Assignment 3 – Semantic

Segmentation Method

Prepared by

**Dylan Gartin**

**Course name** -Computer Vision (SP 2022)

**Course number** - CAP4410.01

**Date of submission** – 3/11/2022

\* Note we will be formatting the reports now with style, Due to major issues that occur even still with trying to implement the template. If you want something added to this template let us know.

# Table Of Contents

[Table Of Contents 2](#_Toc98311853)

[Introduction 3](#_Toc98311854)

[OTSU\_2 (Binary) Segmentation Method – 3](#_Toc98311855)

[Figure 1 – OTSU Method Code Example 4](#_Toc98311856)

[Figure 2 – OTSU Method Images Results 6](#_Toc98311857)

[OTSU\_Multiple Segmentation Method – 6](#_Toc98311858)

[Figure 3 – OTSU Multi Threshold Method 7](#_Toc98311859)

[Figure 4 -OTSU Multi-Threshold Image Results 8](#_Toc98311860)

[Mean Shift Segmentation Method - 8](#_Toc98311861)

[Figure 5 - Mean Shift Code 9](#_Toc98311862)

[Figure 6 – Attempted Code Fixes 11](#_Toc98311863)

[UPDATE – Mean Shift Solution 11](#_Toc98311864)

[Figure 7 – New pyrMeanShiftFiltering Mean Shift Segmentation Solution. 11](#_Toc98311865)

[Figure 8 – Mean Shift Segmentation with Blur 12](#_Toc98311866)

[Challenges 13](#_Toc98311867)

[Conclusion 13](#_Toc98311868)

[References - 13](#_Toc98311869)

# Introduction

The Goal of this assignment is to complete the following task with a couple of sample image that are used in order to either detect or segment objects that are within an image. OTSU Methodologies are used in segmentation, where you can make pieces of an image different segments. Within the OTSU Method you can make it a binary image with two classes or can use multiple classes as well. As for mean shift this is mostly used for finding an object and then keeping track of the object in a scene, but it can also be used to segment an image as well though this is a bit involved. Below are the goals of this assignment and as such what will be covered within this report.:

* Otsu Method with 2 classes
* Otsu Method with multiple classes
* Mean Shift Method Implementation

With the brief introduction completed let’s begin study of the OTSU Methods.

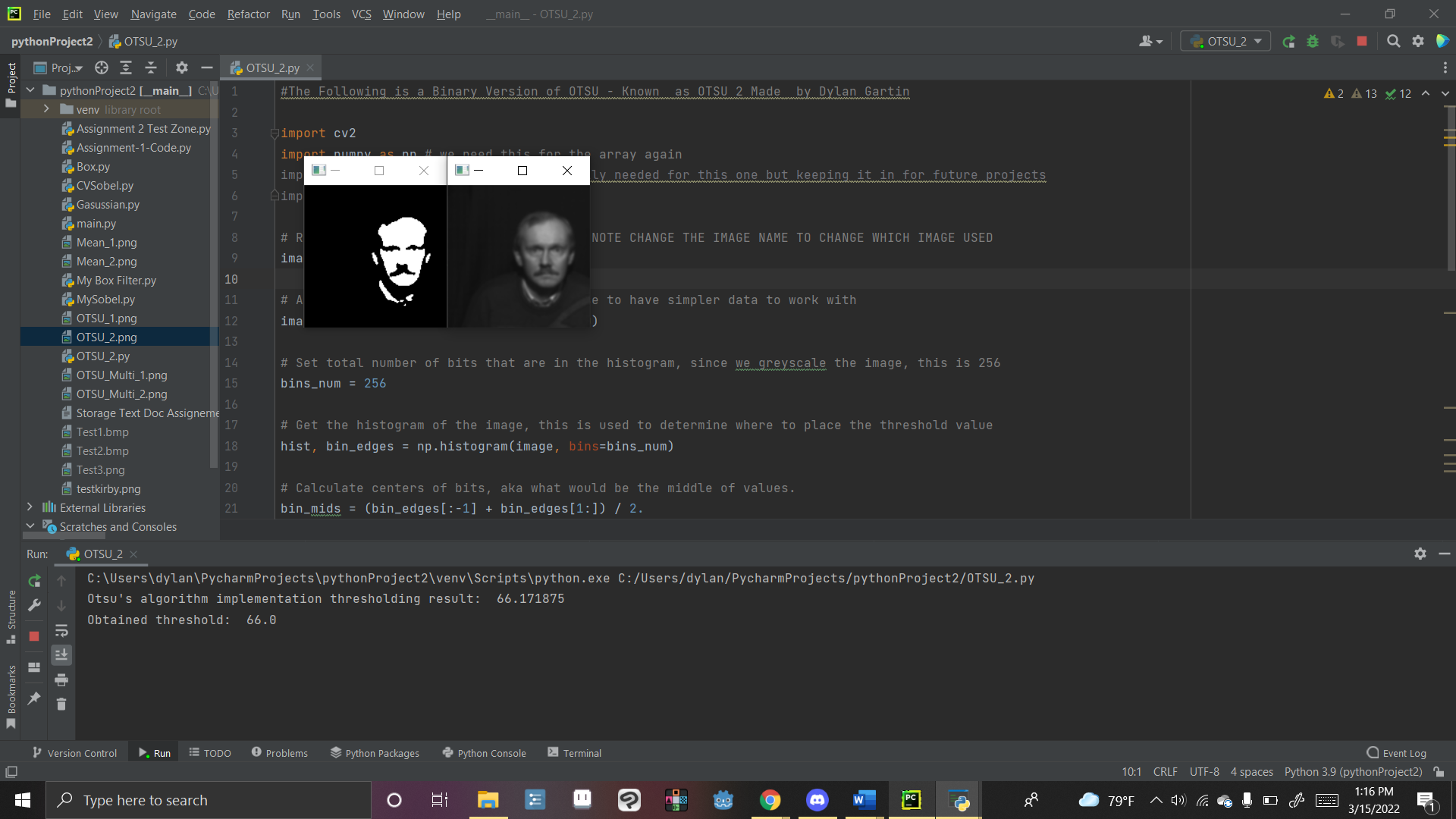
# OTSU\_2 (Binary) Segmentation Method –

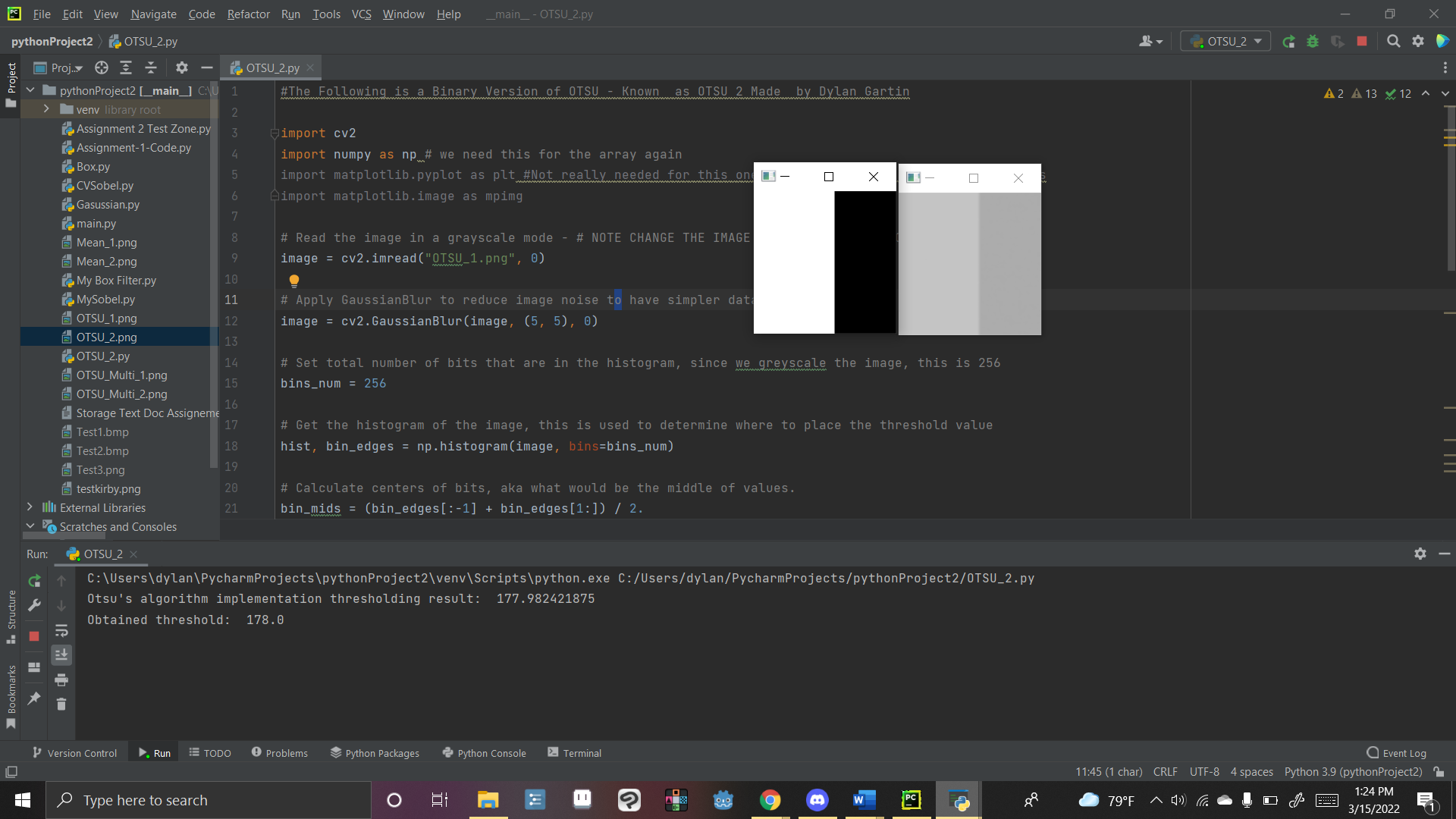
In order for us to apply a binary version of the OTSU method we need to have a method to find where to set the threshold of the image in order to determine which data is set to black and which is set to white. After that open cv has an algorithm that is able to apply the OTSU method to an image itself, so we can use that to assist us. Overall, the code focuses on getting the data needed to determine the threshold, which includes getting the histogram of the bit’s data and then applying it with a mean, and then applying that threshold to the method. Below is the code for such:

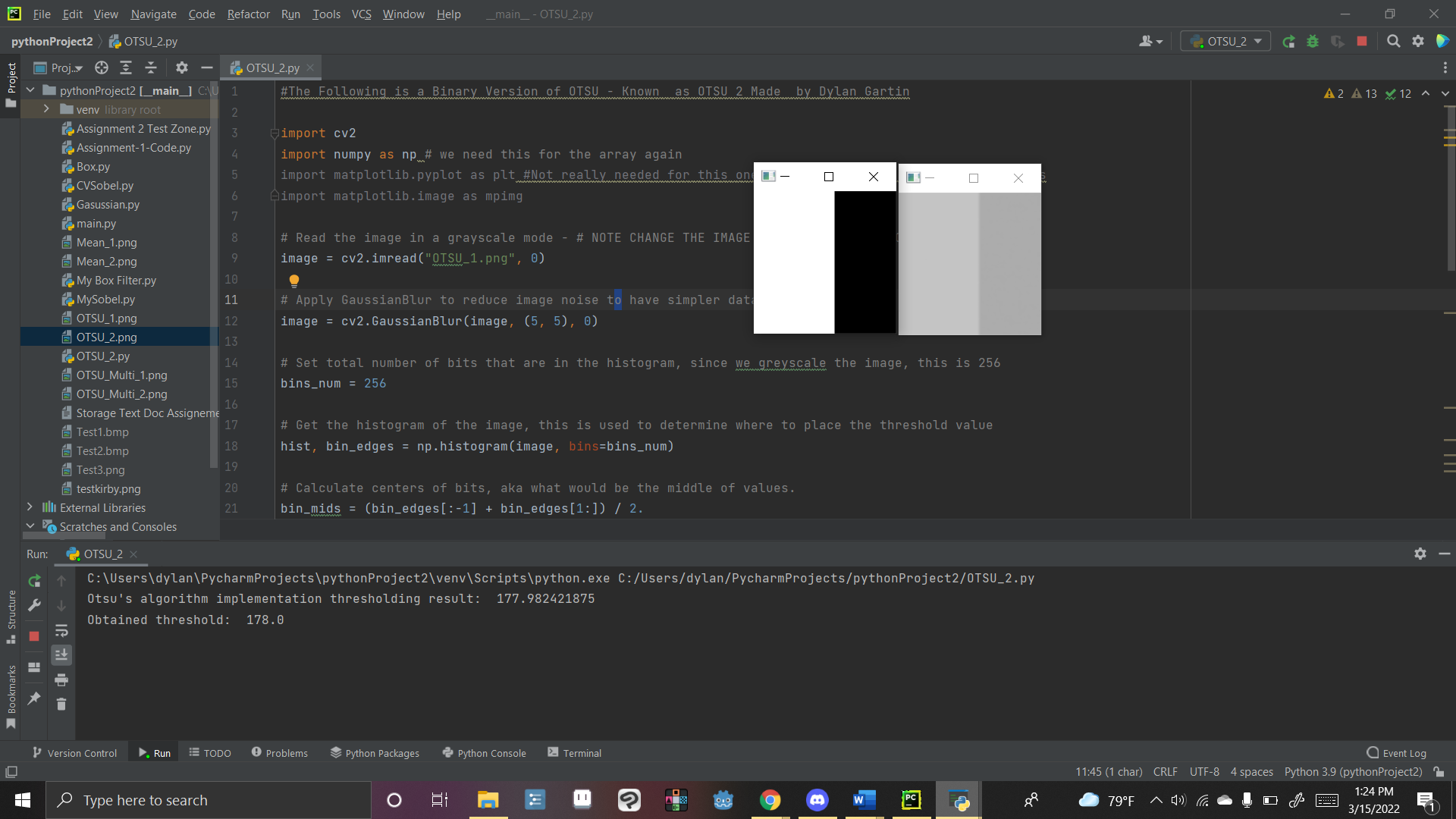
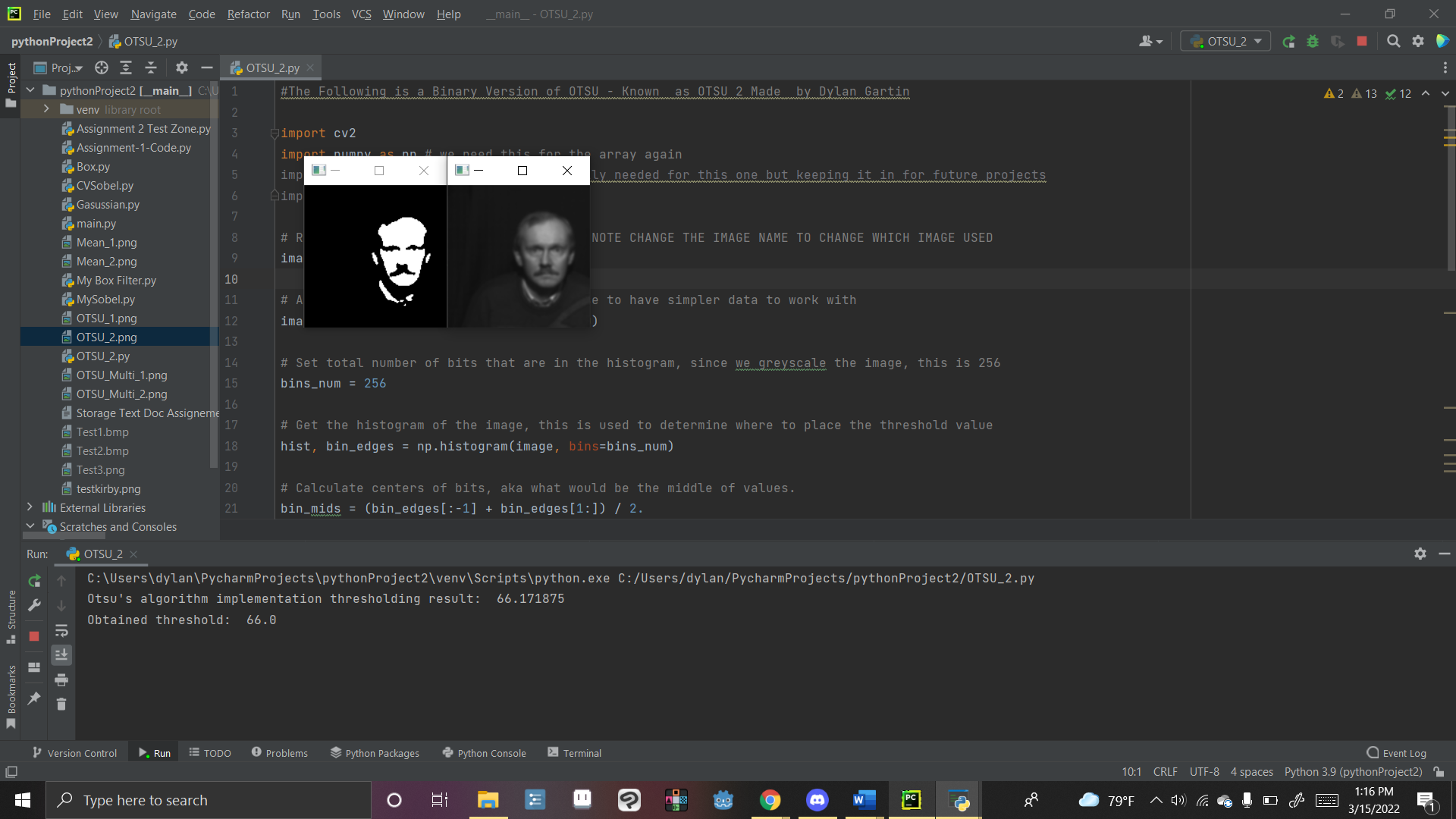
#The Following is a Binary Version of OTSU - Known as OTSU 2 Made by Dylan Gartin  
  
import cv2  
import numpy as np # we need this for the array again  
import matplotlib.pyplot as plt #Not really needed for this one but keeping it in for future projects  
import matplotlib.image as mpimg  
  
# Read the image in a grayscale mode - # NOTE CHANGE THE IMAGE NAME TO CHANGE WHICH IMAGE USED  
image = cv2.imread("test.png", 0)  
  
# Apply GaussianBlur to reduce image noise to have simpler data to work with  
image = cv2.GaussianBlur(image, (5, 5), 0)  
  
# Set total number of bits that are in the histogram, since we greyscale the image, this is 256  
bins\_num = 256  
  
# Get the histogram of the image, this is used to determine where to place the threshold value  
hist, bin\_edges = np.histogram(image, bins=bins\_num)  
  
# Calculate centers of bits, aka what would be the middle of values.  
bin\_mids = (bin\_edges[:-1] + bin\_edges[1:]) / 2.  
  
# Iterate over all thresholds (indices) and get the probabilities.  
weight1 = np.cumsum(hist)  
weight2 = np.cumsum(hist[::-1])[::-1]  
  
#get the means of these using the cumulative sum  
mean1 = np.cumsum(hist \* bin\_mids) / weight1  
mean2 = (np.cumsum((hist \* bin\_mids)[::-1]) / weight2[::-1])[::-1]  
  
# then combine them together  
inter\_class\_variance = weight1[:-1] \* weight2[1:] \* (mean1[:-1] - mean2[1:]) \*\* 2  
  
# Maximize the threshold by using this inter\_class\_variance function val  
index\_of\_max\_val = np.argmax(inter\_class\_variance)  
threshold = bin\_mids[:-1][index\_of\_max\_val]  
print("Otsu's algorithm implementation thresholding result: ", threshold)  
  
# Applying Otsu's method by using the above data point that we got for the threshold and having this passed in for that value.  
otsu\_threshold, image\_result = cv2.threshold(  
 image, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU,  
)  
#print this threshold as well to check it alings with our data  
print("Obtained threshold: ", otsu\_threshold)  
cv2.imshow('Image', image) #Output the blured and greyscale img  
cv2.imshow('OTSU\_2\_RESULTS', image\_result) #Output the OTSU\_2 Method results  
cv2.waitKey(0) # Wait key so we can actually see the image

## Figure 1 – OTSU Method Code Example

With the ability to automatically detect thresholds our code is able to automatically apply the OTSU method to the given images once they are passed through as seen below







## Figure 2 – OTSU Method Images Results

Overall, the main focus of the image is pulled out rather effectively using this type of segmentation and honestly, I myself have been working on making a lot of binary art so I may need to try using this filter out myself.

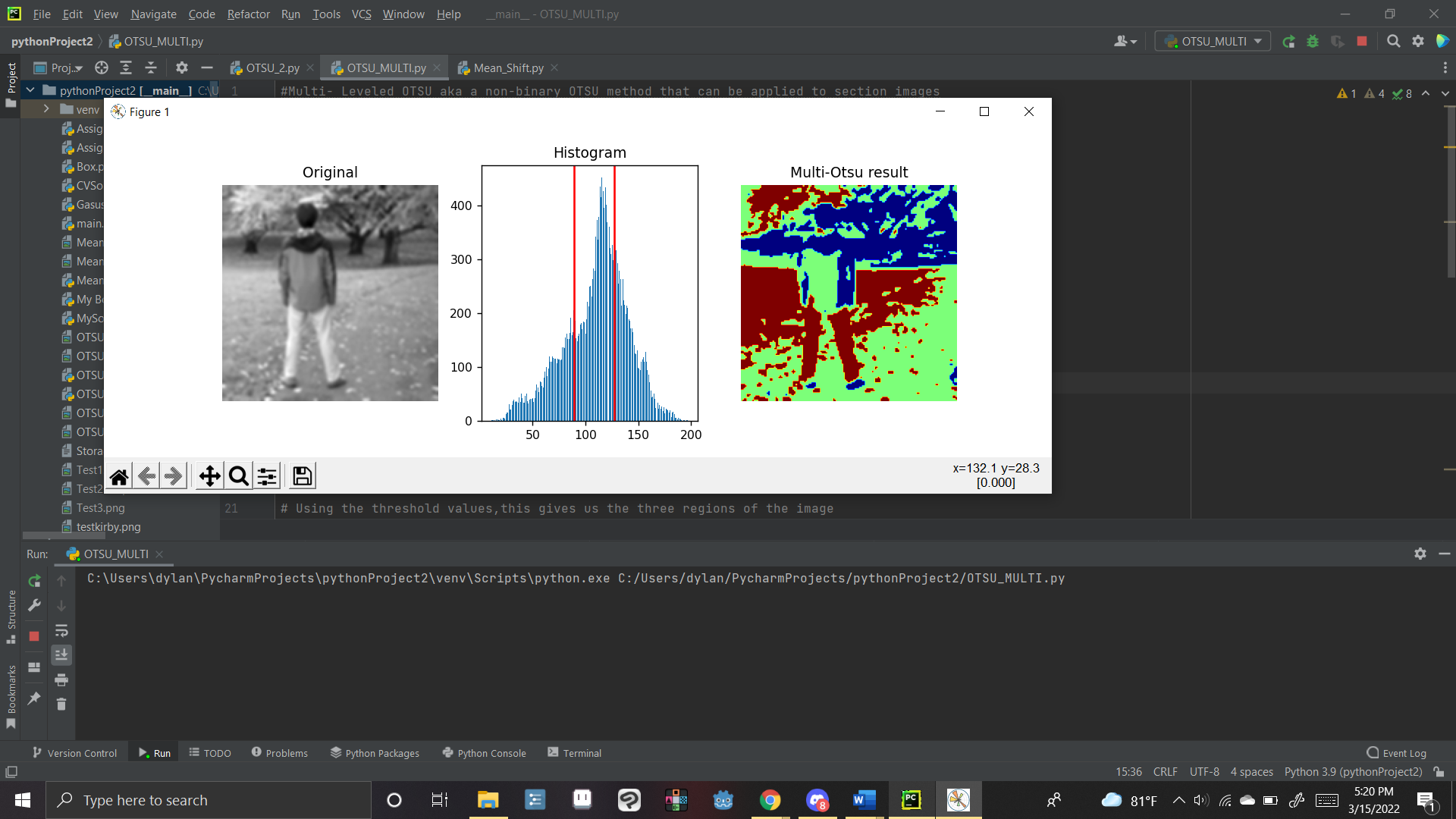
# OTSU\_Multiple Segmentation Method –

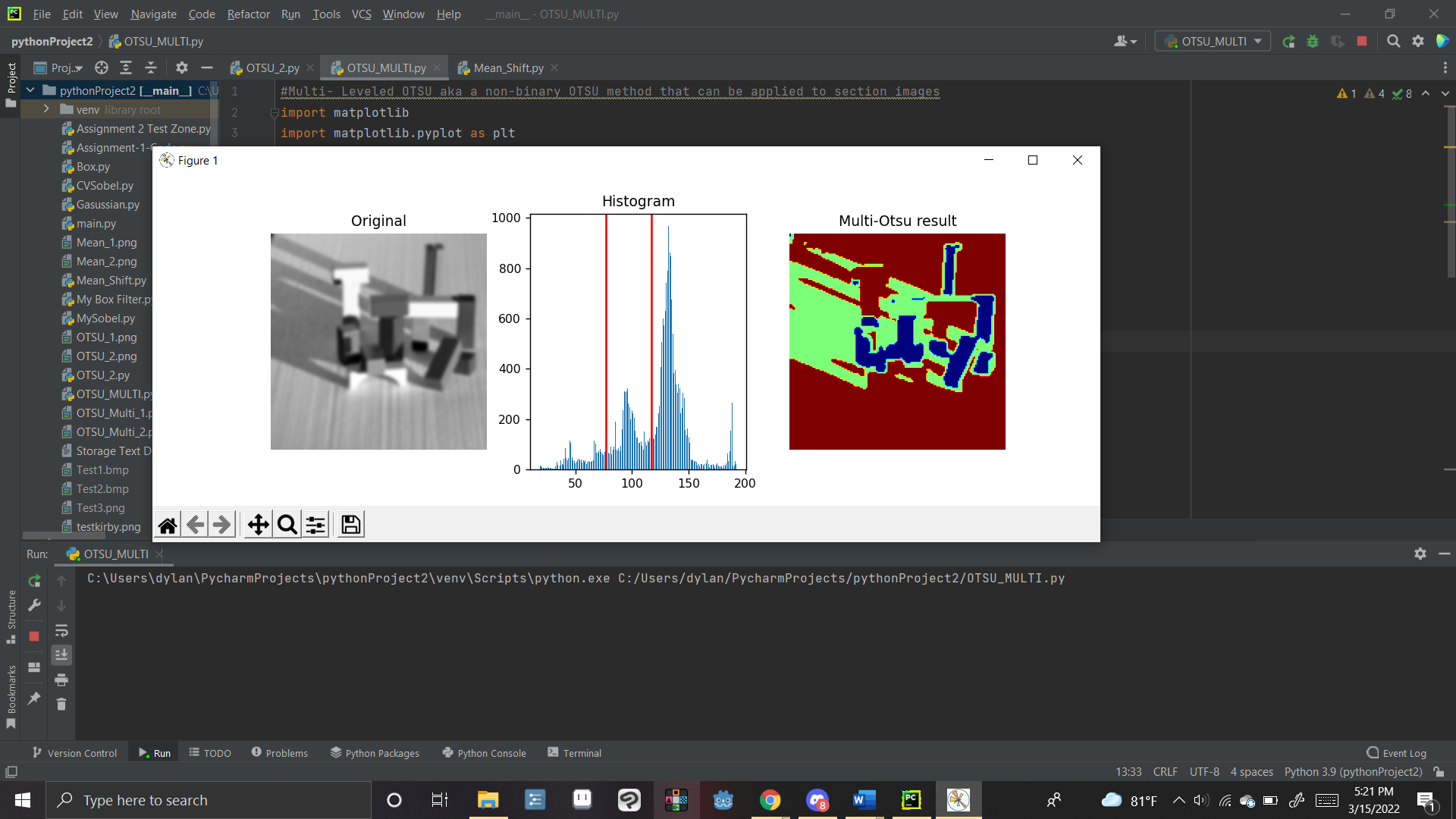
When it comes to the Multi Method of OTSU, it is essentially the same as the other OTSU method, however we now need to have additional segments for the image to be divided by. As such we need to apply multiple thresholds. This results in us using Skimage to be able to find multiple appreciate thresholds that our image can use. We also add in the histogram this time for extra clarity on where the thresholds come from. As such the code can be seen below;

#Multi- Leveled OTSU aka a non-binary OTSU method that can be applied to section images  
import matplotlib  
import matplotlib.pyplot as plt  
import numpy as np  
import cv2  
from skimage import data  
from skimage.filters import threshold\_multiotsu  
  
# this is used to set the font to make it reaable  
matplotlib.rcParams['font.size'] = 9  
  
# The image we want to apply the OTSU to  
image = cv2.imread("OTSU\_Multi\_2.png", 0)  
  
#blur the image to make it clearer?  
image = cv2.GaussianBlur(image, (5, 5), 0)  
  
# For Multi OTSU we can simply use the threshold Multiotsu of Skimage class  
thresholds = threshold\_multiotsu(image)  
  
# Using the threshold values,this gives us the three regions of the image  
regions = np.digitize(image, bins=thresholds)  
  
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 3.5))  
  
# Plotting the original image onto the chart  
ax[0].imshow(image, cmap='gray')  
ax[0].set\_title('Original')  
ax[0].axis('off')  
  
# Plotting the histogram  
ax[1].hist(image.ravel(), bins=255)  
ax[1].set\_title('Histogram')  
for thresh in thresholds:  
 ax[1].axvline(thresh, color='r')  
  
# Plotting the Multi Otsu result.  
ax[2].imshow(regions, cmap='jet')  
ax[2].set\_title('Multi-Otsu result')  
ax[2].axis('off')  
  
plt.subplots\_adjust()  
  
plt.show()  
cv2.waitKey(0) # Wait key so we can actually see the image

## Figure 3 – OTSU Multi Threshold Method

` This results in us getting the following images and histograms for our sample pictures:





## Figure 4 -OTSU Multi-Threshold Image Results

It important to note that our multi leveled segmentation data seems to have a bit of trouble distinguishing shadows. If there was a need for it, we may want to make it where the shadow data is removed from the set. However overall, for basic computer image object detection, most of if not all the data is kept. As such I would say this would be effective for image detection. We may however want to see if there is a way to segment objects based on data besides colors as well, but besides such we are able to use this data as is for this assignment.

# Mean Shift Segmentation Method -

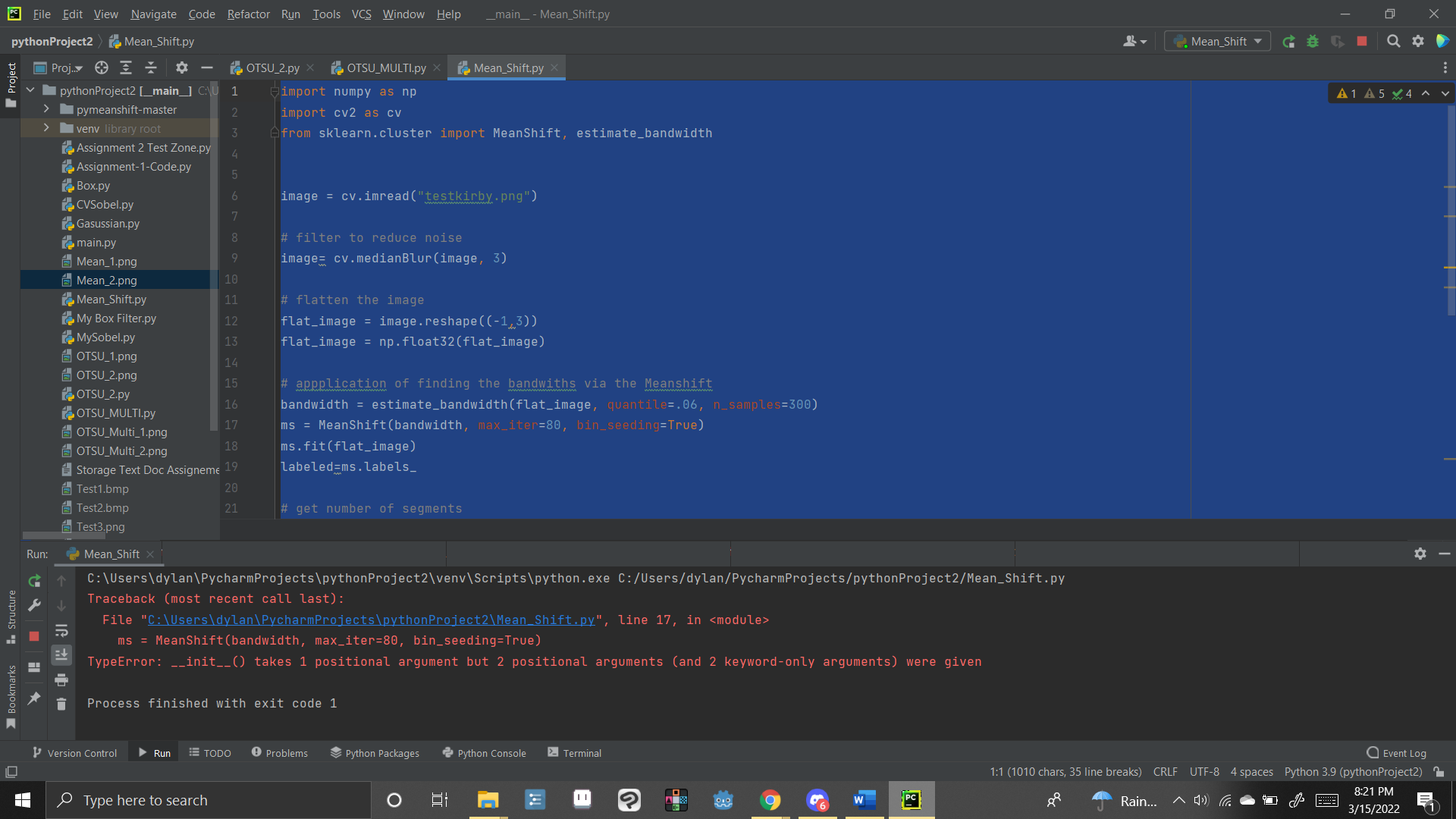
When it comes to using mean shift segmentation, this is where things get a bit tricky, and in fact we were at first. Mean shift is normally used for object detection within videos rather then for segmentation. As such using it for such requires a decent bit of know how and proper code. Overall though we were able to find code that should work for doing this method, using the same sklearn libraries similar to how we used them in OTSU\_MULTI… but this simply did not work. The baseline code that we came up with after doing extensive research on the topic is the following.

import numpy as np  
import cv2 as cv  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
  
  
image = cv.imread("testkirby.png")  
  
# filter to reduce noise  
image= cv.medianBlur(image, 3)  
  
# flatten the image  
flat\_image = image.reshape((-1,3))  
flat\_image = np.float32(flat\_image)  
  
# appplication of finding the bandwiths via the Meanshift  
bandwidth = estimate\_bandwidth(flat\_image, quantile=.06, n\_samples=300)  
ms = MeanShift(bandwidth, max\_iter=80, bin\_seeding=True)  
ms.fit(flat\_image)  
labeled=ms.labels\_  
  
# get number of segments  
segments = np.unique(labeled)  
print('Number of segments: ', segments.shape[0])  
  
# get the average color of each segment  
total = np.zeros((segments.shape[0], 3), dtype=float)  
count = np.zeros(total.shape, dtype=float)  
for i, label in enumerate(labeled):  
 total[label] = total[label] + flat\_image[i]  
 count[label] += 1  
avg = total/count  
avg = np.uint8(avg)  
  
# cast the labeled image into the corresponding average color  
res = avg[labeled]  
result = res.reshape((image.shape))

## Figure 5 - Mean Shift Code

Apparently the issue is caused by “TypeError: method() takes 1 positional argument but 2 were given” Python gives itself another argument called self when it calls the function which is likely why the issue is occurring here, thus passing in 4 arguments instead of 3. The issue is that however despite us knowing the issue we don’t know how to solve it.

We tried multitudes of alterations to this code in order to make it work the one that was able to go the farthest without error, up to line 33 is seen below.



import numpy as np  
import cv2 as cv  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
  
  
image = cv.imread("testkirby.png")  
  
# filter to reduce noise  
image= cv.medianBlur(image, 3)  
  
# flatten the image  
flat\_image = image.reshape((-1,3))  
flat\_image = np.float32(flat\_image)  
print('test1')  
  
# appplication of finding the bandwiths via the Meanshift  
bandwidth = estimate\_bandwidth(flat\_image, quantile=.06, n\_samples=300)  
print('test2')  
bandwidth = MeanShift( max\_iter=80, bin\_seeding=True) # THIS IS DEFIENTLY WHERE THE ERROR IS AT BUT HOW TO FIX  
print('test3')  
# bandwidth.fit(flat\_image)  
print('test4')  
labeled = bandwidth  
print('test5')  
  
# get number of segments  
segments = np.unique(labeled)  
print('Number of segments: ', segments.shape[0])  
  
# get the average color of each segment  
total = np.zeros((segments.shape[0], 3), dtype=float)  
count = np.zeros(total.shape, dtype=float)  
for i, label in enumerate(labeled):  
 total[label] = total[label] + flat\_image[i]  
 count[label] += 1  
avg = total/count  
avg = np.uint8(avg)  
  
# cast the labeled image into the corresponding average color  
res = avg[labeled]  
result = res.reshape((image.shape))

## Figure 6 – Attempted Code Fixes

As of this time we have no solution to fix this problem. And as such is unable to be solved as this is the only lead, I can find for doing mean shift with open.cv. we will be going to office hours tomorrow to ask if there is any way to fix this and get help, and we will give an update from here.

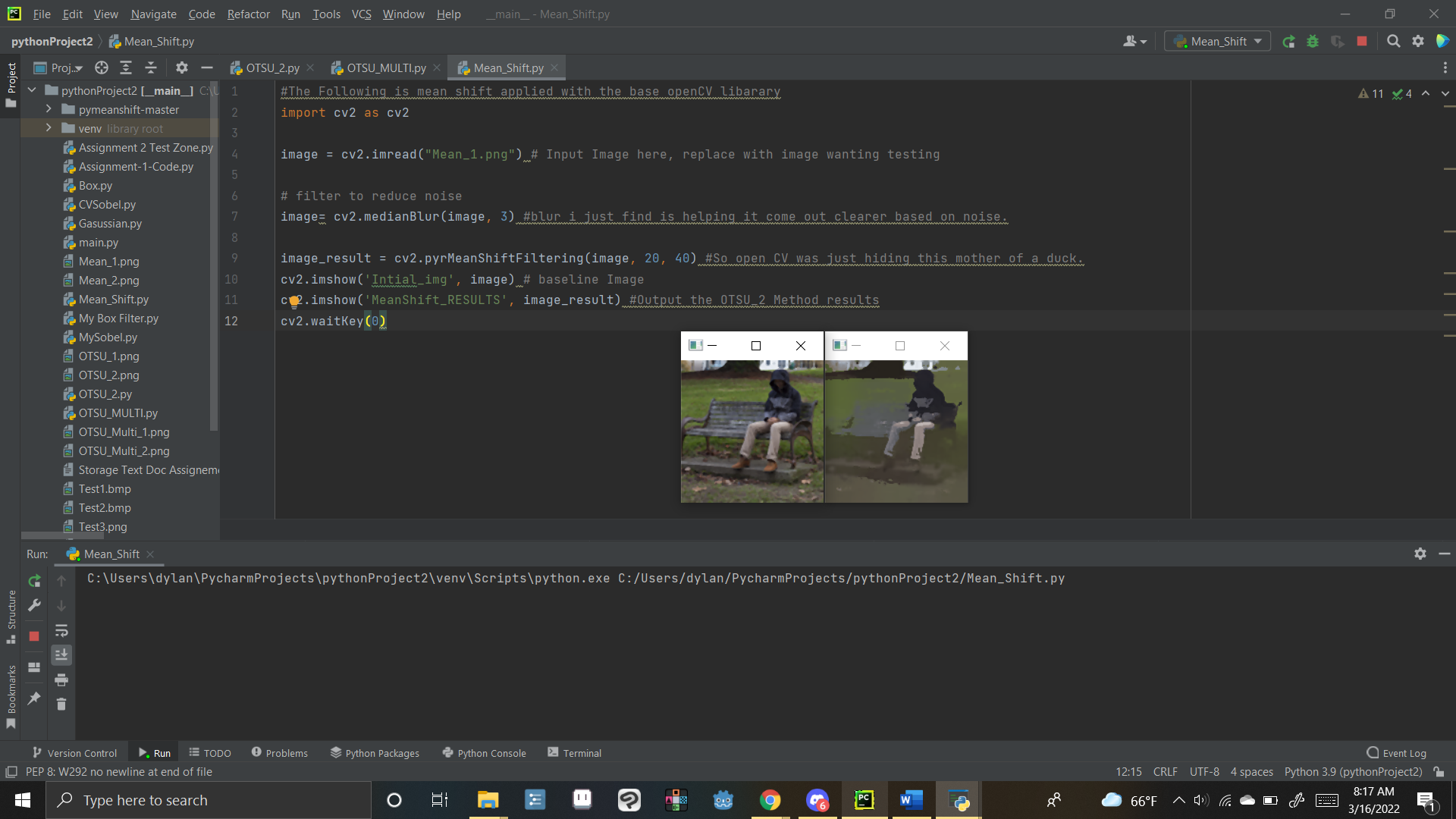
# UPDATE – Mean Shift Solution

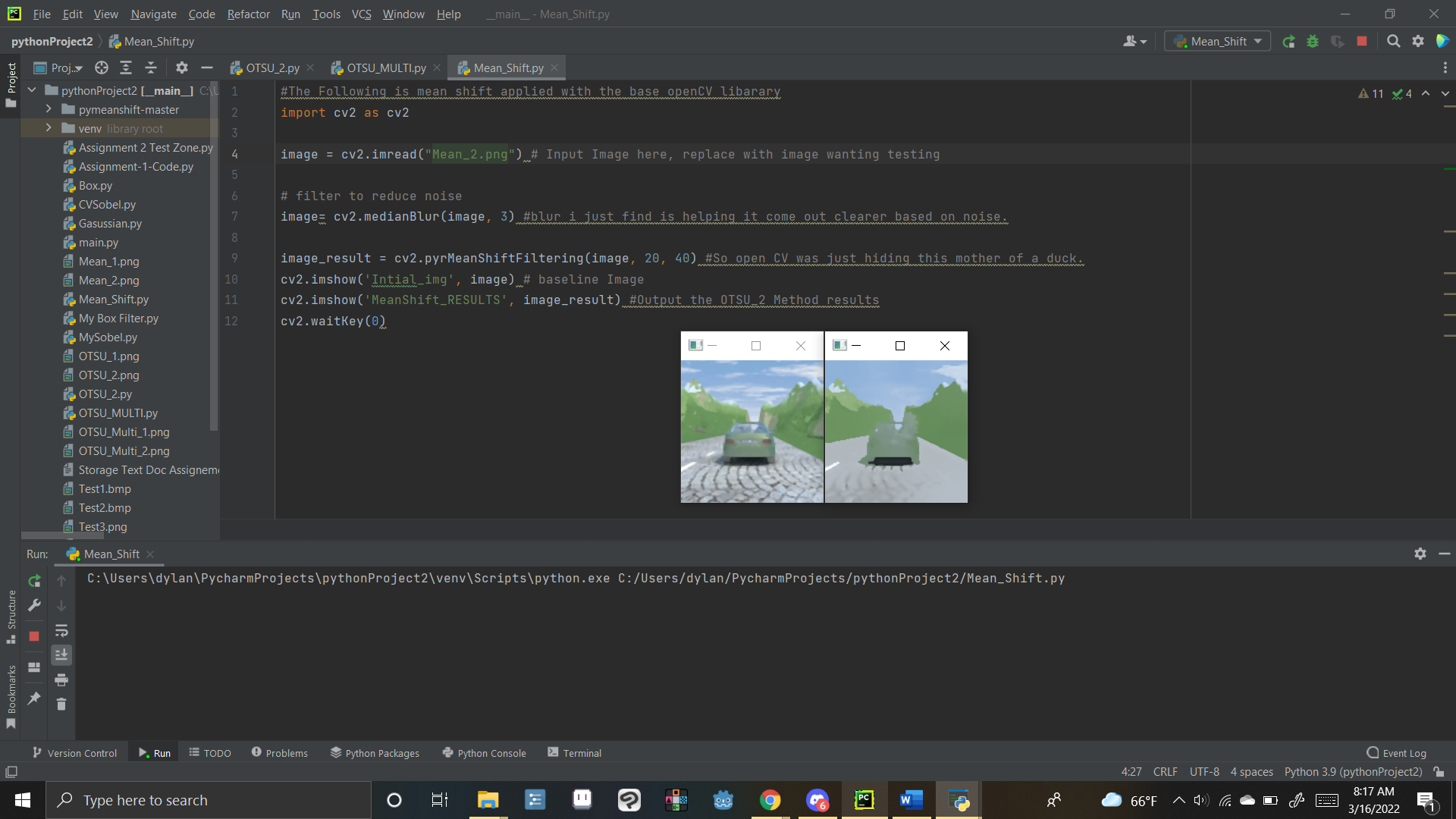
After discussing with classmates about this, we found a version of mean shift that is able to work. Quite simply in fact, OpenCV seems to have a mean shift library out of the box that is used for segmentation. This also lines up with our research as this function was mentioned on it. While I tired to use it with the old code apparently you simply need to use it by itself. Below is the code for this new method;

#The Following is mean shift applied with the base openCV libarary  
import cv2 as cv2  
  
image = cv2.imread("testkirby.png") # Input Image here, replace with image wanting testing  
  
# filter to reduce noise  
image= cv2.medianBlur(image, 3) #blur i just find is helping it come out clearer based on noise.  
  
image\_result = cv2.pyrMeanShiftFiltering(image, 20, 40) #So open CV was just hiding this mother of a duck.  
cv2.imshow('Intial\_img', image) # baseline Image  
cv2.imshow('MeanShift\_RESULTS', image\_result) #Output the OTSU\_2 Method results  
cv2.waitKey(0)

## Figure 7 – New pyrMeanShiftFiltering Mean Shift Segmentation Solution.

With our newfound power we now use it on the sample images to get the following results:





## Figure 8 – Mean Shift Segmentation with Blur

Interestingly enough it seems the blur removes a bit of the data here on the image even though it was very useful on the OTSU method. As such we also tried it without the blur which led to pretty much the exact same results. As such no image of such is included. But we think the reason why it might be a bit blurry is due to the small file size of the image. For our purposes of simply applying mean shift this is fine, however a more professional setting would need to fine tune this to better segment up the image. In the second image in particular I would be concerned it detects the car as ground. However. It is a different color from all the things around it and would likely end up as such with even improved settings due to the base color it has. Overall, this could serve it purpose rather well provided you color detect the image to check that it is not a perfect match to the car’s value based on time of day and other factors.

# Challenges

The main challenge that has occurred besides what was already described in the mean shift section, was simply not having the skit library installed and forgetting how to install libraries. Besides this the OTSU based stuff was rather easy to find data and resources on and to implement them. As for mean shift while it did have a struggle where I couldn’t do it to start with, by the help of my fellow classmate we were able to get the one line piece of code that we were missing in order to pull off some form of mean shift.

# Conclusion

In conclusion, the OTSU method is rather useful and easy to implement into working for object detection in an image baseline. OTSU\_2 seems to be the most useful of the data points in my opinion as it is cleanly able divide images into appropriate sections. OTSU\_MULTI is also rather effective though it may be able to pick up additional data that you were not expecting it too. Meanwhile applying the Mean Shift method of segmentation is rather challenging as it seems to not be built for this type of use.

# References -

The following Sources were used within this document to help with making the code and try to investigate our mean shift error

Anastasia Murzova Sakshi Seth, et al. “Otsu's Thresholding Technique.” *LearnOpenCV*, 5 May 2021, https://learnopencv.com/otsu-thresholding-with-opencv/.

“General Examples¶.” *General Examples - Skimage v0.19.2 Docs*, https://scikit-image.org/docs/stable/auto\_examples/index.html.

Hissaan AliHissaan Ali 1, et al. “How to Find Clusters in Image Using Mean Shift in Python Opencv?” *Stack Overflow*, 1 May 1968, https://stackoverflow.com/questions/62575894/how-to-find-clusters-in-image-using-mean-shift-in-python-opencv.

“Multi-Otsu Thresholding¶.” *Multi-Otsu Thresholding - Skimage v0.19.2 Docs*, https://scikit-image.org/docs/stable/auto\_examples/segmentation/plot\_multiotsu.html.

“Scikit-Learn.” *PyPI*, https://pypi.org/project/scikit-learn/.

Zero PiraeusZero Piraeus 51.2k2626 gold badges143143 silver badges152152 bronze badges, et al. “Typeerror: Method() Takes 1 Positional Argument but 2 Were Given.” *Stack Overflow*, 1 Apr. 1962, https://stackoverflow.com/questions/23944657/typeerror-method-takes-1-positional-argument-but-2-were-given.

Zero PiraeusZero Piraeus 51.2k2626 gold badges143143 silver badges152152 bronze badges, et al. “Typeerror: Method() Takes 1 Positional Argument but 2 Were Given.” *Stack Overflow*, 1 Apr. 1962, https://stackoverflow.com/questions/23944657/typeerror-method-takes-1-positional-argument-but-2-were-given.