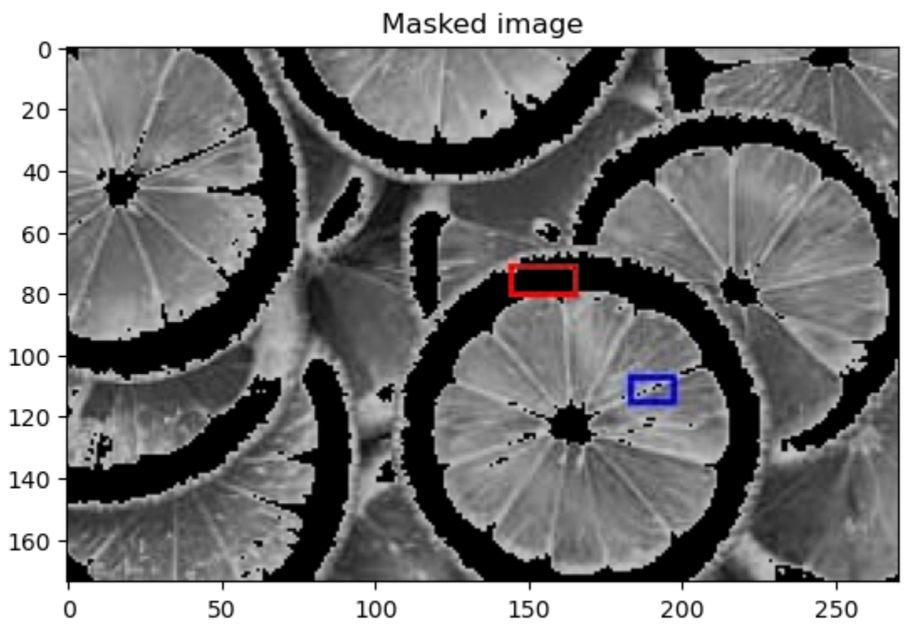
# Use white mask and invert to black to get the desired output

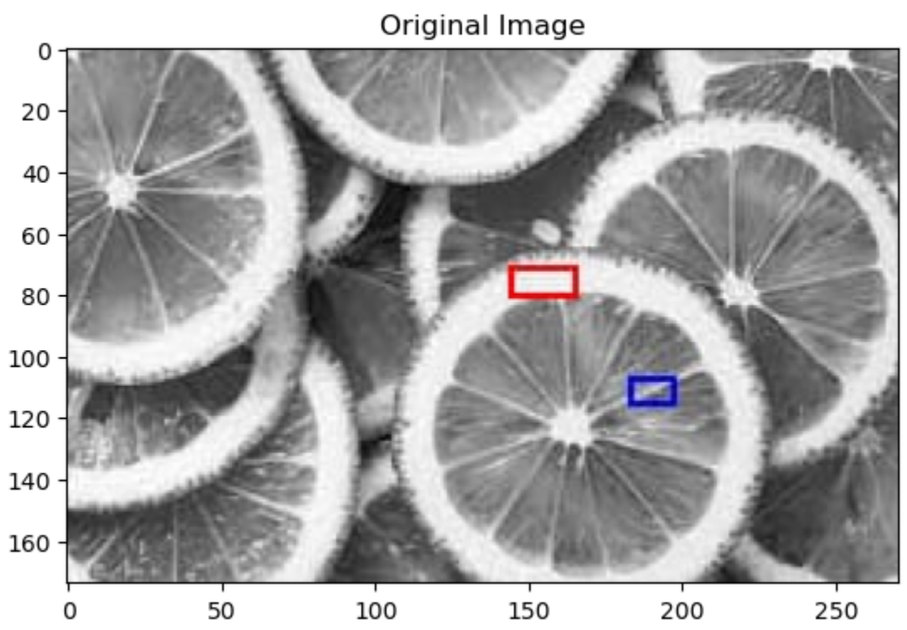
Convert-->grayscale -->create a black and white mask based on pixel intensity --> return an image with white region painted black

In [29]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def detect_and_paint_white(image_path):
    original_image = cv2.imread(image_path)
    grayscale_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY) #-->to grayscale

    _, white_mask = cv2.threshold(grayscale_image, 200, 255, cv2.THRESH_BINARY) #-->inverting white to black
    black_mask = cv2.bitwise_not(white_mask)

    black_canvas = np.zeros_like(original_image) #-->black canvas with same dim as white
    black_canvas = cv2.bitwise_and(original_image, original_image, mask=black_mask)

    return black_canvas

result_image = detect_and_paint_white('image_new.jpeg')

plt.plot(150)
plt.imshow(cv2.cvtColor(result_image, cv2.COLOR_BGR2RGB))
plt.title('Masked image')
plt.show()
```



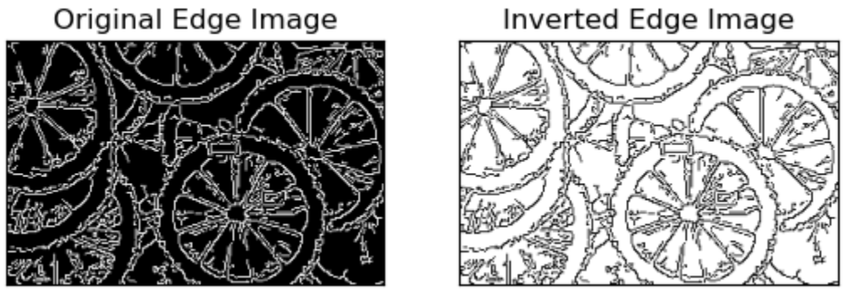

In [30]:
```
plt.plot(121)
plt.imshow(cv2.cvtColor(cv2.imread('image_new.jpeg'), cv2.COLOR_BGR2RGB))
plt.title('Original Image')
```

Out[30]:
```
Text(0.5, 1.0, 'Original Image')
```




## Edge detetction

In [31]:
```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('image_new.jpeg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"

edges = cv.Canny(img, 100, 200)
inverted_edges = cv.bitwise_not(edges)

plt.subplot(121), plt.imshow(edges, cmap='gray')
plt.title('Original Edge Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(inverted_edges, cmap='gray')
plt.title('Inverted Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```




In [ ]:
```
```