An-Najah National University
Department of Computer Engineering
Digital Image Processing – 10636318
First Semester 2024/2025 – P2
OpenCV Project

Name: Ibrahim Mashaqi **Registration number: 12218206**

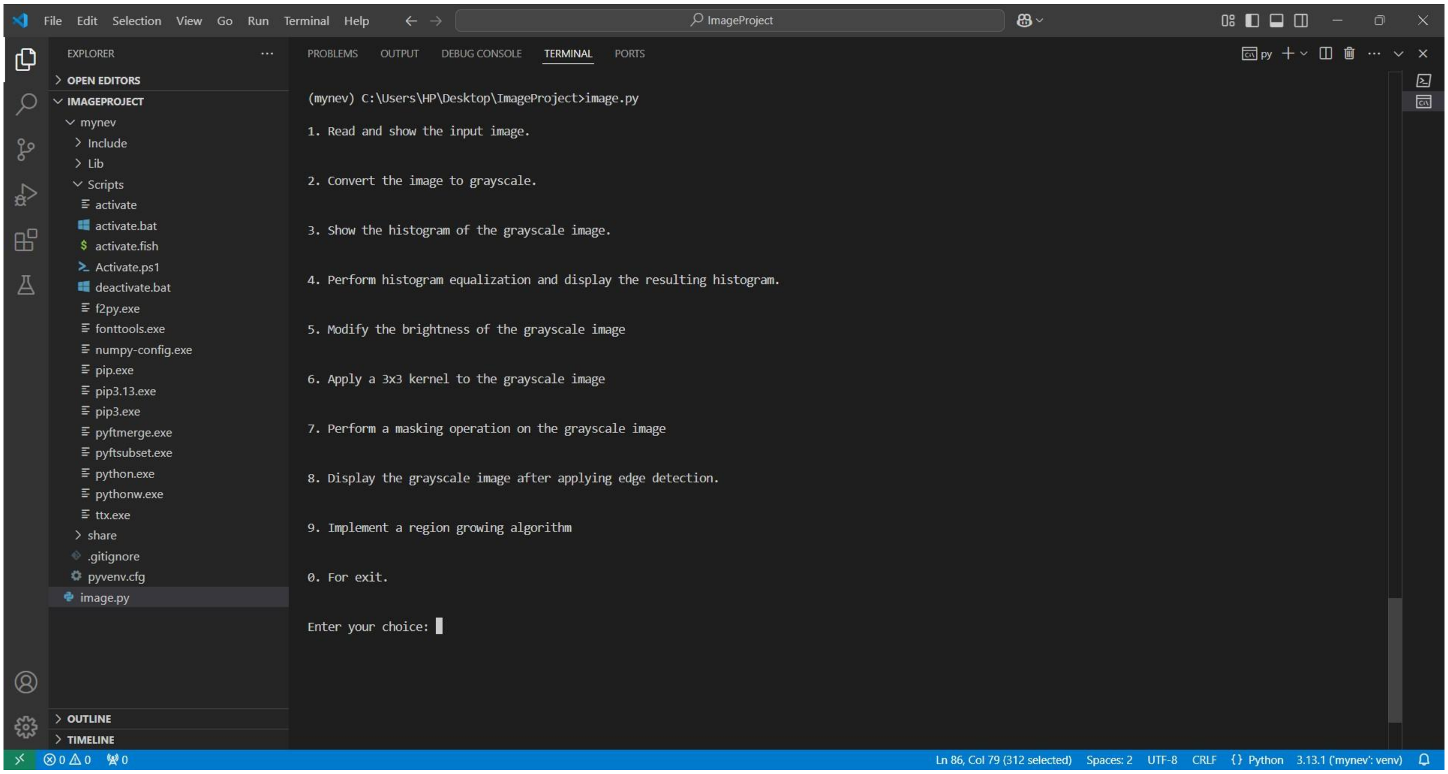
Instructor name: Anas Toma **Date: 1/9/2025**

Introduction:

This report is an interactive Python program dealing with basic implementations of image processing techniques using an OpenCV library. A user working on this program may apply multiple operations to these images: grayscale transformation, histogram equalization, adjustment of brightness, edge detection, and applying kernels. Herein, a menu-driven interface provides a platform where users can perform a practical investigation of the process for understanding simple transformations and analyses of images.

Program Features

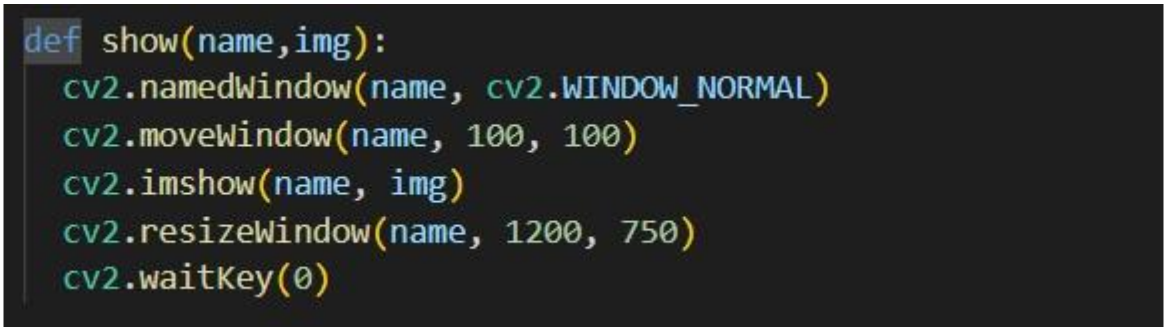
This program displays a menu to the user for selecting any of the particular tasks by entering the corresponding option. The menu for the Program as the following:



```
(mynev) C:\Users\HP\Desktop\ImageProject>image.py

1. Read and show the input image.


2. Convert the image to grayscale.


3. Show the histogram of the grayscale image.


4. Perform histogram equalization and display the resulting histogram.


5. Modify the brightness of the grayscale image


6. Apply a 3x3 kernel to the grayscale image


7. Perform a masking operation on the grayscale image


8. Display the grayscale image after applying edge detection.


9. Implement a region growing algorithm


0. For exit.


Enter your choice: 
```

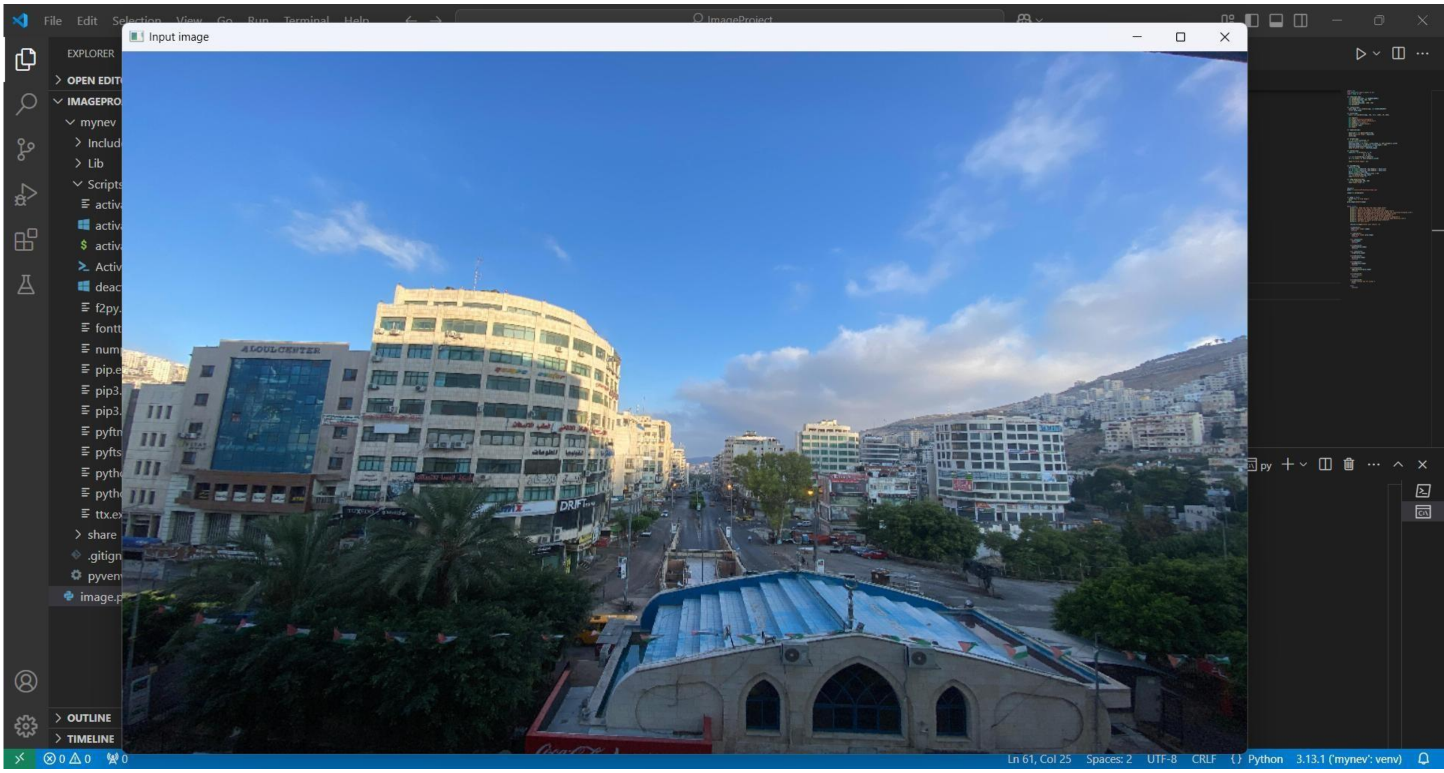
1. Read and Display the Input Image

- The program reads an image from the specified file path and displays it in a resizable window.
- Function Used: show(name, img)
- Code:

- 

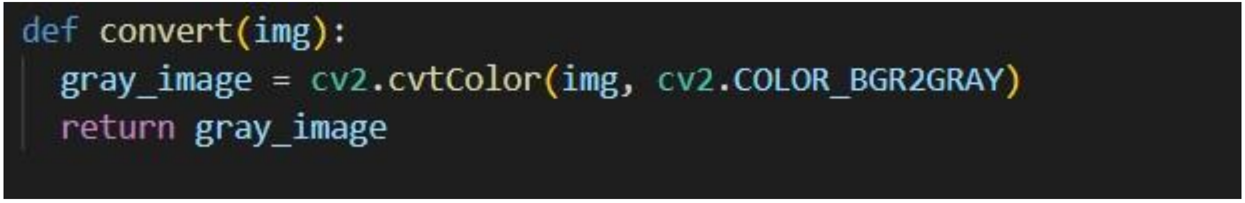
```
def show(name,img):
  cv2.namedWindow(name, cv2.WINDOW_NORMAL)
  cv2.moveWindow(name, 100, 100)
  cv2.imshow(name, img)
  cv2.resizeWindow(name, 1200, 750)
  cv2.waitKey(0)
```

Output image:

- 

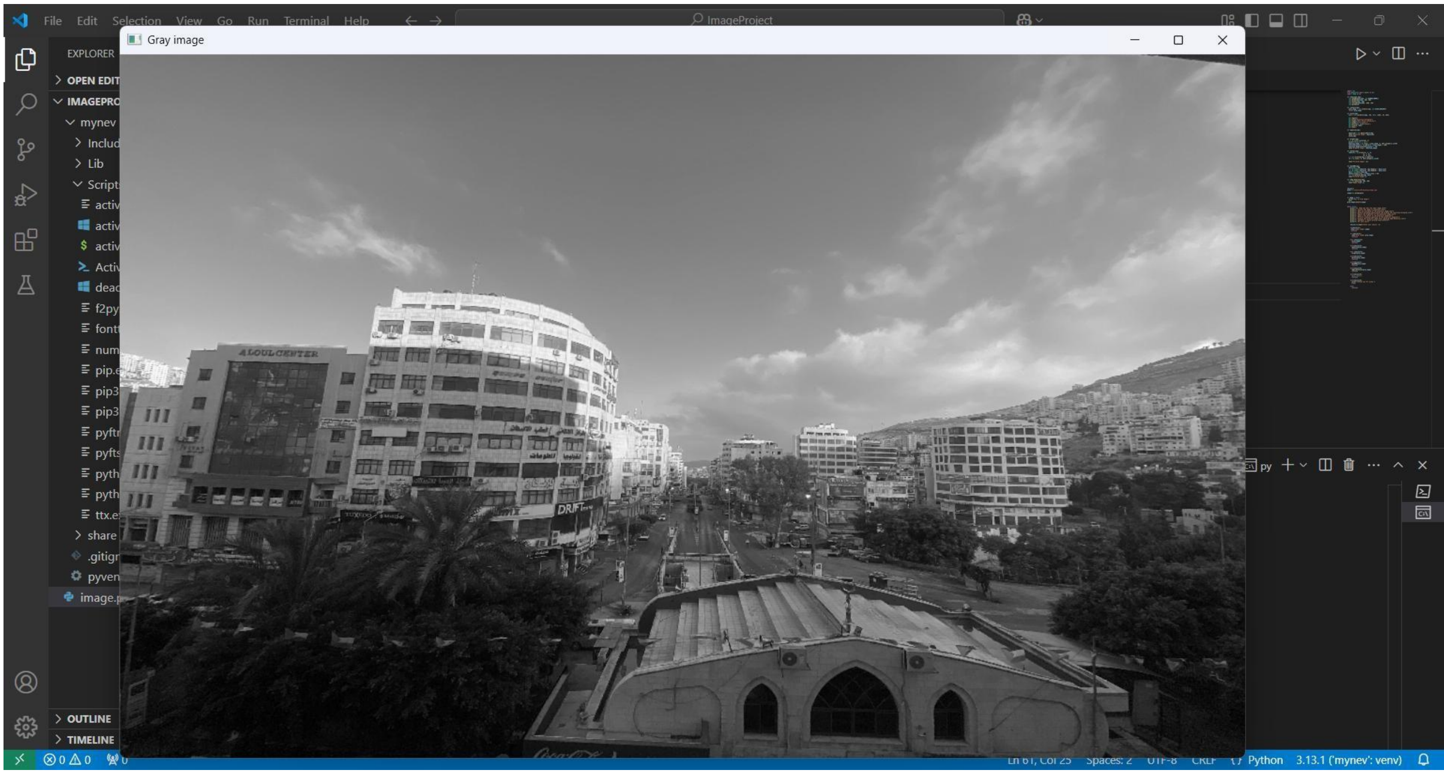
2. Convert the Image to Grayscale

- Converts the input image to grayscale using the OpenCV function cv2.cvtColor().
- Function Used: convert(img) • Code:



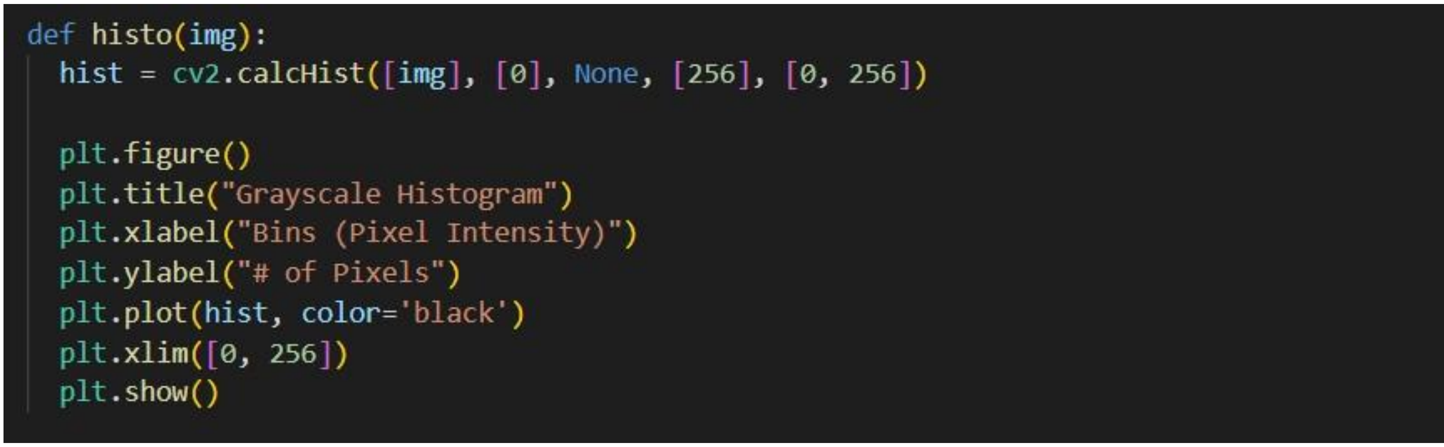
```
def convert(img):
  gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  return gray_image
```

Output image:

- 

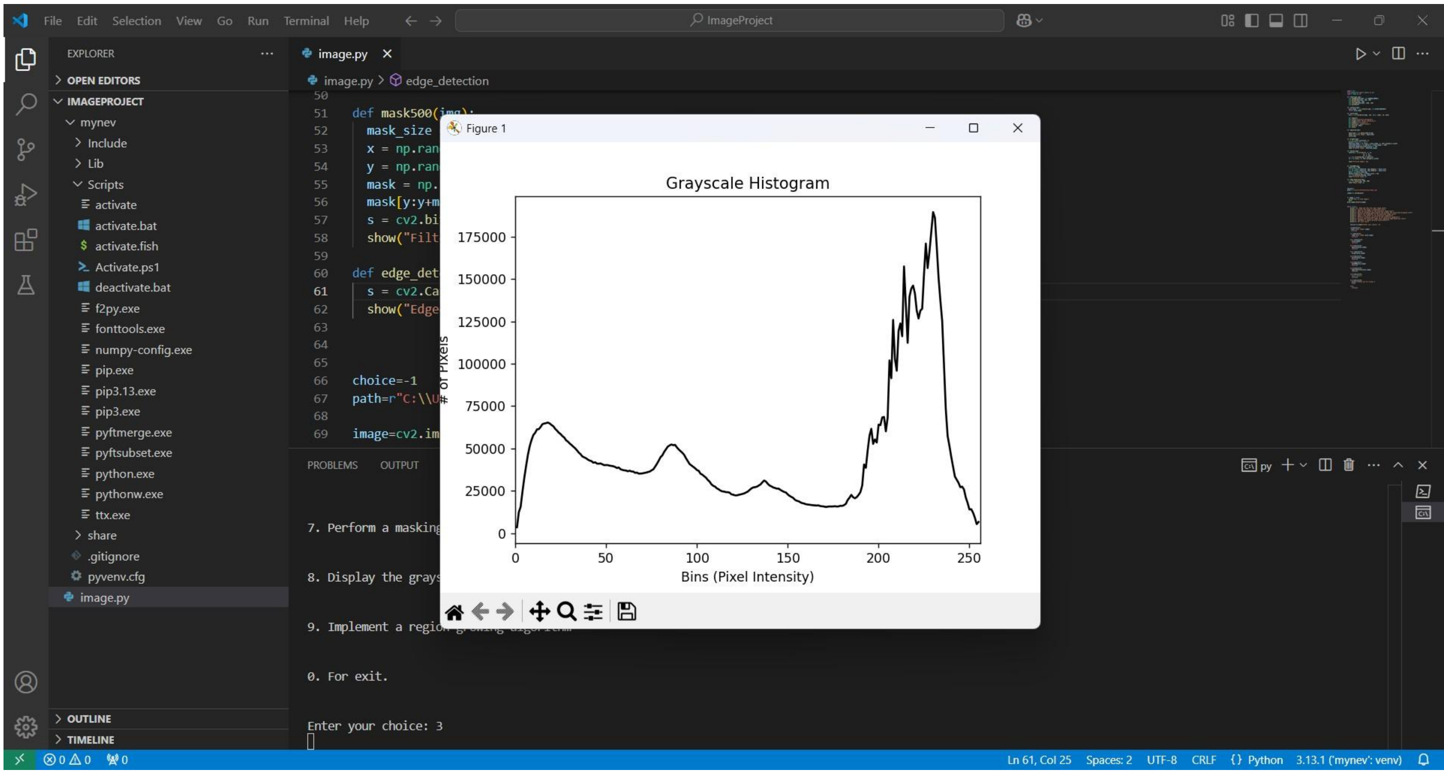
3. Display the Grayscale Histogram

- Computes and visualizes the histogram of the grayscale image to understand pixel intensity distribution.
- Function Used: histo(img) • Code:

- 

```python
def histo(img):
  hist = cv2.calcHist([img], [0], None, [256], [0, 256])

  plt.figure()
  plt.title("Grayscale Histogram")
  plt.xlabel("Bins (Pixel Intensity)")
  plt.ylabel("# of Pixels")
  plt.plot(hist, color='black')
  plt.xlim([0, 256])
  plt.show()
```

Output image:



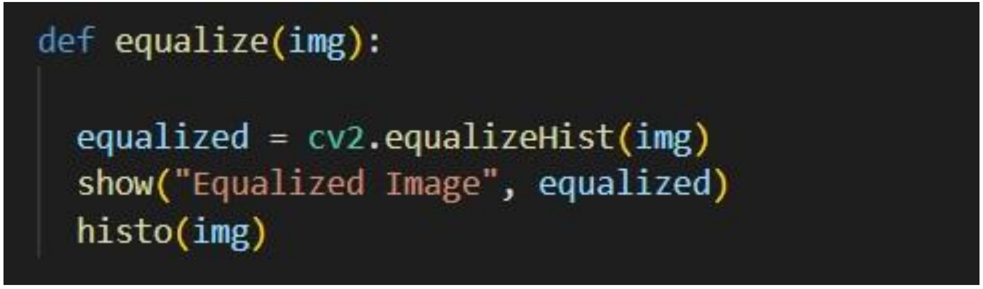
```
def mask500(img):
  mask_size
  x = np.ran
  y = np.ran
  mask = np.
  mask[y:y+m
  s = cv2.bi
  show("Filt

def edge_det
  s = cv2.Ca
  show("Edge



choice=-1
path=r"C:\\U

image=cv2.im
```

```
7. Perform a masking

8. Display the grays

9. Implement a region growing algorithm

0. For exit.

Enter your choice: 3
```

-

4. Perform Histogram Equalization

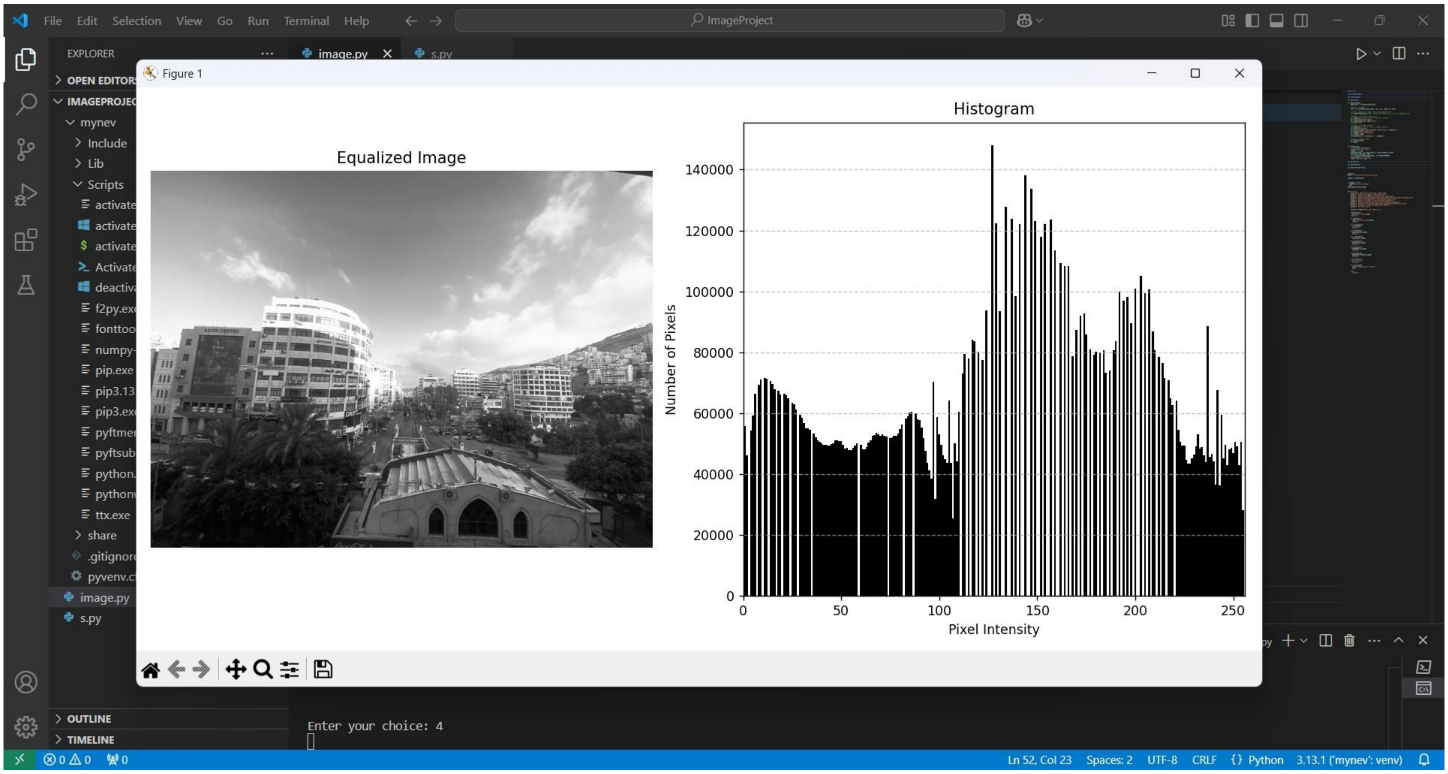
- Enhances the contrast of the grayscale image using histogram equalization.
- Displays the resulting image along with its updated histogram.
- Function Used: equalize(img) • Code:



```
def equalize(img):

  equalized = cv2.equalizeHist(img)
  show("Equalized Image", equalized)
  histo(img)
```

-

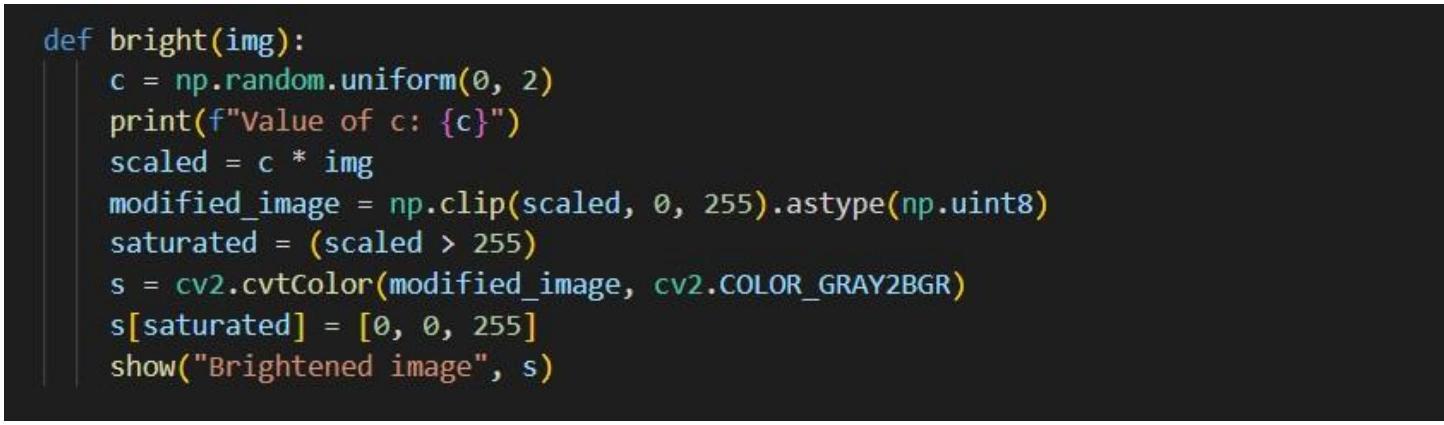
Output image:



```
Enter your choice: 4
```

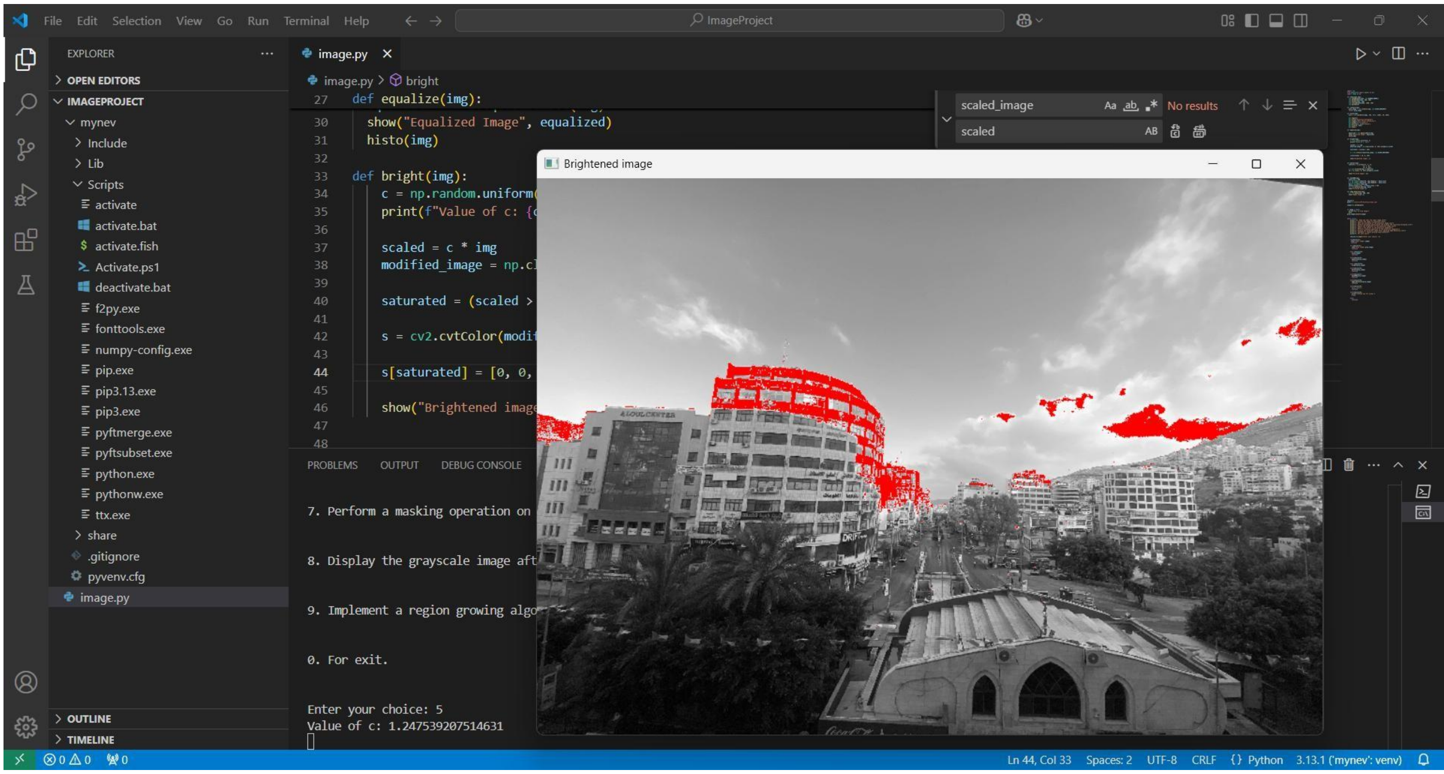
5. Modify the Brightness of the Grayscale Image

- This modifies the brightness by multiplying all the intensities by a random value c
- Uses np.clip() to ensure pixel values remain valid (0-255).
- Uses np.clip() to ensure pixel values remain valid (0-255).
- Highlights, in red, those areas where pixel values exceed 255-saturated regions.
- Converts the grayscale image to a color (BGR) image to allow highlighting.



```
def bright(img):
    c = np.random.uniform(0, 2)
    print(f"Value of c: {c}")
    scaled = c * img
    modified_image = np.clip(scaled, 0, 255).astype(np.uint8)
    saturated = (scaled > 255)
    s = cv2.cvtColor(modified_image, cv2.COLOR_GRAY2BGR)
    s[saturated] = [0, 0, 255]
    show("Brightened image", s)
```

Output image:



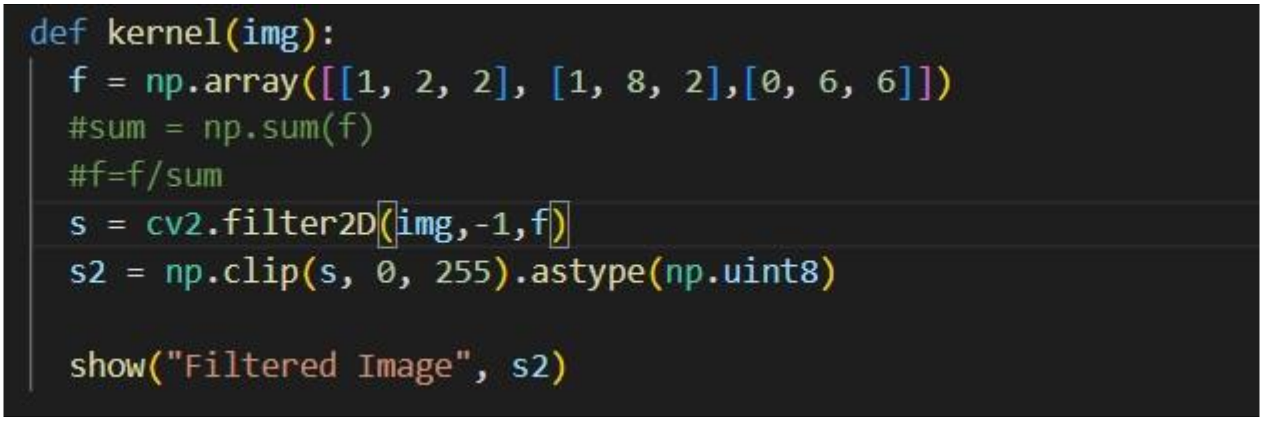
```
def equalize(img):
  show("Equalized Image", equalized)
  histo(img)

def bright(img):
    c = np.random.uniform(
    print(f"Value of c: {c

    scaled = c * img
    modified_image = np.cl

    saturated = (scaled >

    s = cv2.cvtColor(modif

    s[saturated] = [0, 0,

    show("Brightened image
```

```
7. Perform a masking operation on

8. Display the grayscale image aft

9. Implement a region growing algo

0. For exit.

Enter your choice: 5
Value of c: 1.247539207514631
```

- In the example c=1.247 • Function Used: bright(img) • Code:

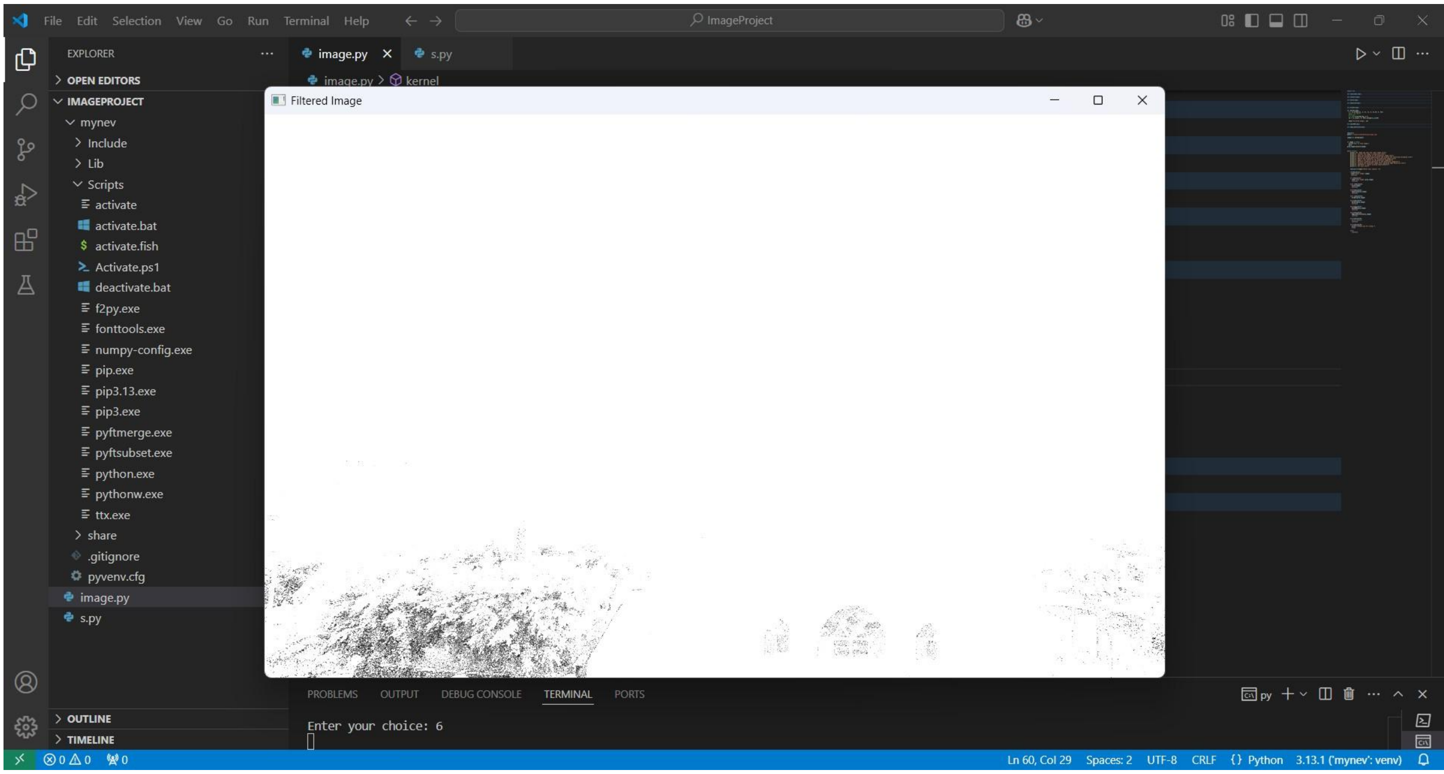
6. Apply a 3x3 Kernel to the Grayscale Image

- Applies a kernel filter with my registration number (12218206) to the grayscale image.
- It focuses on the center pixel so it won’t affect that much
- I didn’t use normalization so the output will be all white
- Function Used: kernel(img) • Code:

- 

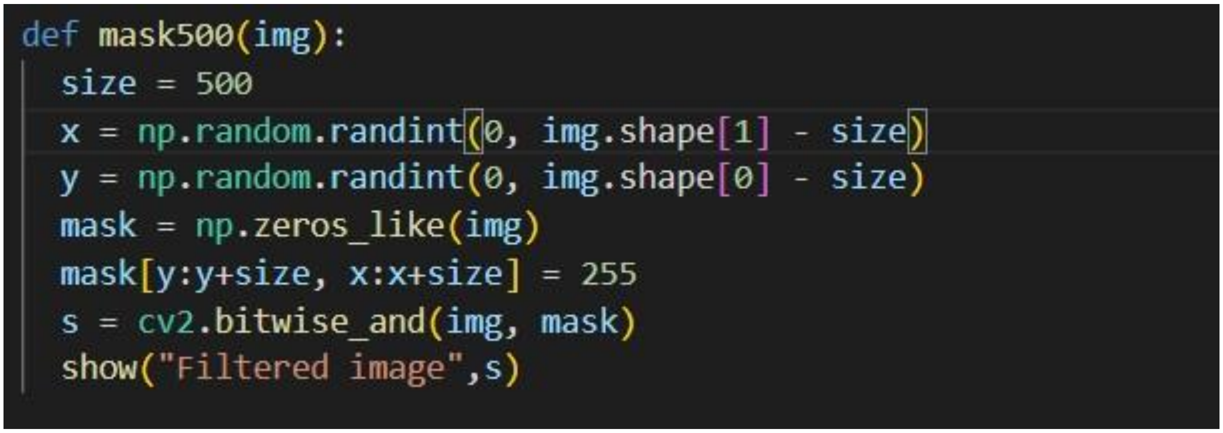
```
def kernel(img):
  f = np.array([[1, 2, 2], [1, 8, 2],[0, 6, 6]])
  #sum = np.sum(f)
  #f=f/sum
  s = cv2.filter2D(img,-1,f)
  s2 = np.clip(s, 0, 255).astype(np.uint8)

  show("Filtered Image", s2)
```

Output image:

- 

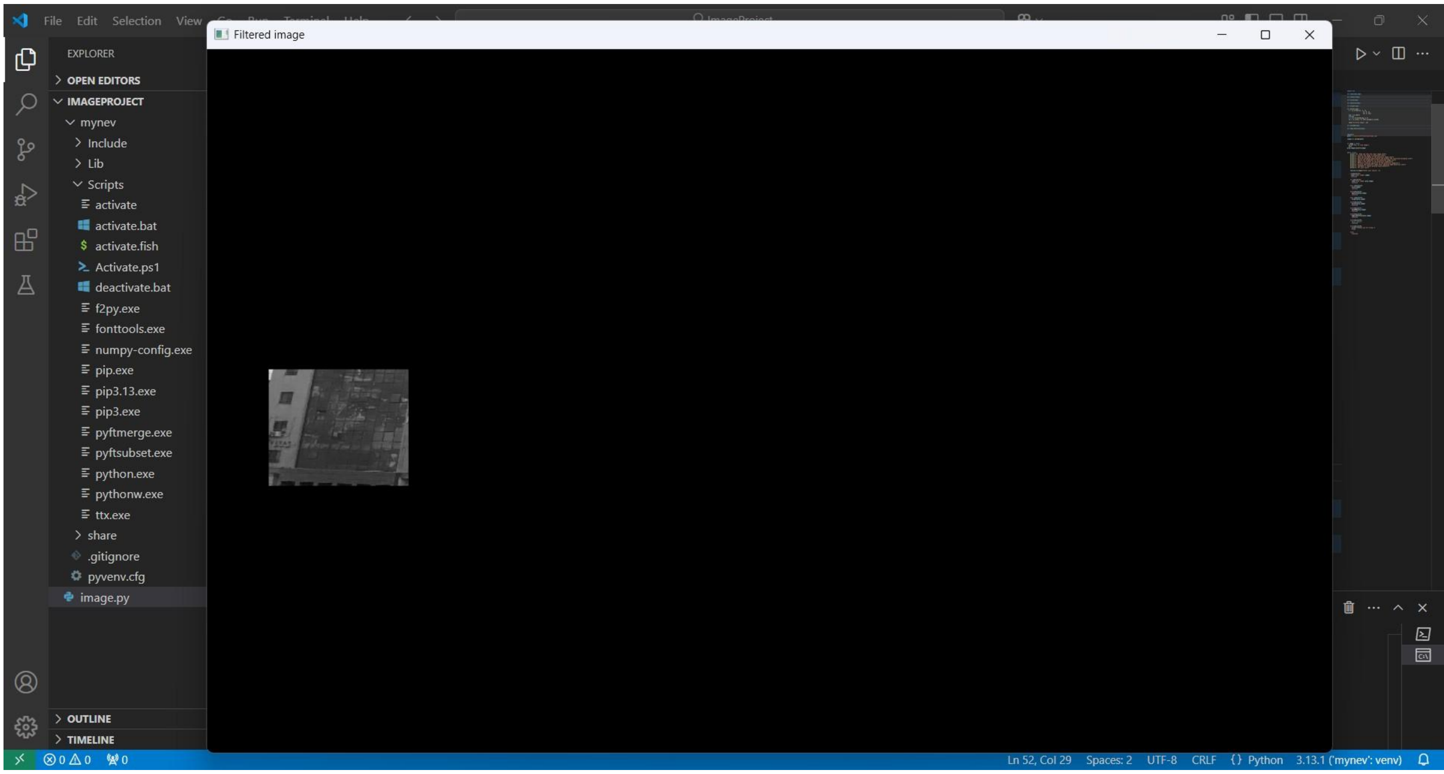
```
Enter your choice: 6
```

7. Perform Masking Operation

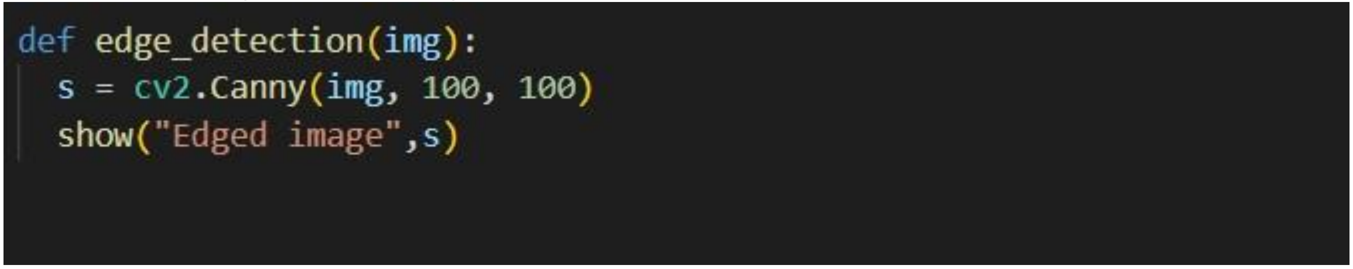
- Creates a random square mask of size 500x500 pixels and applies it to the grayscale image.
- Function Used: mask500(img)
- Code:
- 

```
def mask500(img):
  size = 500
  x = np.random.randint(0, img.shape[1] - size)
  y = np.random.randint(0, img.shape[0] - size)
  mask = np.zeros_like(img)
  mask[y:y+size, x:x+size] = 255
  s = cv2.bitwise_and(img, mask)
  show("Filtered image",s)
```

Output image:

- 

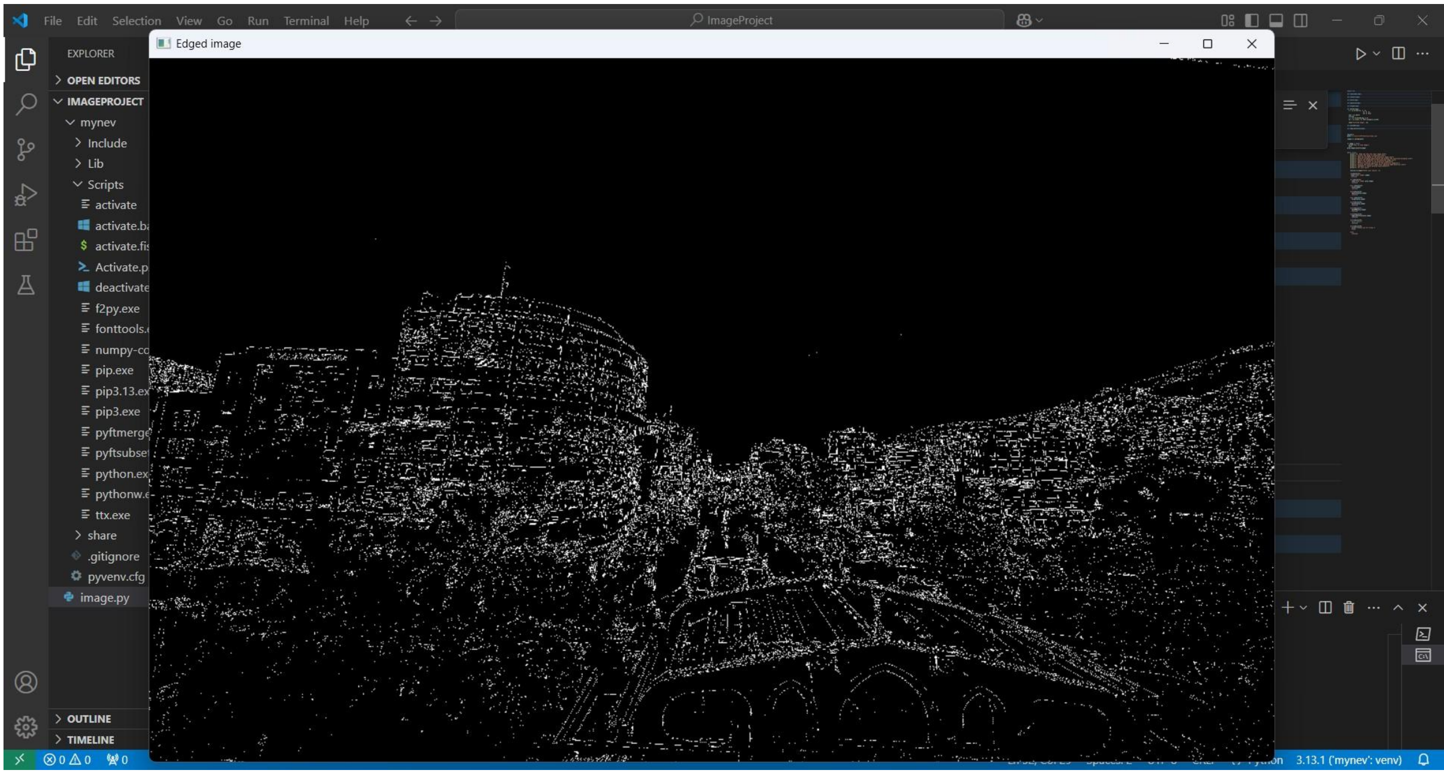
8. Edge Detection

- Detects edges in the grayscale image using the Canny edge detection algorithm with filter size 100*100.
- Function Used: edge_detection(img) • Code:
- 

```
def edge_detection(img):
  s = cv2.Canny(img, 100, 100)
  show("Edged image",s)
```

-

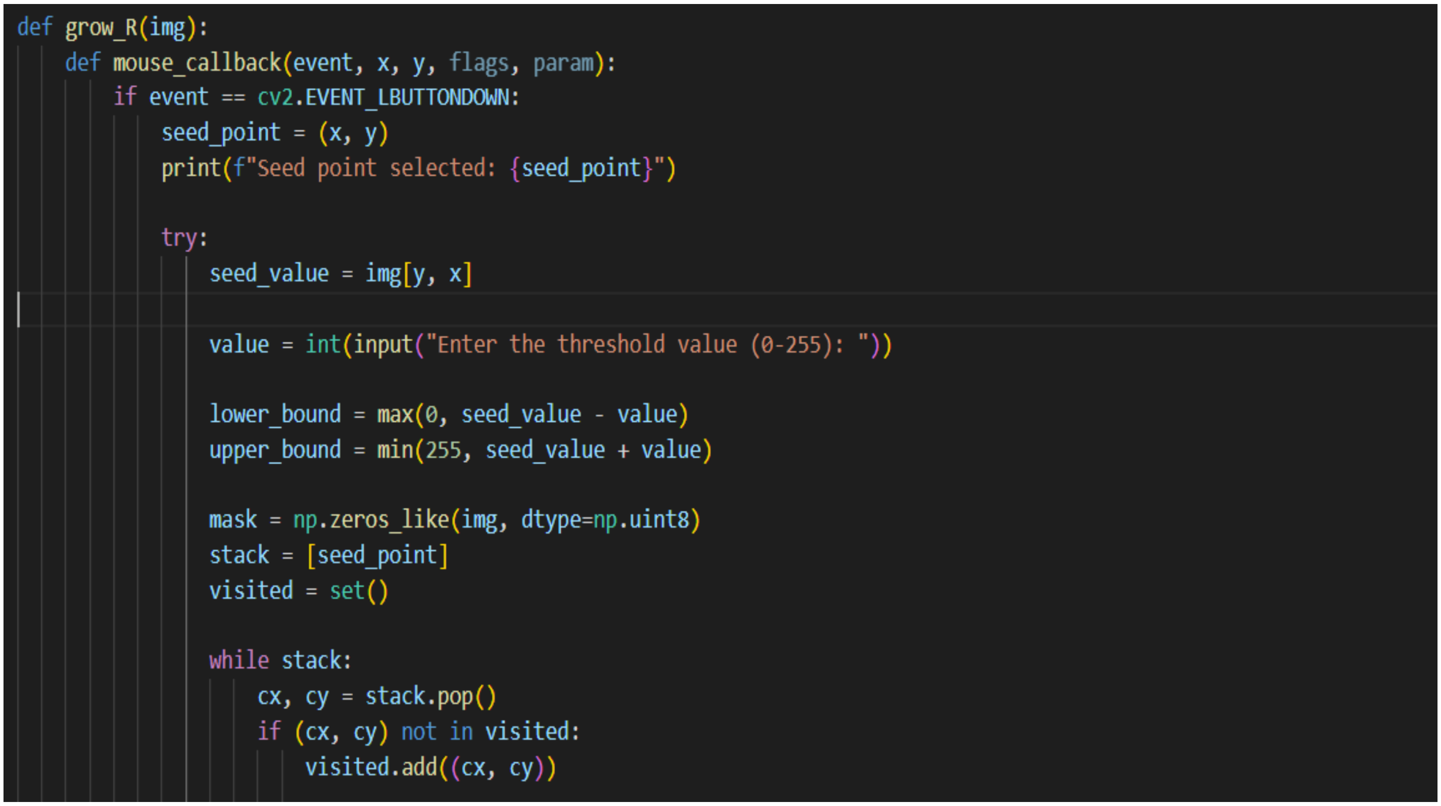
Output image:



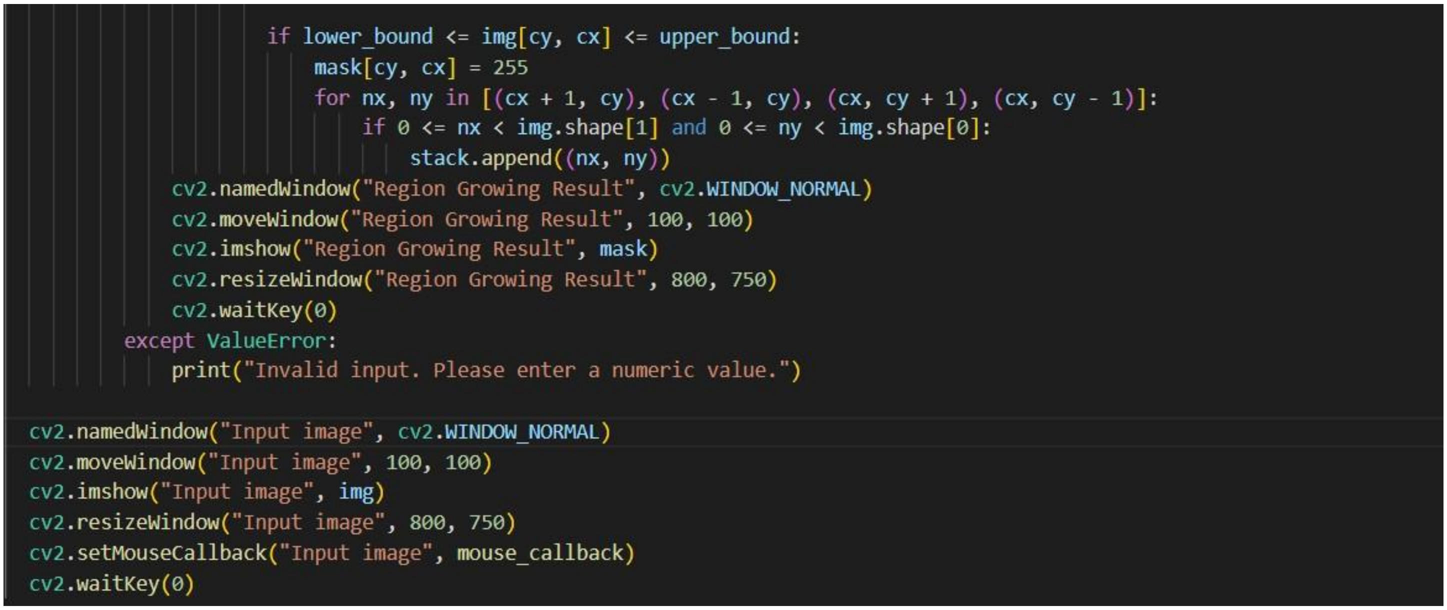
9. Region Growing Algorithm (Placeholder)

- Placeholder for implementing a region-growing algorithm to segment regions in the image.
- User selects a seed point on the image by clicking the left mouse button.
- Allows the user to define the lower and upper bounds for pixel intensity.
- Explores neighboring pixels recursively to determine whether they fall within the specified intensity range .
- Constructs a binary mask where the region is displayed in white (255) and the rest is black (0).
- Displays the input image and the resulting binary mask in separate windows.
- grow(img): Main function to perform region growing on the given input image.

- Code:



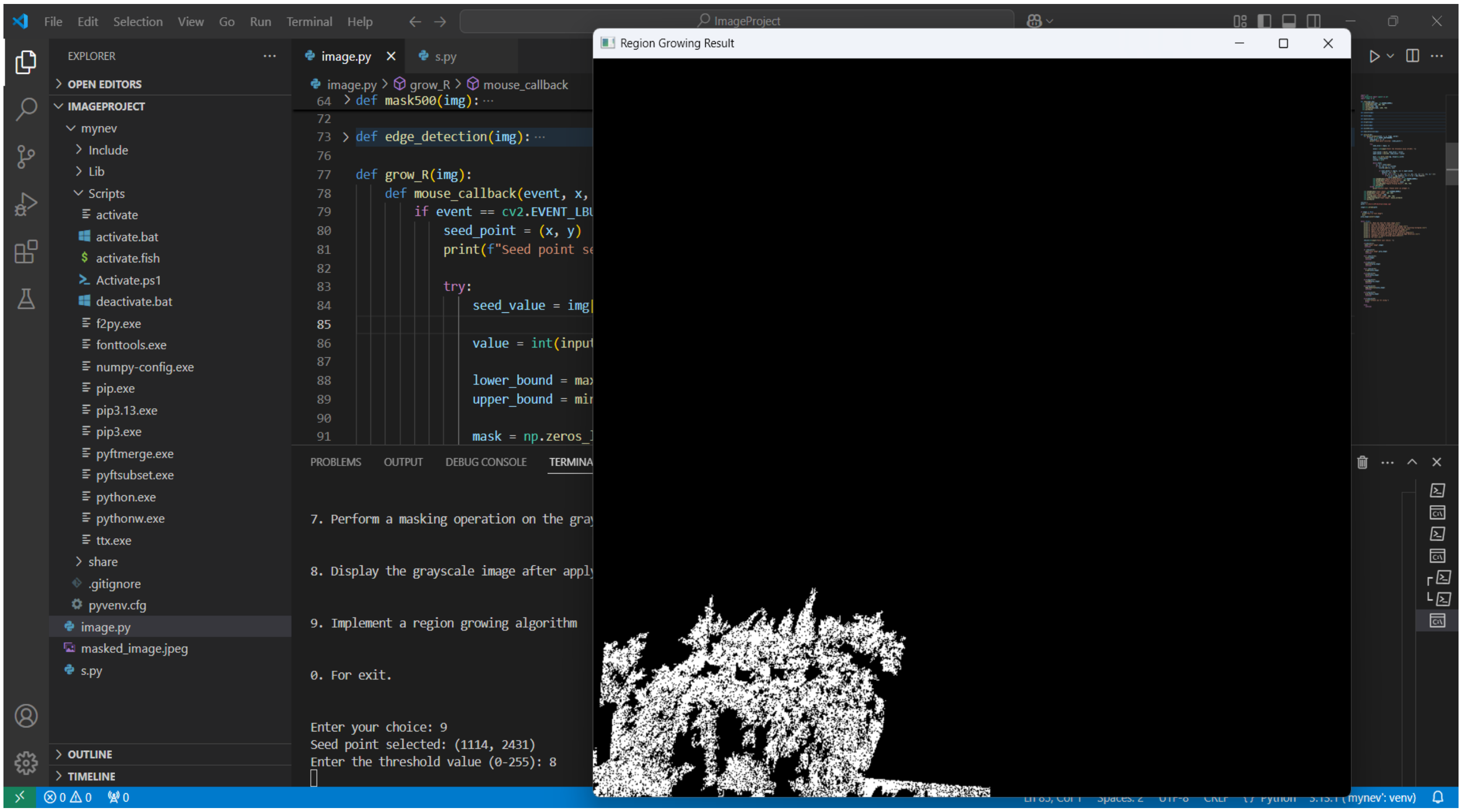
```
def grow_R(img):
    def mouse_callback(event, x, y, flags, param):
        if event == cv2.EVENT_LBUTTONDOWN:
            seed_point = (x, y)
            print(f"Seed point selected: {seed_point}")

            try:
                seed_value = img[y, x]

                value = int(input("Enter the threshold value (0-255): "))

                lower_bound = max(0, seed_value - value)
                upper_bound = min(255, seed_value + value)

                mask = np.zeros_like(img, dtype=np.uint8)
                stack = [seed_point]
                visited = set()

                while stack:
                    cx, cy = stack.pop()
                    if (cx, cy) not in visited:
                        visited.add((cx, cy))
```



```
                        if lower_bound <= img[cy, cx] <= upper_bound:
                            mask[cy, cx] = 255
                            for nx, ny in [(cx + 1, cy), (cx - 1, cy), (cx, cy + 1), (cx, cy - 1)]:
                                if 0 <= nx < img.shape[1] and 0 <= ny < img.shape[0]:
                                    stack.append((nx, ny))
                cv2.namedWindow("Region Growing Result", cv2.WINDOW_NORMAL)
                cv2.moveWindow("Region Growing Result", 100, 100)
                cv2.imshow("Region Growing Result", mask)
                cv2.resizeWindow("Region Growing Result", 800, 750)
                cv2.waitKey(0)
            except ValueError:
                print("Invalid input. Please enter a numeric value.")

    cv2.namedWindow("Input image", cv2.WINDOW_NORMAL)
    cv2.moveWindow("Input image", 100, 100)
    cv2.imshow("Input image", img)
    cv2.resizeWindow("Input image", 800, 750)
    cv2.setMouseCallback("Input image", mouse_callback)
    cv2.waitKey(0)
```

-

Output image if we click at the trees and enter the threshold value (15):



```
> def mask500(img): ···

> def edge_detection(img): ···

def grow_R(img):
    def mouse_callback(event, x,
        if event == cv2.EVENT_LBU
            seed_point = (x, y)
            print(f"Seed point se

            try:
                seed_value = img[

                value = int(input

                lower_bound = max
                upper_bound = mir

                mask = np.zeros_l
```

```
7. Perform a masking operation on the gray

8. Display the grayscale image after apply

9. Implement a region growing algorithm

0. For exit.

Enter your choice: 9
Seed point selected: (1114, 2431)
Enter the threshold value (0-255): 8
```

10. Exit the Program

- Ends the program successfully.

Resources:

- OpenCV
- Stack overflow

Thank you.