Counting Molecules Report.

Using machine learning based tools in the python programming language, the objectives of this project was to remove the mean, isolate the blobs if any and, calculate and analyse the clusters in order to gain some insight and draw conclusions. This report will walk you through the methods I used and the problems I encountered as result, as well as the conclusions drawn.

Blob Detection.

There are 3 kinds of methods to detect blobs:

- Laplacian of Gaussian (LoG)
- Difference of Gaussian (DoG)
- Determinant of Hessian (DoH)

My approach to this task was to use all the methods mentioned and compare the results. The data was provided in .sxm files. This proved to be a hurdle as all the image processing codes are made for jpeg and jpg.

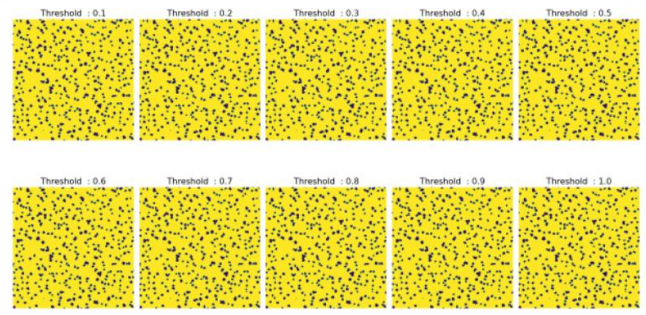
Before applying the blob detection, I binarized the images provided and iterated over several threshold levels to find one that might have produced a better mask.

Binarization.

I binarized the image in order to remove the background noise. This was done in preparation for blob detection. However, I came across some difficulties as the images were often too large and so they had to be binarized while zoomed in to get better visuals. In order to do this, I used 2 different codes as the first did not work with a zoomed in graph/image.

```
def threshold_checker(image):
    thresholds =  np.arange(0.1,1.1,0.1)
    image_gray = bw
    fig, ax = plt.subplots(2, 5, figsize=(17, 10))
    for n, ax in enumerate(ax.flatten()):
        ax.set_title(f'Threshold  : {round(thresholds[n],2)}',
                     fontsize = 16)
        threshold_tree = image_gray < thresholds[n]
        ax.imshow(threshold_tree);
        ax.axis('off')
    fig.tight_layout()

threshold_checker(im2)
```

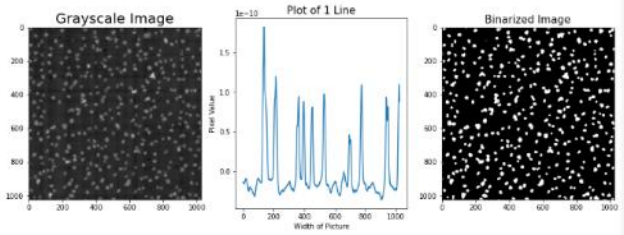


The threshold_checker iterates over several threshold levels and finds the optimum threshold to be used on the image. This ensures that the best mask is produced.

The threshold_checker shows that the mask produced will be the same at any threshold. Thus, meaning that when binarized the threshold can be set from 0.1-1.0 and the binarization will be the same.

```
fig, ax = plt.subplots(1,3,figsize=(15,5))
sample_b = imslice > 0.6
ax[0].set_title('Grayscale Image',fontsize=20)
ax[0].imshow(imslice,cmap='gray')
ax[1].plot(imslice[600])
ax[1].set_ylabel('Pixel Value')
ax[1].set_xlabel('Width of Picture')
ax[1].set_title('Plot of 1 Line',fontsize=15)
ax[2].set_title('Binarized Image',fontsize=15)
ax[2].imshow(bw,cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x23fb06affa0>
```

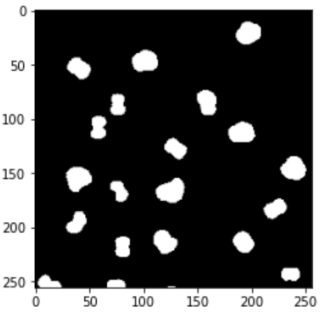


This code shows the width and grayscale image, this helps remove the unnecessary noise and thus can be used to show the different blobs in an image.

Since there is a lot going on in the binarized image, I opted for the binarization of a zoomed image. However, this code is cannot display the binarized image at [0:256,0:256].

So, I opted to use the code below.

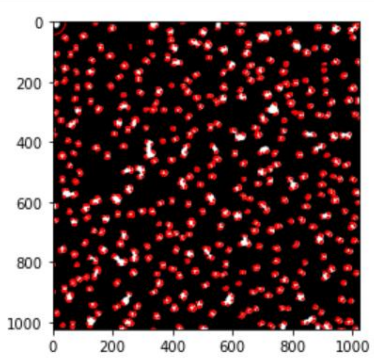
```
image= rgb2gray(imslice[0:256,0:256])
image_gray1= plt.imshow(clean_border[0:256,0:256],
                        cmap='gray')
thresh = threshold_otsu(imslice)
bw = closing(imslice > thresh, square(3))
# image
```



While this code does not show the grayscale image and the width of the picture, it uses Otsu's method to find the optimal threshold value as opposed to the code above. This helped me avoid running the threshold_checker function seen above.

Laplacian of the Gaussian

Determines the blobs by using the LoG method. This method proved to be the wrong approach as there was a lot of overlapping thus preventing the blobs to be detected properly. I attempted to fix this by adjusting max_sigma, num_sigma and threshold however, nothing changed.

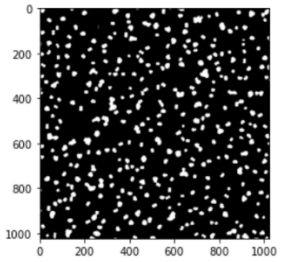


```
# Laplacian of the gaussian
# Using the functions imported above, we can adjust the blob detection using the parameters max_sigma,num_sigma and threshold
blobs = blob_log(bw, max_sigma=30, min_sigma =2, num_sigma=2, threshold=0.3, overlap = 0.6)
fig, ax = plt.subplots()
ax.imshow(bw, cmap='gray')
for blob in blobs:
    y, x, area = blob
    ax.add_patch(plt.Circle((x, y), area*np.sqrt(2), color='r', fill=False))
# write on what it identifies....
# blob detection (medium)
```

The difference of the Hessian

This approach did not work as it did not detect a single blob. The aforementioned parameters were continuously adjusted however no blobs were detected.

```
# the difference of the hessian
blobs = blob_doh(bw, max_sigma=30, min_sigma =2, num_sigma=2, threshold=0.3, overlap = 0.3)
fig, ax = plt.subplots()
ax.imshow(bw, cmap='gray')
for blob in blobs:
    y, x, area = blob
    ax.add_patch(plt.Circle((x, y), area*np.sqrt(2), color='r', fill=False))
```

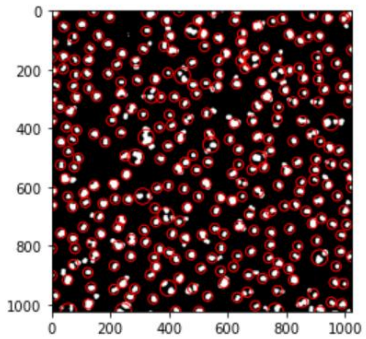


The difference of the Gaussian

This proved to be the best blob detector as it circled most blobs individually. It works better than the Laplacian of the Gaussian because there is no overlap evident. Furthermore, when the parameters are adjusted the number of blobs detected change thus showing that the code works.

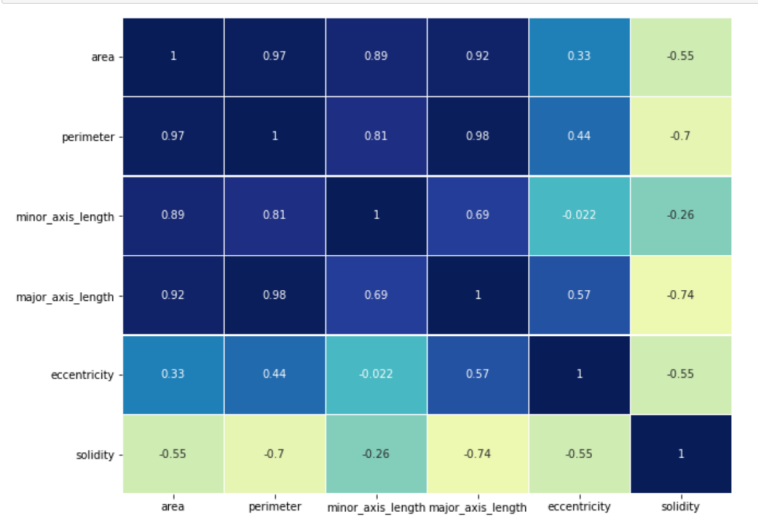
```
# the difference of the gausssian
blobs = blob_dog(bw, max_sigma=30, min_sigma =12, threshold=0.3, overlap = 0.3)
fig, ax = plt.subplots()
ax.imshow(bw, cmap='gray')
for blob in blobs:
    y, x, area = blob
    ax.add_patch(plt.Circle((x, y), area*np.sqrt(2), color='r', fill=False))

#     THE difference of the gaussian works abit better than the laplacian of gaussian
```



Exploratory Data Analysis for Kmeans Clustering

The data provided was relatively clean as there were no missing or repeated values. Additionally, there were negligible outliers.



	area	perimeter	minor_axis_length	major_axis_length	eccentricity	solidity
count	333.000000	333.000000	333.000000	333.000000	333.000000	333.000000
mean	351.297297	71.071663	17.848290	24.939505	0.627339	0.932368
std	184.534236	23.865005	4.163392	9.027917	0.162057	0.043375
min	12.000000	10.656854	3.813415	4.270321	0.189291	0.711538
25%	239.000000	58.627417	15.180631	20.318789	0.518471	0.929012
50%	298.000000	65.455844	16.848639	23.181540	0.655629	0.945513
75%	400.000000	75.597980	20.397378	25.441092	0.740349	0.954430
max	1368.000000	192.409163	38.150161	69.489147	0.952680	0.995305

```
<seaborn.axisgrid.PairGrid at 0x23fb16b9c40>
```