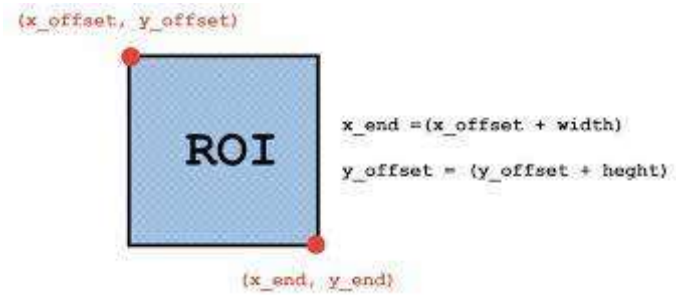
```
# Region of Intrest

# (x_offset, y_offset) - starting point
# x_end = (x_offset + width ) # width of the small image
# y_offset = (y_offset + height ) # height of the small image
# X_end, y_end -> ending point ( calculate this )
```

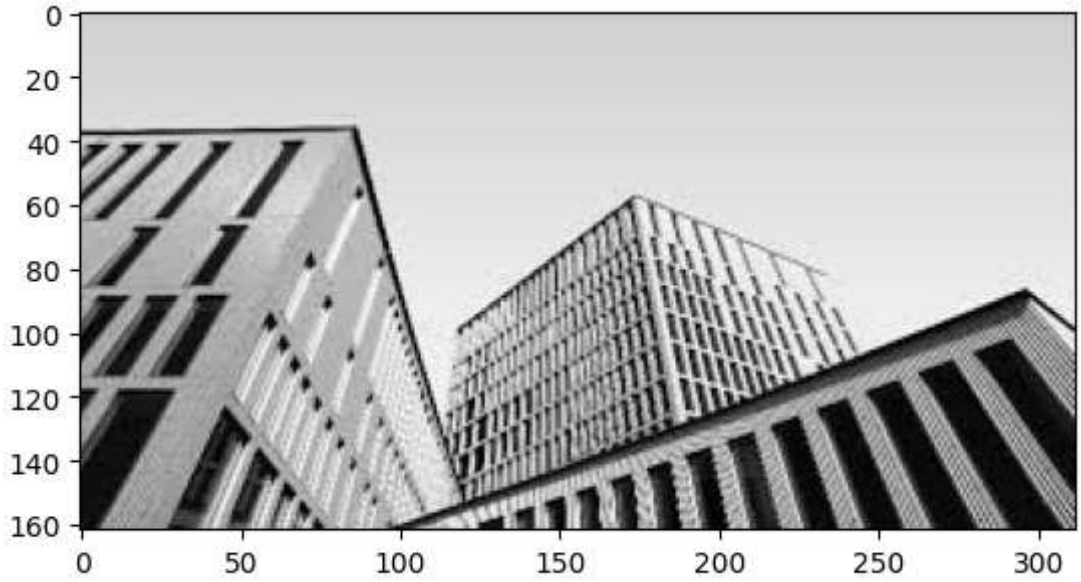



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


img=cv2.imread('/content/randomimage.jpeg') # small image
img1=cv2.imread('/content/b&wimages.jpeg') # larger image



# converting to different color spaces

# THIS IS A SMALLER IMAGE

img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.imshow(img)
img=cv2.resize(img,(57,54))
img.shape
```

```
(54, 57, 3)
```


```
# THIS IS GOING TO BE THE LARGER IMAGE

# converting to different color spaces
img1=cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
plt.imshow(img1)
img1=cv2.resize(img1,(157,154))

img1.shape
```

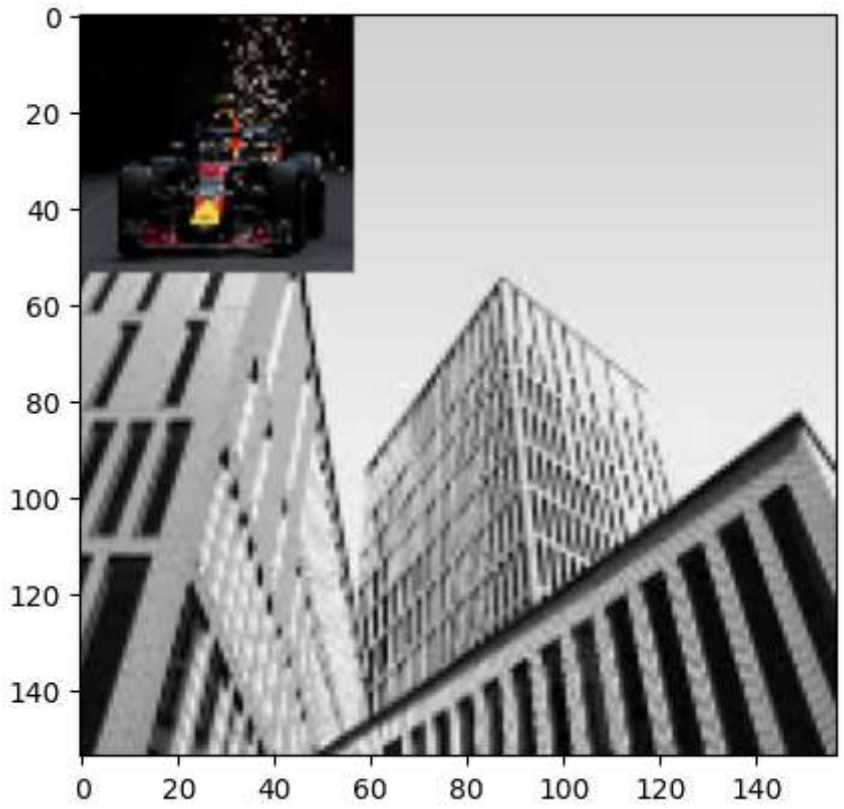
```
(154, 157, 3)
```




```
# Set the initial x and y offsets to 0
x_offset=0
y_offset=0

# Calculate the ending y-coordinate of the image by adding the image's height to the y off
y_end=y_offset+img.shape[0]

# Calculate the ending x-coordinate of the image by adding the image's width to the x offs
x_end=x_offset+img.shape[1]

# x_offset and y_offset are starting points for positioning an image on a larger canvas.
# y_end is calculated by adding the height of the image to the y offset. This gives the y-
# x_end is calculated by adding the width of the image to the x offset. This gives the x-c


img1[y_offset:y_end,x_offset:x_end]=img

# This line of code is copying the contents of the img (presumably an image) and pasting i
# The positioning of this copy-paste action is determined by the values of y_offset, y_enc

# y_offset:y_end specifies a range of rows in the img1 image where the copied img will be
```

```
# It starts from the y_offset row and goes up to (but doesn't include) the y_end row.

# x_offset:x_end specifies a range of columns in the img1 image where the copied img will
# It starts from the x_offset column and goes up to (but doesn't include) the x_end columr

# So, effectively, this line of code is placing the contents of img onto img1 at a specifi
# The result is that img will appear within the defined region of img1.


plt.imshow(img1)
```

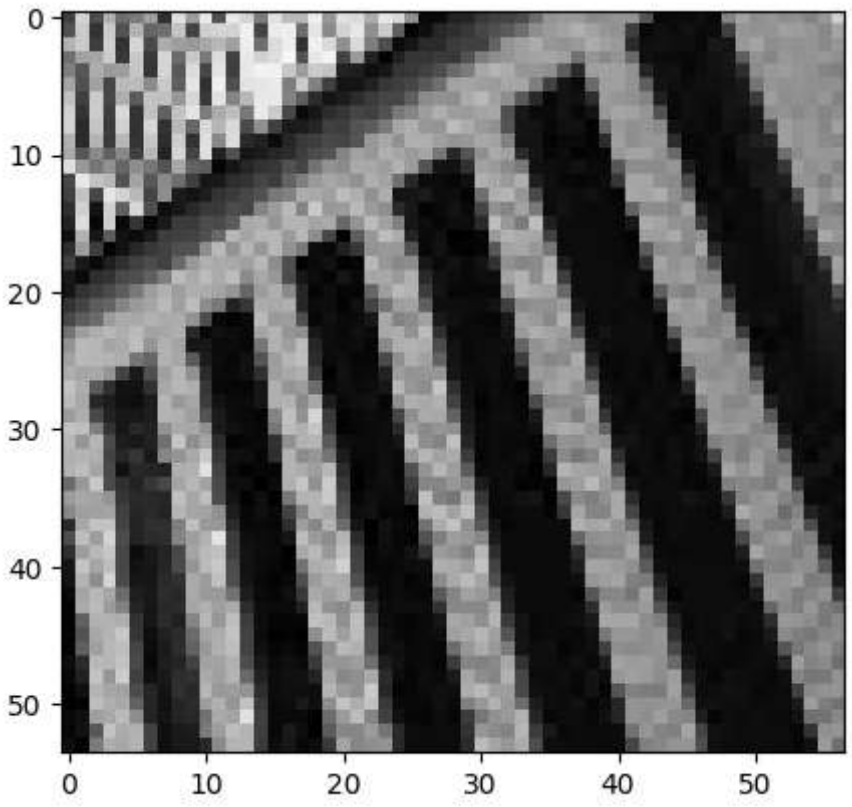
```
<matplotlib.image.AxesImage at 0x7affa4bb4a90>
```




```
# REGION OF INTREST

# x_offset and y_offset are being calculated based on the difference between certain value
# These values are used to determine where the smaller image (foreground image) will be pl

x_offset = 154 - 54
y_offset = 157-57
```

```
# Creating an ROI of the same size of the
# foreground image ( smaller image that will go on top )
rows,cols,channels = img.shape

# roi = img1[0:rows,0:cols] # TOP LEFT CORNER
# The Region of Interest (ROI) is defined in the larger image (img1) using the calculated
# The ROI is essentially a subsection of img1 that matches the size of the foreground imag

roi = img1[y_offset:154,x_offset:157]
plt.imshow(roi)
```

```
<matplotlib.image.AxesImage at 0x7affa507dae0>
```

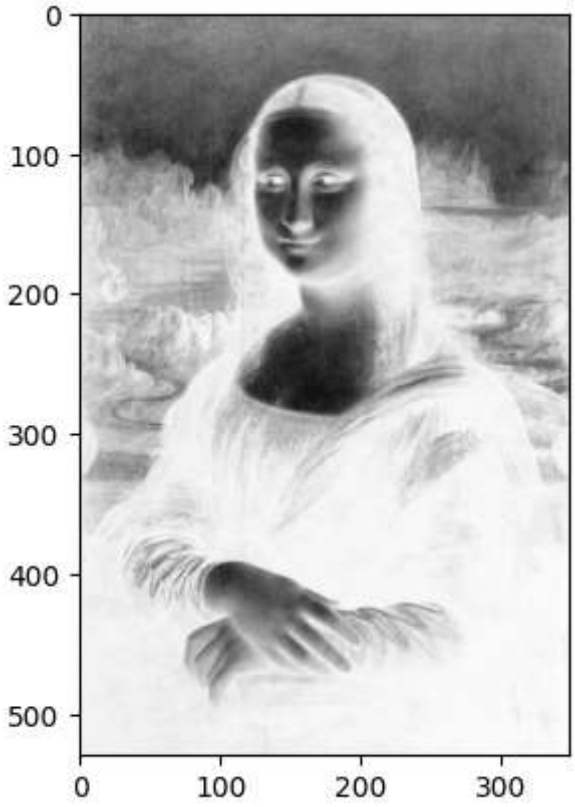



```
img=cv2.imread('/content/image.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
plt.imshow(img,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7affa484bd60>
```


```
# creating a mask -> its an inverse layer ( black and white to white and black)

# cv2.bitwise_not(img) is a function that takes an image (img) and creates an inverse vers
# This is commonly used to create masks where the background becomes the foreground and vi
mask_inv=cv2.bitwise_not(img)

# The code then displays the resulting inverted mask using plt.imshow(mask_inv, cmap='gray
# The cmap='gray' argument specifies that the colormap used for display should be grayscal
plt.imshow(mask_inv,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7affa479b280>
```




```
# Creating a 3-channel white background image with the same dimensions as img
white_bg = np.full(img.shape, 255, dtype=np.uint8)

# Using bitwise OR operation to combine the white background image with itself,
# using the mask_inv as a mask
bg = cv2.bitwise_or(white_bg, white_bg, mask=mask_inv)

# np.full(img.shape, 255, dtype=np.uint8) creates an all-white image with the same dimensi

# The cv2.bitwise_or() function performs a bitwise OR operation between two images. In thi
```

```
# This means that where the mask is black (corresponding to the original image), no change
# the white background gets combined.
```

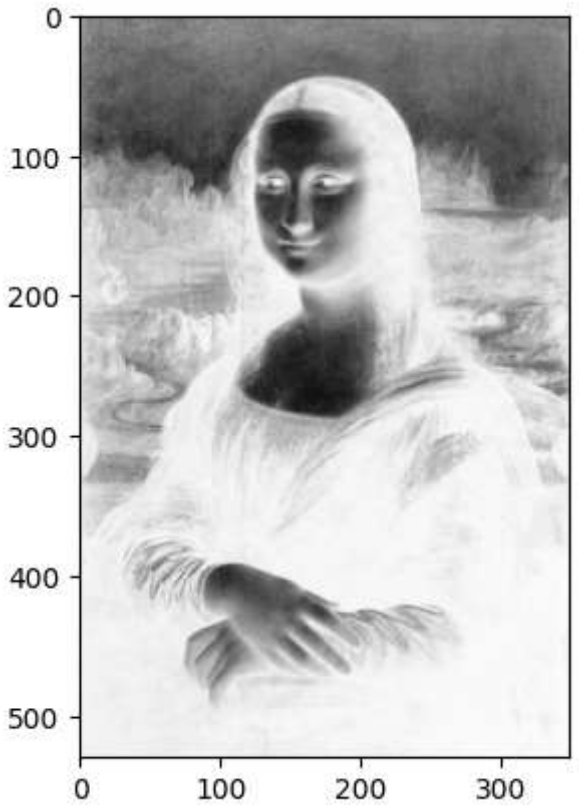
```
bg.shape
```

```
(530, 350)
```

```
# Grab Original FG image and place on top of Mask
```

```
plt.imshow(mask_inv,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7affa3b464a0>
```




```
fg = cv2.bitwise_or(img, img, mask=mask_inv)
```

```
plt.imshow(fg)
```

```
<matplotlib.image.AxesImage at 0x7affa3a25ba0>
```


```
fg.shape
```

```
(530, 350)
```

```
# rest continued in 01-Blending-and-Pasting-Images.ipynb
```