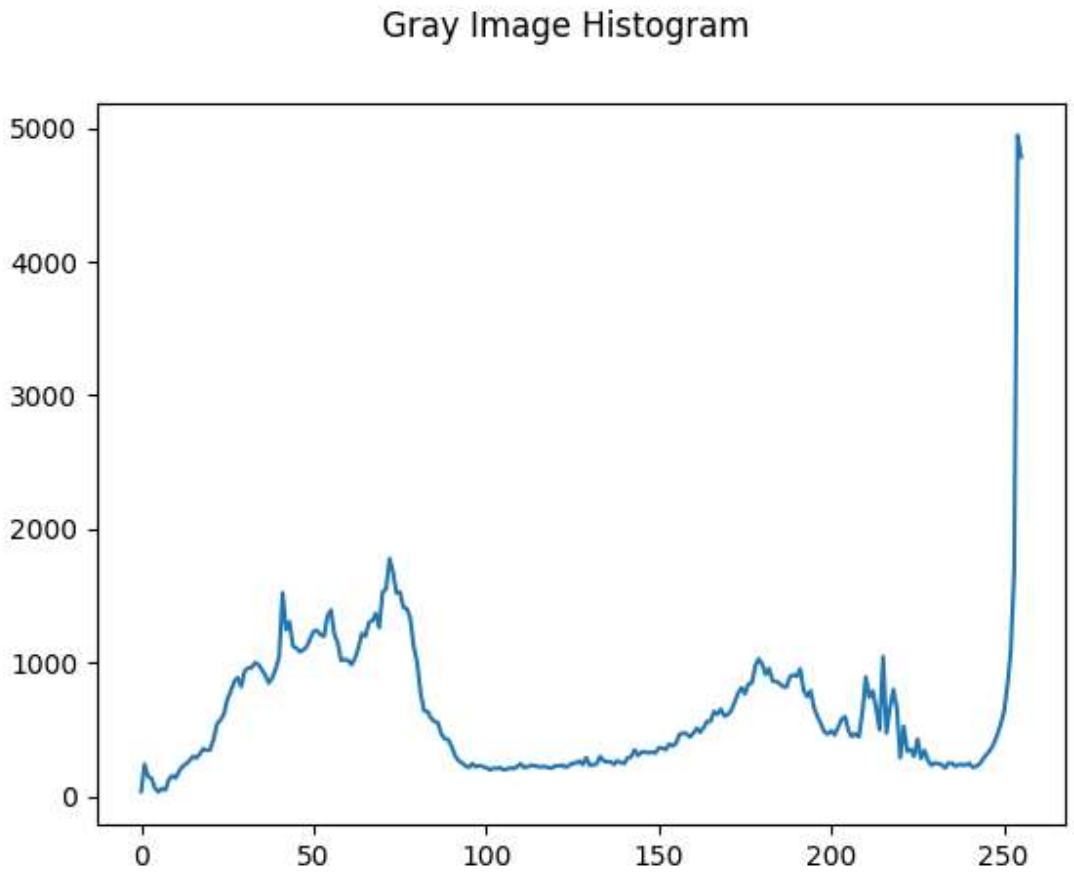
```
# Draw Colored Image Histogram
plot1 = plt.figure(1)
plot1.suptitle('Colored Image Histogram')
plt.plot(scales_colored)

# Draw Gray Image Histogram
plot2 = plt.figure(2)
plot2.suptitle('Gray Image Histogram')
plt.plot(scales_gray)
```

سپس مقادیر ذخیره شده در آرایه ها را که همان مقادیر فراوانی است را پلات می کنیم.

نتیجه محاسبه هیستوگرام



نمودار هیستوگرام رسم شده برای عکس رنگی



نمودار رسم شده برای عکس سیاه و سفید

توضیحات کد بازسازی تصویر با آستانه مناسب

```
# Calculate Colored Image Threshold
max_val_colored = 0
for i in range(256):
    if scales_colored[i] > scales_colored[max_val_colored]:
        max_val_colored = i

# Calculate Gray Image Threshold
max_val_gray = 0
for i in range(256):
    if scales_gray[i] > scales_gray[max_val_gray]:
        max_val_gray = i
```

با محاسبه مقدار ماکسیمم ذخیره شده در آرایه ها، آستانه هر کدام از تصاویر رنگی و سیاه و سفید را محاسبه می کنیم.

```
# Rebuild Colored Image With Threshold
new_image_colored = image_colored

for i in range(len(image_colored)):
    for j in range(len(image_colored[0])):
        for k in range(len(image_colored[0][0])):
            if new_image_colored[i][j][k] >= threshold_colored:
                new_image_colored[i][j][k] = 0

# Rebuild Gray Image With Threshold
new_image_gray = image_gray

for i in range(len(image_gray)):
    for j in range(len(image_gray[0])):
        if new_image_gray[i][j] > threshold_gray:
            new_image_gray[i][j] = 0
```

سپس در قسمت هایی که از آستانه مقدار بیشتری دارند را صفر می کنیم.

```
# Save Colored Image With Threshold
io.imsave("KimiaMahdinejad_removed_noise_colored.jpg",
new_image_colored.astype(np.uint8))

# Save Gray Image With Threshold
io.imsave("KimiaMahdinejad_removed_noise_gray.jpg",
new_image_gray.astype(np.uint8))
```

نتیجه را در قالب دو تصویر ذخیره می کنیم.

نتیجه بازسازی تصویر با آستانه مناسب

تصویر اصلی

تصویر رنگی با آستانه ۲۵۵

تصویر سیاه و سفید با آستانه ۲۵۴