Histogram is a graph that shows the distribution of pixel intensities of an image.

Grayscale Histograms:

1) Script:Load the necessary package and the image path

```
# import the necessary packages
from matplotlib import pyplot as plt
import imutils
import cv2

img_path = "beach.png"
```

2) Script : load the image and convert to grayscale

```
image = cv2.imread(img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

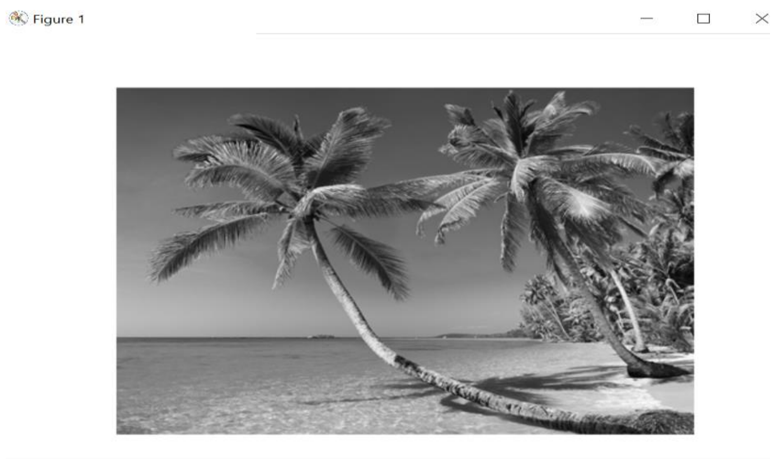
3) Script : we will apply `cv2.calcHist`

```
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
```

4) Script: first showing the image in matplotlib but need to be converted back to rgb

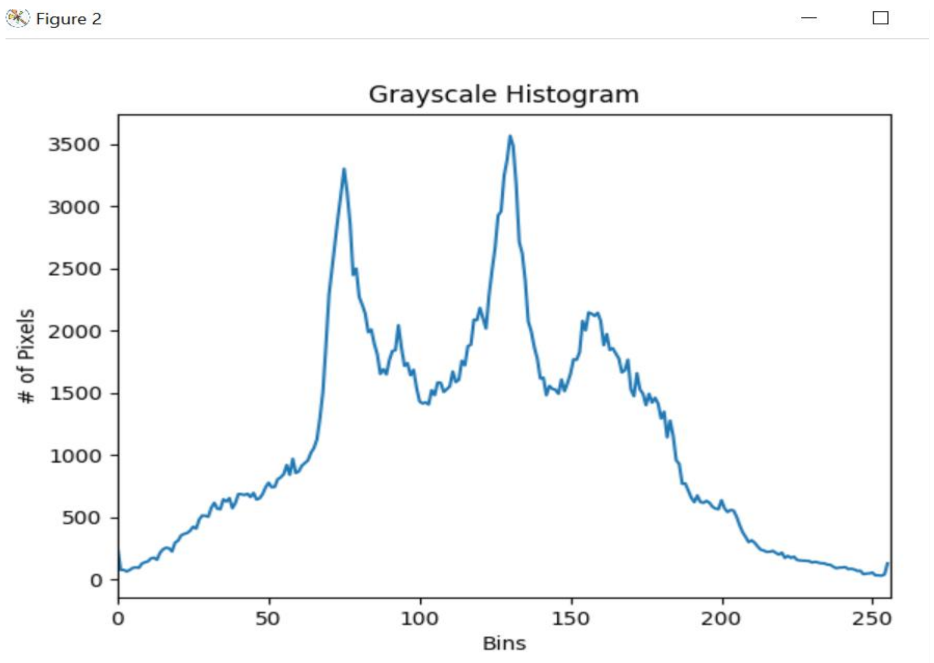
```
# matplotlib expects RGB images so convert and then display the image
# with matplotlib
plt.figure()
plt.axis("off")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_GRAY2RGB))
```

Result:



5) Script:Plotting the hist array of pixels intensities

```
# plot the histogram
plt.figure()
plt.title("Grayscale Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
```

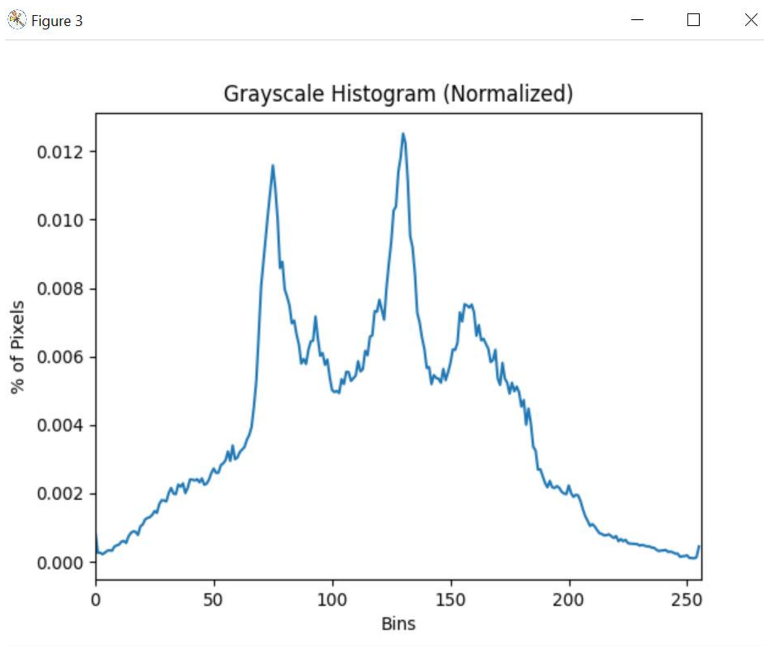


6) Script: Normalizing the histogram using (pixel count)/sum of all the counts

```
# normalize the histogram
        hist /= hist.sum()
```

7) Script :show the normalize histogram array

```
# plot the normalized histogram
plt.figure()
plt.title("Grayscale Histogram (Normalized)")
plt.xlabel("Bins")
plt.ylabel("% of Pixels")
plt.plot(hist)
plt.xlim([0, 256])
plt.show()
```



Color histograms with OpenCV

Using script 1 above with the rest of this
2) Script:Loading the image

```
# load the input image from disk
image = cv2.imread(args["image"])
```

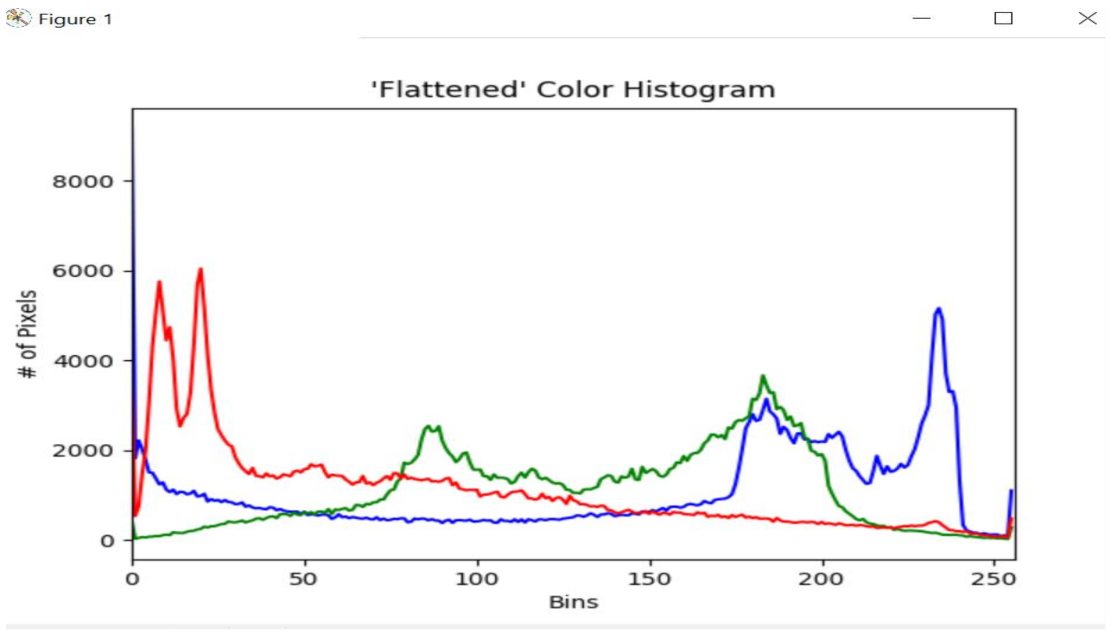
3) Script:- Splitting the rgb image to its respective channel and initializing color which will be used in drawing the histogram.

```
# split the image into its respective channels, then initialize the
# tuple of channel names along with our figure for plotting
chans = cv2.split(image)
colors = ("b", "g", "r")
plt.figure()
plt.title("'Flattened' Color Histogram")
plt.xlabel("Bins")
plt.ylabel("# of Pixels")
```

4) Script:drawing the histogram for individual channel in the loop

```
# loop over the image channels
for (chan, color) in zip(chans, colors):
    # create a histogram for the current channel and plot it
    hist = cv2.calcHist([chan], [0], None, [256], [0, 256])
    plt.plot(hist, color=color)
    plt.xlim([0, 256])
```

Result



Because color image have more than one channel aside from getting histogram for individual channel we can get for 2 channels which is known as multidimensional histogram.

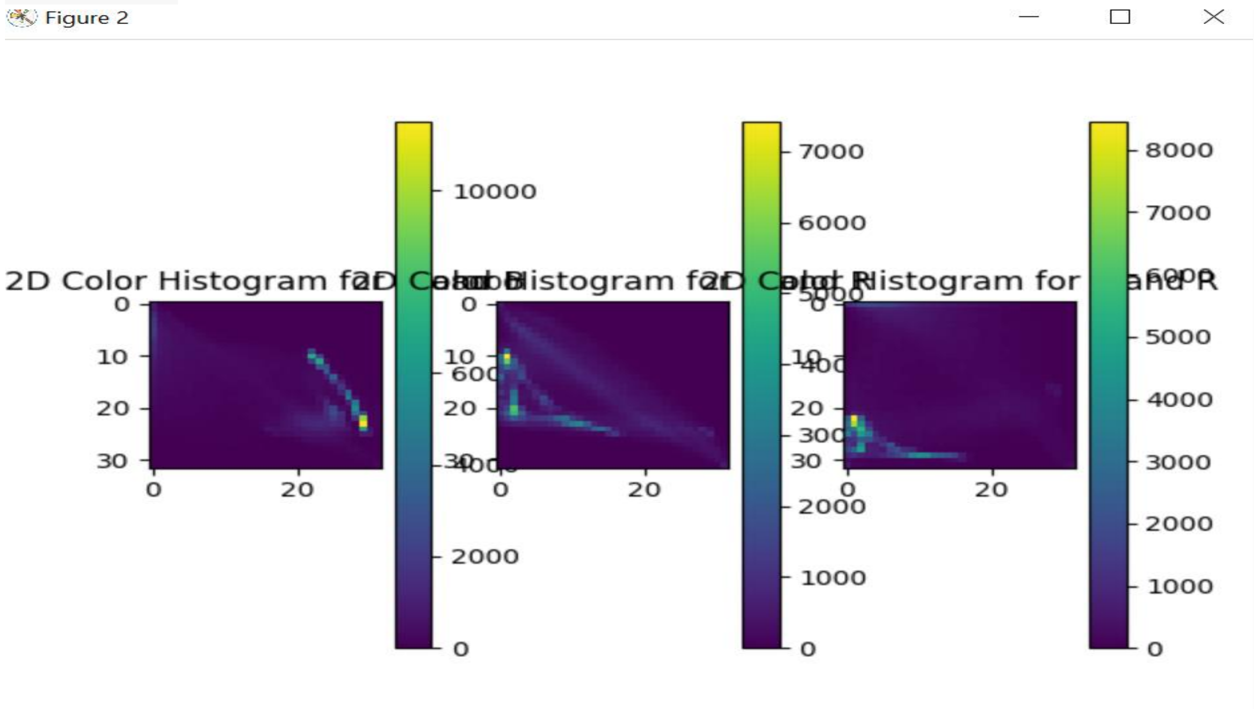
5) Script: drawing 2d histogram for (g,b),(b,r) and (g,r)

```
# create a new figure and then plot a 2D color histogram for the
# green and blue channels
fig = plt.figure()
ax = fig.add_subplot(131)
hist = cv2.calcHist([chans[1], chans[0]], [0, 1], None, [32, 32],
    [0, 256, 0, 256])
p = ax.imshow(hist, interpolation="nearest")
ax.set_title("2D Color Histogram for G and B")
plt.colorbar(p)
```

```
# plot a 2D color histogram for the green and red channels
ax = fig.add_subplot(132)
hist = cv2.calcHist([chans[1], chans[2]], [0, 1], None, [32, 32],
    [0, 256, 0, 256])
p = ax.imshow(hist, interpolation="nearest")
ax.set_title("2D Color Histogram for G and R")
plt.colorbar(p)

# plot a 2D color histogram for blue and red channels
ax = fig.add_subplot(133)
hist = cv2.calcHist([chans[0], chans[2]], [0, 1], None, [32, 32],
    [0, 256, 0, 256])
p = ax.imshow(hist, interpolation="nearest")
ax.set_title("2D Color Histogram for B and R")
plt.colorbar(p)
```

Result:



The color image can also have a 3d histogram

6) Script:

```
# our 2D histogram could only take into account 2 out of the 3
# channels in the image so now let's build a 3D color histogram
# (utilizing all channels) with 8 bins in each direction -- we
# can't plot the 3D histogram, but the theory is exactly like
# that of a 2D histogram, so we'll just show the shape of the
# histogram
hist = cv2.calcHist([image], [0, 1, 2],
    None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
print("3D histogram shape: {}, with {} values".format(
    hist.shape, hist.flatten().shape[0]))

# display the original input image
plt.figure()
plt.axis("off")
plt.imshow(imutils.opencv2matplotlib(image))

# show our plots
plt.show()
```

Result:

image histograms for masked regions with OpenCV:

Just as one can get the histogram of an image, one can get the histogram of the masked region of an image with the same open cv histogram function applied on the masked region.

1) Script:Import the necessary packages

```
# import the necessary packages
from matplotlib import pyplot as plt
import numpy as np
import cv2

img_path = "beach.png"
```

2) Script:defining a function that can take an image,and title with optional masking region.The code before the loop and the loop is similar to the script 3 and 4nfor image histogram for color image respectively.The difference is making it a function so it is reusable and the cv2.calcHist is function will use mask depending on if it is passed or not

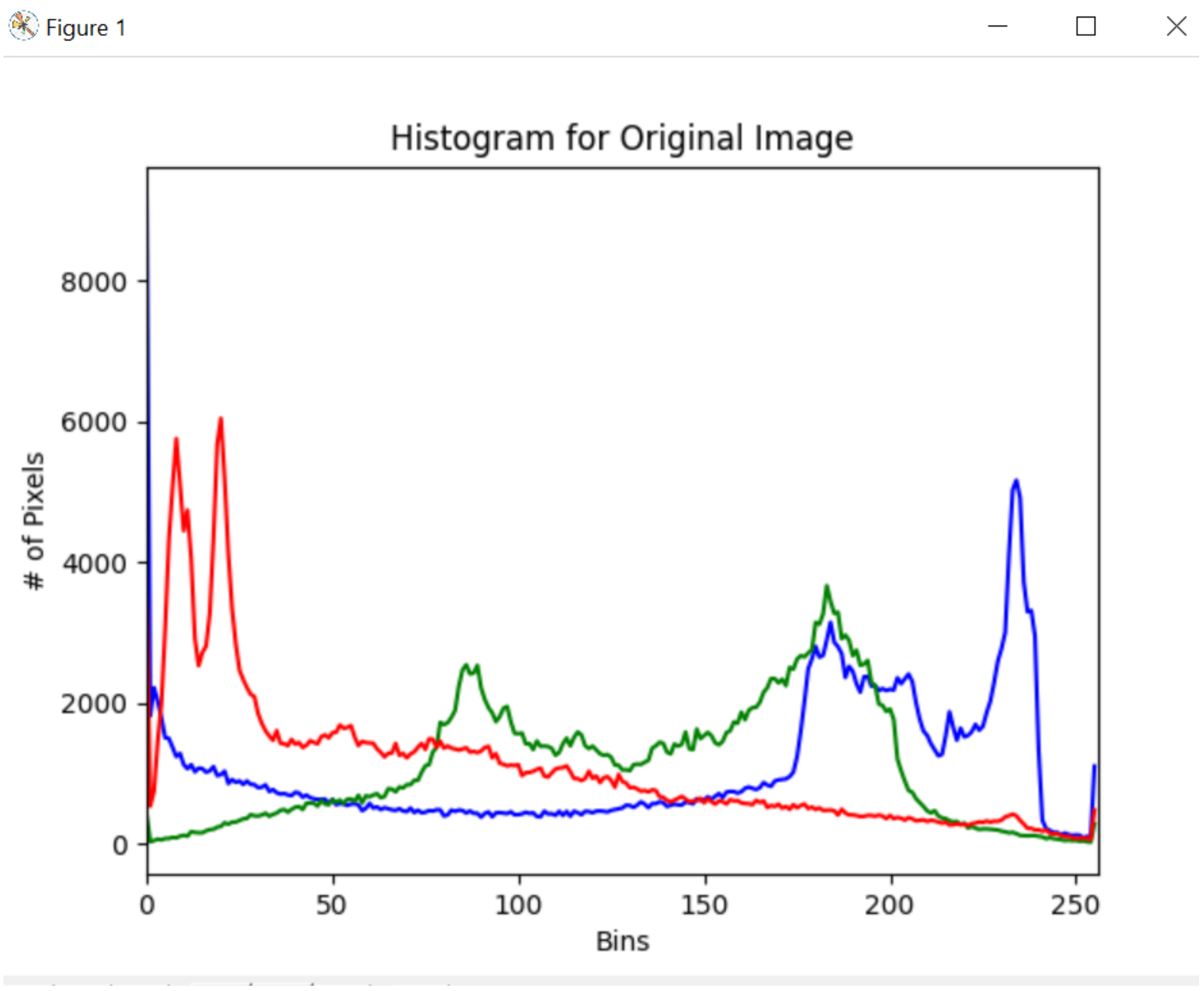
```
def plot_histogram(image, title, mask=None):
    # split the image into its respective channels, then initialize
    # the tuple of channel names along with our figure for plotting
    chans = cv2.split(image)
    colors = ("b", "g", "r")
    plt.figure()
    plt.title(title)
    plt.xlabel("Bins")
    plt.ylabel("# of Pixels")

    # loop over the image channels
    for (chan, color) in zip(chans, colors):
        # create a histogram for the current channel and plot it
        hist = cv2.calcHist([chan], [0], mask, [256], [0, 256])
        plt.plot(hist, color=color)
        plt.xlim([0, 256])
```

3) Script:Read the image,show the image,plot the histogram of the individual channel

```
# load the beach image and plot a histogram for it
image = cv2.imread("beach.png")
plot_histogram(image, "Histogram for Original Image")
cv2.imshow("Original", image)
```

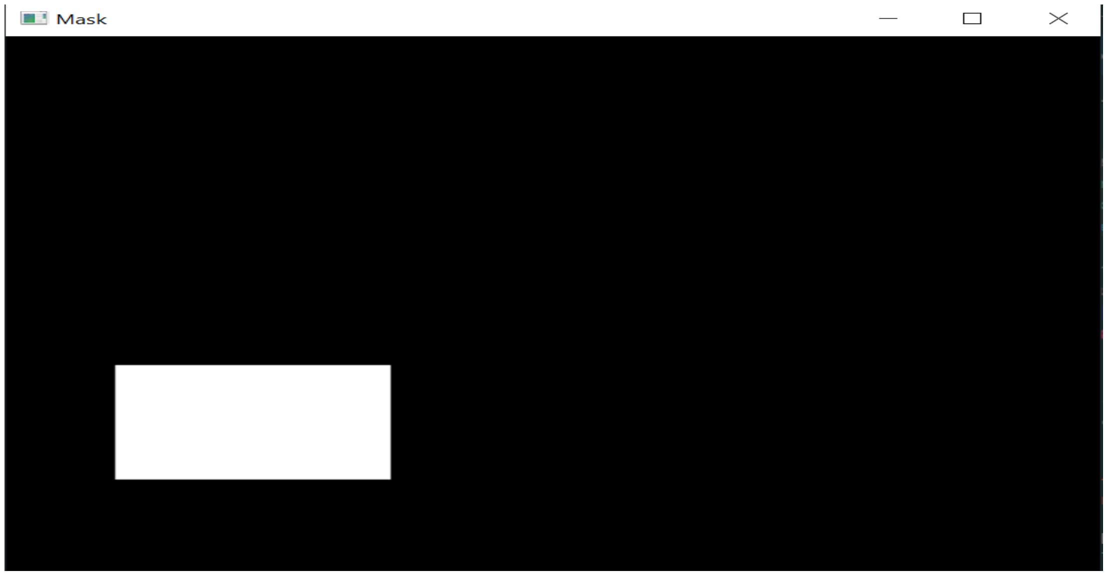
Result:



4) Script:Here we construct a mask and a rectangular region and then show the mask

```
# construct a mask for our image; our mask will be *black* for
regions
# we want to *ignore* and *white* for regions we want to *examine*
mask = np.zeros(image.shape[:2], dtype="uint8")
cv2.rectangle(mask, (60, 290), (210, 390), 255, -1)
cv2.imshow("Mask", mask)
```

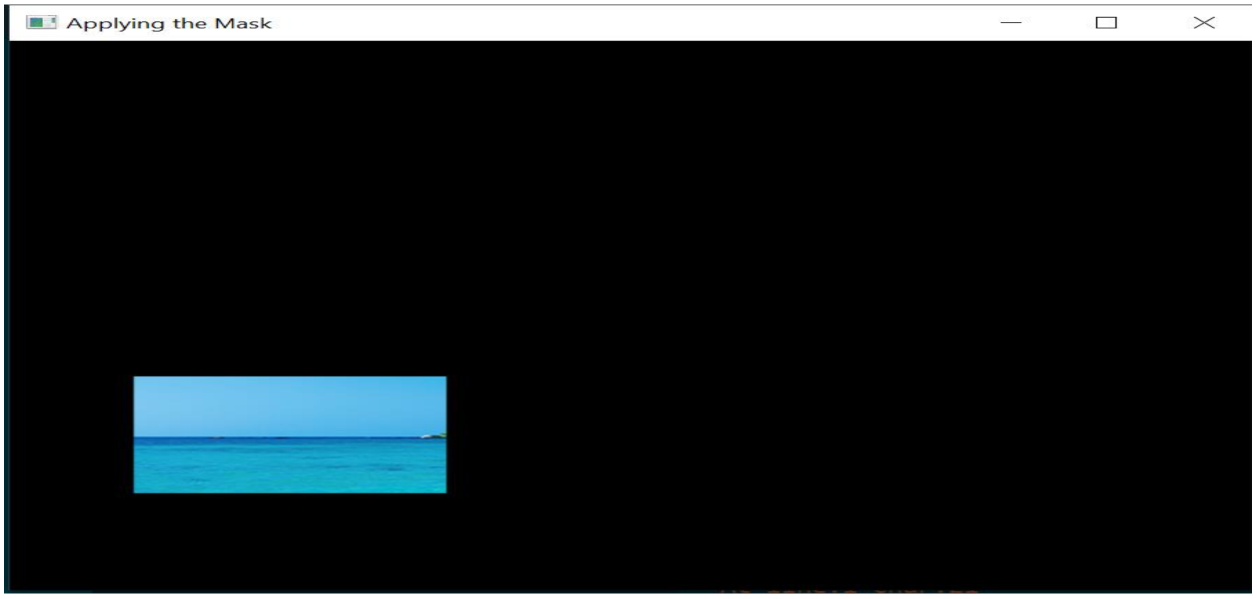
Result:



5) Script:Showing the masked region

```
# display the masked region
masked = cv2.bitwise_and(image, image, mask=mask)
cv2.imshow("Applying the Mask", masked)
```

Result:



6) Script:we compute the histogram of the image using the masked region also and then plotting it.

```
# compute a histogram for our image, but we'll only include pixels
in
# the masked region
plot_histogram(image, "Histogram for Masked Image", mask=mask)
```

```
# show our plots
plt.show()
```

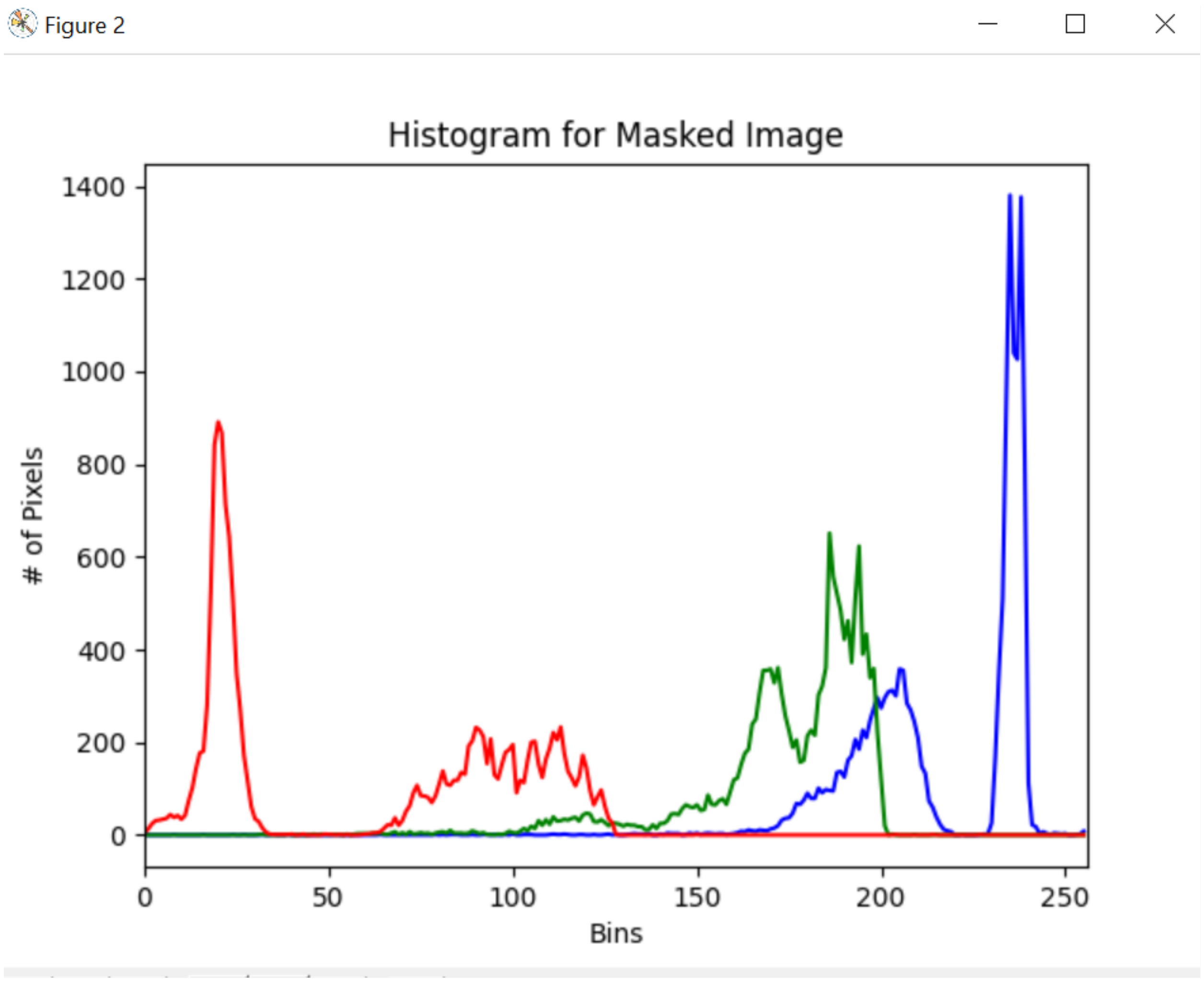


Figure 2
Histogram for Masked Image
# of Pixels
Bins