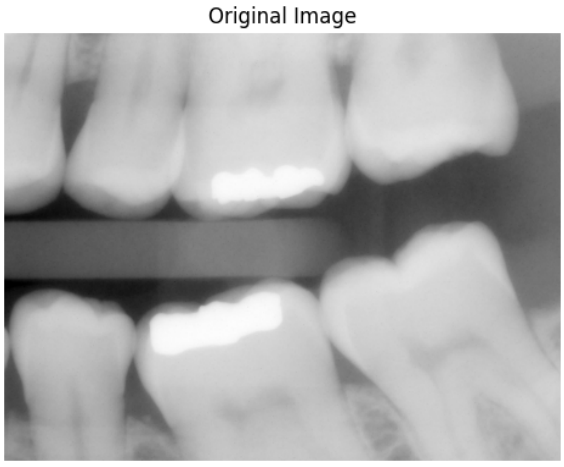
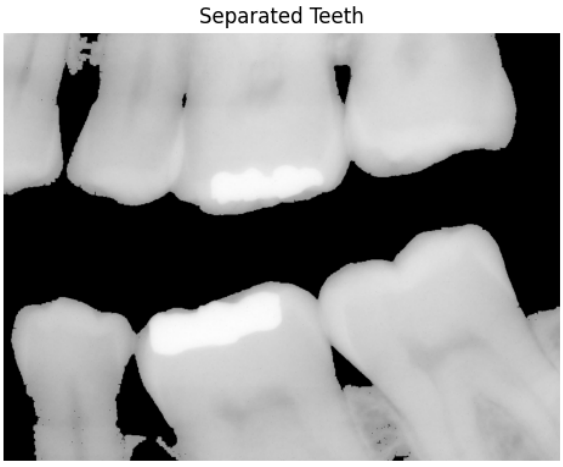
Question# 1

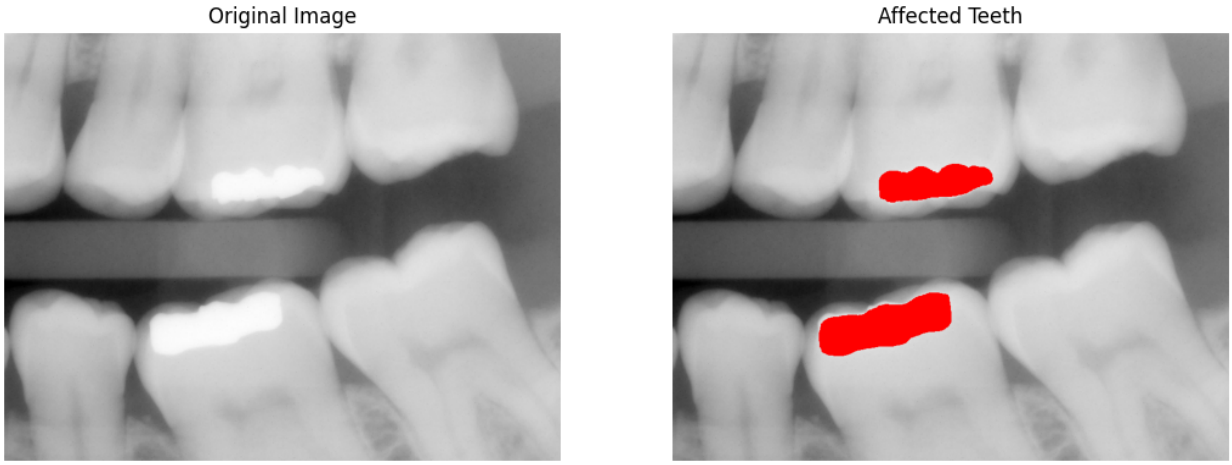
a) Apply thresholding to the grayscale image to create a binary image using OpenCV's threshold function. This step converts the grayscale image to a black and white image with clear boundaries.
Use morphological opening to remove small objects from the binary image. This step removes any small white areas that may not be teeth using OpenCV's morphologyEx function.
Find the two largest connected components in the binary image using OpenCV's connectedComponentsWithStats function. This step identifies the two largest objects in the image, which should correspond to the upper and lower sets of teeth.
Create a mask using the two largest connected components. This step creates a binary image where only the two largest objects are white and the rest of the image is black.
Apply the mask to the original image to isolate the teeth using OpenCV's bitwise_and function.
Set the rest of the image to black using NumPy's zeros_like function. This step sets all the pixels outside of the mask to black, so only the teeth remain visible.
Combine the masked image and black image to get the final result.
Display the original image and the resulting image side by side using Matplotlib's subplots function.
The resulting image shows only the teeth in the original x-ray image, with the rest of the image removed.





b) code loads an image of a dental x-ray, converts it to grayscale, and then applies thresholding to obtain a binary mask of the region of interest. The mask is then dilated to include neighboring pixels in the region of interest.The mask is then converted to 3 channels to use it as a color mask, and all the pixels in the original image are set to red

color where the mask is white. The resulting image is displayed along with the original image, and the affected teeth image is saved as a png file in the specified directory.Overall, this code is designed to highlight the affected teeth in the dental x-ray image by creating a binary mask of the region of interest and then applying it as a color mask to the original image.

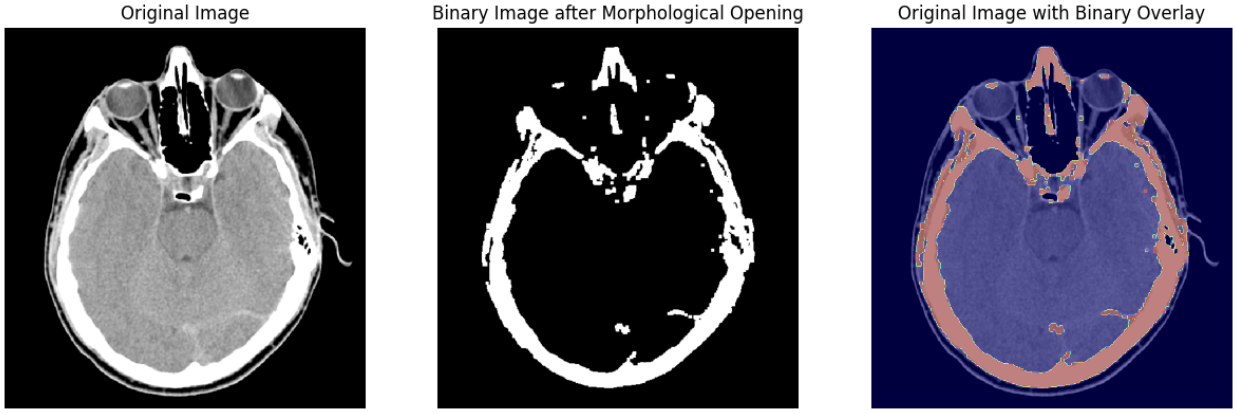


c) function find_red_pixels_percentage that takes the path of an image file as input and returns the percentage of red pixels in the image.
Here is what the function does in detail:
loads the image from the specified file path using cv2.imread.
It converts the color space of the image from BGR to RGB using cv2.cvtColor.
defines the lower and upper bounds of the red color using numpy arrays.
creates a binary mask that identifies the red pixels in the image using cv2.inRange.
calculates the total number of pixels in the image by multiplying the height and width of the image.
counts the number of red pixels in the image using np.count_nonzero.
Calculates the percentage of red pixels in the image by dividing the number of red pixels by the total number of pixels and multiplying by 100.
returns the percentage of red pixels in the image.
The last two lines of code call the function on an image file and print out the result.

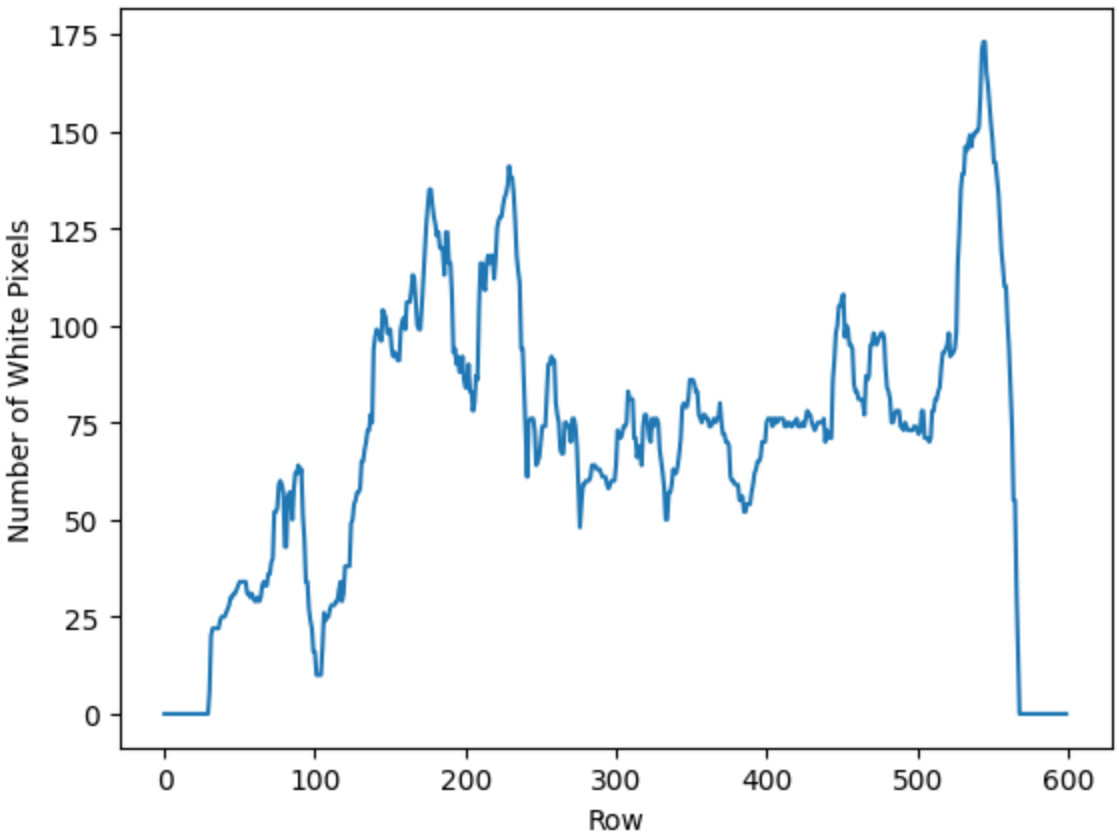
```
Percentage of red pixels in the image: 0.36620978757477274
```

Question# 2

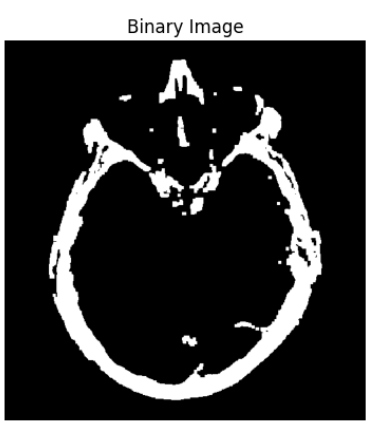
a) It first loads an image in BGR format using the cv2.imread() function, and then converts it to grayscale using cv2.cvtColor(). Next, it sets low and high threshold values for pixel intensities, and applies these thresholds using cv2.threshold() to create a binary image. Morphological opening is applied to the binary image using cv2.morphologyEx() to remove noise. Finally, the original image, binary image after morphological opening, and the original image with binary overlay are displayed using matplotlib, and the binary image after morphological opening is saved using cv2.imwrite(). The input image used in this example is a brain scan, and the output images show the parts of the brain that have pixel intensities above the threshold values as white pixels in the binary image, and as red overlay on the original image.

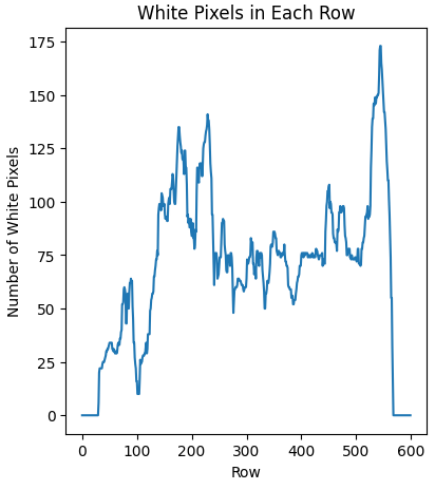


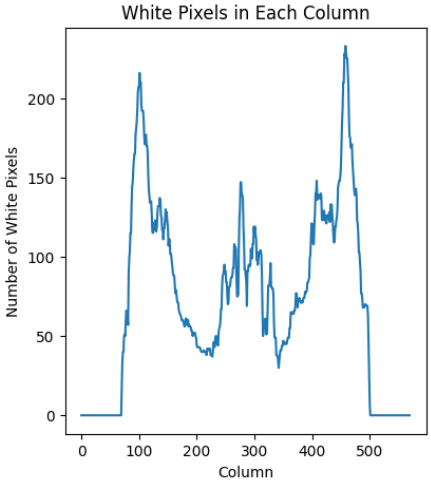
b) defines a function intensity_slicing that takes an image file path as input and performs intensity slicing on it to extract features from the image. The input image is loaded using the OpenCV library, then converted to grayscale. A binary image is created by applying a threshold to the grayscale image, which is then subjected to morphological opening to remove noise. The number of white pixels in each row of the resulting binary image is then counted and plotted as a graph using matplotlib. The purpose of this function is to extract features from the image that can be used for further analysis or processing. The function is then tested on the binary image generated in the previous code block.



c) defines a function intensity_slicing which takes an image path as input, reads an image in BGR format from the path, converts it to grayscale, applies thresholding to create a binary image, removes noise using morphological opening, counts the number of white pixels in each row and column of the binary image, and plots the binary image along with the number of white pixels in each row and column. The function uses the OpenCV library to perform various image processing operations, and the Matplotlib library to display the images and plots. The intensity_slicing function is called with the path of a binary image file, and the resulting binary image and the number of white pixels in each row and column are plotted in a 3-panel figure.





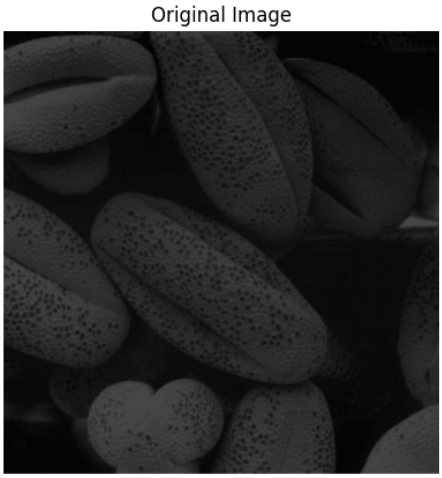


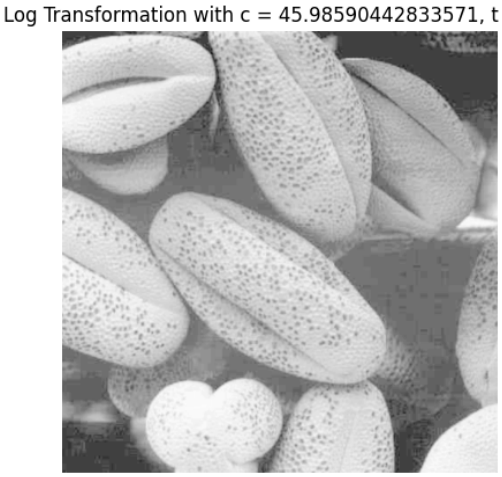
Question# 3

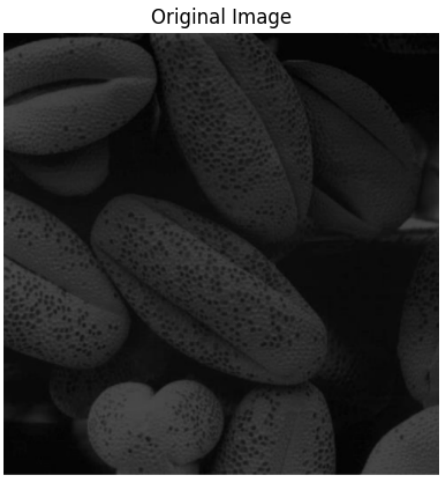
demonstrates the effect of different logarithmic transformations on an input grayscale image. The input image 'grain.png' is loaded and a scaling factor 'c' is calculated using the maximum pixel value of the input image. Then, the logarithmic transformation is applied to the input image with different transformation parameters stored in the 'transformations' list.

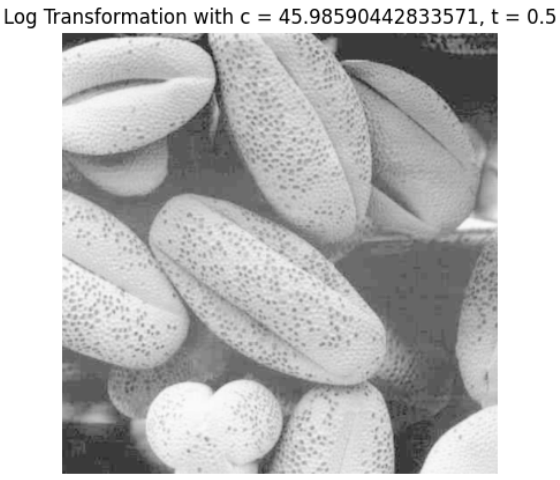
For each transformation parameter value in the list, the logarithmic transformation is applied to the input image, and the resulting transformed image is displayed alongside the original image using matplotlib. The parameters used for the transformation are printed as part of the title of the output image.

The logarithmic transformation is a type of intensity transformation that is often used to expand the dynamic range of an image. It compresses the higher intensity values of the image while expanding the lower intensity values, resulting in an image with more visible details in the darker regions.



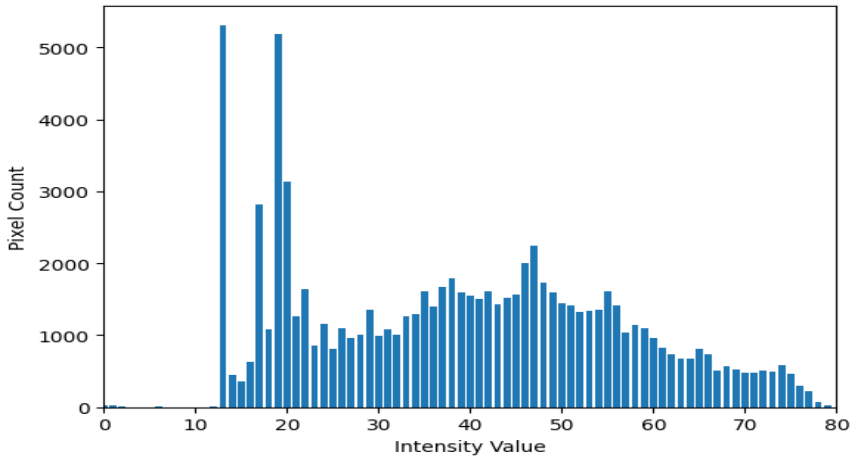




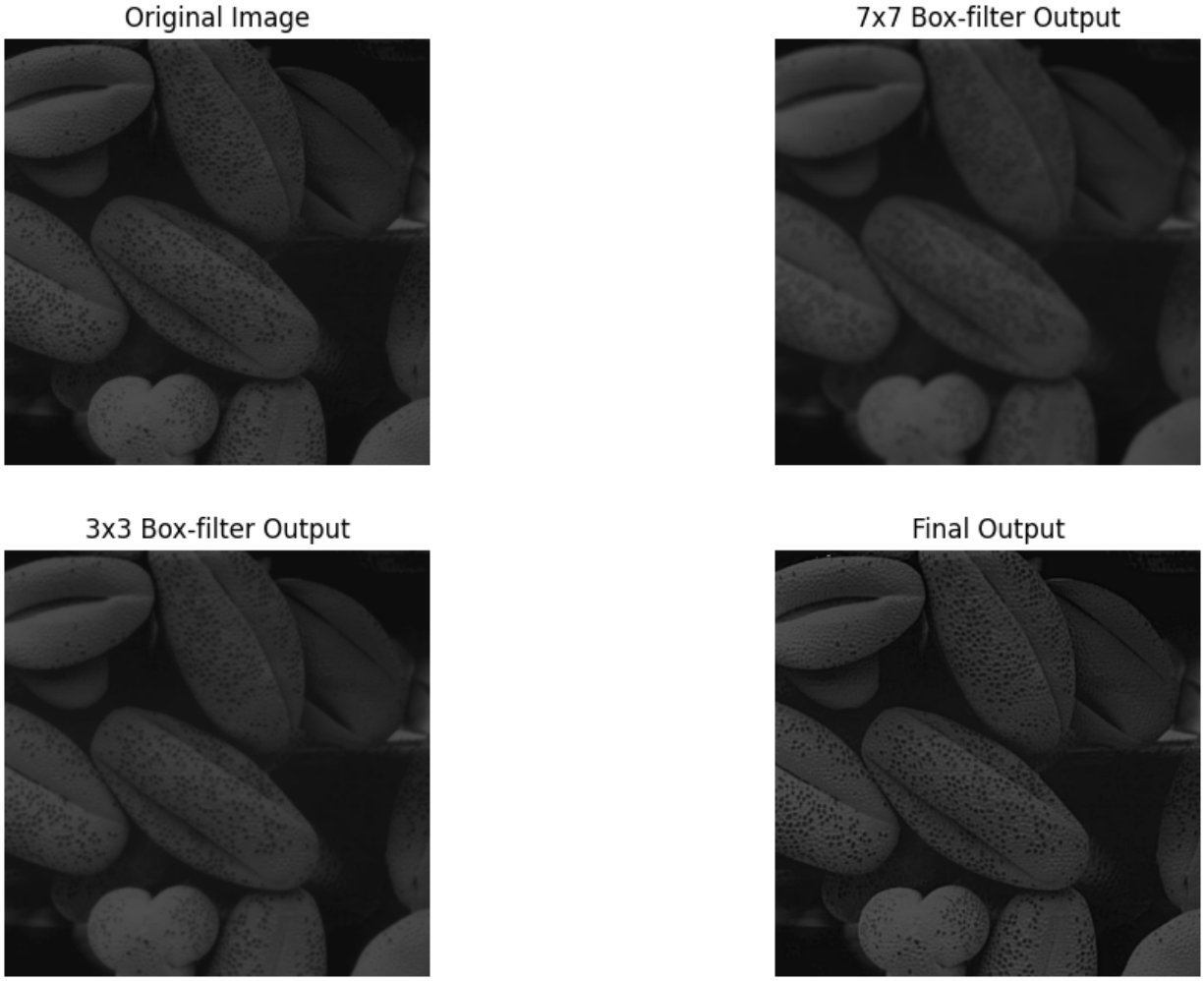


Question# 4

a) code defines a function count_pixel_intensities that takes an image file path as input and returns a list of intensity counts for each possible intensity value (0-255) in the image. The function first reads the header of the image file to extract the width and height of the image and calculates the total number of pixels in the image. It then reads the pixel data from the image file into a buffer and loops over each pixel in the buffer to extract its RGB values, convert them to grayscale intensity, and increment the count for the corresponding intensity value.
The code then calls the count_pixel_intensities function on a specific image file "data/grain3.tif" and assigns the returned list of intensity counts to the variable intensity_counts. Finally, the code plots the intensity counts as a bar graph using matplotlib. The plot is limited to intensity values between 0 and 80 using plt.xlim(0, 80) to make it easier to see the details of the plot.

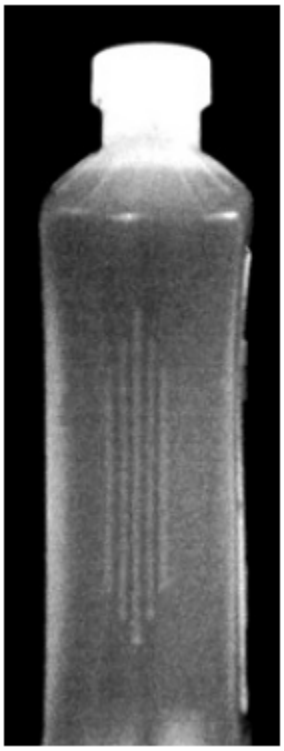


b) code applies a filter to an input image 'grain3.tif' using box filters of sizes 7x7 and 3x3. The 7x7 and 3x3 filtered outputs are then subtracted from each other to compute the final output. The final output image is displayed along with the original image and the filtered outputs using Matplotlib. The 'cv2' module of OpenCV is used to read the image and apply the box filters. The 'plt' module of Matplotlib is used to display the images. The resulting images are displayed in a 2x2 grid using subplots, with the original image in the top left, the 7x7 filtered output in the top right, the 3x3 filtered output in the bottom left, and the final output in the bottom right. The 'cvtColor' function of OpenCV is used to convert the color format of the images from BGR to RGB before displaying them using Matplotlib.



c) defines a function calculate_filled_percentage that takes as input the path to a grayscale image of a bottle and a threshold value, and calculates the percentage of the bottle that is filled with syrup.
The function first reads in the grayscale image using OpenCV's imread function, and then applies a binary threshold to obtain a binary image of the bottle using the cv2.threshold function. The function then applies the cv2.bitwise_not function to the binary image to obtain a binary image of the syrup.

The function then uses the cv2.countNonZero function to count the number of white pixels (i.e., non-zero values) in the syrup image, and the total number of pixels in the bottle image is calculated by multiplying the number of rows and columns of the image array.
Finally, the filled percentage is calculated as the ratio of the number of syrup pixels to the total number of bottle pixels, and the result is printed to the console with a message indicating whether the bottle is properly filled or not based on the input threshold value.
After defining the function, the code calls it with the path to an example image data/bottle2.jpg and a threshold value of 90%. The code then displays the original image using matplotlib's imshow function, with the axis turned off using the axis('off') method.



```
The bottle is 92.97% filled.
Bottle is properly filled.
```