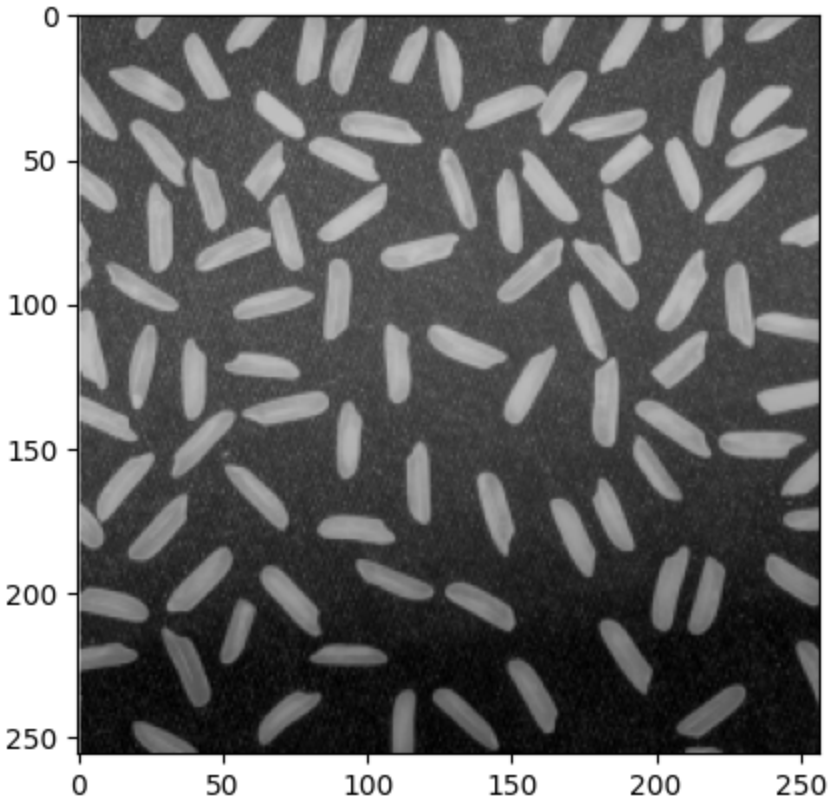
In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
from google.colab.patches import cv2_imshow
from google.colab import files

# uploaded = files.upload()
```

In [ ]:
```
image = cv2.imread('rice.png', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, cmap=plt.cm.gray)
```

Out[ ]: <matplotlib.image.AxesImage at 0x7b247a6b75e0>



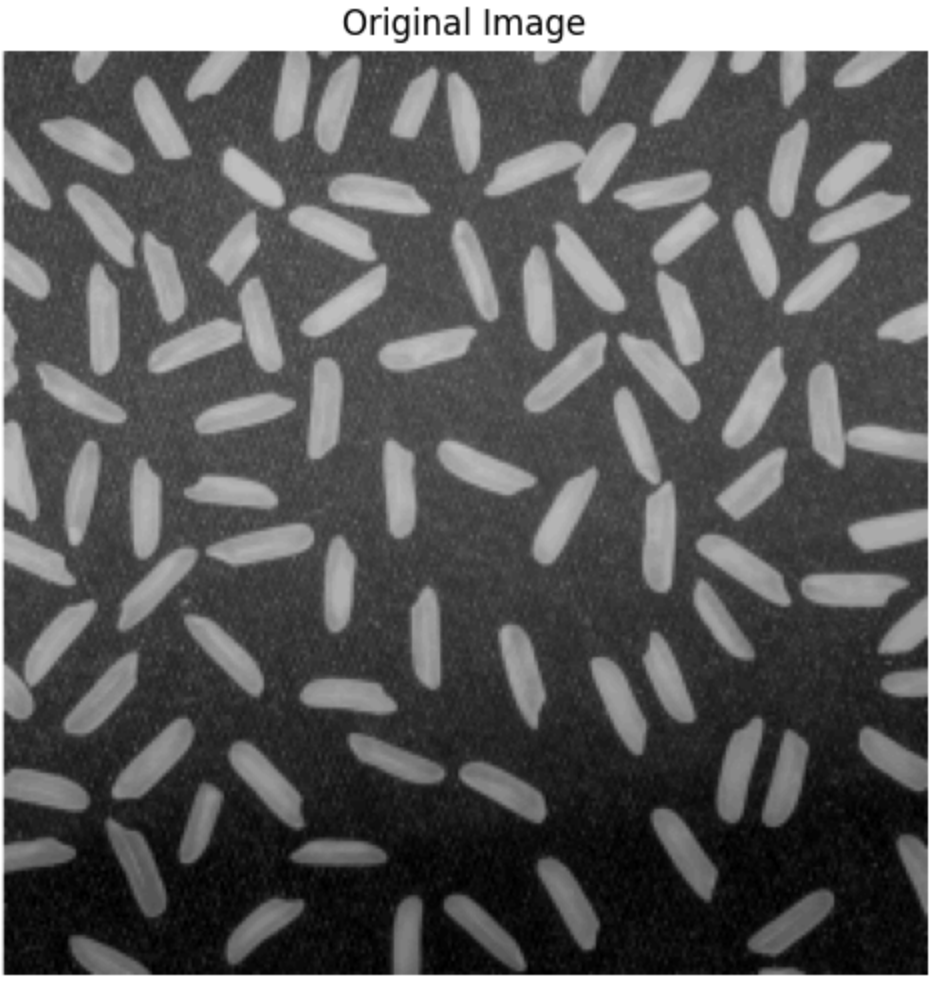
In [ ]:
```
def basic_adaptive_thresholding(image):
    # Compute histogram
    hist = cv2.calcHist([image], [0], None, [256], [0, 256])

    # Select initial estimate threshold T from histogram
    T = np.argmax(hist)

    # Segment the image using T as the threshold
    _, binary_image = cv2.threshold(image, T, 255, cv2.THRESH_BINARY)

    # Divide image into four sub-images
    rows, cols = image.shape
    sub_images = [image[:rows//2, :cols//2],
                  image[:rows//2, cols//2:],
```

```
                image[rows//2:, :cols//2],
                image[rows//2:, cols//2:]]

    # Apply global thresholding independently to each sub-image
    binary_sub_images = [cv2.threshold(sub_image, T, 255, cv2.THRESH_BINARY)[1] for

    # Combine segmented sub-images
    segmented_image = np.vstack((np.hstack((binary_sub_images[0], binary_sub_images
                                np.hstack((binary_sub_images[2], binary_sub_images

    # Connect segmented sub-images to get the full binary image
    full_binary_image = np.zeros_like(image)
    full_binary_image[:rows//2, :cols//2] = binary_sub_images[0]
    full_binary_image[:rows//2, cols//2:] = binary_sub_images[1]
    full_binary_image[rows//2:, :cols//2] = binary_sub_images[2]
    full_binary_image[rows//2:, cols//2:] = binary_sub_images[3]

    return T, binary_image, binary_sub_images, segmented_image, full_binary_image,
```

In [ ]:
```
# Perform basic adaptive thresholding
T, binary_image, binary_sub_images, segmented_image, full_binary_image, hist = basi
```

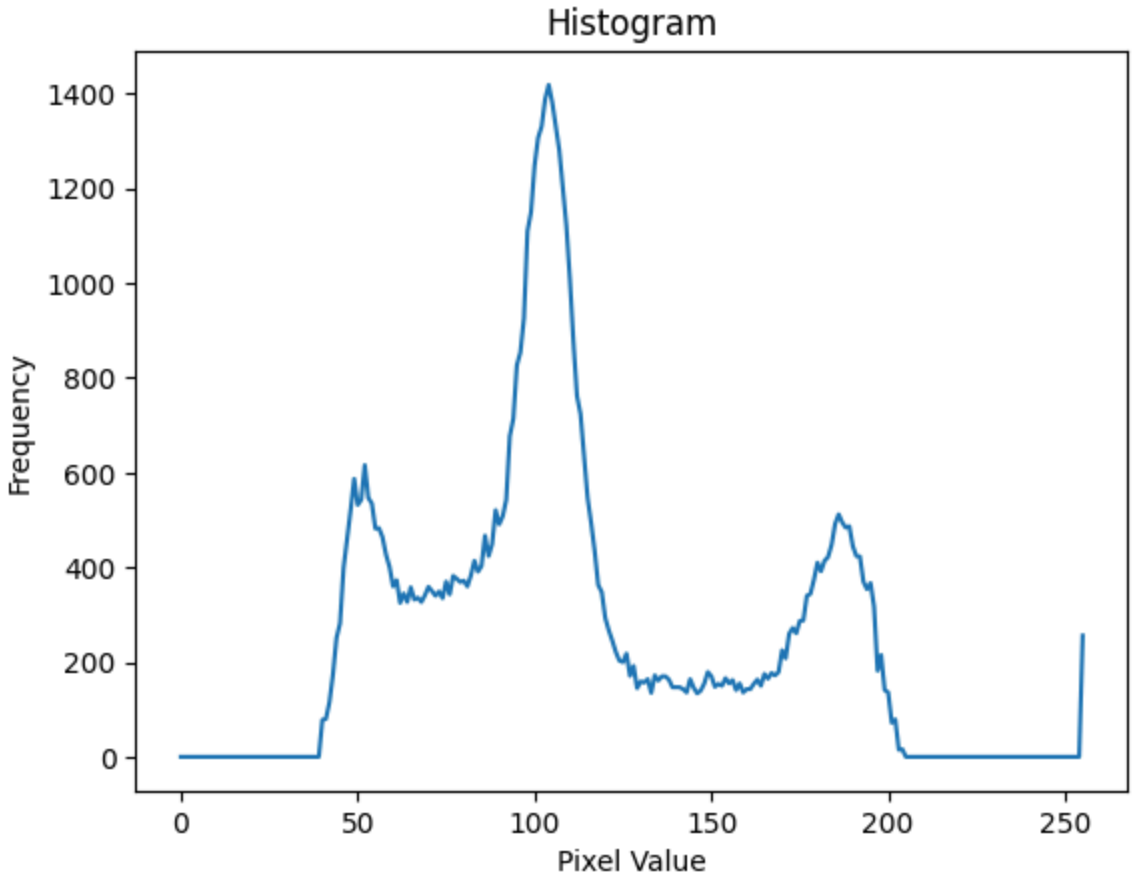
In [ ]:
```
# Display the original image
plt.figure(figsize=(8, 6))
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')
```

Out[ ]: (-0.5, 256.5, 255.5, -0.5)



In [ ]:
```
# Display the histogram
plt.plot(hist)
plt.title('Histogram')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
```

Out[ ]:
```
Text(0, 0.5, 'Frequency')
```

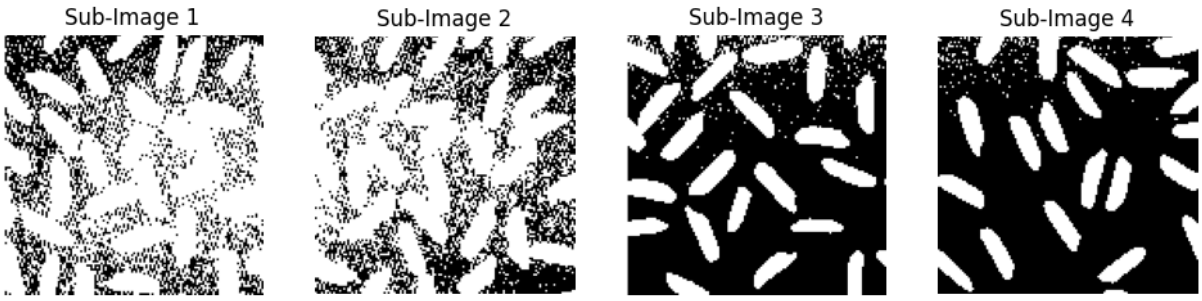


In [ ]:
```
# Display the binary image
plt.imshow(binary_image, cmap='gray')
plt.title('Binary Image (Threshold = {})'.format(T))
plt.axis('off')
```

Out[ ]: (-0.5, 256.5, 255.5, -0.5)

In [ ]:
```
# Display the sub-images after thresholding
fig, ax = plt.subplots(figsize=(12, 8))
for i in range(4):
    ax = plt.subplot(1, 4, i+1)
    ax.imshow(binary_sub_images[i], cmap='gray')
    ax.set_title('Sub-Image {}'.format(i+1))
    ax.axis('off')
```

```
<ipython-input-8-22808f59bf9e>:4: MatplotlibDeprecationWarning: Auto-removal of over
lapping axes is deprecated since 3.6 and will be removed two minor releases later; e
xplicitly call ax.remove() as needed.
  ax = plt.subplot(1, 4, i+1)
```



In [ ]:
```
# Display the segmented image
plt.imshow(segmented_image, cmap='gray')
plt.title('Segmented Image')
plt.axis('off')
```

Out[ ]:
```
(-0.5, 256.5, 255.5, -0.5)
```

In [ ]:
```
# Display the full binary image
plt.imshow(full_binary_image, cmap='gray')
plt.title('Full Binary Image')
plt.axis('off')

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

In [ ]: