Make images come alive with scikit-image

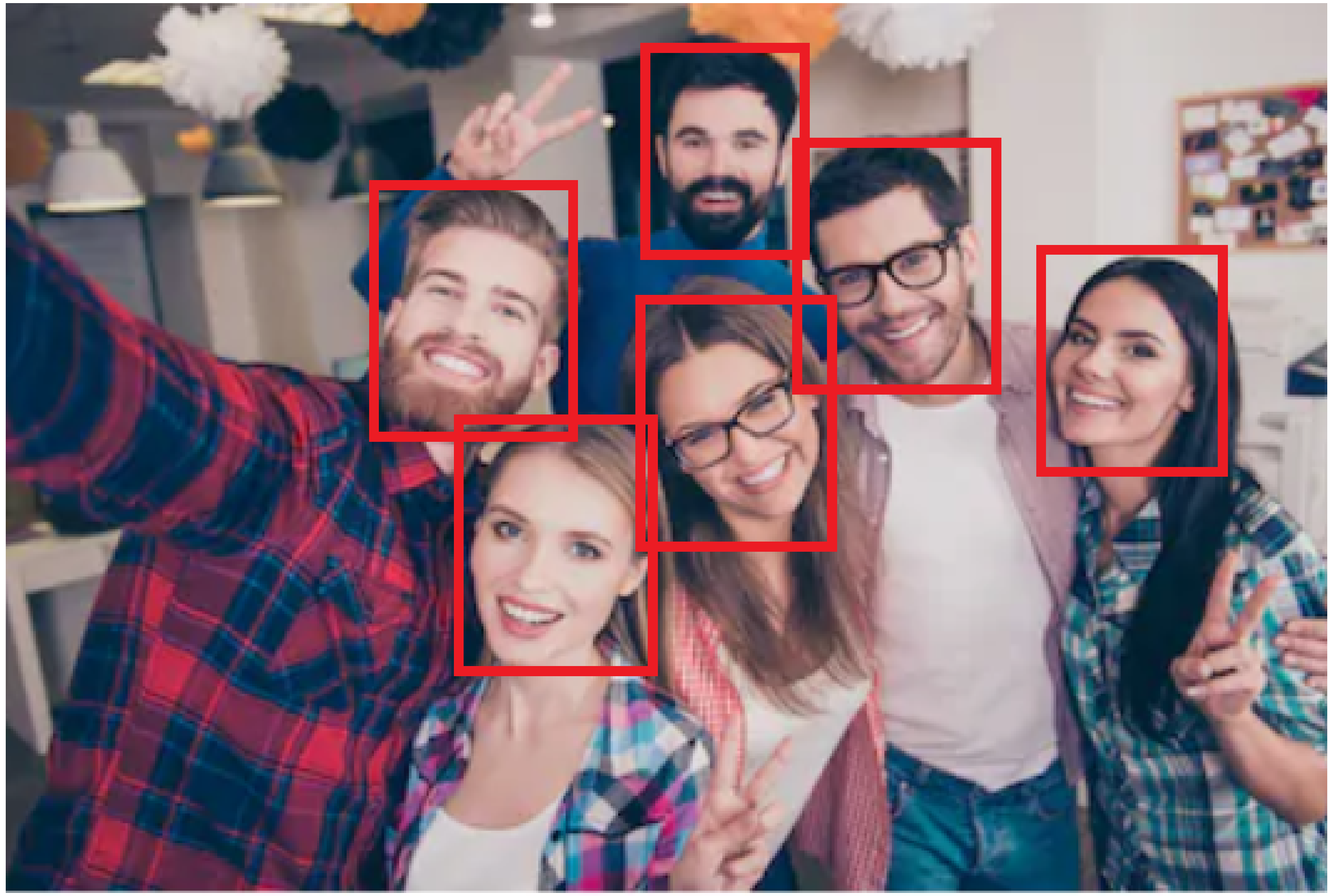
IMAGE PROCESSING IN PYTHON

Rebeca Gonzalez
Data Engineer

What is image processing?

Operations on images and videos to:

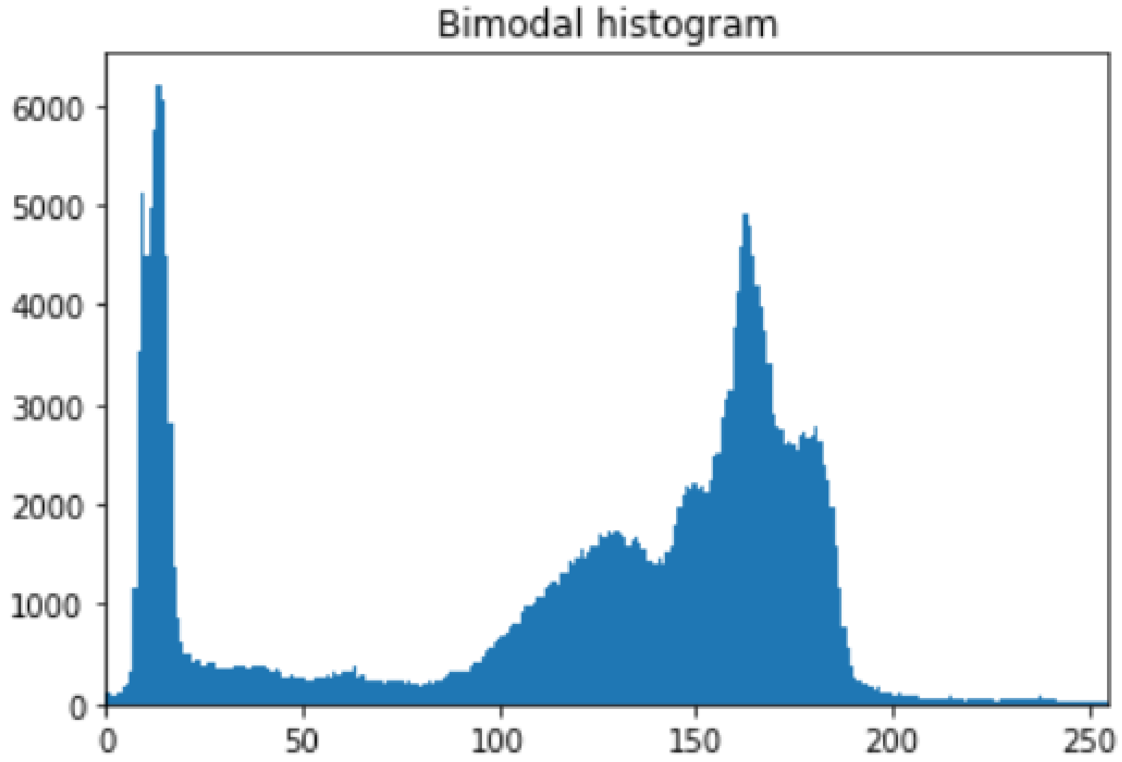
- Enhance an image
- Extract useful information
- Analyze it and make decisions



What is image processing?

Operations to on images and videos to:

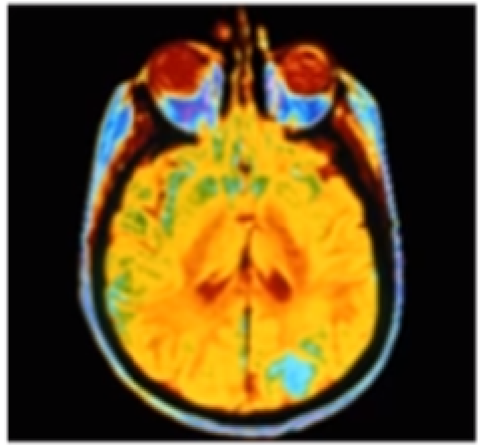
- Enhance an image
- Extract useful information
- Analyze it and make decisions

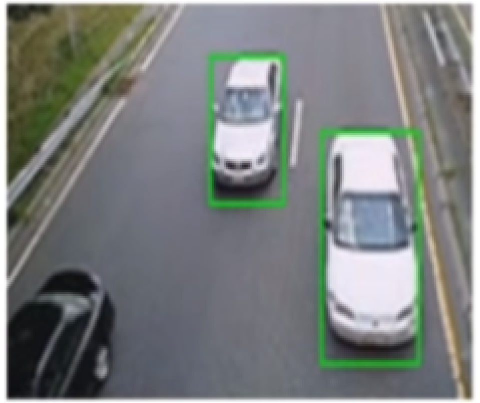


Applications

- Medical image analysis
- Artificial intelligence
- Image restoration and enhancement
- Geospatial computing
- Surveillance
- Robotic vision
- Automotive safety
- And many more...





Purposes

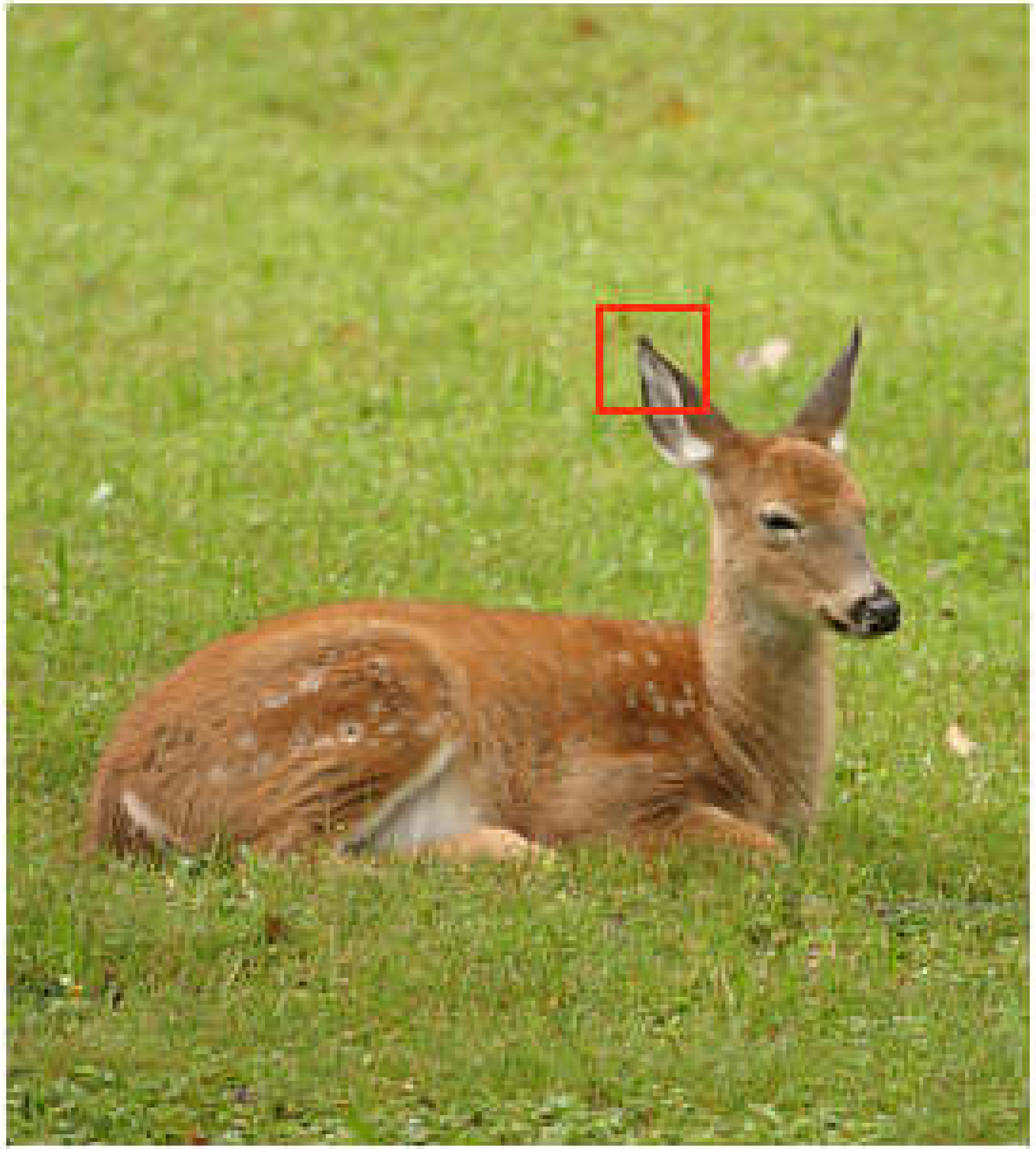
1. Visualization:
   - Objects that are not visible
2. Image sharpening and restoration
   - A better image
3. Image retrieval
   - Seek for the image of interest
4. Measurement of pattern
   - Measures various objects
5. Image Recognition
   - Distinguish objects in an image

Intro to scikit-image

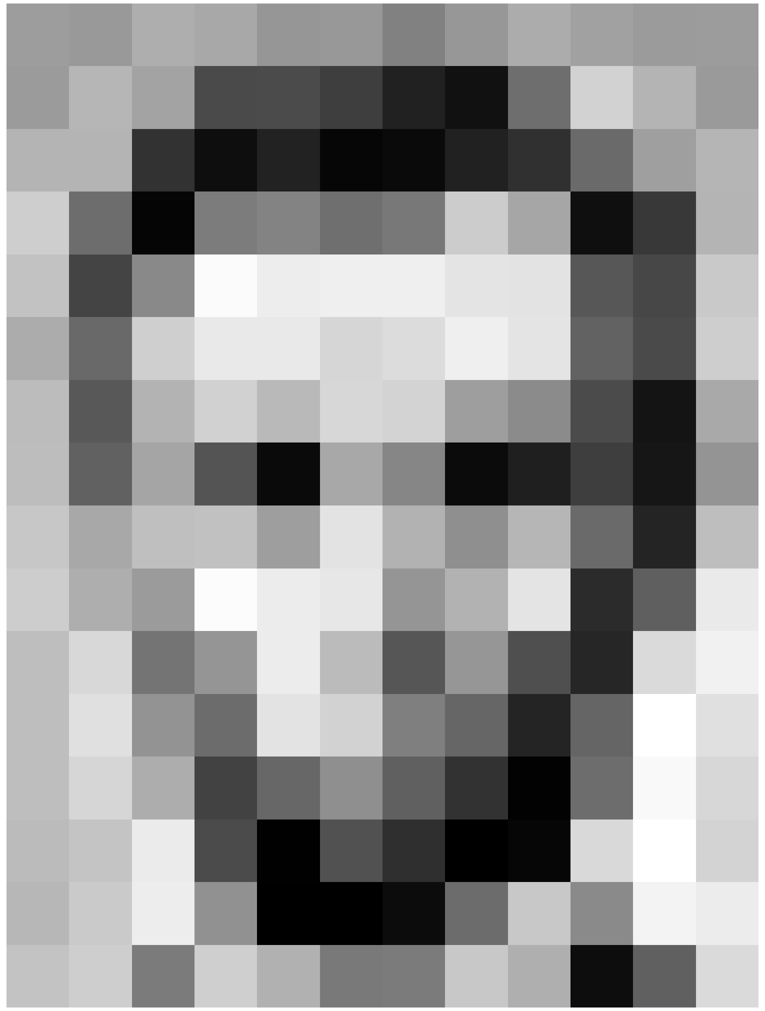
- Easy to use
- Makes use of Machine Learning
- Out of the box complex algorithms

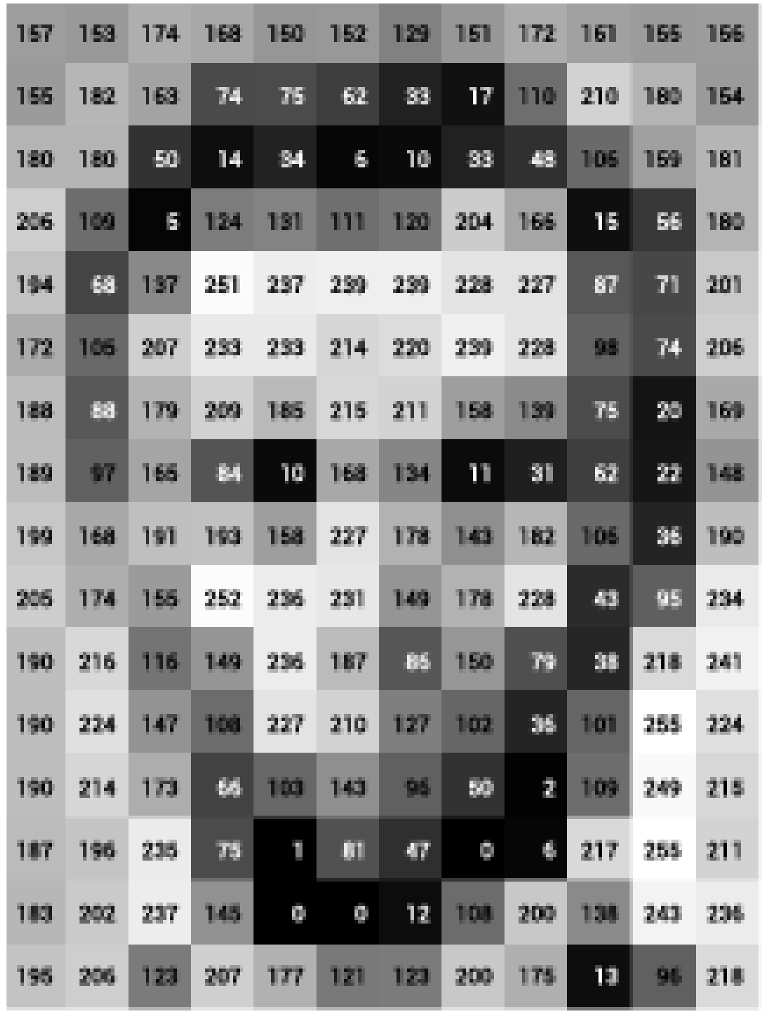
scikit-image
image processing in python

What is an image?



What is an image?





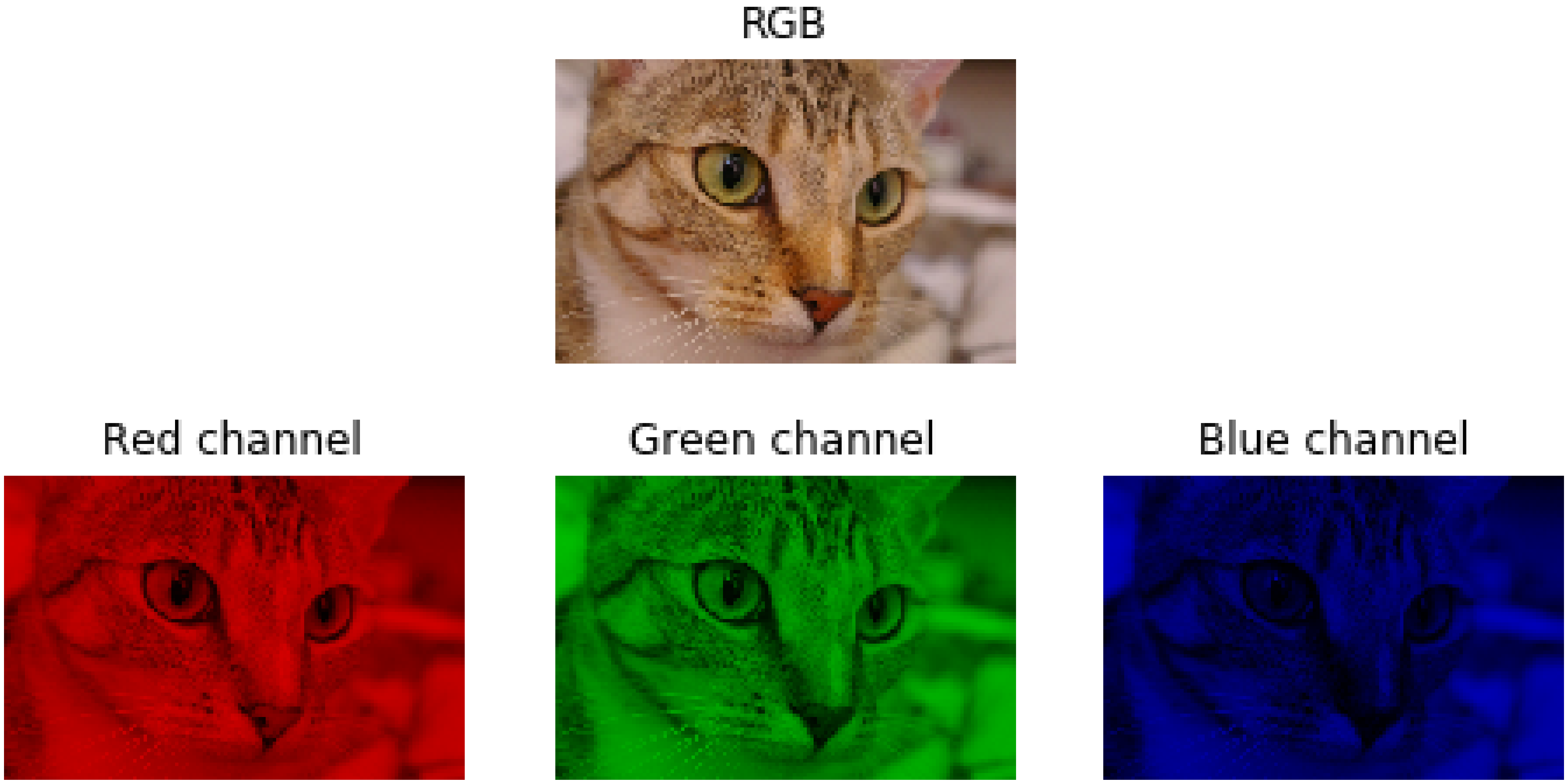
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Images in scikit-image

```
from skimage import data
rocket_image = data.rocket()
```

Rocket

RGB channels

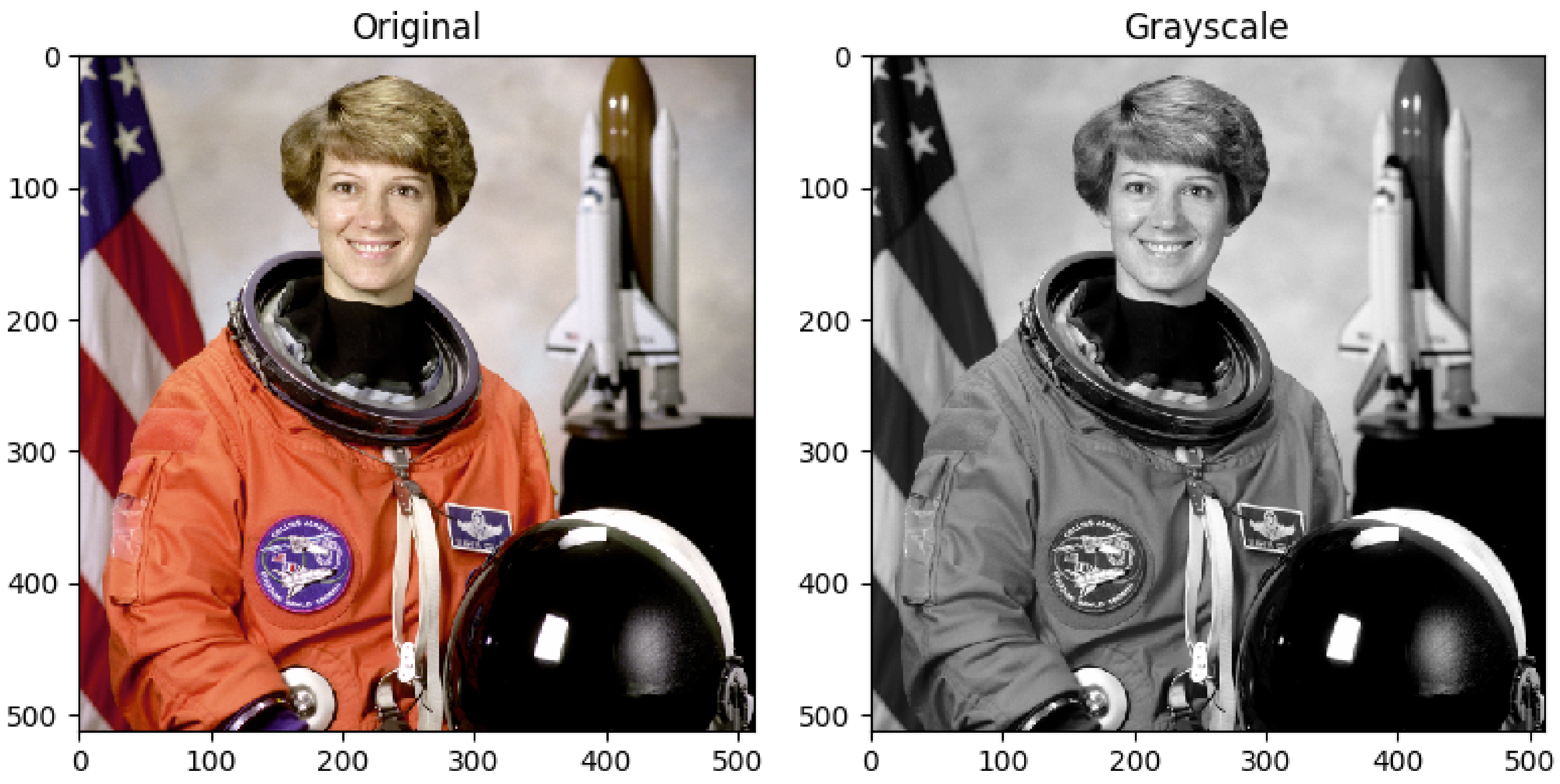


Grayscaled images

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

RGB vs Grayscale

```python
from skimage import color
grayscale = color.rgb2gray(original)
rgb = color.gray2rgb(grayscale)
```



Visualizing images in the course

Don't worry about Matplotlib!

```
def show_image(image, title='Image', cmap_type='gray'):
    plt.imshow(image, cmap=cmap_type)
    plt.title(title)
    plt.axis('off')
    plt.show()
```

Visualizing images in the course

```
from skimage import color
grayscale = color.rgb2gray(original)

show_image(grayscale, "Grayscale")
```

Grayscale

Let's practice!

IMAGE PROCESSING IN PYTHON

NumPy for images

IMAGE PROCESSING IN PYTHON

Rebeca Gonzalez
Data Engineer

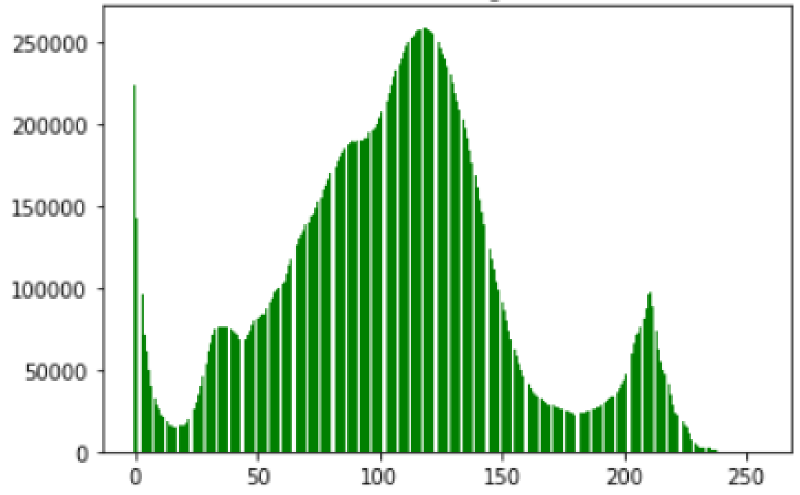
NumPy for images

- Fundamentals of image processing techniques
  - Flipping
  - Extract and analyze features

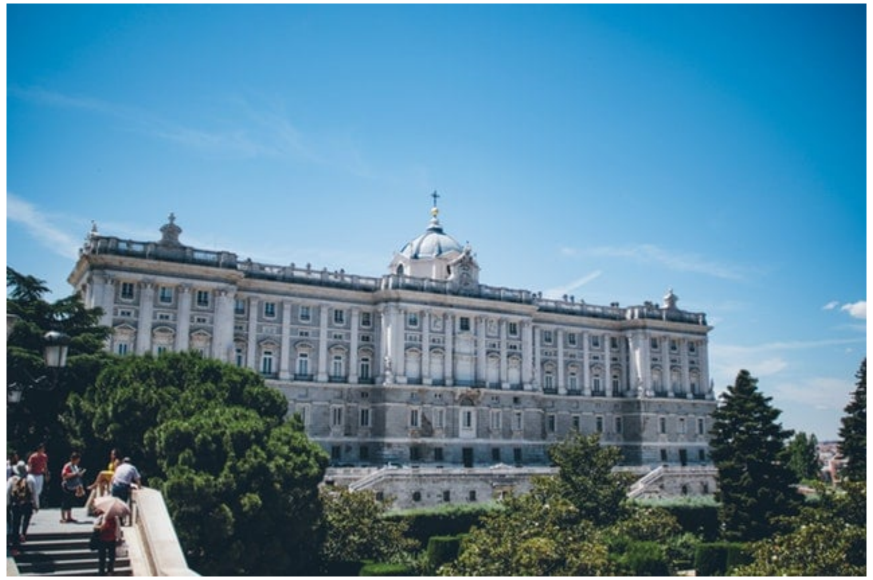
Original

Flipped

Green Histogram



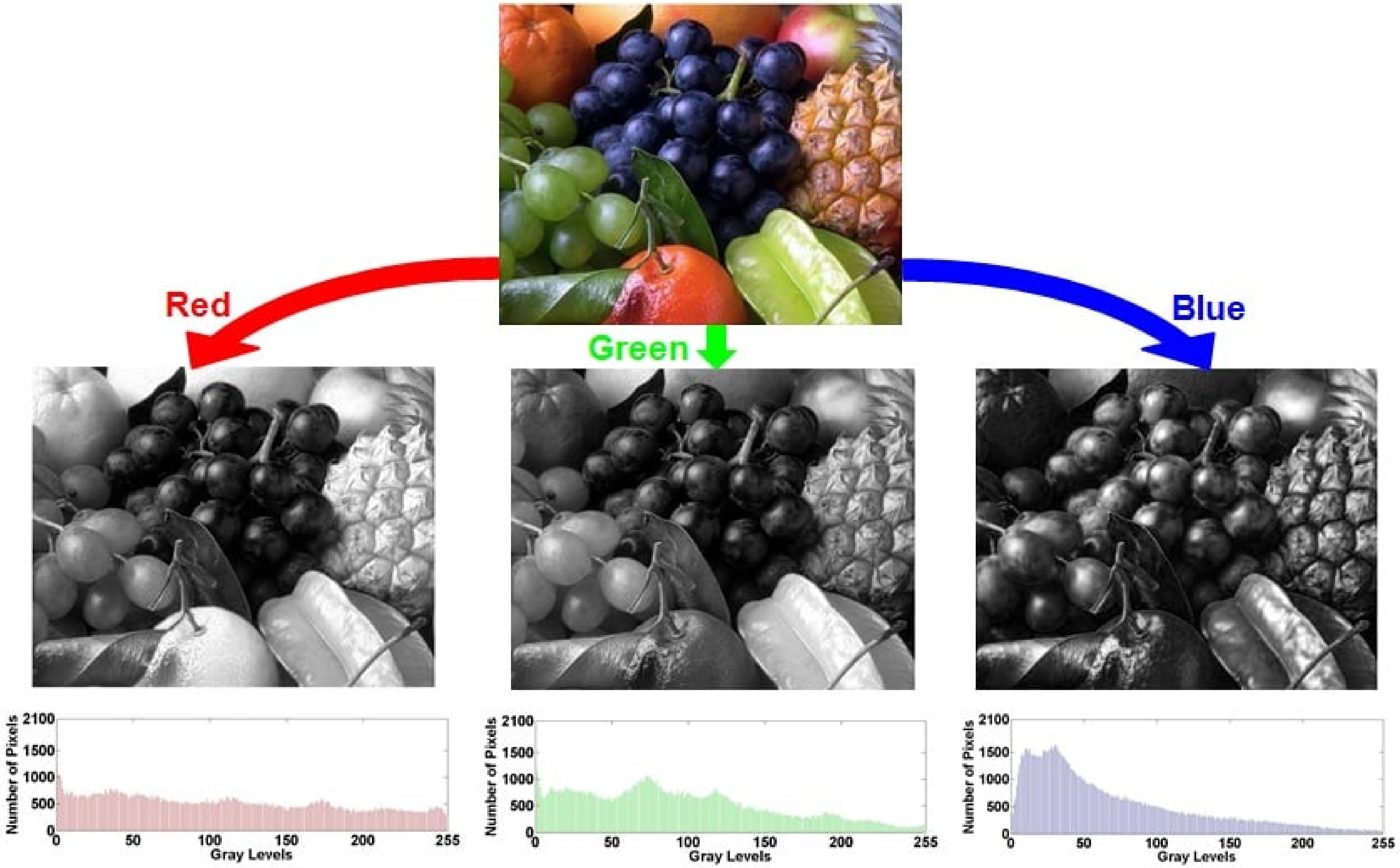
Images as NdArrays



```
# Loading the image using Matplotlib
madrid_image = plt.imread('/madrid.jpeg')


type(madrid_image)
```

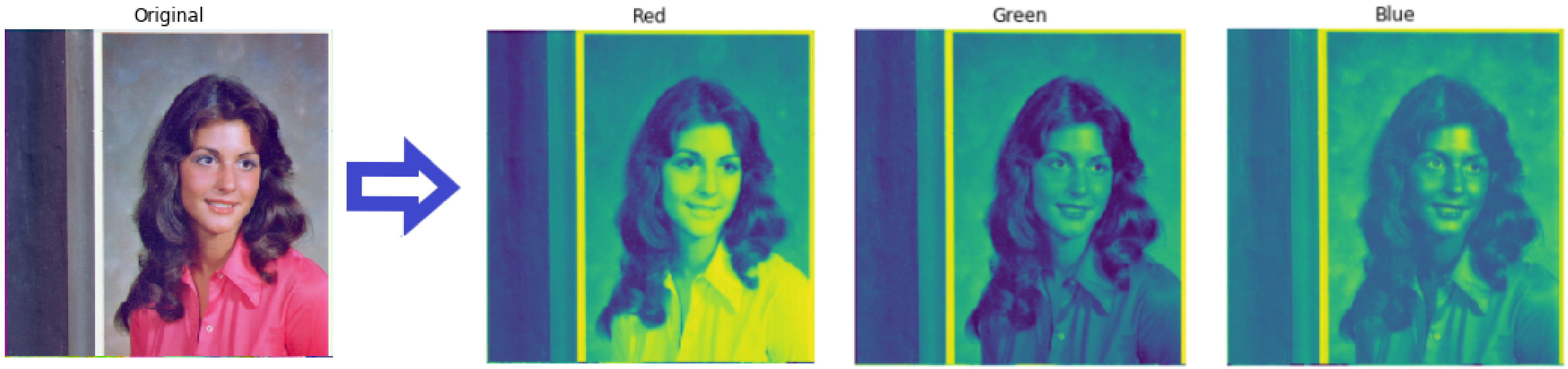
```
<class 'numpy.ndarray'>
```

Colors with NumPy



Colors with NumPy

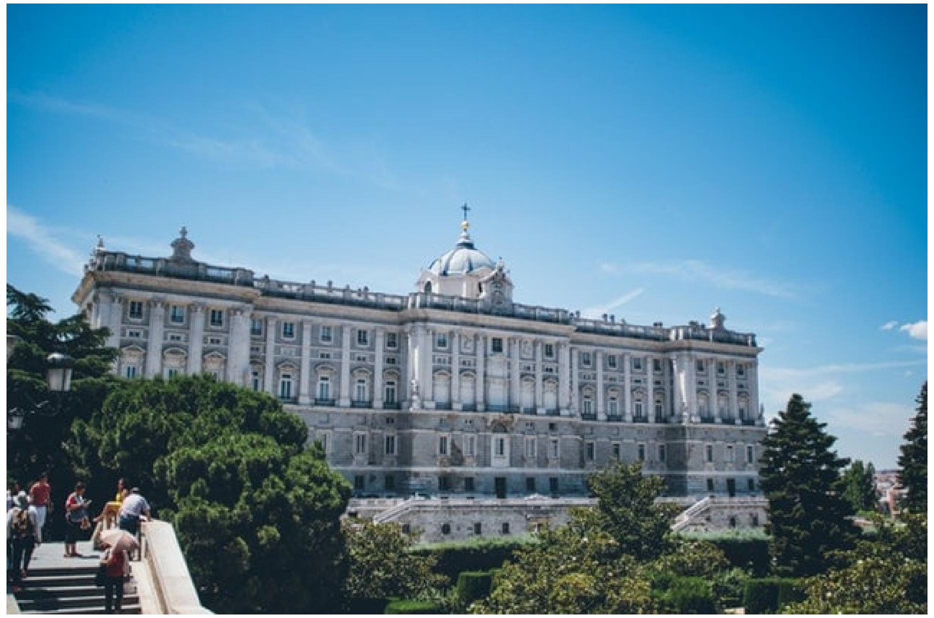
```
# Obtaining the red values of the image
red = image[:, :, 0]

# Obtaining the green values of the image
green = image[:, :, 1]

# Obtaining the blue values of the image
blue = image[:, :, 2]
```



Colors with NumPy


```
plt.imshow(red, cmap="gray")
plt.title('Red')
plt.axis('off')
plt.show()
```

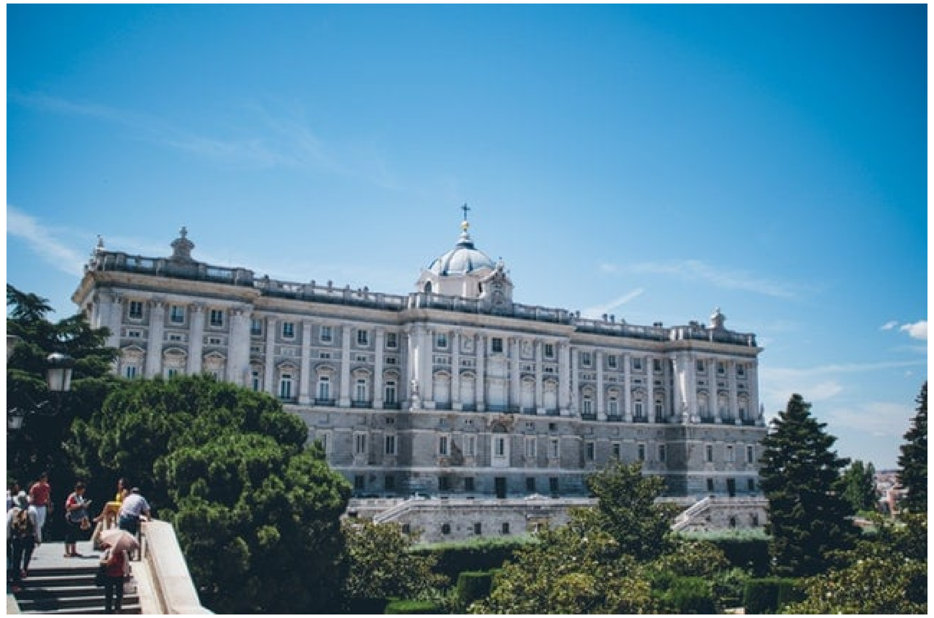
Shapes



```python
# Accessing the shape of the image
madrid_image.shape
```

```
(426, 640, 3)
```

Sizes



```
# Accessing the shape of the image
madrid_image.size
```

```
817920
```

Flipping images: vertically

```
# Flip the image in up direction
vertically_flipped = np.flipud(madrid_image)

show_image(vertically_flipped, 'Vertically flipped image')
```

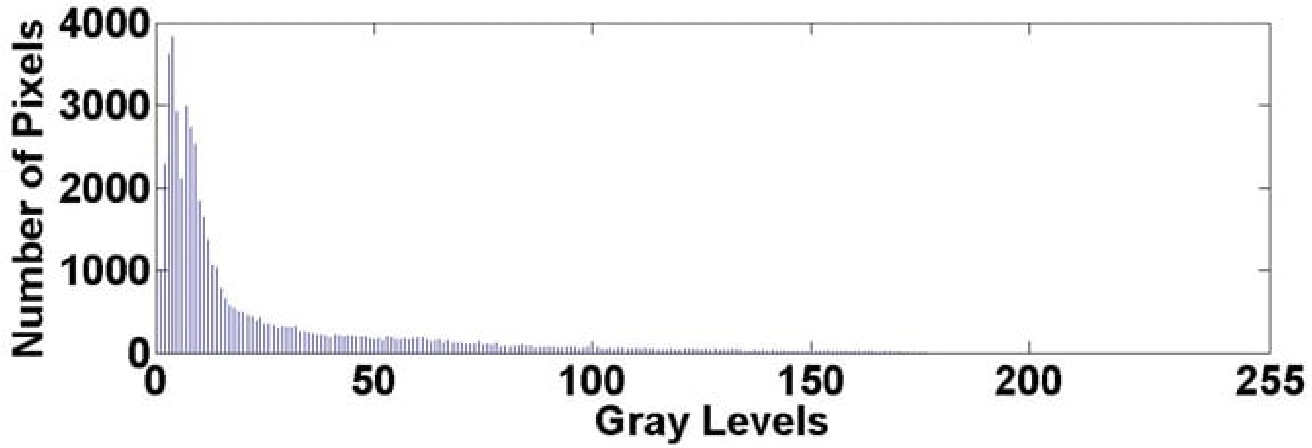
Vetically flipped image

Flipping images: horizontally

```
# Flip the image in left direction
horizontally_flipped = np.fliplr(madrid_image)


show_image(horizontally_flipped, 'Horizontally flipped image')
```

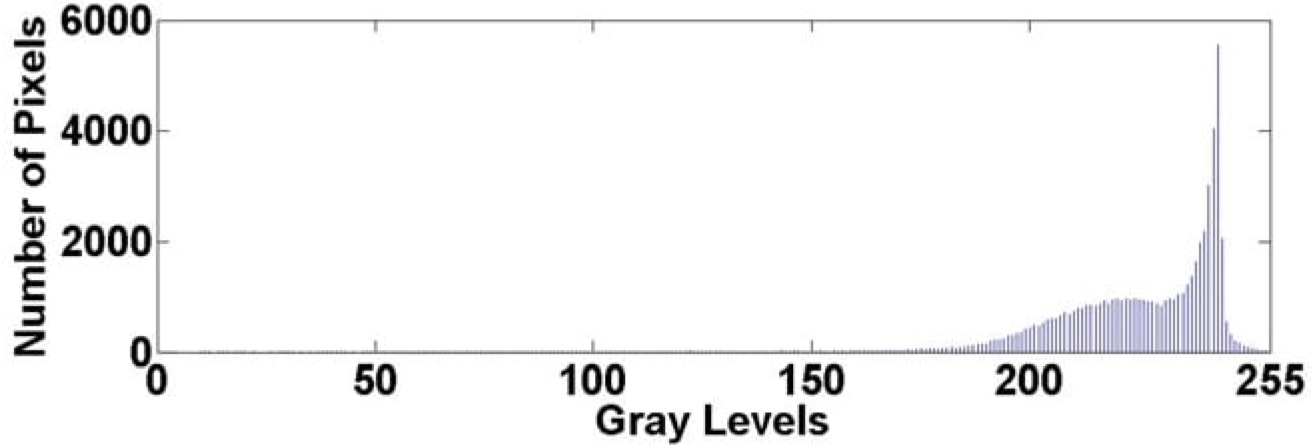
Horizontally flipped image

What is a histogram?



(a)

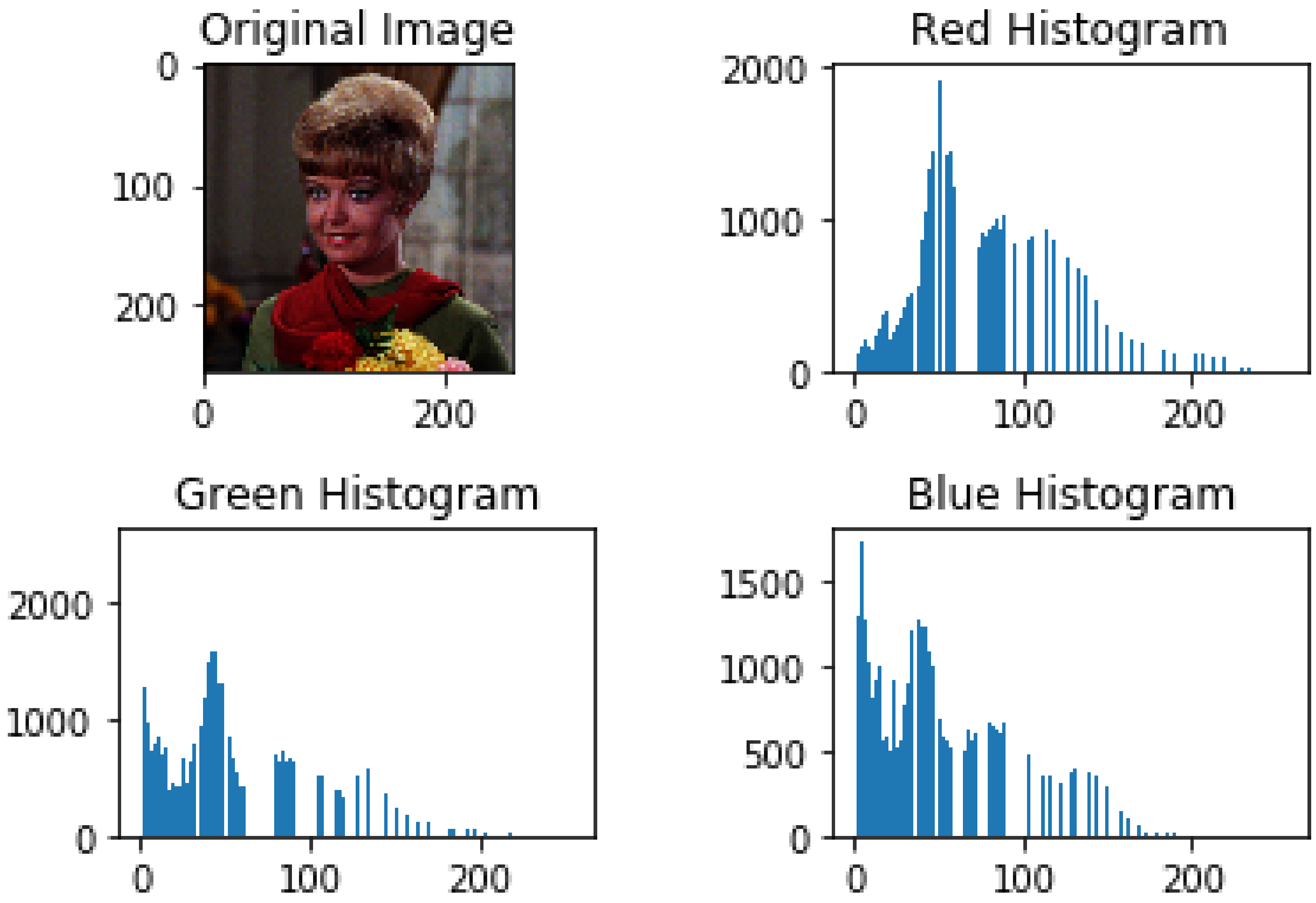
(b)



(a)

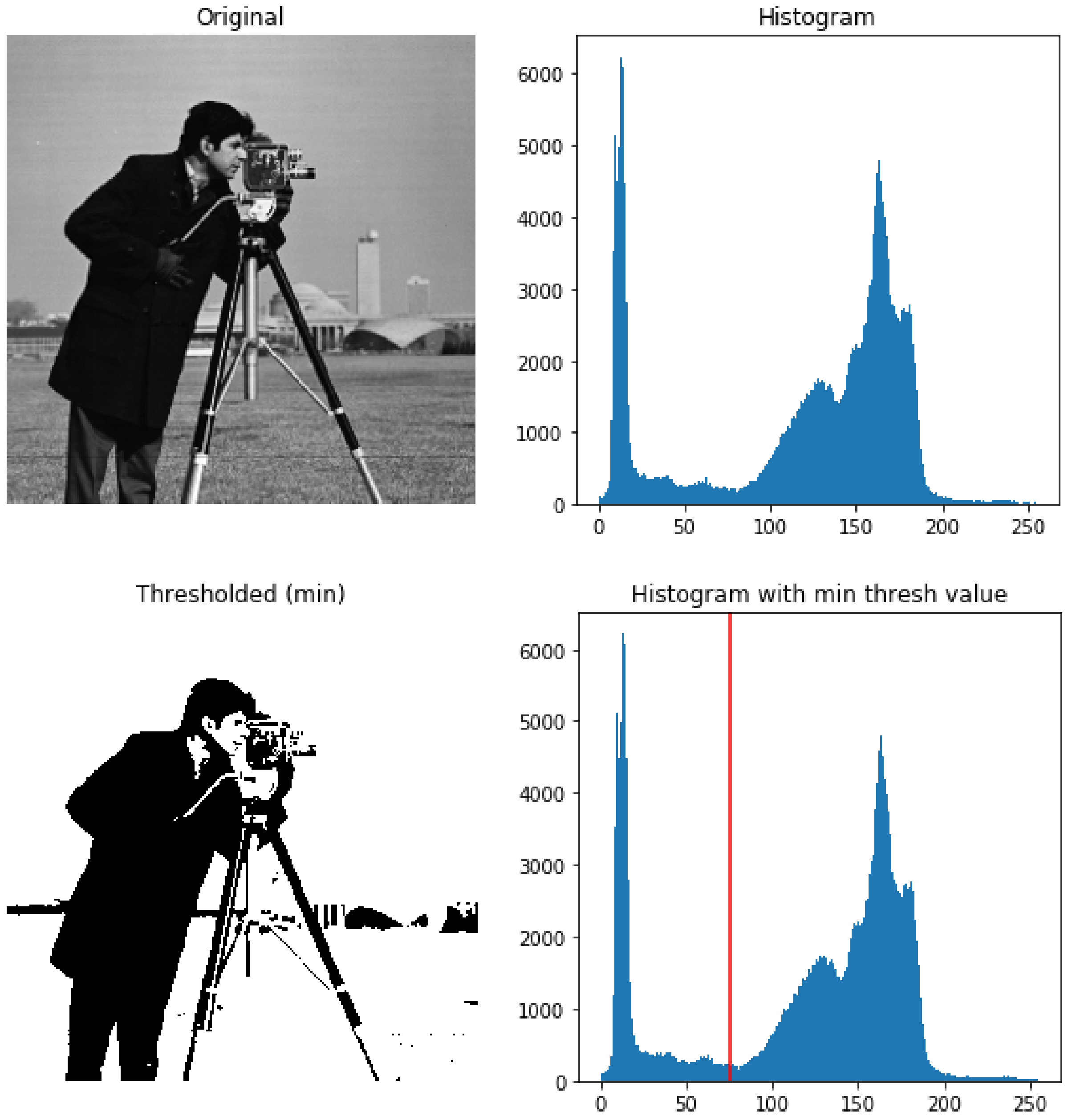
(b)

Color histograms

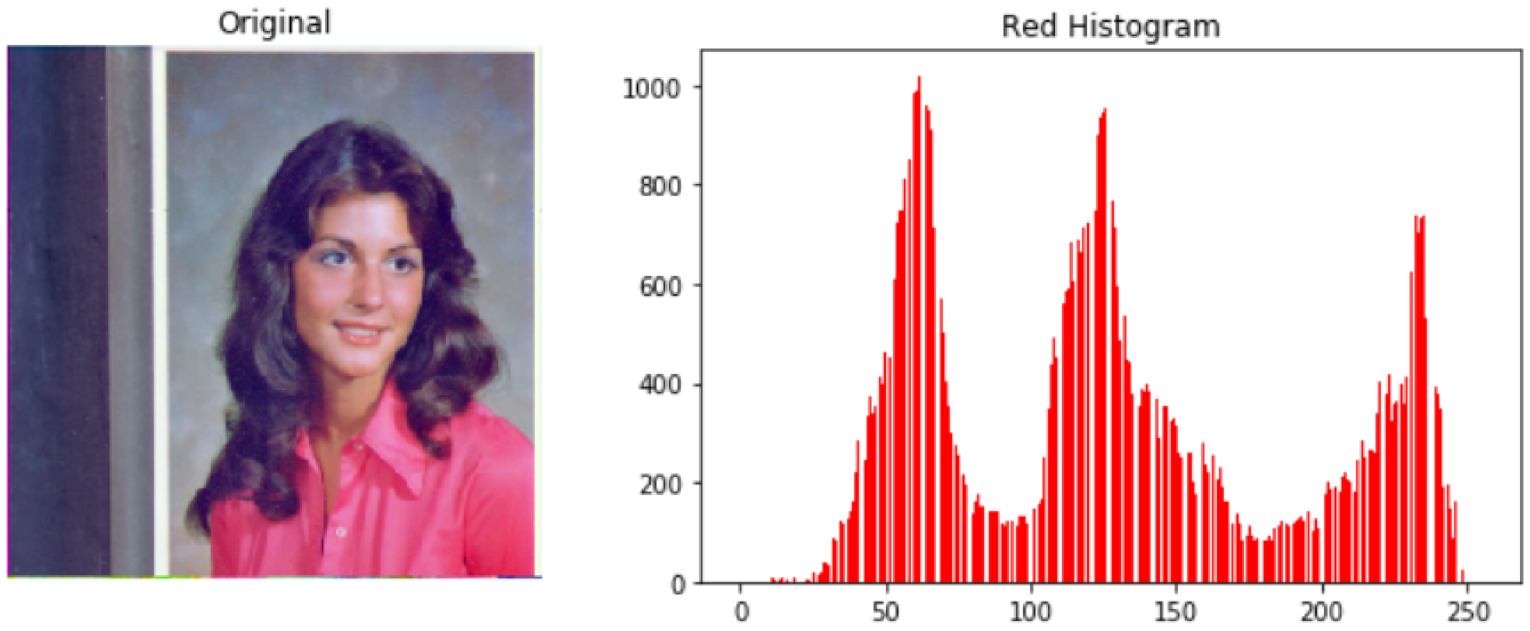


Applications of histograms

- Analysis
- Thresholding
- Brightness and contrast
- Equalize an image



Histograms in Matplotlib

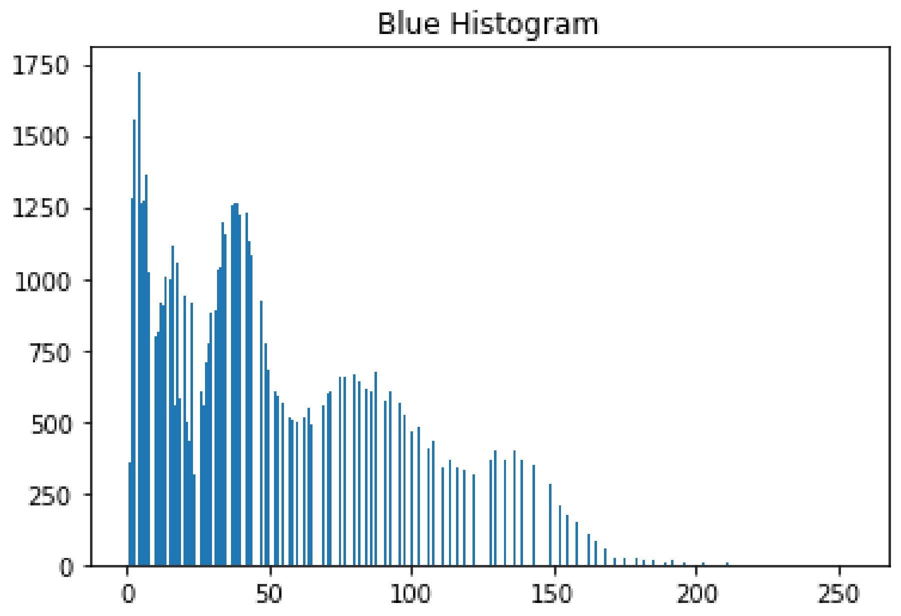



```
# Red color of the image
red = image[:, :, 0]

# Obtain the red histogram
plt.hist(red.ravel(), bins=256)
```

Visualizing histograms with Matplotlib

```
blue = image[:, :, 2]

plt.hist(blue.ravel(), bins=256)
plt.title('Blue Histogram')
plt.show()
```

```
import numpy as np

# 2D array
array = np.array([[1, 2],
                  [3, 4]])
print(array.shape)  # (2, 2)
print(array.ravel())  # [1, 2, 3, 4]
print(array.ravel().shape)  # (4,)
```



Let's practice!

Getting started with thresholding

IMAGE PROCESSING IN PYTHON

Rebeca Gonzalez
Data Engineer

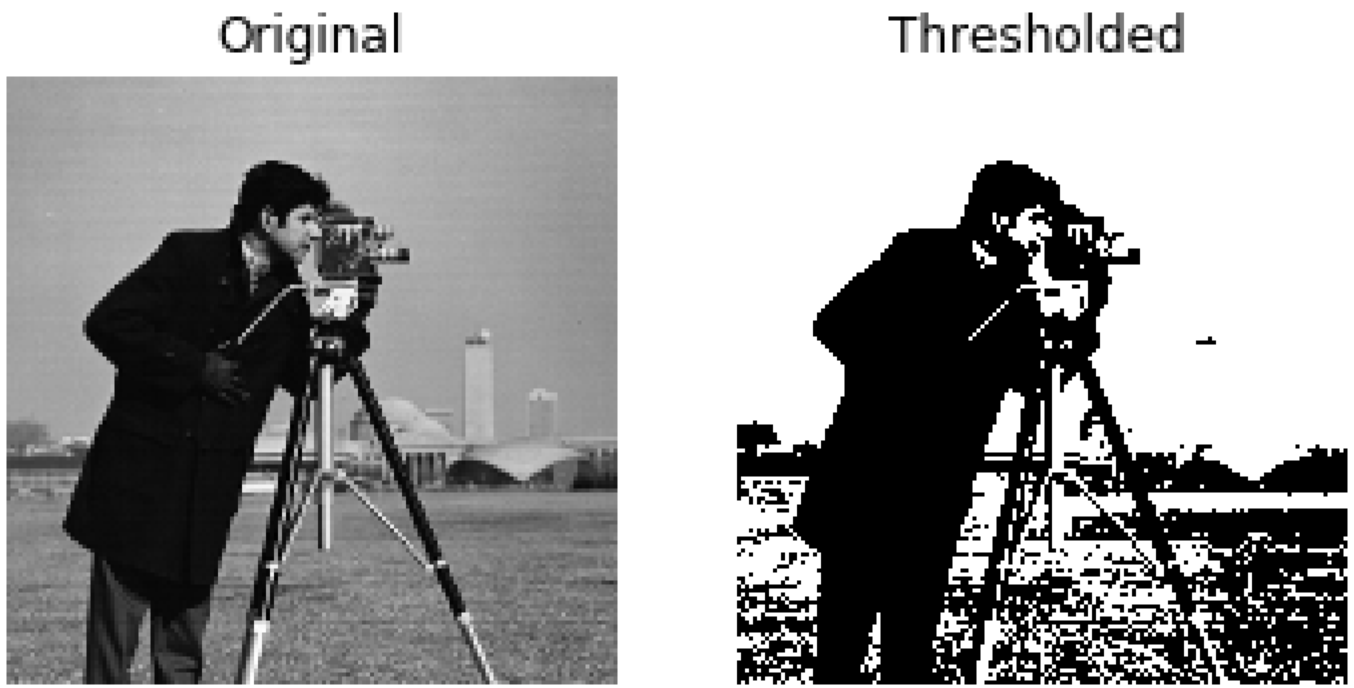
Thresholding

Partitioning an image into a foreground and background

By making it **black and white**

We do so by setting each pixel to:

- 255 (white) if pixel $>$ thresh value
- 0 (black) if pixel $<$ thresh value



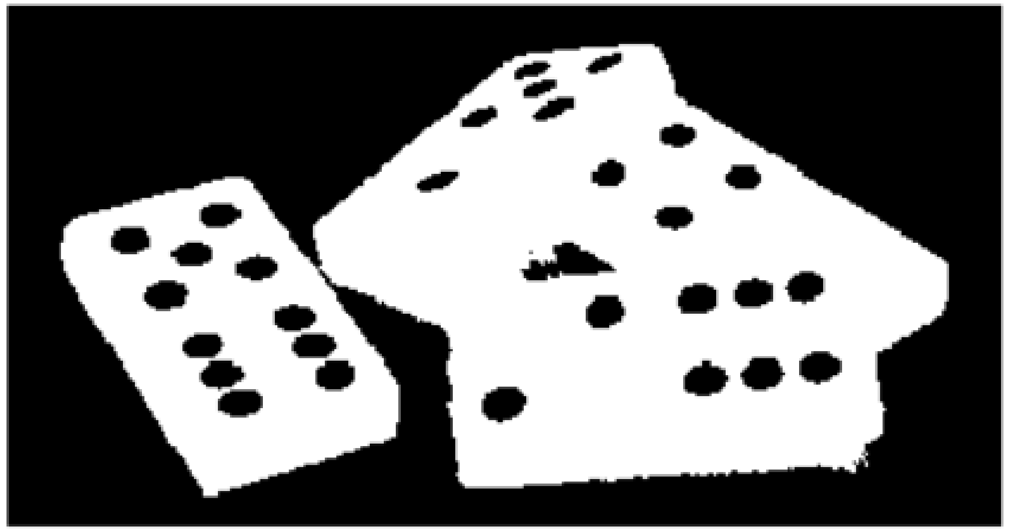
Thresholding

Simplest method of image segmentation

- Isolate objects
  - Object detection
  - Face detection
  - Etc.

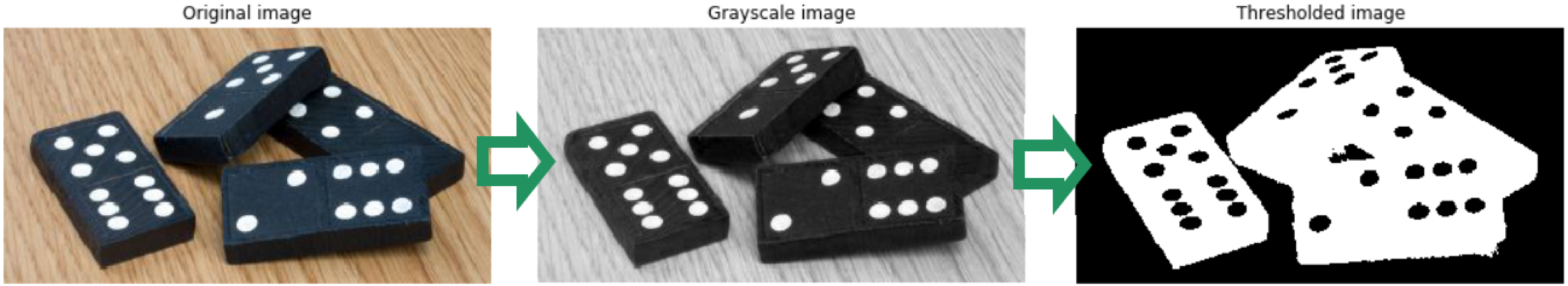
Original image

Thresholded image



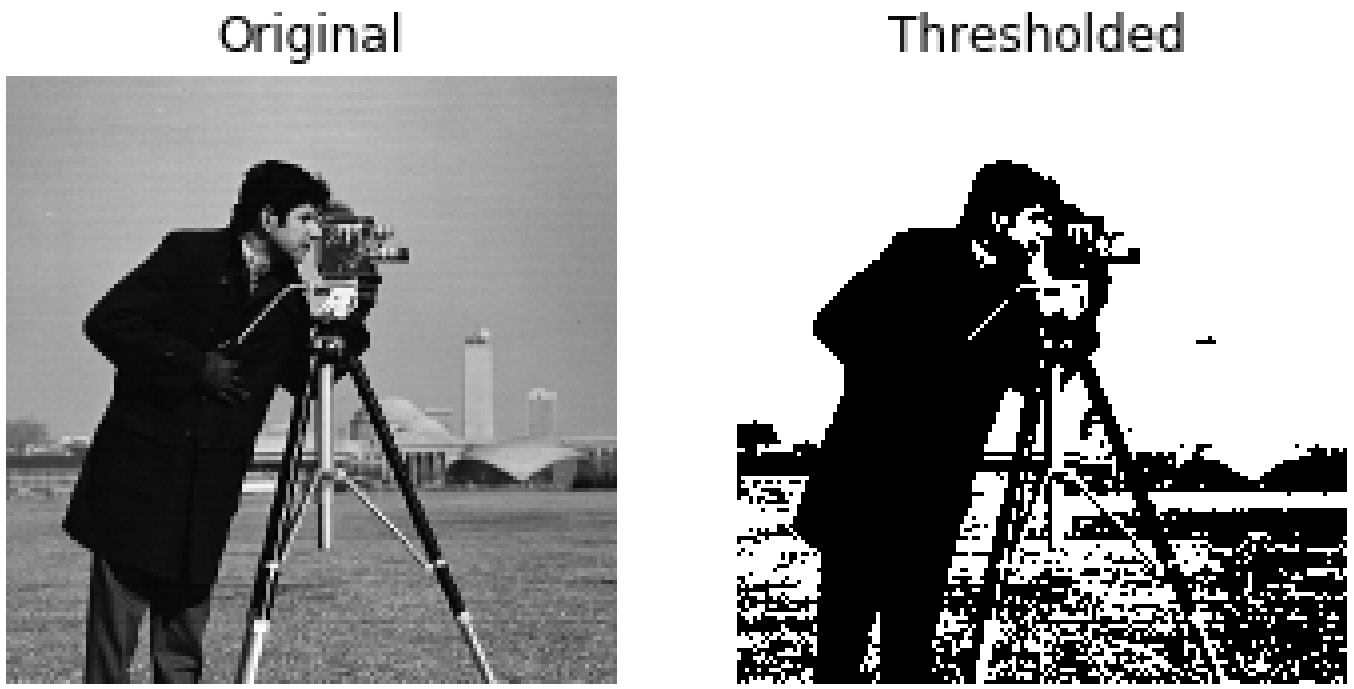
Thresholding

Only from **grayscale** images



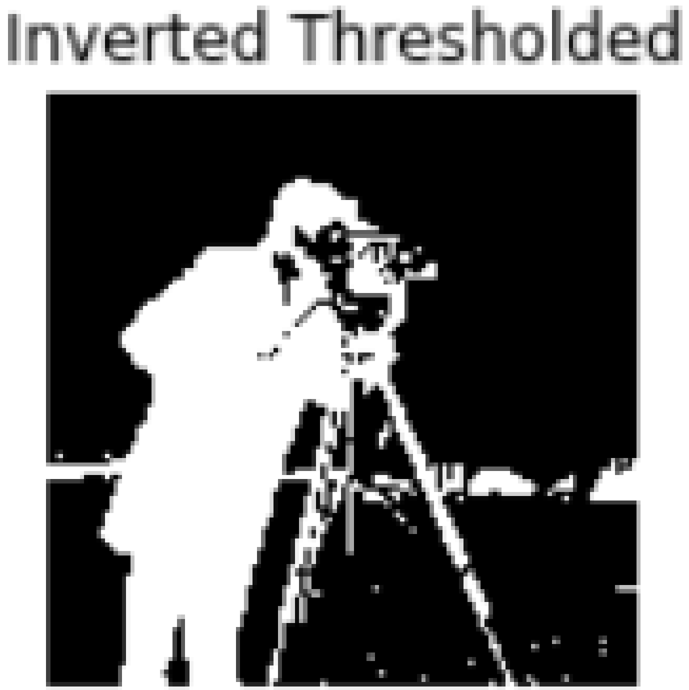
Apply it

```
# Obtain the optimal threshold value
thresh = 127

# Apply thresholding to the image
binary = image > thresh

# Show the original and thresholded
show_image(image, 'Original')
show_image(binary, 'Thresholded')
```



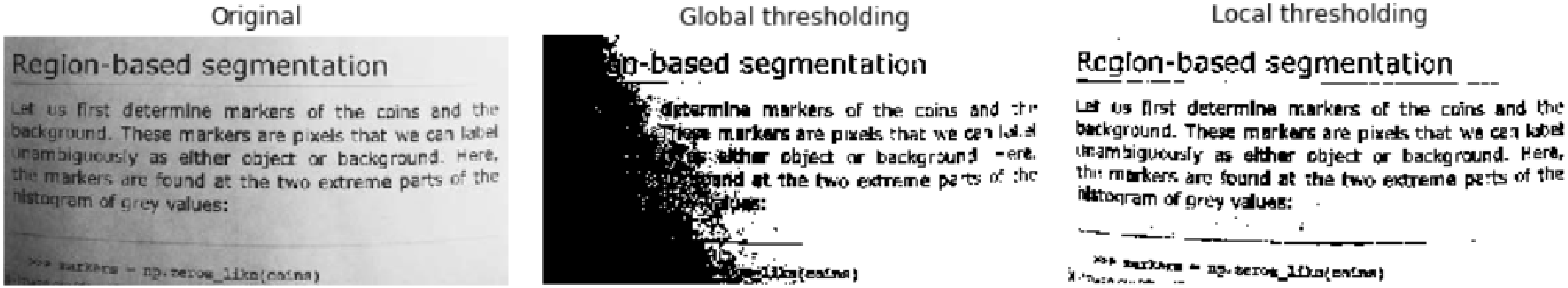
Inverted thresholding

```
# Obtain the optimal threshold value
thresh = 127

# Apply thresholding to the image
inverted_binary = image <= thresh

# Show the original and thresholded
show_image(image, 'Original')
show_image(inverted_binary,
           'Inverted thresholded')
```



Categories

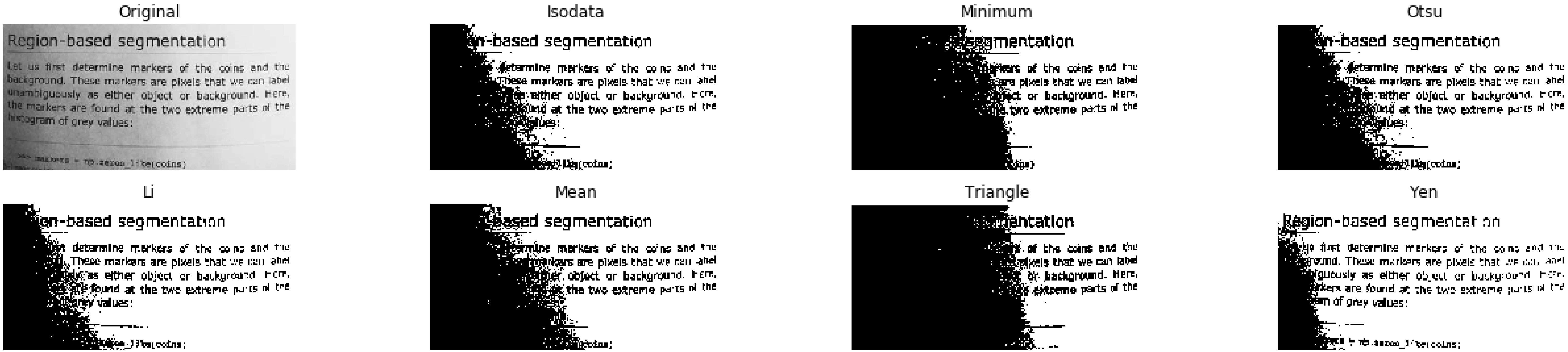
- **Global or histogram based:** good for uniform backgrounds
- **Local or adaptive:** for uneven background illumination



Try more thresholding algorithms

```
from skimage.filters import try_all_threshold

# Obtain all the resulting images
fig, ax = try_all_threshold(image, verbose=False)

# Showing resulting plots
show_plot(fig, ax)
```

Try more thresholding algorithms



Optimal thresh value

Global

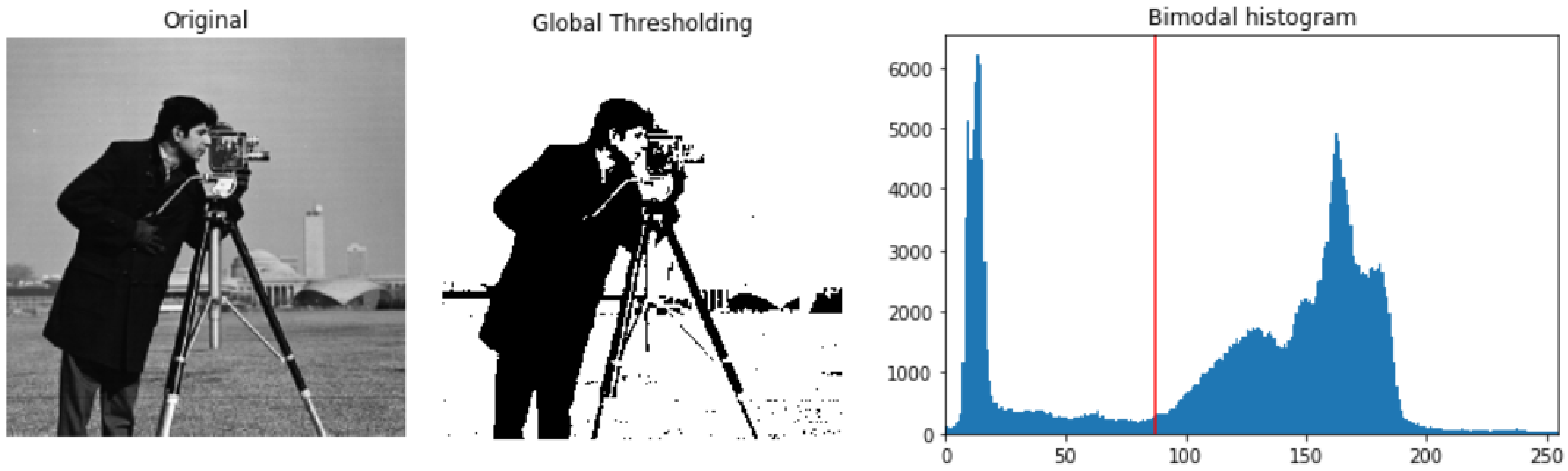
Uniform background

```
# Import the otsu threshold function
from skimage.filters import threshold_otsu

# Obtain the optimal threshold value
thresh = threshold_otsu(image)

# Apply thresholding to the image
binary_global = image > thresh
```

Optimal thresh value

Global

```python
# Show the original and binarized image
show_image(image, 'Original')
show_image(binary_global, 'Global thresholding')
```



Optimal thresh value

Local

Uneven background

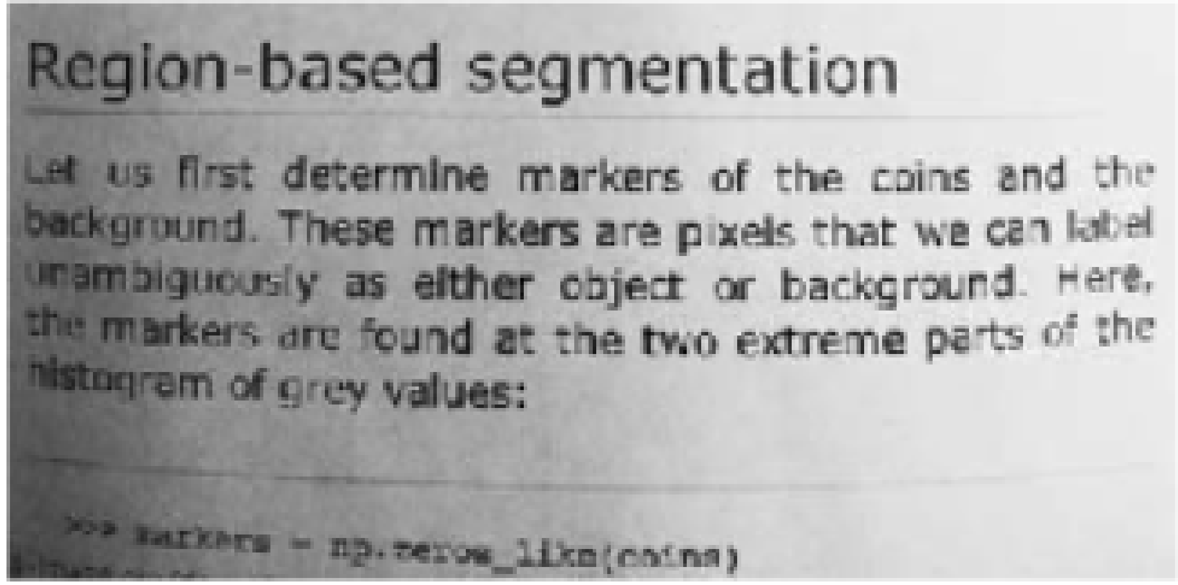
```
# Import the local threshold function
from skimage.filters import threshold_local

# Set the block size to 35
block_size = 35

# Obtain the optimal local thresholding
local_thresh = threshold_local(text_image, block_size, offset=10)

# Apply local thresholding and obtain the binary image
binary_local = text_image > local_thresh
```

Optimal thresh value

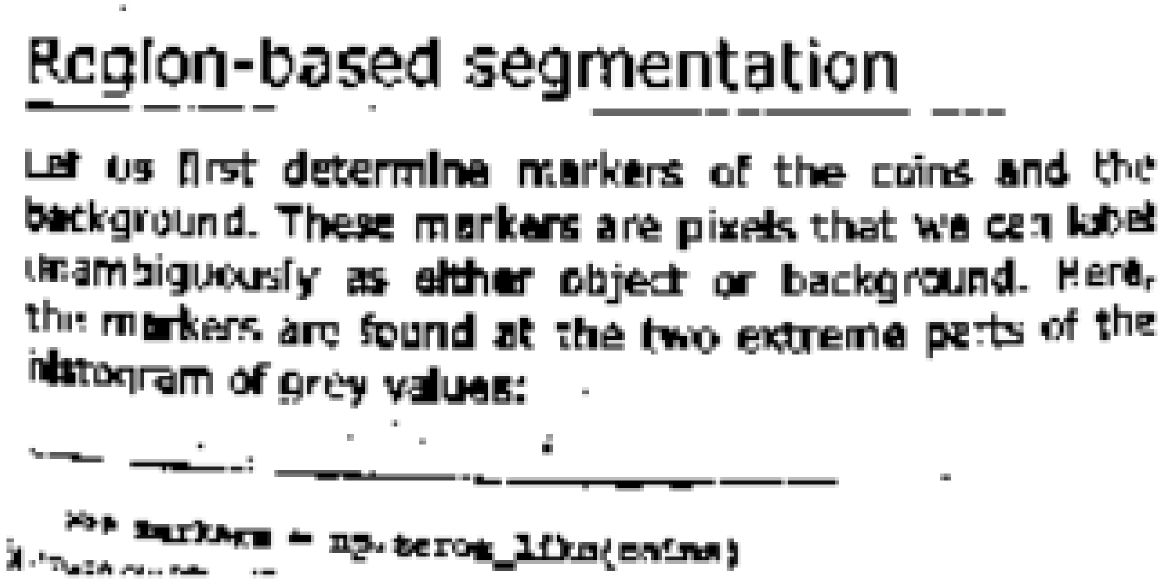
Local

```
# Show the original and binarized image
show_image(text_image, 'Original')
show_image(binary_local, 'Local thresholding')
```

Original



Local thresholding



Let's practice!

IMAGE PROCESSING IN PYTHON