Automated Feature Extraction (Part 3)

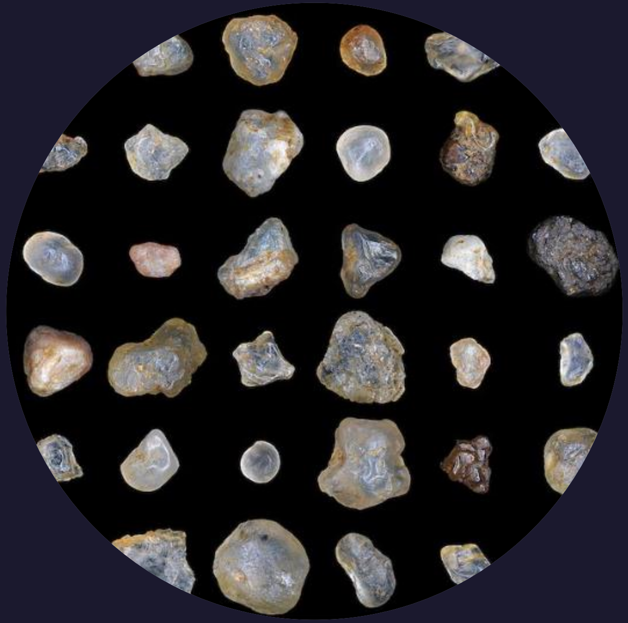
Ron Michael V. Acda

2019-03839

Objectives

1. Combine image segmentation techniques to label and count the number of features in an image.

2. Perform basic statistical analysis on image segmentation data.



Results and Discussion

General Workflow

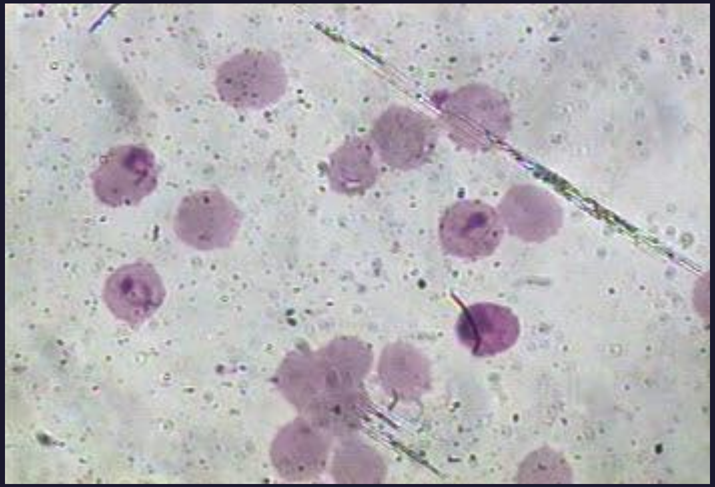
1. Load image as grayscale
2. Perform initial segmentation, which can be:
    1. Grayscale thresholding
    2. Parametric/non-parametric segmentation
3. Perform morphological operations to clean the binarized segmentation
4. Use skimage.measure to count and label features.
5. Use pandas to perform basic statistical analysis on segmentation data

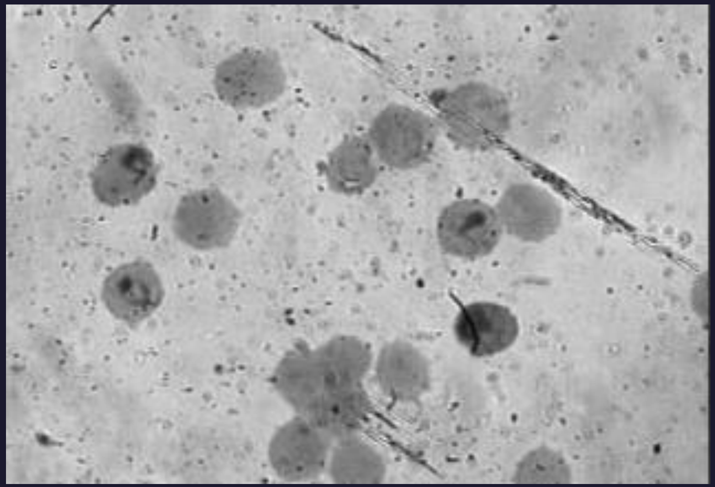
Statistical Analysis

1. The mean, median, and the standard deviation of various characteristics of a blob/feature was calculated using the pandas and numpy library.
2. The following characteristics were summarized:
   1. Area
   2. Perimeter
   3. Coordinates of the centroid (geometric center of the blob/feature)
   4. Eccentricity - measures how "deformed" or "elongated" the blob is. If the eccentricity is close to 0, it appears spherical. If it is close to 1, then it is highly elliptical.
   5. Major axis and minor axis lengths – this assumes that the blob or feature can be approximated as an ellipse

Example 1: Malaria-infected Cells

Step 1: Load the image as grayscale

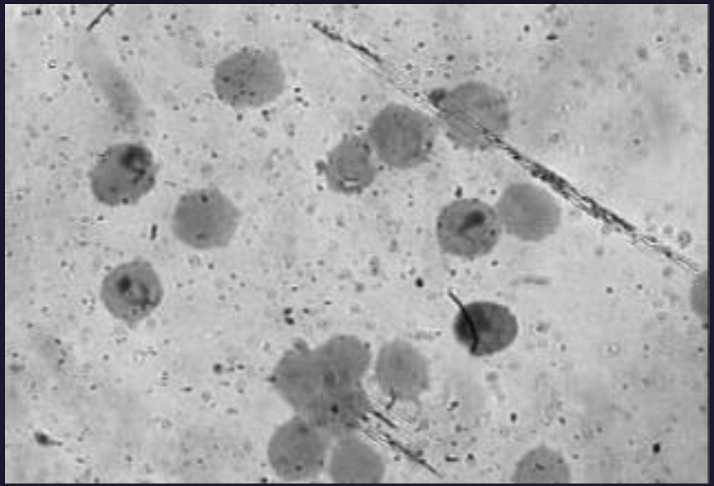

Original


Grayscale

In Python, this can be done by using the opencv2 library.

Example 1: Malaria-infected Cells

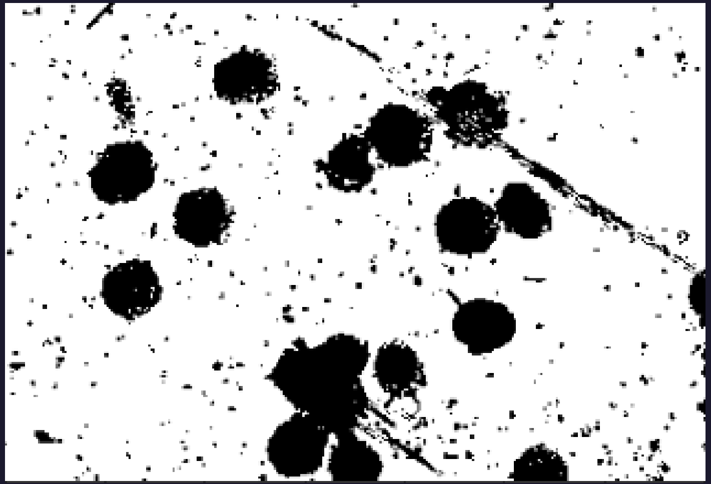
Step 1: Load the image as grayscale



```
import cv2

malaria = cv2.imread(path+file1,0) #load
image in grayscale
```

Example 1: Malaria-infected Cells

Step 2: Perform initial segmentation and binarize the resulting image

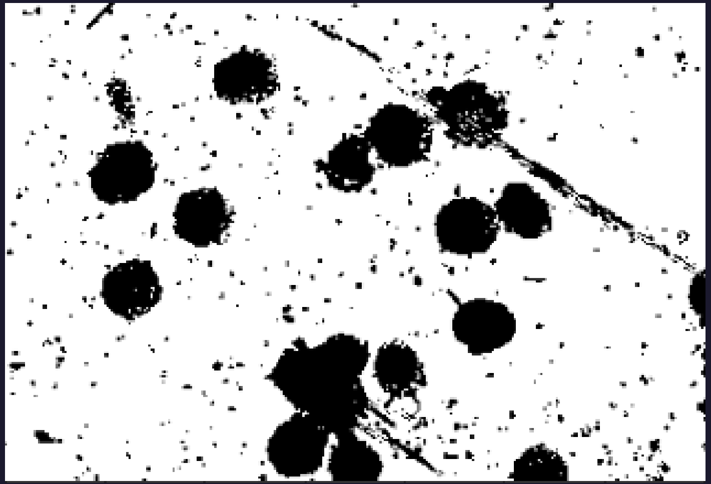


```
from skimage import filters

def gray_thresh(image, low, high):
    image[image<low] = 0
    image[image>high] = 255
    return image

thresholded= gray_thresh(malaria, 121, 164)
val= filters.threshold_otsu(thresholded[np.isfinite(thresholded)])
BW = thresholded > val
plt.imshow(BW, cmap='gray')
```

Usually, grayscale thresholding suffices, especially when the features or blobs stand well against the background.

Example 1: Malaria-infected Cells

Step 2: Perform initial segmentation and binarize the resulting image

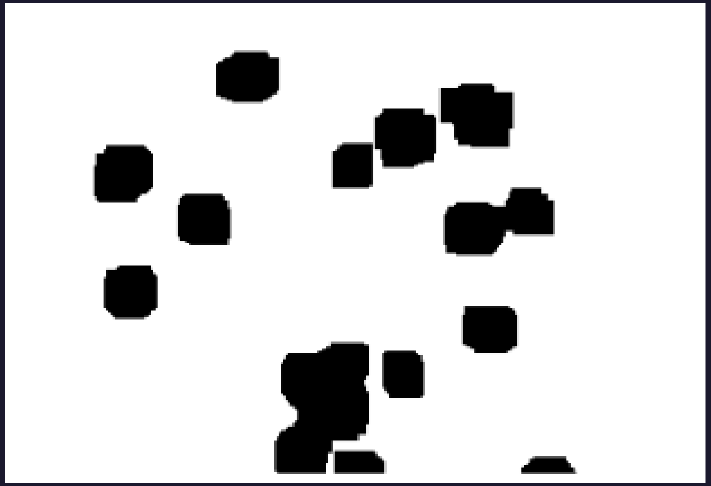


```
from skimage import filters

def gray_thresh(image, low, high):
    image[image<low] = 0
    image[image>high] = 255
    return image

thresholded= gray_thresh(malaria, 121, 164)
val= filters.threshold_otsu(thresholded[np.isfinite(thresholded)])
BW = thresholded > val
plt.imshow(BW, cmap='gray')
```

The equivalent of Matlab's imbinarize() in Python is the skimage.filters.threshold_otsu function. By binarizing the image, we can now perform morphological segmentation.

Example 1: Malaria-infected Cells

Step 3: Cleaning the segmented image

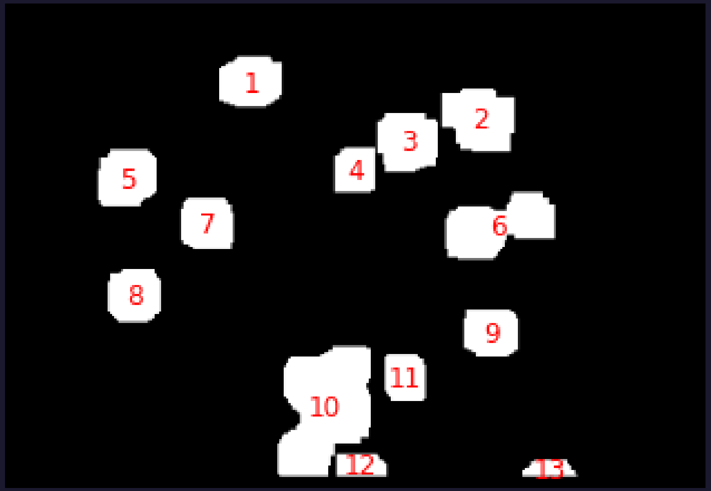


```
strel1 = np.ones((3,3))
plt.imshow(BW, cmap='gray')
BW2= ndimage.binary_opening(BW, strel1)
plt.imshow(BW2, cmap='gray')
BW3 = ndimage.binary_closing(BW2, strel1, iterations=7)
BW3[0:7,:] = 1
BW3[:, 0:7] = 1
BW3[:, -7::] = 1
BW3[-7::, :] = 1
plt.imshow(BW3, cmap='gray')
```

A sequence of morphological operations is performed on the binarized segmented image to remove artifacts. The holes or gaps in between the blobs must be closed; otherwise, the separated pieces are counted as different blobs or features.

Example 1: Malaria-infected Cells

Step 4: Count and label features

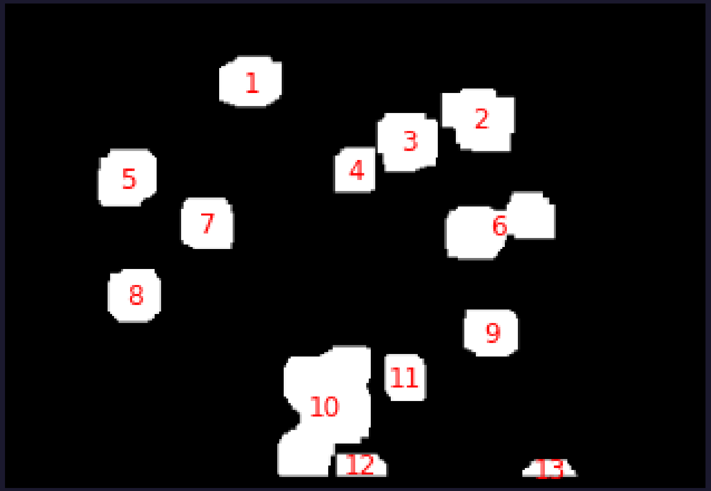


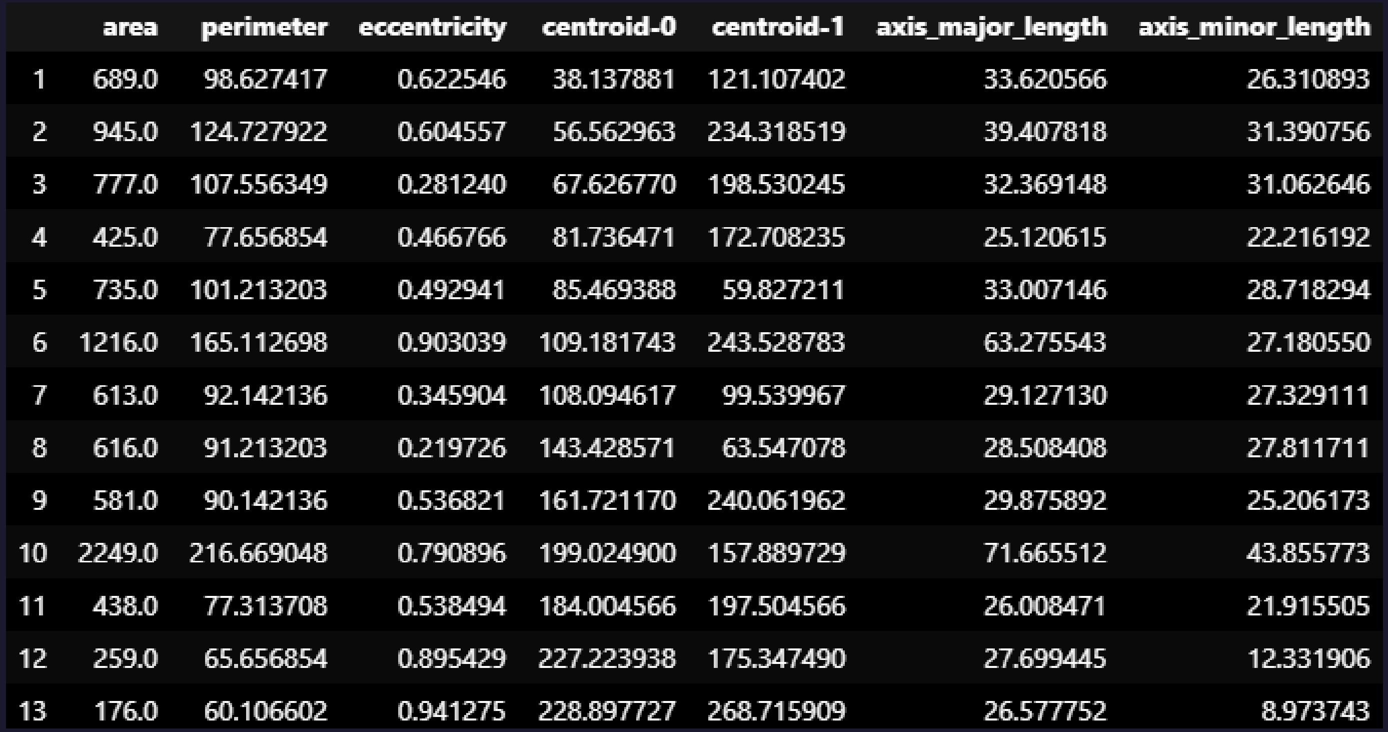
```
image = np.logical_not(BW3) # Inverts the image
label_img = label(image)
regions = regionprops(label_img)
props = regionprops_table(label_img, properties = ('area', 'perimeter', 'eccentricity',
'centroid', 'axis_major_length', 'axis_minor_length'))

plt.imshow(image,cmap='gray')

for region in regions:
    centroid = region.centroid
    label_value = region.label
    plt.text(centroid[1], centroid[0], str(label_value), color='red', fontsize=12,
ha='center', va='center')

plt.show()
```

Using skimage.measure.regionprops(), the blobs or features in the image are counted and labeled. The image needs to be inverted before labeling and counting, because it counts the "white" regions in the image.

Example 1: Malaria-infected Cells

Step 4: Count and label features



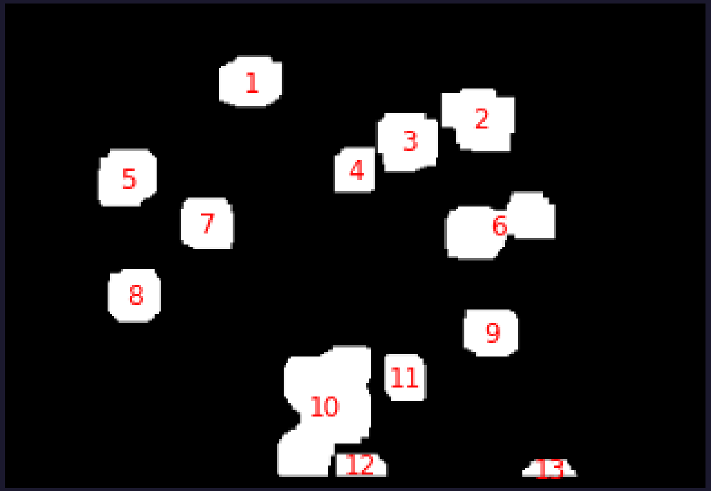


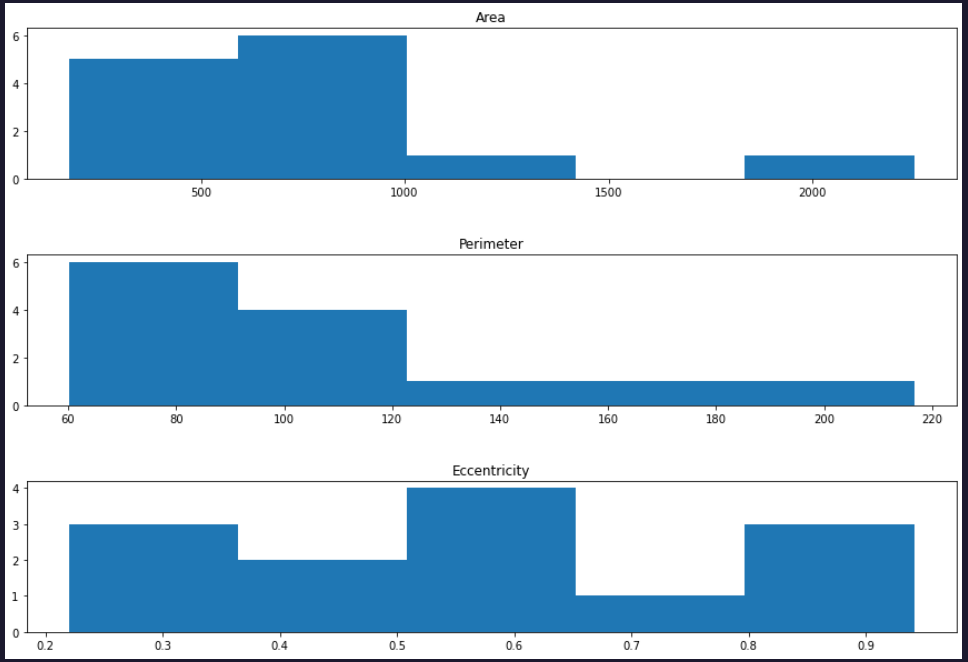
	area	perimeter	eccentricity	centroid-0	centroid-1	axis_major_length	axis_minor_length
1	689.0	98.627417	0.622546	38.137881	121.107402	33.620566	26.310893
2	945.0	124.727922	0.604557	56.562963	234.318519	39.407818	31.390756
3	777.0	107.556349	0.281240	67.626770	198.530245	32.369148	31.062646
4	425.0	77.656854	0.466766	81.736471	172.708235	25.120615	22.216192
5	735.0	101.213203	0.492941	85.469388	59.827211	33.007146	28.718294
6	1216.0	165.112698	0.903039	109.181743	243.528783	63.275543	27.180550
7	613.0	92.142136	0.345904	108.094617	99.539967	29.127130	27.329111
8	616.0	91.213203	0.219726	143.428571	63.547078	28.508408	27.811711
9	581.0	90.142136	0.536821	161.721170	240.061962	29.875892	25.206173
10	2249.0	216.669048	0.790896	199.024900	157.889729	71.665512	43.855773
11	438.0	77.313708	0.538494	184.004566	197.504566	26.008471	21.915505
12	259.0	65.656854	0.895429	227.223938	175.347490	27.699445	12.331906
13	176.0	60.106602	0.941275	228.897727	268.715909	26.577752	8.973743

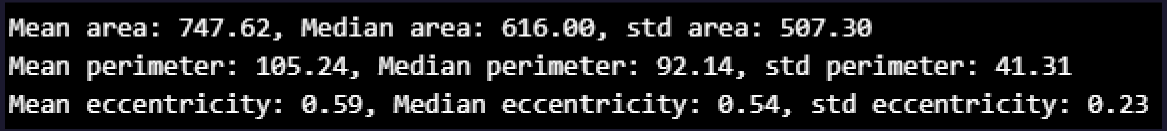
By loading `props` as a Pandas dataframe, we can then display the measurements on each blob. Since no scale was used, the unit of distance used for the calculations is a pixel.

Example 1: Malaria-infected Cells

Step 5: Perform statistical analysis







```
Mean area: 747.62, Median area: 616.00, std area: 507.30
Mean perimeter: 105.24, Median perimeter: 92.14, std perimeter: 41.31
Mean eccentricity: 0.59, Median eccentricity: 0.54, std eccentricity: 0.23
```

By treating each column of the dataFrame as a numpy array, the statistics can be calculated.

Example 1: Malaria-infected Cells

Step 5: Perform statistical analysis

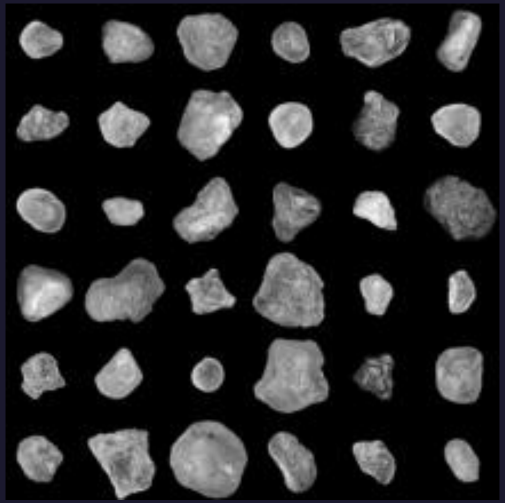
```
df= pd.DataFrame(props)
df.index = df.index + 1
bins = 5
fig, ax = plt.subplots(3,1, figsize=(15,10))
ax[0].hist(df['area'], bins=bins)
ax[0].set_title('Area')
ax[1].hist(df['perimeter'], bins=bins)
ax[1].set_title('Perimeter')
ax[2].hist(df['eccentricity'], bins=bins)
ax[2].set_title('Eccentricity')
fig.subplots_adjust(hspace=0.5)
print('Mean area: {a:.2f}, Median area: {c:.2f}, std area: {b:.2f}'.format(a=np.mean(df['area']), c=np.median(df['area']),
b=np.std(df['area'])))
print('Mean perimeter: {a:.2f}, Median perimeter: {c:.2f}, std perimeter: {b:.2f}'.format(a=np.mean(df['perimeter']),
c=np.median(df['perimeter']), b=np.std(df['perimeter'])))
print('Mean eccentricity: {a:.2f}, Median eccentricity: {c:.2f}, std eccentricity: {b:.2f}'.format(a=np.mean(df['eccentricity']),
c=np.median(df['eccentricity']), b=np.std(df['eccentricity'])))
```

This code snippet displays the histograms and prints the statistics of the segmentation data

Example 2: Pebbles

Step 1: Load the image as grayscale

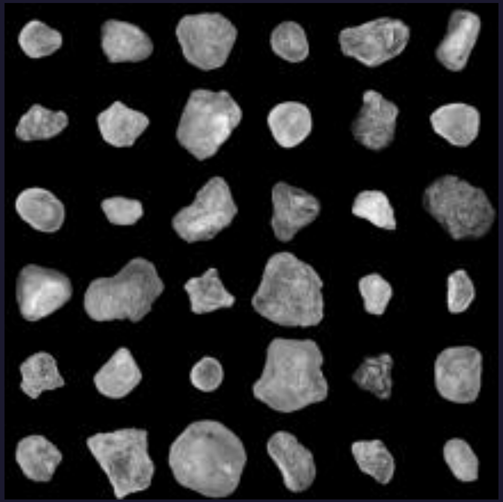
Original


Grayscale

In Python, this can be done by using the opencv2 library.

Example 2: Pebbles

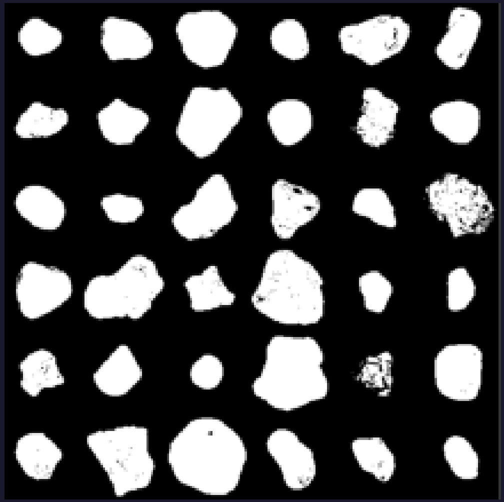
Step 1: Load the image as grayscale



```
import cv2

image = cv2.imread(path+file1,0) #load image
in grayscale
```

Example 2: Pebbles

Step 2: Perform initial segmentation and binarize the resulting image

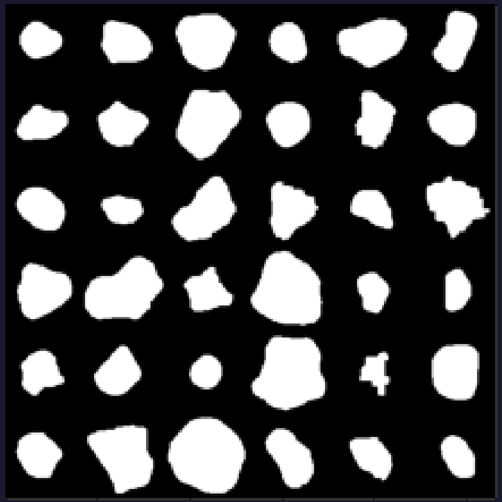


```
thresholded = gray_thresh(image, 18, 255)
val=
filters.threshold_otsu(thresholded[np.isfinite(
thresholded)])
plt.figure()
BW = thresholded > val
plt.imshow(BW, cmap='gray')
```

The equivalent of Matlab’s imbinarize() in Python is the skimage.filters.threshold_otsu function. By binarizing the image, we can now perform morphological segmentation.

Example 2: Pebbles

Step 3: Cleaning the segmented image

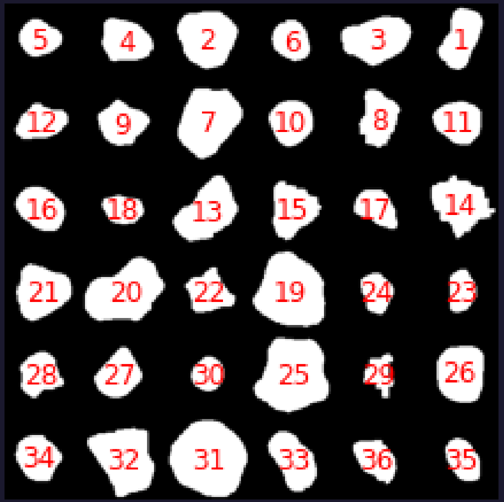


```
strel1 = np.ones((3,3))
plt.figure()
plt.imshow(BW, cmap='gray')
BW2= ndimage.binary_opening(BW, strel1)
plt.figure()
plt.imshow(BW2, cmap='gray')
BW3 = ndimage.binary_closing(BW2, strel1, iterations=5)
plt.figure()
plt.imshow(BW3, cmap='gray')
```

A sequence of morphological operations is performed on the binarized segmented image to remove artifacts. The holes or gaps in between the blobs must be closed; otherwise, the separated pieces are counted as different blobs or features.

Example 2: Pebbles

Step 4: Count and label features

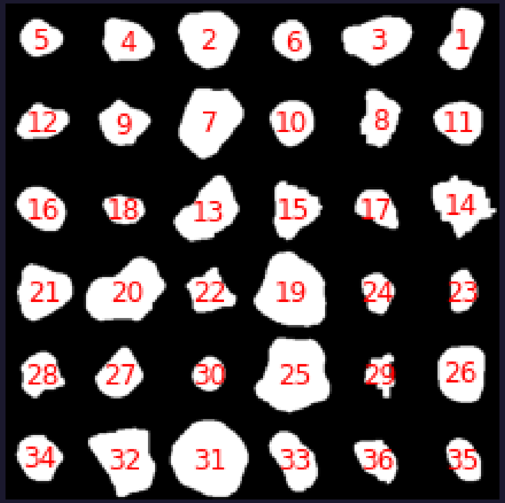


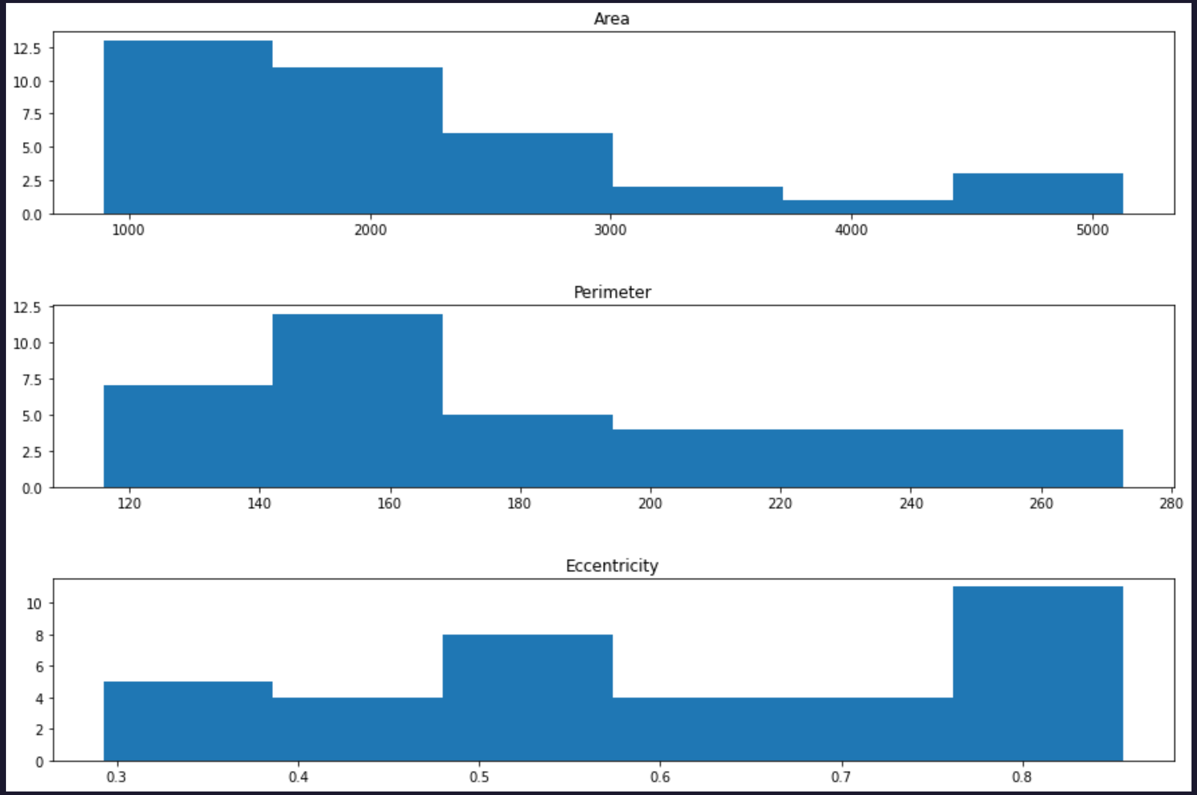
```
image = np.logical_not(BW3) # Inverts the image
label_img = label(image)
regions = regionprops(label_img)
props = regionprops_table(label_img, properties = ('area', 'perimeter', 'eccentricity',
'centroid', 'axis_major_length', 'axis_minor_length'))

plt.imshow(image,cmap='gray')

for region in regions:
    centroid = region.centroid
    label_value = region.label
    plt.text(centroid[1], centroid[0], str(label_value), color='red', fontsize=12,
ha='center', va='center')

plt.show()
```

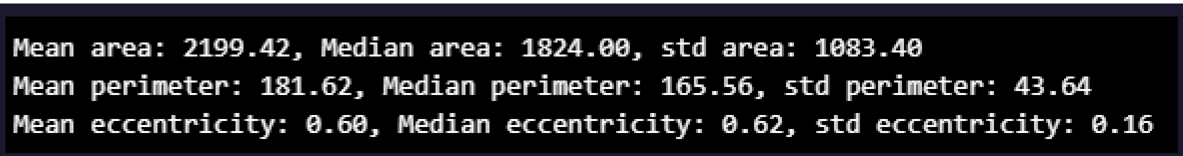
Using skimage.measure.regionprops(), the blobs or features in the image are counted and labeled. The image needs to be inverted before labeling and counting, because it counts the "white" regions in the image.

Example 2: Pebbles

Step 5: Perform statistical analysis



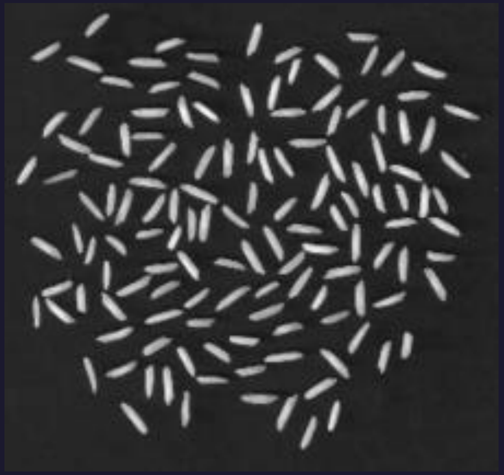



```
Mean area: 2199.42, Median area: 1824.00, std area: 1083.40
Mean perimeter: 181.62, Median perimeter: 165.56, std perimeter: 43.64
Mean eccentricity: 0.60, Median eccentricity: 0.62, std eccentricity: 0.16
```

By treating each column of the dataFrame as a numpy array, the statistics can be calculated.

Example 3: Rice Grains

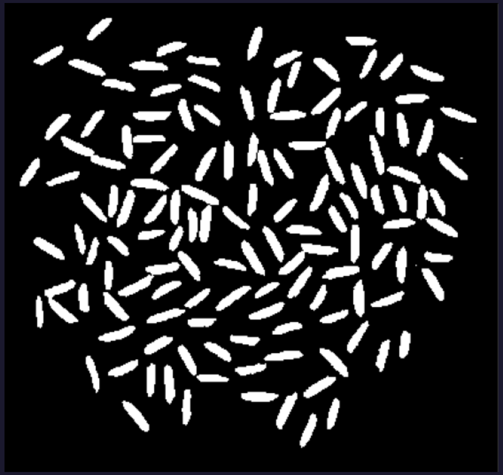
Step 1: Load the image as grayscale



```
import cv2

image = cv2.imread(path+file1,0) #load image
in grayscale
```

Example 3: Rice Grains

Step 2: Perform initial segmentation and binarize the resulting image

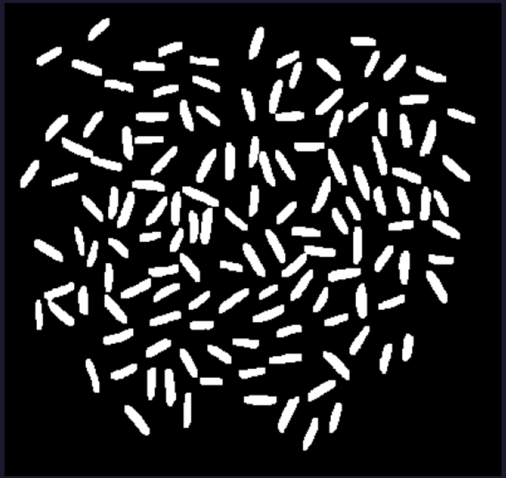


```
thresholded = gray_thresh(image, 18, 255)
val=
filters.threshold_otsu(thresholded[np.isfinite(
thresholded)])
plt.figure()
BW = thresholded > val
plt.imshow(BW, cmap='gray')
```

The equivalent of Matlab's imbinarize() in Python is the skimage.filters.threshold_otsu function. By binarizing the image, we can now perform morphological segmentation.

Example 3: Rice Grains

Step 3: Cleaning the segmented image

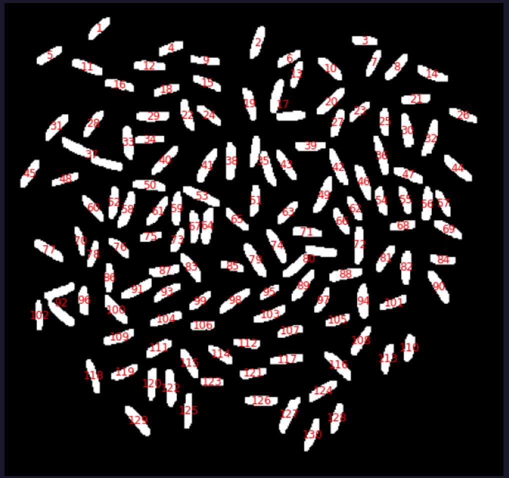


```
strel1 = np.ones((3,2))
plt.figure(figsize=(10,10))
plt.imshow(BW, cmap='gray')
BW2= ndimage.binary_opening(BW, strel1, iterations=2)
plt.figure(figsize=(10,10))
plt.imshow(BW2, cmap='gray')
```

A sequence of morphological operations is performed on the binarized segmented image to remove artifacts. The holes or gaps in between the blobs must be closed; otherwise, the separated pieces are counted as different blobs or features. Note that the image is already clean in this case, so minimal cleaning is done. Too many morphological operations may alter the rice grain sizes too much.

Example 3: Rice Grains

Step 4: Count and label features

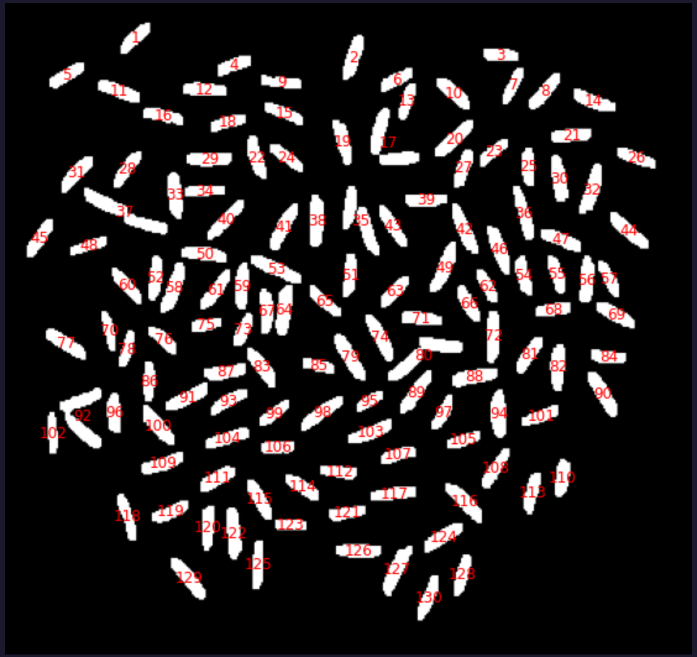


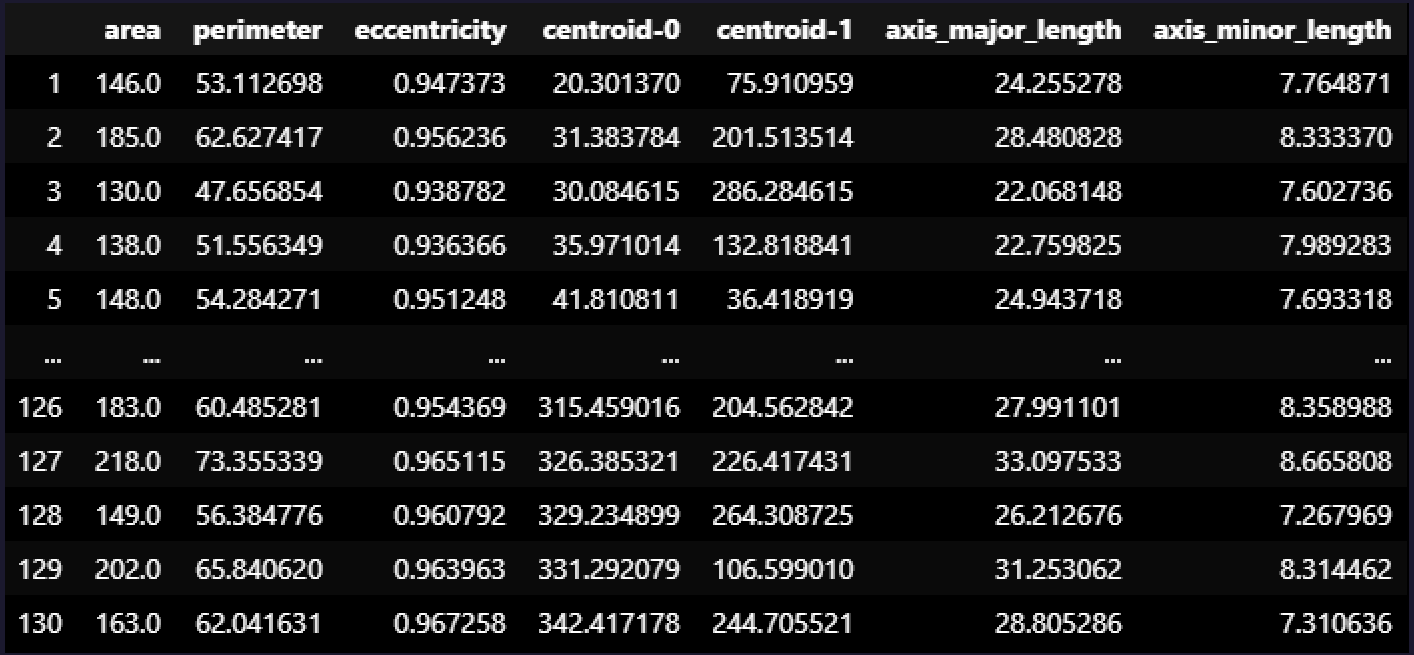
```
image = np.logical_not(BW3) # Inverts the image
label_img = label(image)
regions = regionprops(label_img)
props = regionprops_table(label_img, properties = ('area', 'perimeter', 'eccentricity',
'centroid', 'axis_major_length', 'axis_minor_length'))

plt.imshow(image,cmap='gray')

for region in regions:
    centroid = region.centroid
    label_value = region.label
    plt.text(centroid[1], centroid[0], str(label_value), color='red', fontsize=12,
ha='center', va='center')

plt.show()
```

Using skimage.measure.regionprops(), the blobs or features in the image are counted and labeled. The image needs to be inverted before labeling and counting, because it counts the "white" regions in the image. 130 rice grains were identified in the image.

Example 3: Rice Grains

Step 4: Count and label features



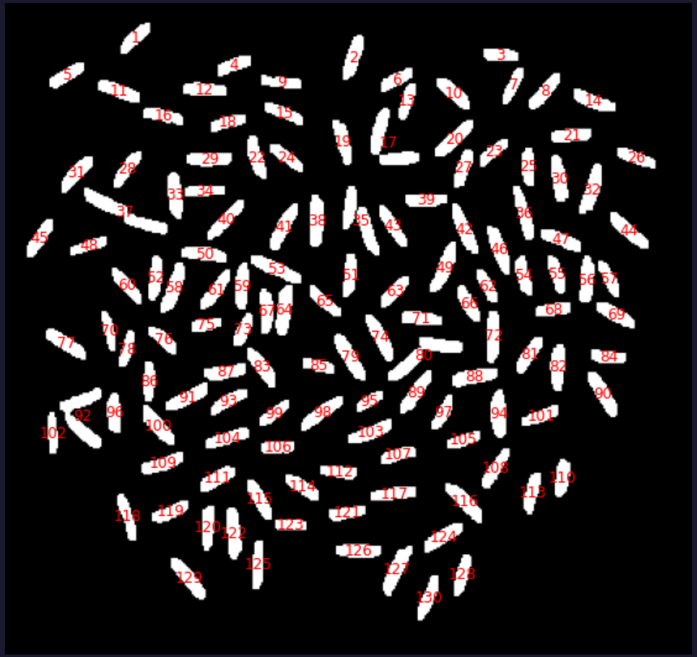


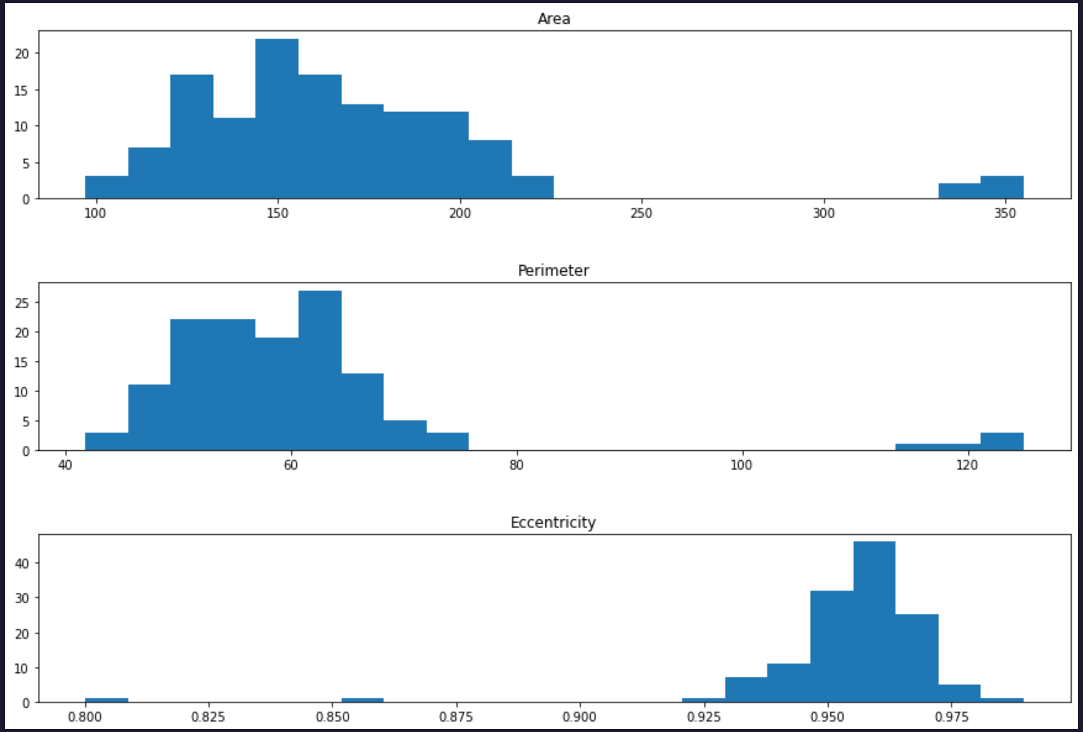
	area	perimeter	eccentricity	centroid-0	centroid-1	axis_major_length	axis_minor_length
1	146.0	53.112698	0.947373	20.301370	75.910959	24.255278	7.764871
2	185.0	62.627417	0.956236	31.383784	201.513514	28.480828	8.333370
3	130.0	47.656854	0.938782	30.084615	286.284615	22.068148	7.602736
4	138.0	51.556349	0.936366	35.971014	132.818841	22.759825	7.989283
5	148.0	54.284271	0.951248	41.810811	36.418919	24.943718	7.693318
...	...	...	...	...	...	...	...
126	183.0	60.485281	0.954369	315.459016	204.562842	27.991101	8.358988
127	218.0	73.355339	0.965115	326.385321	226.417431	33.097533	8.665808
128	149.0	56.384776	0.960792	329.234899	264.308725	26.212676	7.267969
129	202.0	65.840620	0.963963	331.292079	106.599010	31.253062	8.314462
130	163.0	62.041631	0.967258	342.417178	244.705521	28.805286	7.310636

Using skimage.measure.regionprops(), the blobs or features in the image are counted and labeled. The image needs to be inverted before labeling and counting, because it counts the "white" regions in the image. 130 rice grains were identified in the image.

Example 3: Rice Grains

Step 4: Count and label features





```
Mean area: 167.48, Median area: 161.00, std area: 45.39
Mean perimeter: 60.39, Median perimeter: 58.28, std perimeter: 13.90
Mean eccentricity: 0.95, Median eccentricity: 0.96, std eccentricity: 0.02
```

Note that save for some outliers, the area, perimeter, and eccentricities of the rice grains obey a normal distribution. Normal distributions are prevalent in nature, and rice grain characteristics should not be an exception.

Self-Reflection

Overall, I think I did okay in this activity. This was just a showcase of the techniques I've learned in the previous two modules. The only new thing here was to label and perform quantitative measurements on the segmented images.

However, there are some things that I wanted to experiment and try, but were unable to do so because of time constraints. Here are some of them.

1. Using a scale in the images.
2. Additional images.
3. Performing statistical analysis on the features that meet a certain criteria. This way, outliers can be eliminated.

Self-score: 95/100