Project - 2

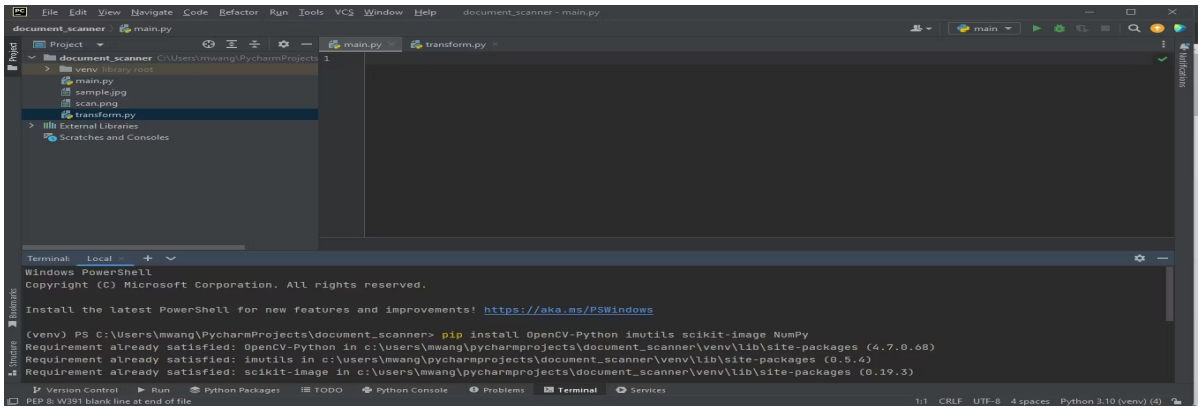
Create a document scanner

PROJECT IMPLEMENTATION

Preparing Your Environment

You will use OpenCV-Python to take the image input and perform some image processing. Imutils to resize the input and output images. scikit-image to apply a threshold on the image. NumPy will help you work with arrays.

In [ ]:
```
pip install OpenCV-Python imutils scikit-image NumPy
```



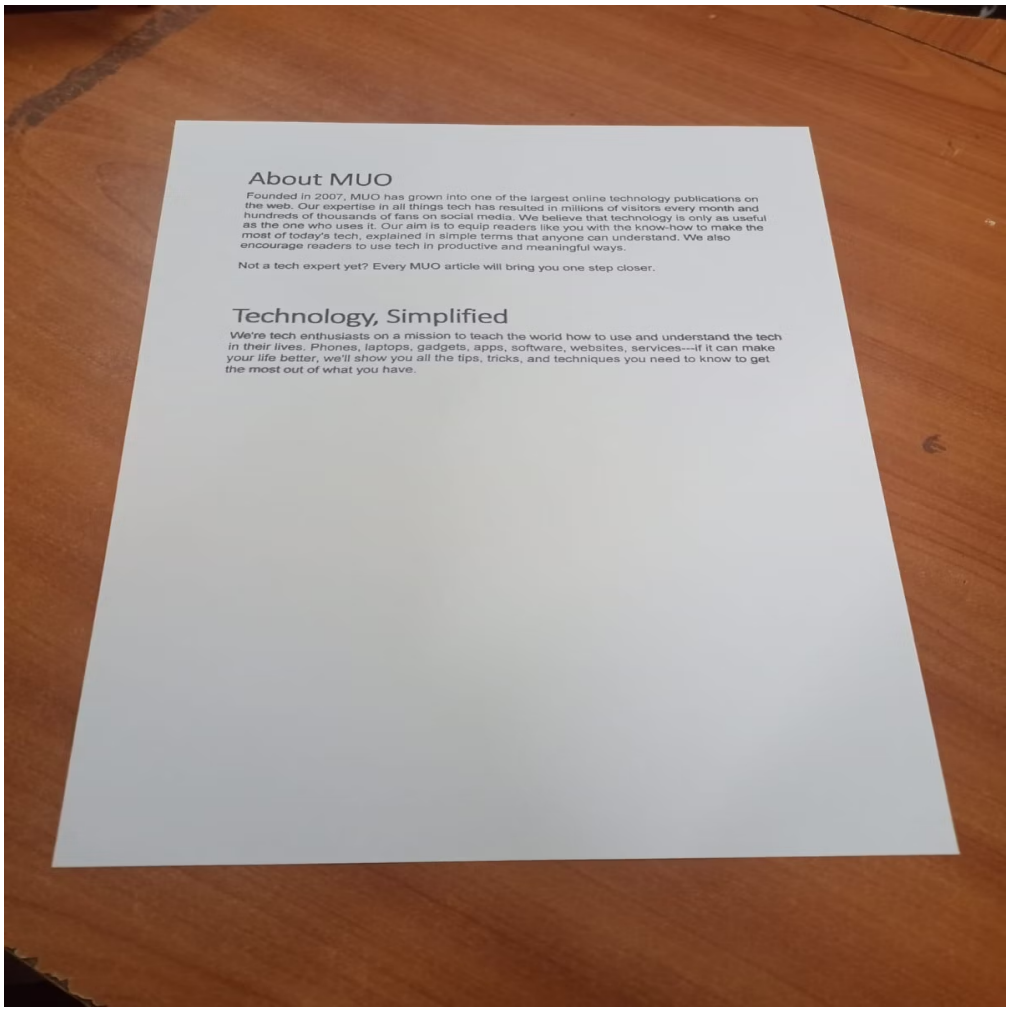
Importing the Installed Libraries

In [ ]:
```
import cv2
import imutils
from skimage.filters import threshold_local
from transform import perspective_transform
```

Taking and Resizing the Input

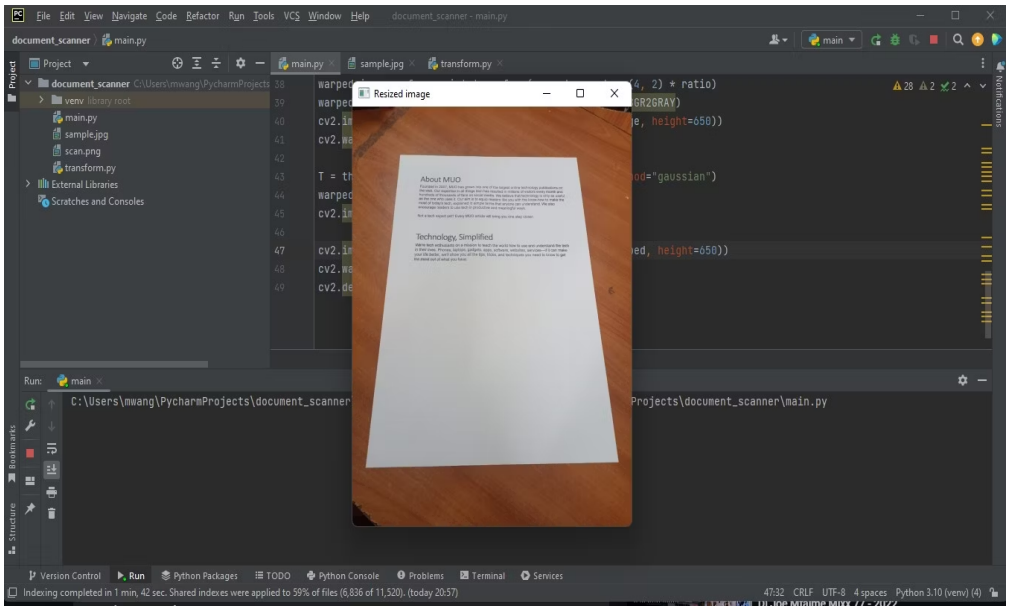
Take a clear image of the document you want to scan. Ensure the four corners of the document and its contents are visible. Copy the image to the same folder you are storing the program files.

Pass the input image path to OpenCV. Make a copy of the original image as you will need it during perspective transformation. Divide the height of the original image by the height you wish to resize it to. This will maintain the aspect ratio. Finally, output the resized image.



In [ ]:

```
# Passing the image path
original_img = cv2.imread('images/sample.jpg')
copy = original_img.copy()

# The resized height in hundreds
ratio = original_img.shape[0] / 500.0
img_resize = imutils.resize(original_img, height=500)

# Displaying output
cv2.imshow('Resized image', img_resize)

# Waiting for the user to press any key
cv2.waitKey(0)
```



You've now resized the height of the original image to 500 pixels.

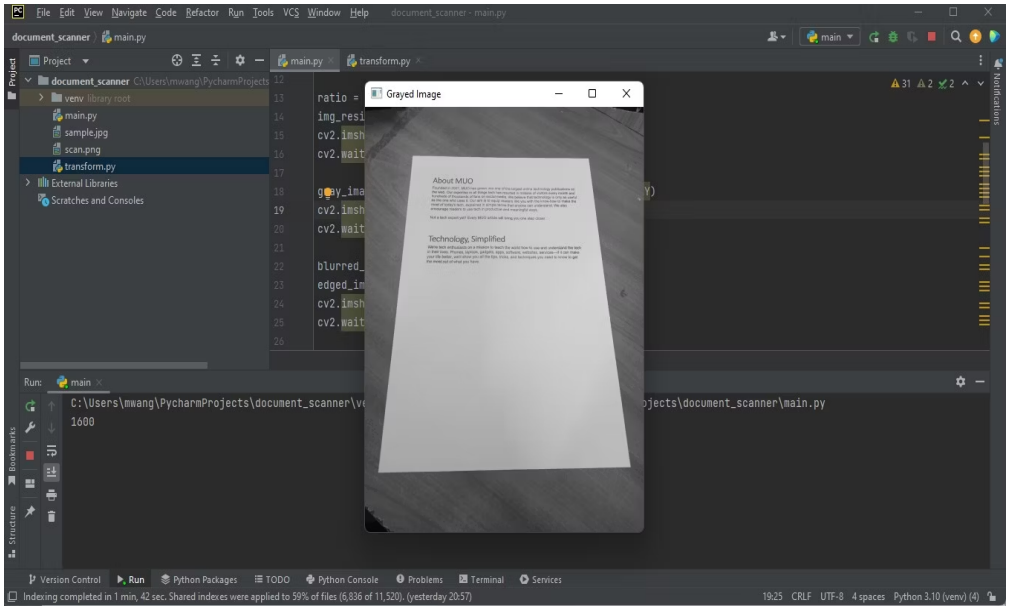
Converting the Resized Image to Grayscale

Convert the resized RGB image to grayscale. Most image-processing libraries only work with grayscale images as they are easier to process.

In [ ]:
```
gray_image = cv2.cvtColor(img_resize, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grayed Image', gray_image)
cv2.waitKey(0)
```

Notice the difference between the original image and the grayed one.

The colored table has turned to black and white.

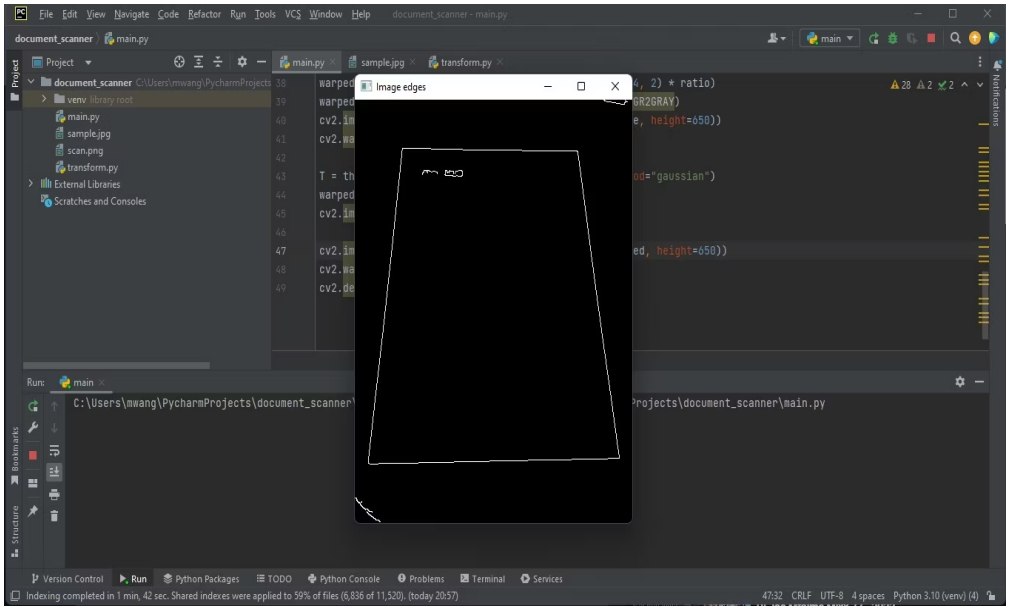


The colored table has turned to black and white.

Applying an Edge Detector

Apply a Gaussian blur filter on the grayed image to remove noise. Then call the OpenCV canny function to detect the edges present in the image.

In [ ]:
```
blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
edged_img = cv2.Canny(blurred_image, 75, 200)
cv2.imshow('Image edges', edged_img)
cv2.waitKey(0)
```



Finding the Largest Contour

Detect the contours present in the edged image. Sort them in descending order keeping only the five largest contours. Approximate the largest contour with four sides by looping through the sorted contours.

The contour with four sides is likely to contain the document.

In [ ]:
```
cnts, _ = cv2.findContours(edged_img, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:5]

for c in cnts:
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)

    if len(approx) == 4:
        doc = approx
        break
```

Circling the Four Corners of the Document Contour

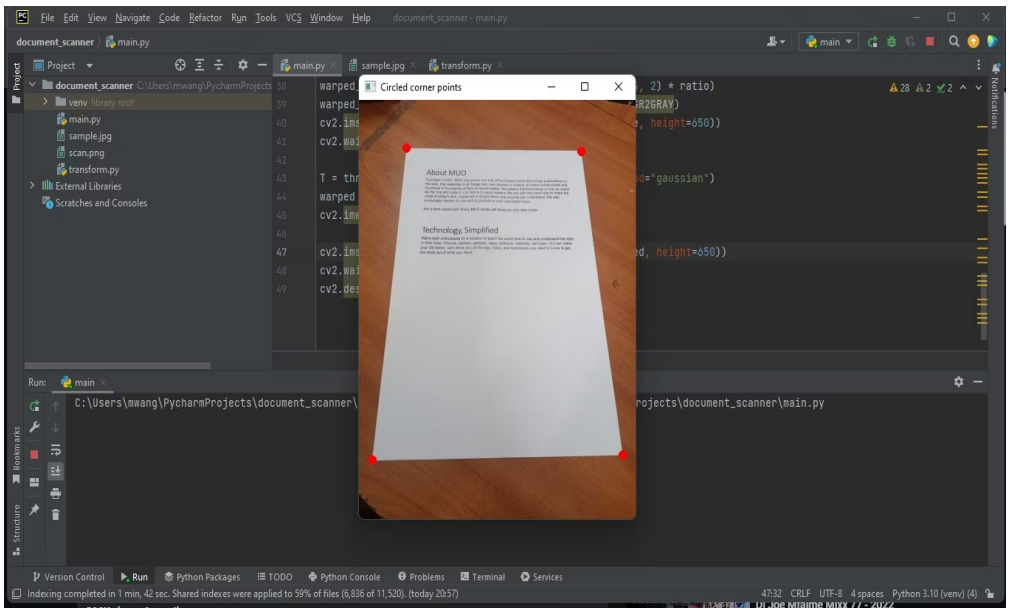
Circle the corners of the detected document contour. This will help you determine whether your program was able to detect the document in the image.

In [ ]:
```
p = []
```

```
for d in doc:
    tuple_point = tuple(d[0])
    cv2.circle(img_resize, tuple_point, 3, (0, 0, 255), 4)
    p.append(tuple_point)

cv2.imshow('Circled corner points', img_resize)
cv2.waitKey(0)
```

Implement circling on the resized RGB image.

Having detected the document, you now need to extract the document from the image.



Using Warp Perspective to Get the Desired Image

Warp perspective is a computer vision technique for transforming an image to correct distortions. It transforms an image into a different plane allowing you to view the image from a different angle.

In [ ]:

```
warped_image = perspective_transform(copy, doc.reshape(4, 2) * ratio)
warped_image = cv2.cvtColor(warped_image, cv2.COLOR_BGR2GRAY)
cv2.imshow("Warped Image", imutils.resize(warped_image, height=650))
cv2.waitKey(0)
```

To obtain a warped image, you need to create a simple module that will perform the perspective transformation.

Transformation Module

The module will order the points of the document corners. It will also transform the document image into a different plane and change the camera angle to an overhead shot.

Open the transform.py file you created earlier. Import OpenCV and NumPy libraries.

In [ ]:

```
import numpy as np
import cv2
```

This module will contain two functions. Create a function that will order the coordinates of the document corner points. The first coordinate will be that of the top left corner, the second will be that of the top right corner, the third will be of the bottom right corner, and the fourth coordinate will be that of the bottom left corner.

In [ ]:
```
def order_points(pts):
   # initializing the list of coordinates to be ordered
   rect = np.zeros((4, 2), dtype = "float32")

   s = pts.sum(axis = 1)

   # top-left point will have the smallest sum
   rect[0] = pts[np.argmin(s)]

   # bottom-right point will have the largest sum
   rect[2] = pts[np.argmax(s)]

   '''computing the difference between the points, the
   top-right point will have the smallest difference,
   whereas the bottom-left will have the largest difference'''
   diff = np.diff(pts, axis = 1)
   rect[1] = pts[np.argmin(diff)]
   rect[3] = pts[np.argmax(diff)]

   # returns ordered coordinates
   return rect
```

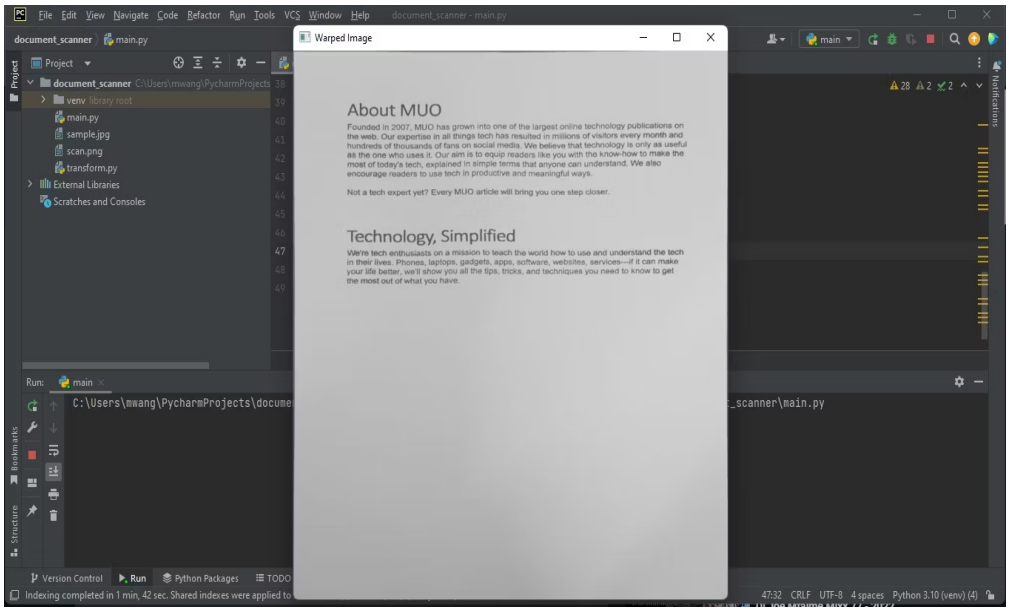
Create a second function that will compute the corner coordinates of the new image and obtain an overhead shot. It will then calculate the perspective transform matrix and return the warped image.

In [ ]:
```
def perspective_transform(image, pts):
   # unpack the ordered coordinates individually
   rect = order_points(pts)
   (tl, tr, br, bl) = rect

   '''compute the width of the new image, which will be the
   maximum distance between bottom-right and bottom-left
   x-coordinates or the top-right and top-left x-coordinates'''
   widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
   widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
   maxWidth = max(int(widthA), int(widthB))

   '''compute the height of the new image, which will be the
   maximum distance between the top-left and bottom-left y-coordinates'''
   heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
   heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
   maxHeight = max(int(heightA), int(heightB))

   '''construct the set of destination points to obtain an overhead shot'''
   dst = np.array([
      [0, 0],
      [maxWidth - 1, 0],
      [maxWidth - 1, maxHeight - 1],
      [0, maxHeight - 1]], dtype = "float32")

   # compute the perspective transform matrix
   transform_matrix = cv2.getPerspectiveTransform(rect, dst)
```

```
    # Apply the transform matrix
    warped = cv2.warpPerspective(image, transform_matrix, (maxWidth, maxHeight))

    # return the warped image
    return warped
```

You have now created the transform module. The error on the perspective_transform import will now disappear.

Notice that the image displayed has an overhead shot.



Applying Adaptive Threshold and Saving the Scanned Output

In the main.py file, apply the Gaussian threshold to the warped image. This will give the warped image a scanned look. Save the scanned image output to the folder containing the program files.

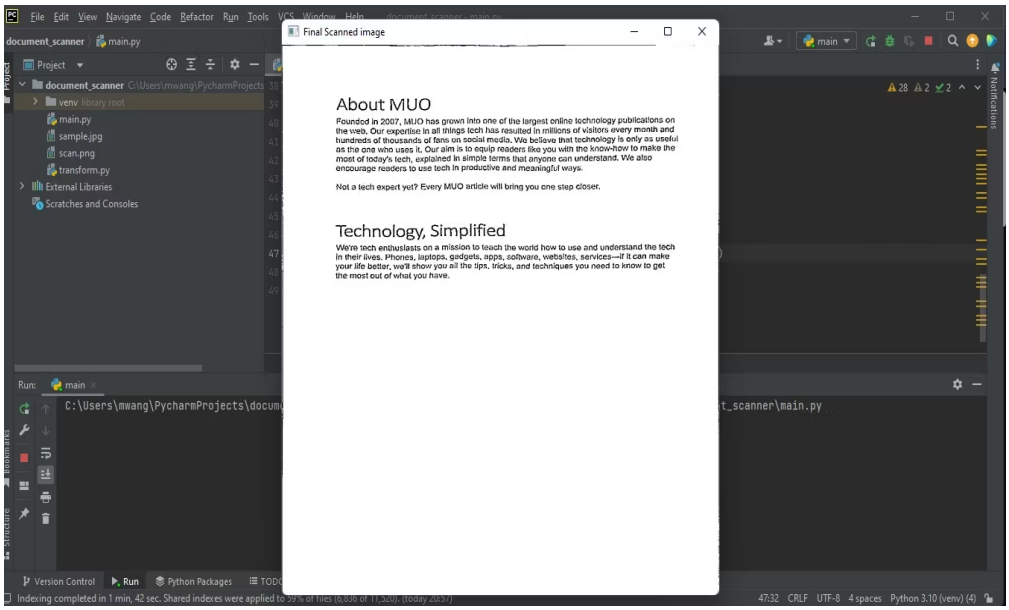
Saving the scan in PNG format maintains the document quality.

In [ ]:
```
T = threshold_local(warped_image, 11, offset=10, method="gaussian")
warped = (warped_image > T).astype("uint8") * 255
cv2.imwrite('./'+'scan'+'.png',warped)
```

Displaying the Output

Output the image of the scanned document:

In [ ]:
```
cv2.imshow("Final Scanned image", imutils.resize(warped, height=650))
cv2.waitKey(0)
cv2.destroyAllWindows()
```



The following image shows the output of the program, an overhead shot of the scanned document.