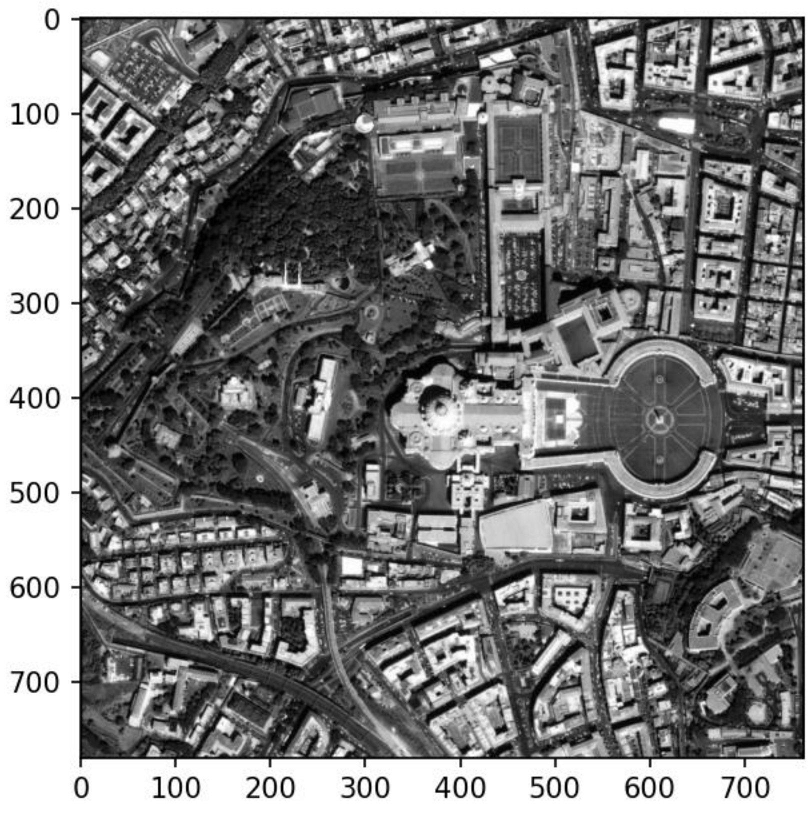
CODE IMPLEMENTATION

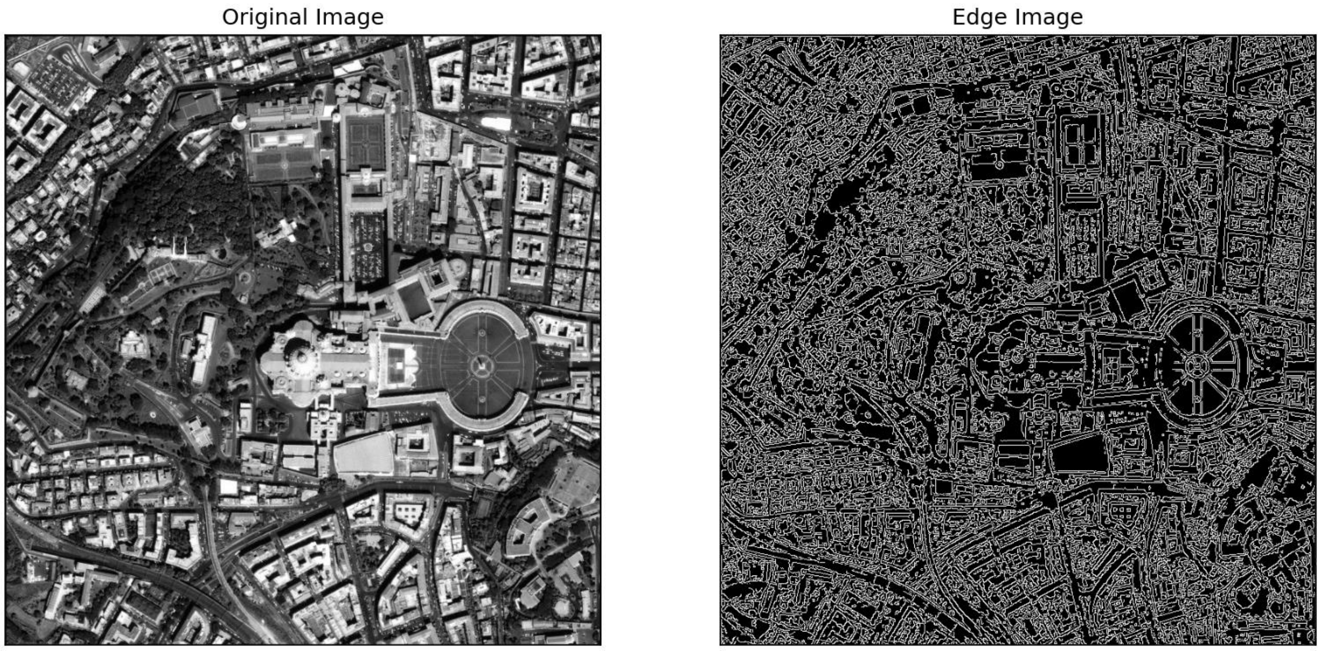
GRAYSCALE CONVERSION:

```
import cv2
import matplotlib.pyplot as plot

img = cv2.imread('data.jpg', cv2.IMREAD_GRAYSCALE)

#diplaying the gray image
plot.imshow(img,cmap="gray")
plot.show()
```



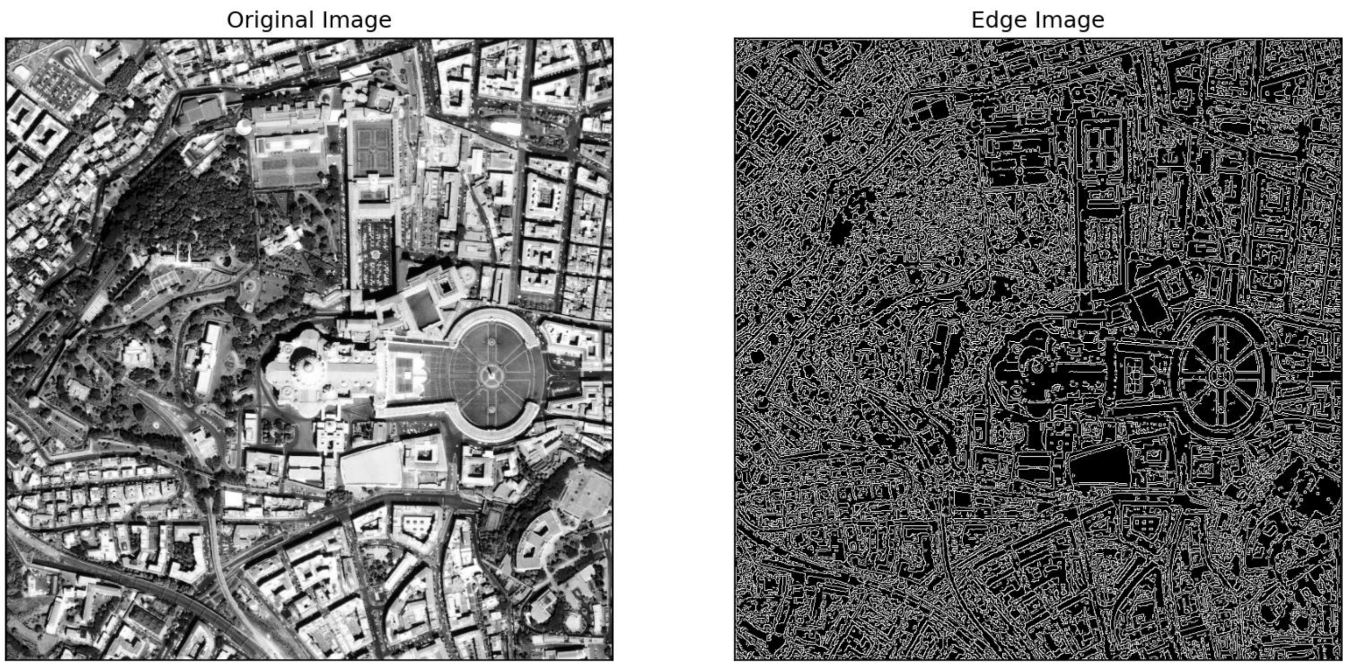
Canny edge deduction:

```
import cv2 as cv
from matplotlib import pyplot as plt

#gray scale conversion
img = cv.imread('data.jpg', cv.IMREAD_GRAYSCALE)

assert img is not None, "file could not be read, check with os.path.exists()"

edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



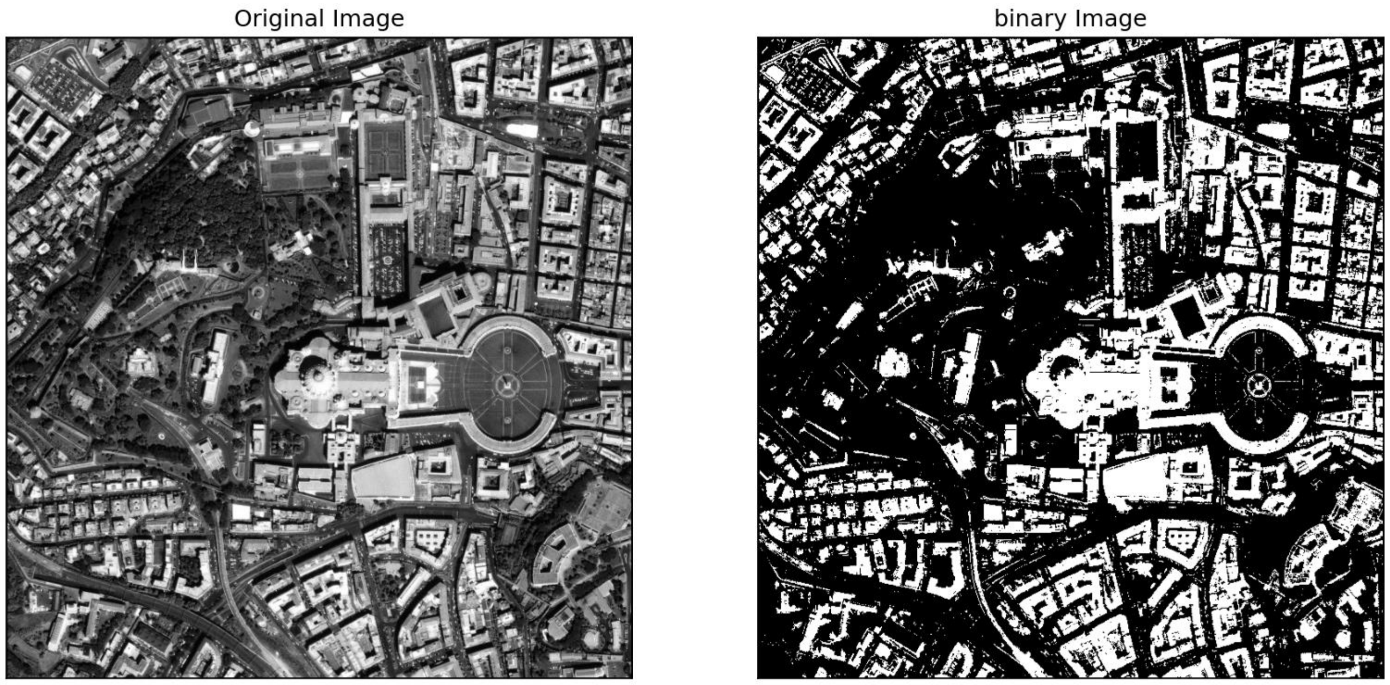
Canny edge deduction of histogram equalized image:

```
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread('data.jpg',0)
img = cv.equalizeHist(img)

edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```



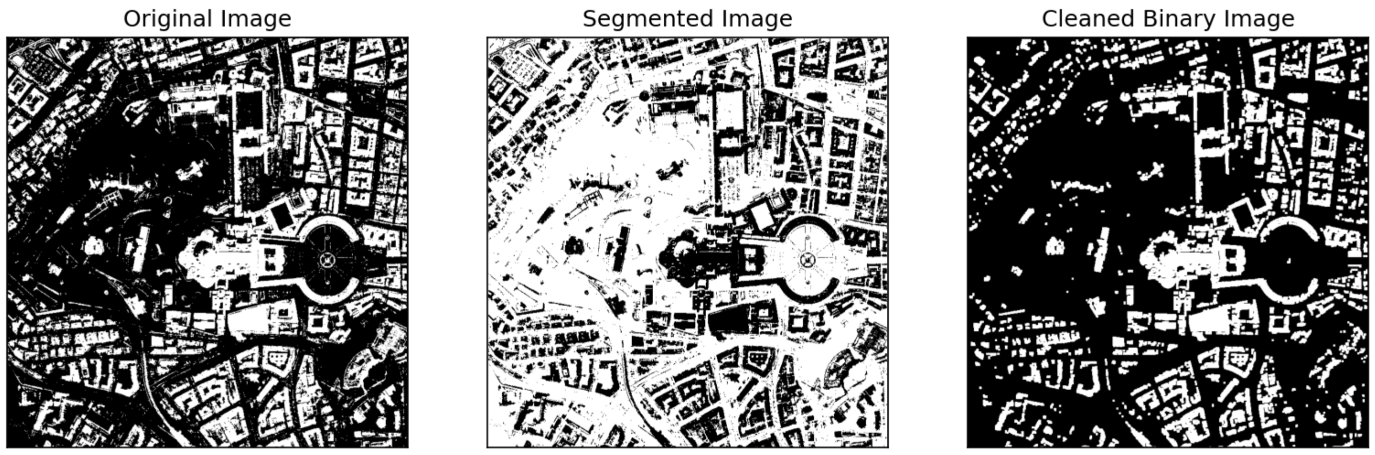
Binary image conversion:

```
import cv2
import matplotlib.pyplot as plt

def convert_to_binary(image_path, threshold_value=127):
    # Read the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply binary thresholding
    _, binary_image = cv2.threshold(image, threshold_value, 255,
cv2.THRESH_BINARY)

    plt.subplot(121), plt.imshow(image, cmap='gray')
    plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(binary_image, cmap="gray")
    plt.title('binary Image'), plt.xticks([]), plt.yticks([])
    plt.show()

# Example usage


image_path = 'data.jpg'
convert_to_binary(image_path)
```



Noise removed from segmented image :

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt


def convert_to_binary(image_path, threshold_value=127):
    # Read the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Apply binary thresholding
    _, binary_image = cv2.threshold(image, threshold_value, 255,
cv2.THRESH_BINARY)

    return binary_image


def remove_noise(image, kernel_size=4):
    # Define the kernel for morphological operations
    kernel = np.ones((kernel_size, kernel_size), np.uint8)

    # Apply erosion followed by dilation to remove noise
    cleaned_image = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

    return cleaned_image


def segment_image(image, num_clusters=3):
    # Reshape the image to be a list of pixels
    pixels = image.reshape((-1, 1))

    # Apply k-means clustering
    kmeans = KMeans(n_clusters=num_clusters)
    kmeans.fit(pixels)

    # Get the labels for each pixel
    labels = kmeans.labels_

    # Reshape labels to the shape of the original image
    segmented_image = labels.reshape(image.shape)

    # Convert labels to 8-bit for display
    segmented_image = (segmented_image * (255 / (num_clusters -
1))).astype(np.uint8)

    # Display the original, noisy, and segmented images
    plt.subplot(131), plt.imshow(image, cmap="gray")
    plt.title('Original Image'), plt.xticks([]), plt.yticks([])

    # Remove noise from the binary image
    cleaned_binary_image = remove_noise(image)
    plt.subplot(133), plt.imshow(cleaned_binary_image, cmap="gray")
    plt.title('Cleaned Binary Image'), plt.xticks([]), plt.yticks([])

    plt.subplot(132), plt.imshow(segmented_image, cmap="gray")
    plt.title('Segmented Image'), plt.xticks([]), plt.yticks([])

    plt.show()


# Example usage
image_path = 'data.jpg'
binary_image = convert_to_binary(image_path)
segment_image(binary_image)
```



Detecting Building Boundaries:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def preprocess_image(image_path, threshold_value=127,
noise_removal_kernel=3):
    # Read the color image
    color_image = cv2.imread(image_path)

    # Convert to grayscale
    gray_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding
    _, binary_image = cv2.threshold(gray_image, threshold_value, 255,
cv2.THRESH_BINARY)

    # Define the kernel for morphological operations
    kernel = np.ones((noise_removal_kernel, noise_removal_kernel), np.uint8)

    # Apply erosion followed by dilation to remove noise
    cleaned_binary_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN,
kernel)

    return color_image, binary_image, cleaned_binary_image

def detect_buildings(image, cleaned_image):
    # Find contours in the cleaned binary image
    contours, _ = cv2.findContours(cleaned_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw contours on the original image
    building_boundaries_image = image.copy()
    cv2.drawContours(building_boundaries_image, contours, -1, (0, 255, 0),
thickness=2)

    return building_boundaries_image

def segment_and_detect(image_path, threshold_value=127,
noise_removal_kernel=3):
    # Preprocess the image
    color_image, binary_image, cleaned_binary_image =
preprocess_image(image_path, threshold_value, noise_removal_kernel)

    # Detect buildings and draw boundaries on the color image
    building_boundaries_image = detect_buildings(color_image,
cleaned_binary_image)

    # Display the original, binary, and building-boundaries images
    plt.subplot(131), plt.imshow(cv2.cvtColor(color_image,
cv2.COLOR_BGR2RGB))
    plt.title('Original Image'), plt.xticks([]), plt.yticks([])
    plt.subplot(132), plt.imshow(binary_image, cmap="gray")
    plt.title('Binary Image'), plt.xticks([]), plt.yticks([])

    plt.subplot(133), plt.imshow(cv2.cvtColor(building_boundaries_image,
cv2.COLOR_BGR2RGB))
    plt.title('Building Boundaries'), plt.xticks([]), plt.yticks([])

    plt.show()

# Example usage
image_path = 'data.jpg'
segment_and_detect(image_path)
```