IMAGE PROCESSING USING K MEANS CLUSTERING AND EUCLIDEAN DISTANCE METHOD

Rahul Parihar[1], Satvik sawhney[2], Arti Vaish3, Sherry Verma[4]
E-Mail Id: rahulparihar.btech18@sushantuniversity.edu.in[1], satviksawhney.btech18@sushantuniversity.edu.in[2], artivaish@sushantuniversity.edu.in[3], sherryverma@sushantuniversity.edu.in[4]
Department of Computer Science & Engineering, Sushant University Gurugram, Haryana, India

Abstract- Image processing refers to the processing of images, which includes many different techniques that are used till we reach our goal. The processing output can either be in the form of an image, or a feature of the given image. This can be used for decision making and a more comprehensive image analysis. The Image compression process is a variation of data compression that is used on images without affecting their quality. Reduced size of the file makes sure that a lot of images can be put in a certain amount of memory / disk space. Processing also decreases the required time for the images to travel over the Internet or to be downloaded from different web pages. K - means clustering using Euclidean distance method includes vector quantization, which is originally a part of signal processing. It is very popular in cluster analysis for mining data. In this Project, we are using Image compression and segmentation algorithms using K – means clustering and Euclidean Distance method and many more algorithms used to process images.

Keywords: Image, processing, compression, k-means, decision making, clustering, Euclidian distance, image quality, image size.

1. LITERATURE REVIEW

In this part of the research, we will review previous literature working on image compression. Generally, images are compressed using multiple techniques. For example, Vector Quantization (VQ) and K-Means Clustering are commonly used to apply image compression. In research [6] a new algorithm (IDE-LBG) for generating optimum VQ Codebooks to efficiently compress grayscale images is proposed. This algorithm takes less computation time and results an excellent PSNR. The algorithm was tested on 5 different images, at a resolution of 512 x 512 and 6 different Codebook sizes. PSNR was calculated after each different compression. Based on VQ, the researchers presented in [7] a methodology for image compression. The methodology was tested on different image resolutions with a different Block Size. The MSE, PSNR (dB), and CR standards are used in this paper as performance measures. In [8] a scheme was presented to compress images with K-means clustering. The energy efficiency of the sensors was tested when sending data after applying a compression process.

Energy consumption has been reduced by approximately 49% when sending images by the sensors. PSNR, MSE, SSIM was calculated as performance measures in this paper. Research [9] is concerned with medical images and trying to reduce the size of them to the lowest degree while preserving the quality of the images, as they are used in diagnosis. DWT-VQ (Discrete Wavelet Transform - Vector Quantization) technique is proposed for image compression. In the first 276 IJCSNS International Journal of Computer Science and Network Security, VOL.21 No.9, September 2021 stage a preprocessing operation is used to remove the speckle and salt and pepper noises in ultrasound imaging. Then the proposed technique is applied to the image. In [10], the Singular Value Decomposition (SVD) technique for image compression and how to apply it is studied. This technique is based on dividing the image matrix into a number of linearly independent matrices. The researcher used MATLAB to implement an image compression algorithm based on Singular Value Decomposition (SVD). Compression of medical images was the focus of the researchers' attention at [11]. Discrete Cosine Transform was used to compress images as was done by means of exploiting spectra similarity. Image quality evaluation is important in the case of image compression. There are several literatures that have established standards for image quality. For example, the SSIM scale was discussed in [12], which is used to measure the degree of similarity between two images. Also, there are two important metrics for evaluating image quality, MSE and PSNR [13]. [1]

There have been many works done in the area of image segmentation by using different methods. And many are done based on different application of image segmentation. K-means algorithm is the one of the simplest clustering algorithm and there are many methods implemented so far with different method to initialize the centre. And many researchers are also trying to produce new methods which are more efficient than the existing methods, and shows better segmented result. Some of the existing recent works are discussed here. Pallavi Purohit and Ritesh Joshi4 introduced a new efficient approach towards K-means clustering algorithm. They proposed a new method for generating the cluster center by reducing the mean square error of the final cluster without large increment in the execution time. It reduced the means square error without sacrificing the execution time. Many comparisons have been done and it can conclude that accuracy is more for dense dataset rather than sparse dataset. Alan Jose, S. Ravi and M. Sambath5 proposed Brain Tumor Segmentation using K-means Clustering and Fuzzy C-means Algorithm and its area calculation. In the paper, they divide the process into three parts, pre-processing of the image, advanced k-means and fuzzy c-means and lastly the feature extraction. First pre-processing is implemented

by using the filter where it improves the quality of the image. Then the proposed advance K-means algorithm is used, followed by Fuzzy c-means to cluster the image. Then the resulted segment image is used for the feature extraction for the region of interest. They used MRI image for the analysis and calculate the size of the extracted tumor region in the image. Madhu Yedla, Srinivasa Rao Pathakota, T. M. Srinivasa6 proposed Enhancing K-means clustering algorithm with improved initial center. A new method for finding the initial centroid is introduced and it provides an effective way of assigning the data points to suitable clusters with reduced time complexity. They proved their proposed algorithm has more accuracy with less computational time comparatively original k-means clustering algorithm. This algorithm does not require any additional input like threshold value. But this algorithm still initializes the number of cluster k and suggested determination of value of k as one of the future work. K. A. Abdul Nazeer, M. P. Sebastian7 proposed an enhanced algorithm to improve the accuracy and efficiency of the k-means clustering algorithm.
They present an enhanced k-means algorithm which combines a systematic method consisting two approaches. First one is finding the initial centroid and another is assigning the data point to the clusters. They have taken different initial centroid and tested execution time and accuracy. From the result it can be conclude that the proposed algorithm reduced the time complexity without sacrificing the accuracy of clusters. [2]

2. K – MEANS

K-Means is a non-hierarchical data clustering method. Grouping a data using the K-Means method in general can be done with the following basic algorithm (Aguste, 2007): 1. Determine the number of initial clusters. 2. Placing centroids according to the number of clusters randomly. 3. Get the data that has the closest distance. 4. Allocate data into clusters according to the centroid using the nearest distance calculation. 5. Allocate data into clusters according to the new centroid using the nearest distance calculation. 6. Return to step 4, if there is still moved data from cluster or in the centroid value above the specified threshold value or if the objective function value changes above the threshold value. [3]

3. METHODOLOGY

The internet is full of large quantities of data that is in the form of images and videos. Users upload billions of media every day, on the social media websites like Facebook, Instagram, google drive, etc. With such large amounts of data files, it becomes important to apply data compression techniques on the images in order to reduce their cost of storage and transmission.[4]

4. IMAGE COMPRESSION

This project is designed for compressing images using their pixel values. Each pixel, containing RGB values is of 3 bytes (or 24 bits). Our goal is to reduce the number of bits that each pixel occupies. K-means clustering is an unsupervised learning algorithm. It is also known as the optimization technique used to find the 'k' clusters in the given data point set. The data points clustered together due to some similarity. It starts, initially, with the making of the 'k' random clusters, and after that, its aim is to minimize / reduce distance from cluster center to every data point in each cluster on the basis of some similarity / Euclidean distance metric. [5]
The two main repetitive steps are:

- The Assignment step - Every data point has been assigned to a particular cluster that has the center nearest to it.
- The Update step – The new centers (centroids) are created / calculated from the assigned data points to new clusters by computing average of the data points.

The steps above repeat until the centroids stop moving away from their respective clusters. After this, we will get many different clusters that are separated out because of some differences. In the same way, some data points are grouped together due to similarities.
These algorithms for compression extract the features of original image in order to improve the results. Lossy and lossless are the two types of image compression methods used.

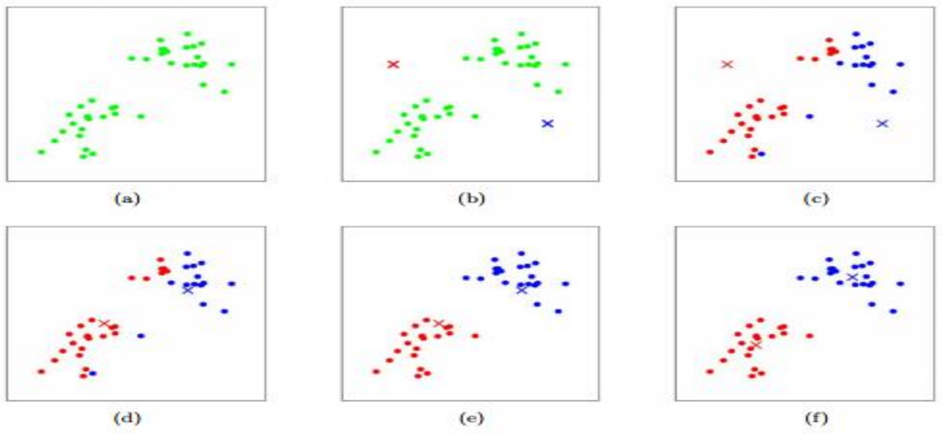


Fig. 4.1 K Means Clustering

4.1 Lossless Compression

This compression is used for reducing size of an image, while keeping the same quality as the original image. Lossless compression doesn't reduce the file quality and the image is able to restore to its original form anytime. [6]The file size is fixed in this compression type.[7] Algorithms used for lossless compression are:

- Run Length encoding
- Arithmetic Coding
- Huffman Coding

4.2 Lossy Compression

The lossy compression is a kind of compression which eliminates data that is not noticeable. In order to give the image an even low size, this compression removes some components of an image that are not as important. The image is unable to restore to its original form and in this compression, the data size changes and the file quality is lowered. [8]This technique is mainly used in video, audio and image compressions. Algorithms used for lossy compression are:

- Discrete Cosine Transform
- Transform Coding
- Fractal compression

Here, we are using K-Means Clustering for compression of images, which is a transformation compression method. We perform quantization of colors of the image using k means clustering, which helps in further compressing the original image.
Using K-means clustering,
K-Means algorithm is a centroid based clustering technique. The dataset is clustered into k different clusters using this technique. A centroid point represents clusters in the clustering algorithm of k means.[9]

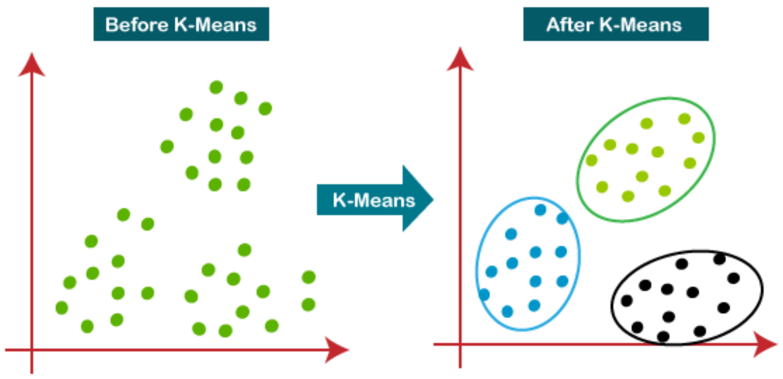


Fig. 4.2 Forming Clusters from a Dataset

5. IMAGE SEGMENTATION

When we use computer vision, the process of partitioning an image into multiple different segments is known as image segmentation. The aim for segmentation of an image is to change the representation of image into something that is more meaningful and is also easy to analyze. This is mostly used for locating the objects and creating some boundaries.
It is not a good idea to process a whole image, since many parts of an image may not contain any information of use. Therefore, we can use only the important segments to process image by segmenting the image.
An image is just a set of pixels, arranged in a meaningful sequence. Pixels which have similar attributes are grouped together in image segmentation. Image segmentation gives us a mask for objects / pixel-wise collection / features inside an image, which gives us a more comprehensive and granular understanding of the object. [10], [11]

6. OBJECTIVES

By the process of image compression, K-means clustering groups the similar RGB colors into clusters of k of different values for RGB (k=16). Thus, the centroid of every cluster is a representation of 3D color vectors in RGB colors in their clusters. After that, the k centroids are going to replace the RGB color vector units from their clusters. Therefore, its only needed to store each pixel label, that lets us know which cluster the pixel belongs to. On top of that, we also keep track of the RGB color units / vectors from every cluster center.
If we take k=16, the final image is going to have 16 different colors in its color space. The compression technique here will be lossy, that is, the intricate details in the image might get vanished post compression. But we can use a relatively high 'k' value to minimize the loss factor and make it as less of an issue as possible. [12]

6.1 Libraries Needed

- NumPy library
- Matplotlib library
- SciPy library

6.2 Image Used

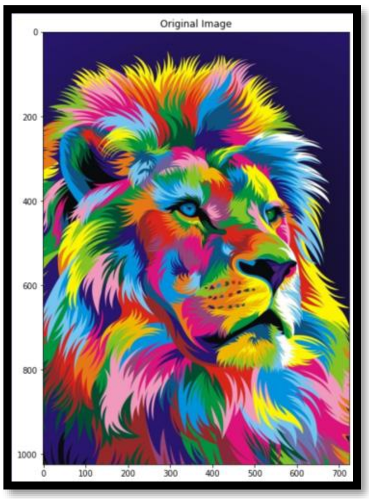


Fig. 6.1 Original Image Used

7. CODE AND IMPLEMENTATION

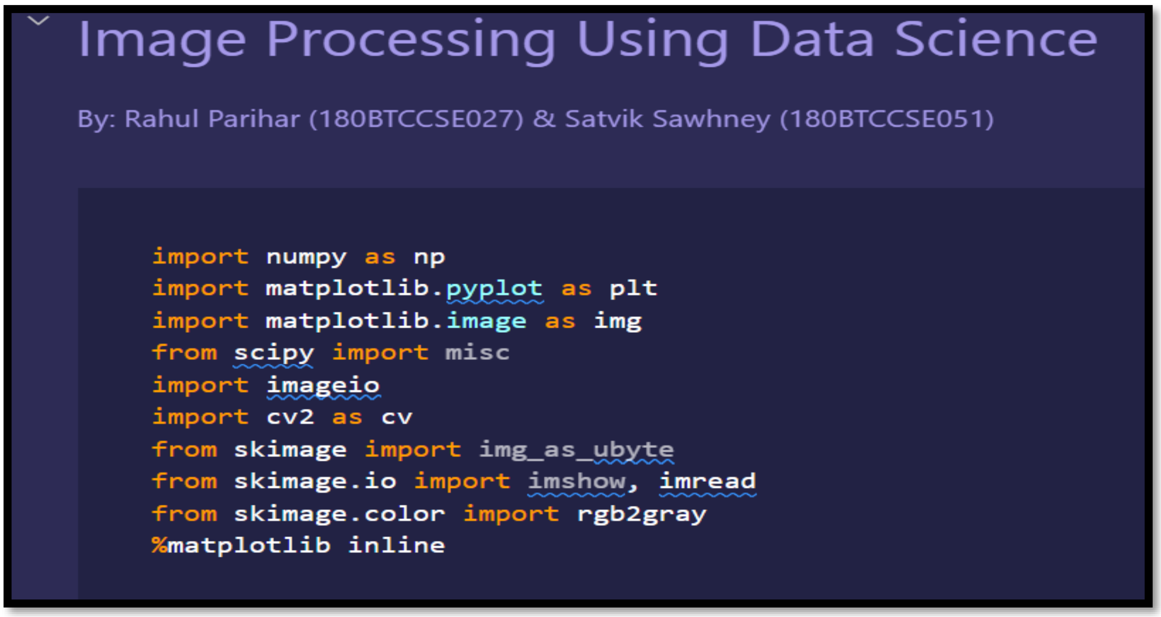
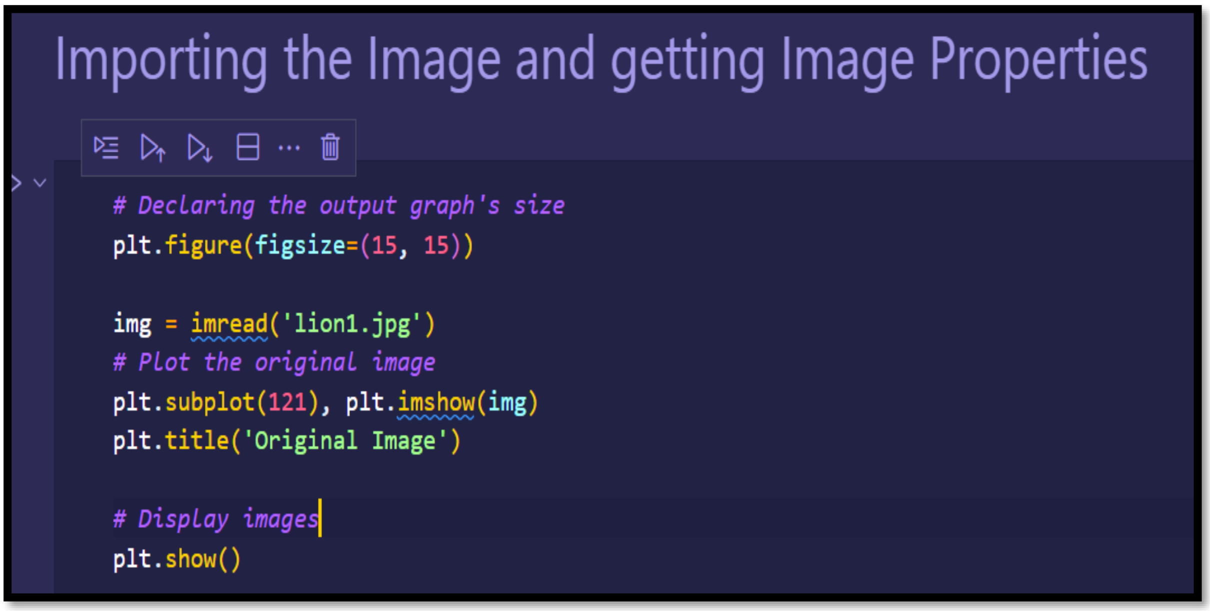


Image Processing Using Data Science

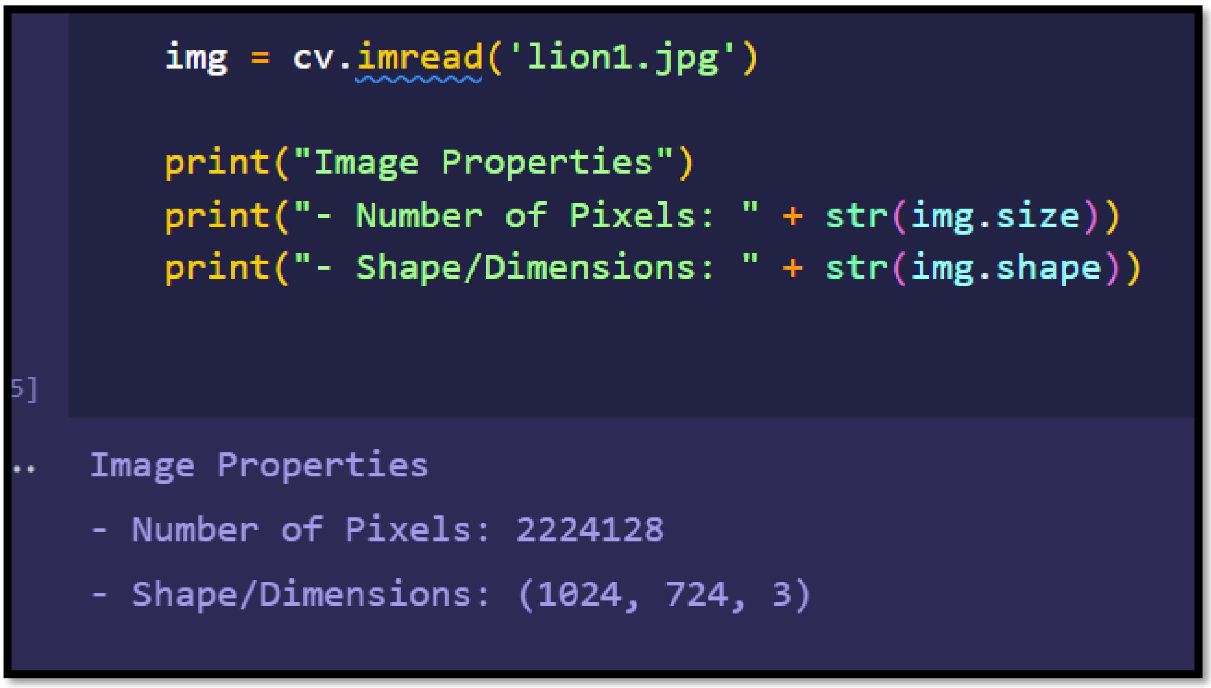
By: Rahul Parihar (180BTCCSE027) & Satvik Sawhney (180BTCCSE051)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
from scipy import misc
import imageio
import cv2 as cv
from skimage import img_as_ubyte
from skimage.io import imshow, imread
from skimage.color import rgb2gray
%matplotlib inline
```



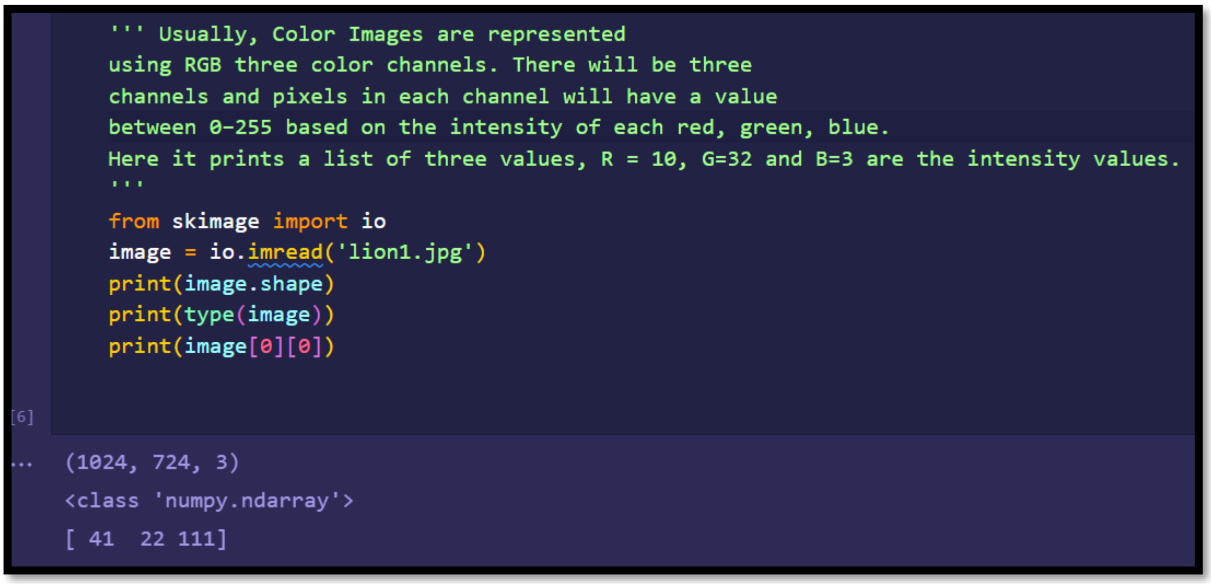
Importing the Image and getting Image Properties

```
# Declaring the output graph's size
plt.figure(figsize=(15, 15))

img = imread('lion1.jpg')
# Plot the original image
plt.subplot(121), plt.imshow(img)
plt.title('Original Image')

# Display images
plt.show()
```



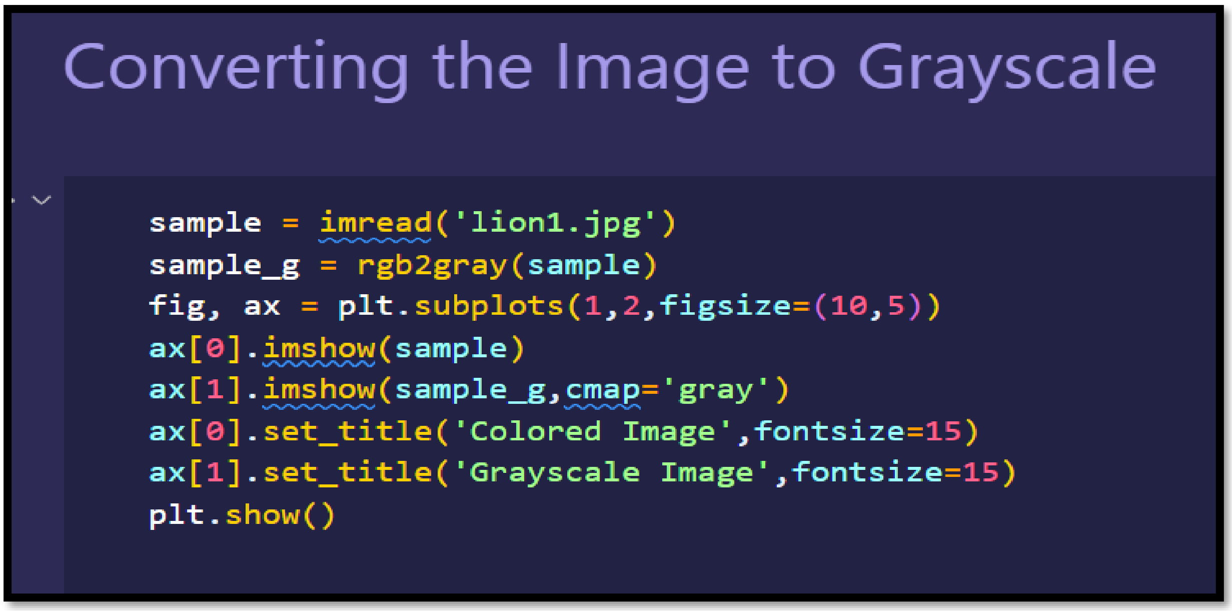
```
img = cv.imread('lion1.jpg')

print("Image Properties")
print("- Number of Pixels: " + str(img.size))
print("- Shape/Dimensions: " + str(img.shape))
```

```
Image Properties
- Number of Pixels: 2224128
- Shape/Dimensions: (1024, 724, 3)
```



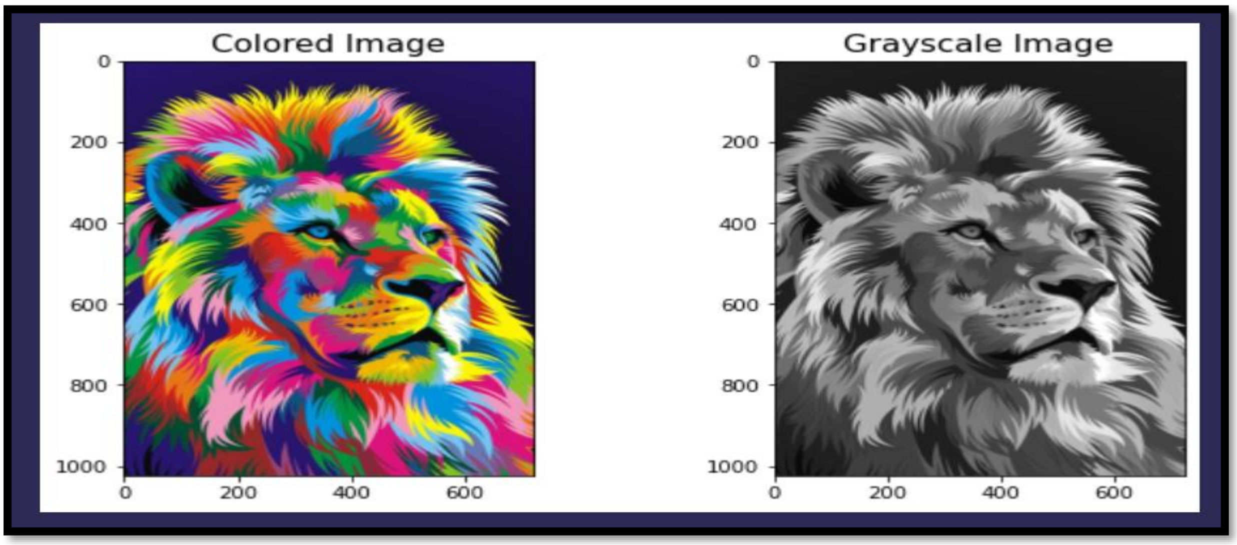
```
''' Usually, Color Images are represented
using RGB three color channels. There will be three
channels and pixels in each channel will have a value
between 0-255 based on the intensity of each red, green, blue.
Here it prints a list of three values, R = 10, G=32 and B=3 are the intensity values.
'''
from skimage import io
image = io.imread('lion1.jpg')
print(image.shape)
print(type(image))
print(image[0][0])
```

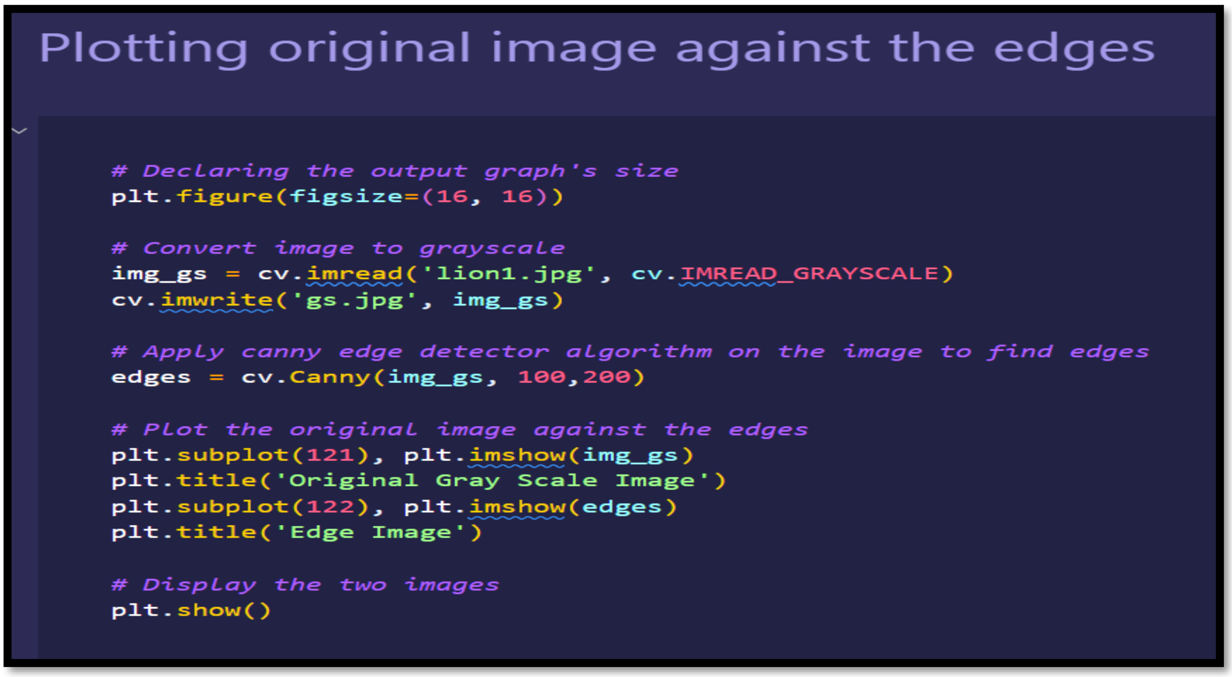
```
(1024, 724, 3)
<class 'numpy.ndarray'>
[ 41  22 111]
```



Converting the Image to Grayscale

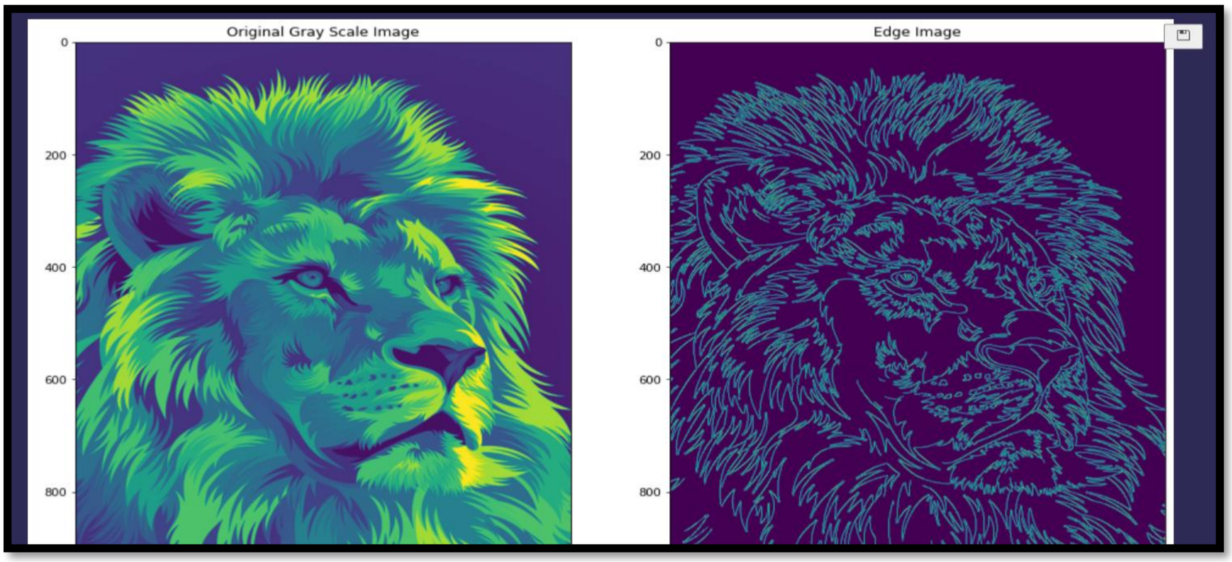
```
sample = imread('lion1.jpg')
sample_g = rgb2gray(sample)
fig, ax = plt.subplots(1,2,figsize=(10,5))
ax[0].imshow(sample)
ax[1].imshow(sample_g,cmap='gray')
ax[0].set_title('Colored Image',fontsize=15)
ax[1].set_title('Grayscale Image',fontsize=15)
plt.show()
```

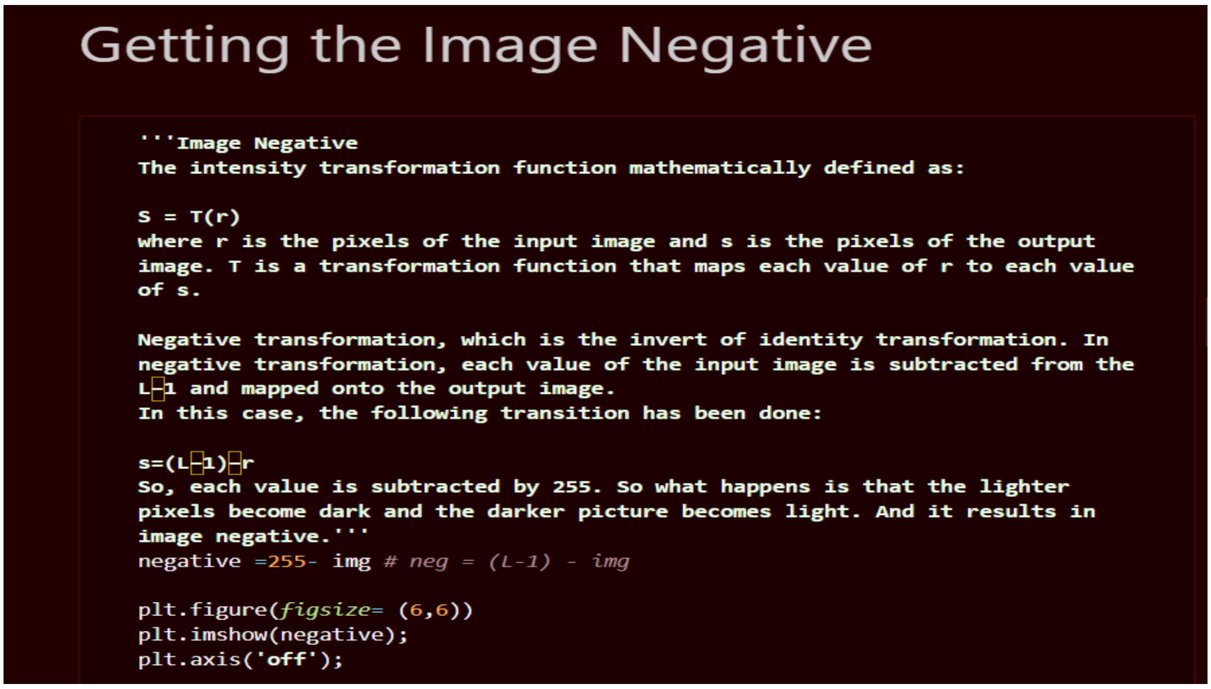




Plotting original image against the edges

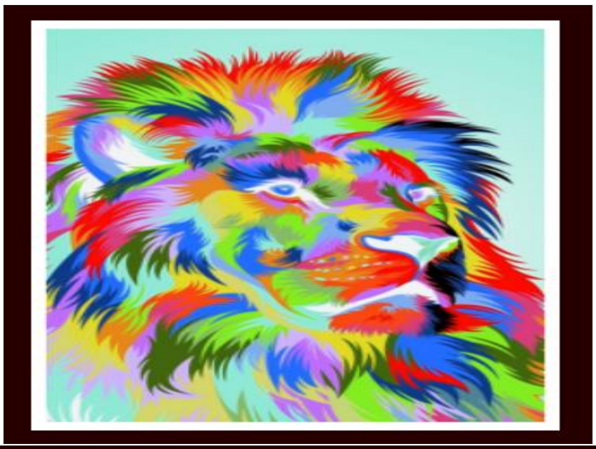
```
# Declaring the output graph's size
plt.figure(figsize=(16, 16))

# Convert image to grayscale
img_gs = cv.imread('lion1.jpg', cv.IMREAD_GRAYSCALE)
cv.imwrite('gs.jpg', img_gs)

# Apply canny edge detector algorithm on the image to find edges
edges = cv.Canny(img_gs, 100,200)

# Plot the original image against the edges
plt.subplot(121), plt.imshow(img_gs)
plt.title('Original Gray Scale Image')
plt.subplot(122), plt.imshow(edges)
plt.title('Edge Image')

# Display the two images
plt.show()
```

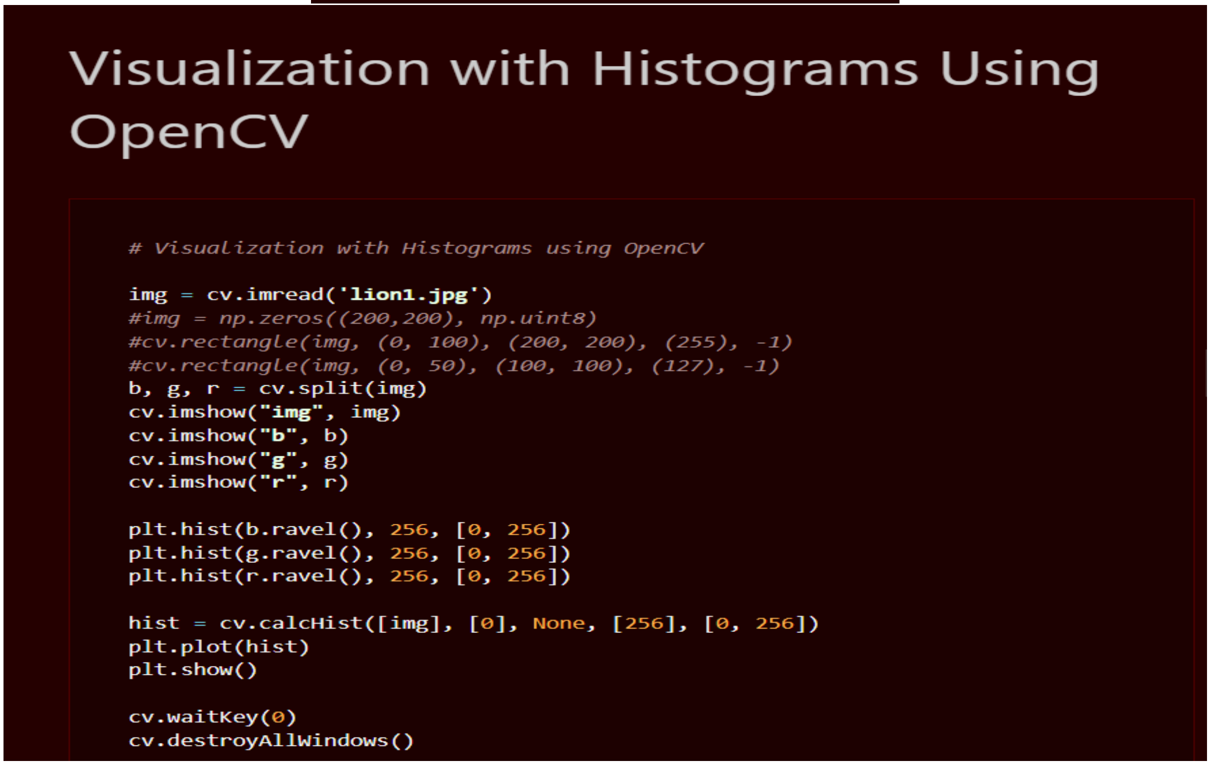




Getting the Image Negative

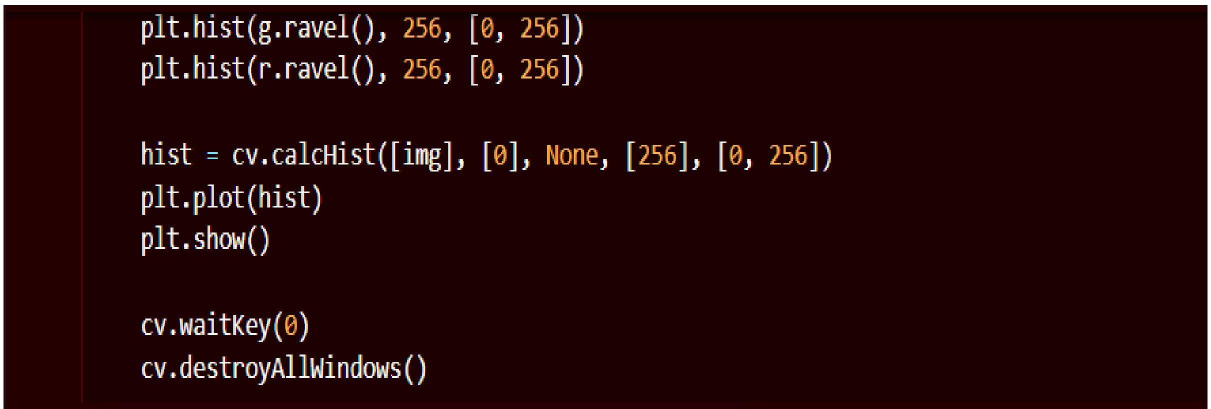
```
'''Image Negative
The intensity transformation function mathematically defined as:

S = T(r)
where r is the pixels of the input image and s is the pixels of the output
image. T is a transformation function that maps each value of r to each value
of s.

Negative transformation, which is the invert of identity transformation. In
negative transformation, each value of the input image is subtracted from the
L□1 and mapped onto the output image.
In this case, the following transition has been done:

s=(L□1)□r
So, each value is subtracted by 255. So what happens is that the lighter
pixels become dark and the darker picture becomes light. And it results in
image negative.'''
negative =255- img # neg = (L-1) - img

plt.figure(figsize= (6,6))
plt.imshow(negative);
plt.axis('off');
```





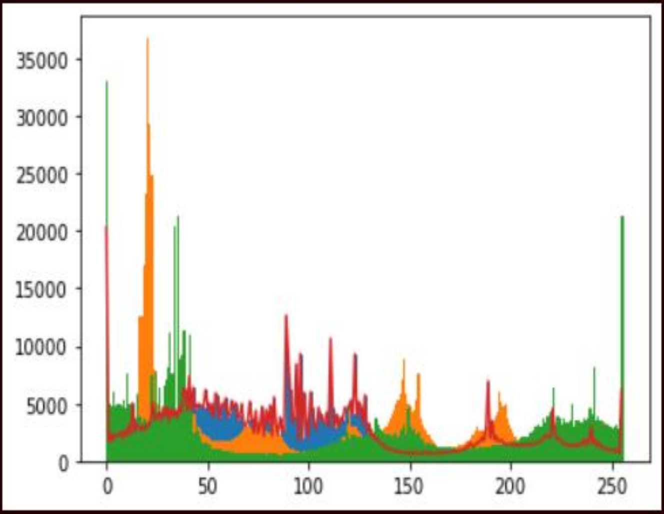
Visualization with Histograms Using OpenCV

```
# Visualization with Histograms using OpenCV

img = cv.imread('lion1.jpg')
#img = np.zeros((200,200), np.uint8)
#cv.rectangle(img, (0, 100), (200, 200), (255), -1)
#cv.rectangle(img, (0, 50), (100, 100), (127), -1)
b, g, r = cv.split(img)
cv.imshow("img", img)
cv.imshow("b", b)
cv.imshow("g", g)
cv.imshow("r", r)

plt.hist(b.ravel(), 256, [0, 256])
plt.hist(g.ravel(), 256, [0, 256])
plt.hist(r.ravel(), 256, [0, 256])

hist = cv.calcHist([img], [0], None, [256], [0, 256])
plt.plot(hist)
plt.show()

cv.waitKey(0)
cv.destroyAllWindows()
```



```
    plt.hist(g.ravel(), 256, [0, 256])
    plt.hist(r.ravel(), 256, [0, 256])

    hist = cv.calcHist([img], [0], None, [256], [0, 256])
    plt.plot(hist)
    plt.show()

    cv.waitKey(0)
    cv.destroyAllWindows()
```

...



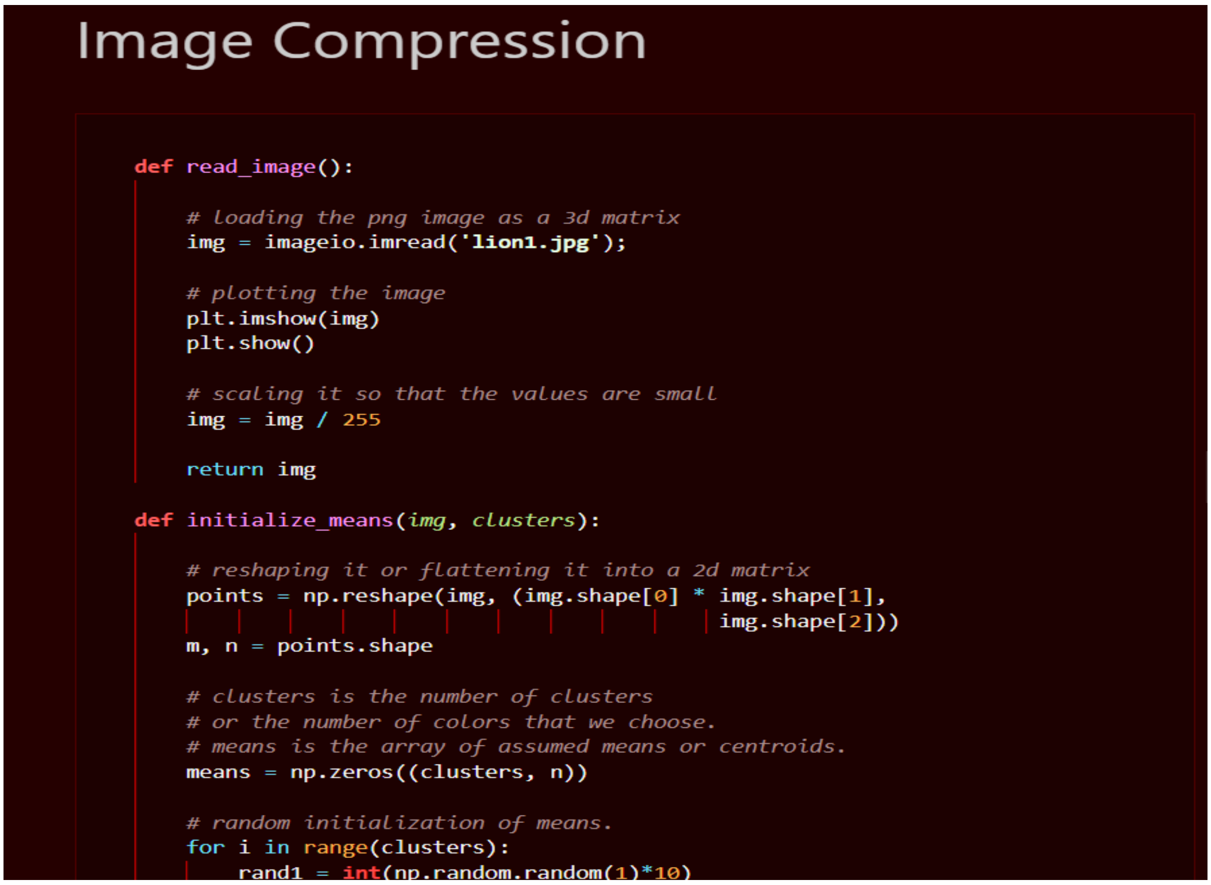
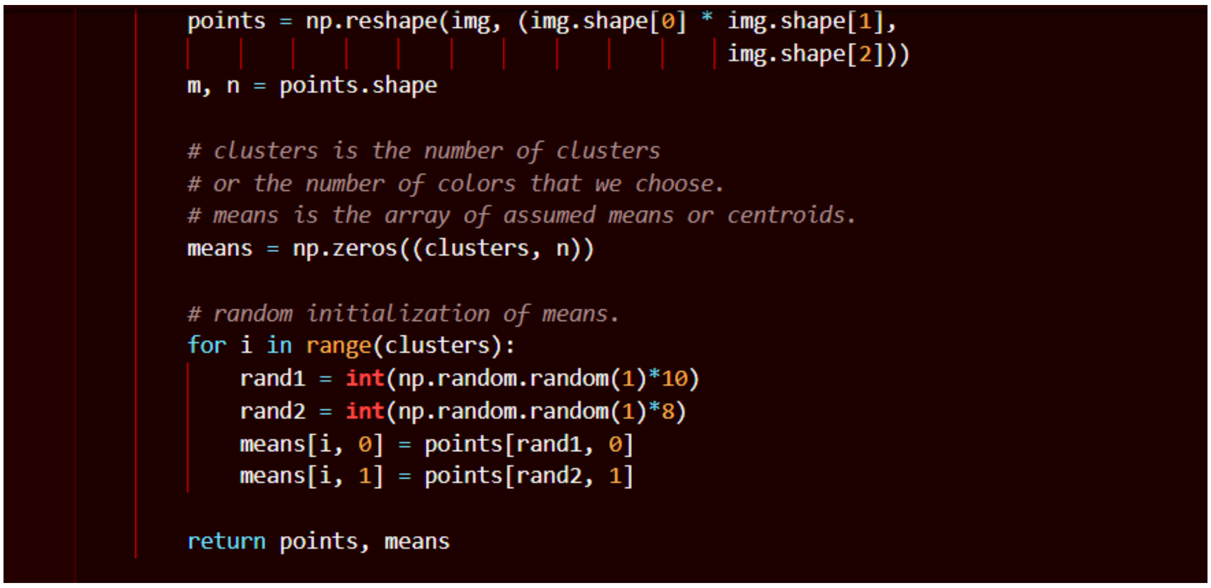
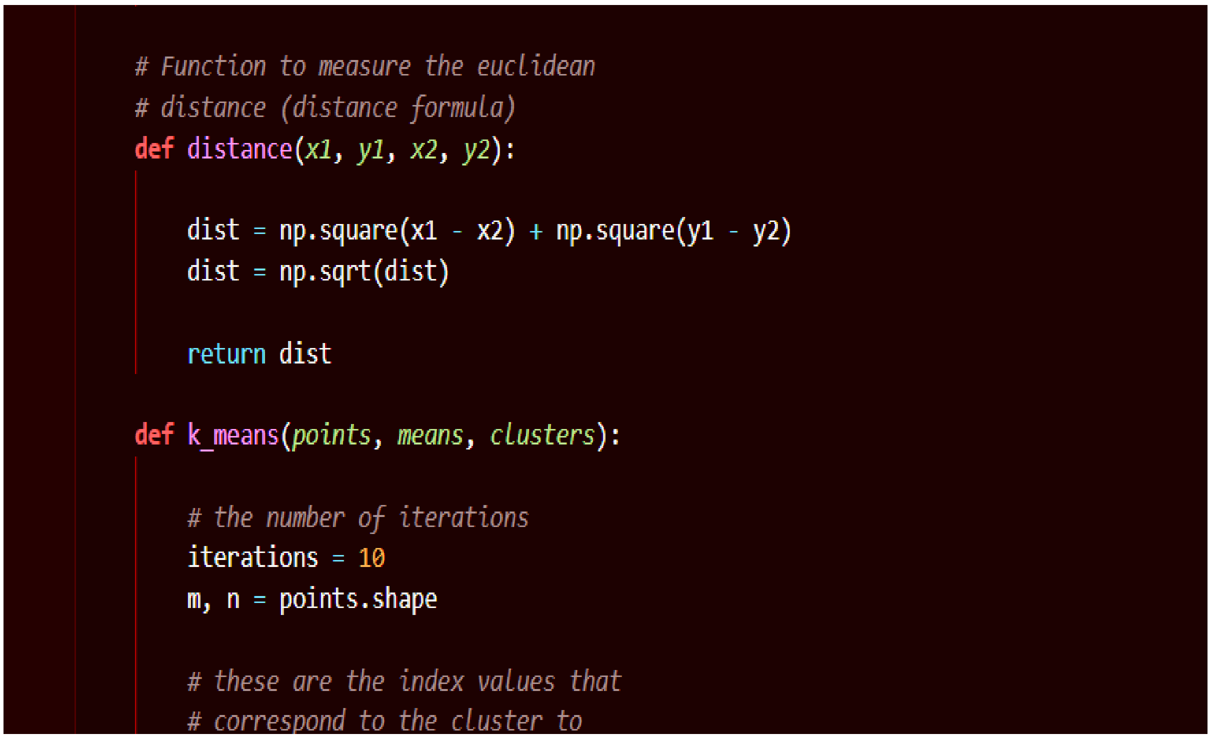


Image Compression

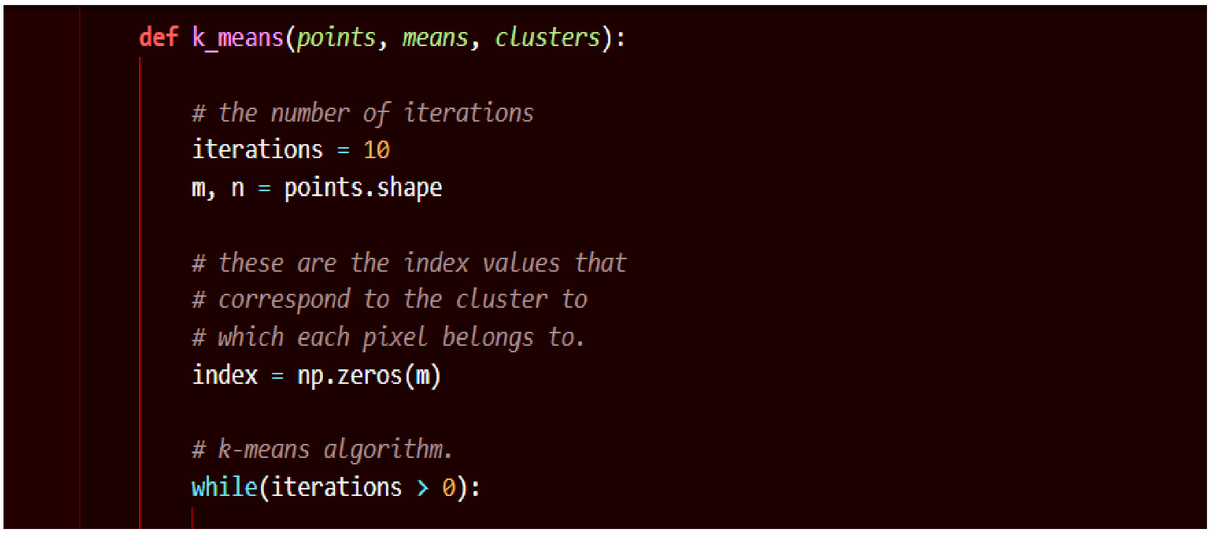
```
def read_image():

    # loading the png image as a 3d matrix
    img = imageio.imread('lion1.jpg');

    # plotting the image
    plt.imshow(img)
    plt.show()

    # scaling it so that the values are small
    img = img / 255

    return img

def initialize_means(img, clusters):

    # reshaping it or flattening it into a 2d matrix
    points = np.reshape(img, (img.shape[0] * img.shape[1],
                                              img.shape[2]))
    m, n = points.shape

    # clusters is the number of clusters
    # or the number of colors that we choose.
    # means is the array of assumed means or centroids.
    means = np.zeros((clusters, n))

    # random initialization of means.
    for i in range(clusters):
        rand1 = int(np.random.random(1)*10)
```



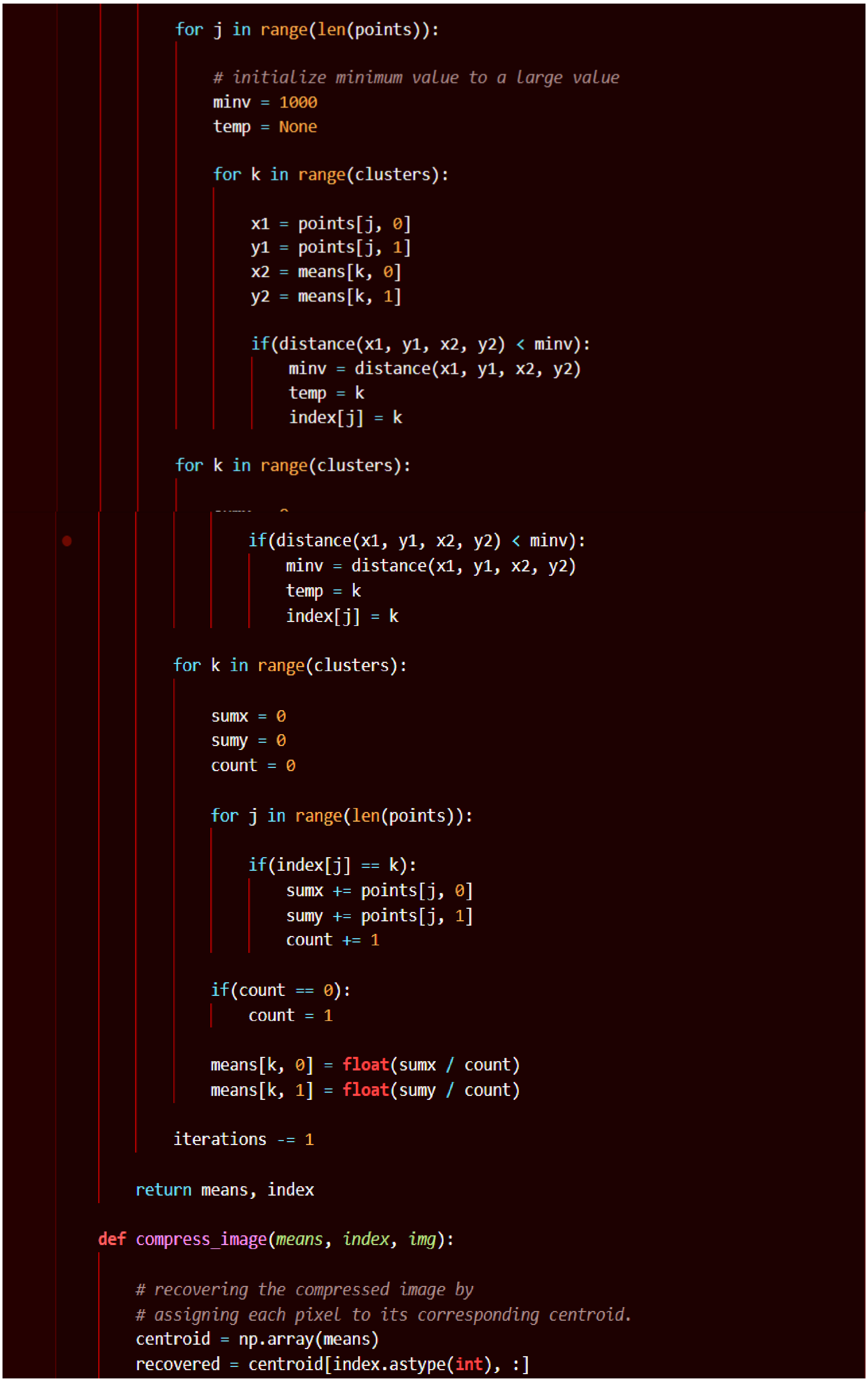
```
    points = np.reshape(img, (img.shape[0] * img.shape[1],
                                             img.shape[2]))
    m, n = points.shape

    # clusters is the number of clusters
    # or the number of colors that we choose.
    # means is the array of assumed means or centroids.
    means = np.zeros((clusters, n))

    # random initialization of means.
    for i in range(clusters):
        rand1 = int(np.random.random(1)*10)
        rand2 = int(np.random.random(1)*8)
        means[i, 0] = points[rand1, 0]
        means[i, 1] = points[rand2, 1]

    return points, means
```

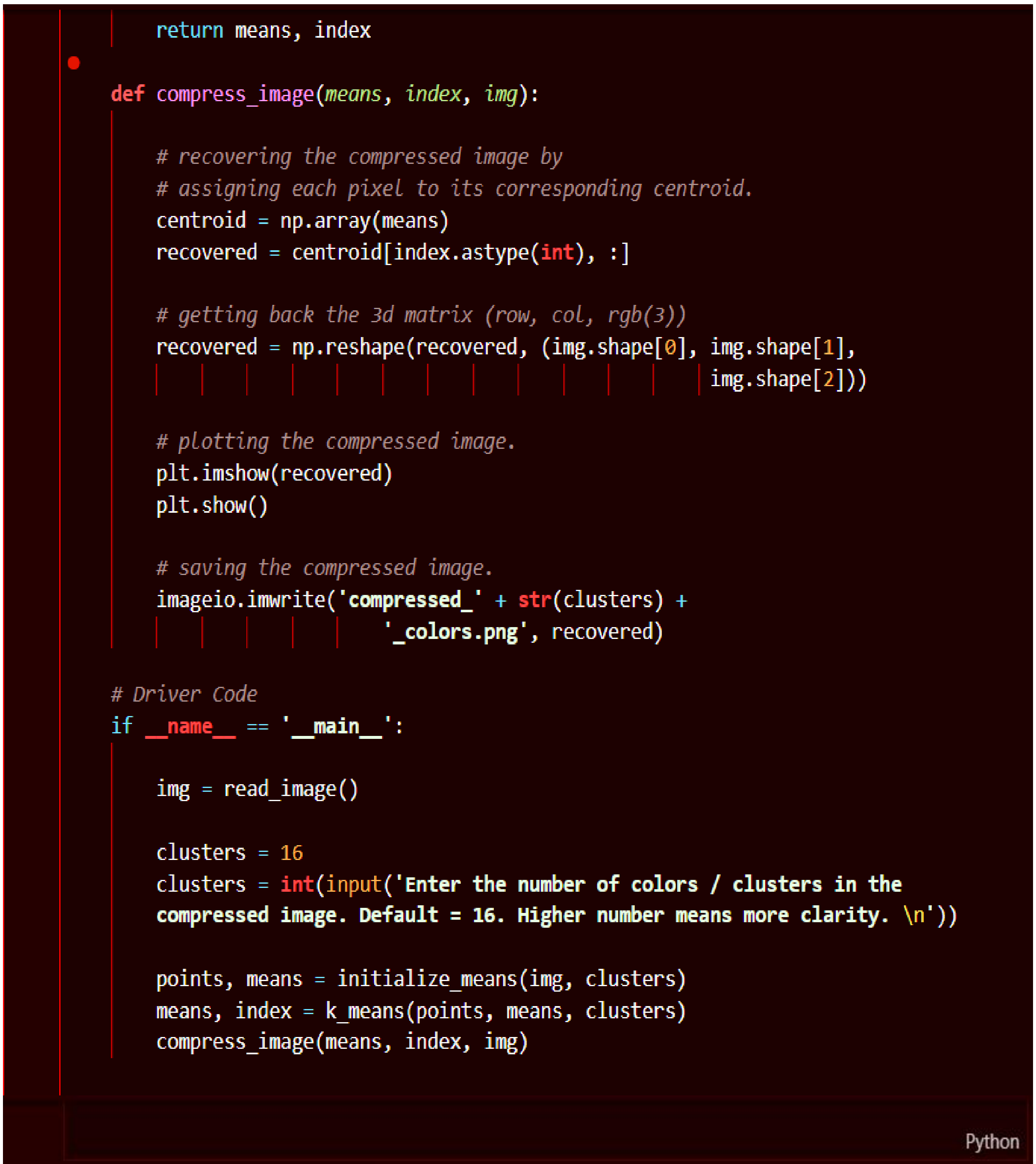


```
# Function to measure the euclidean
# distance (distance formula)
def distance(x1, y1, x2, y2):

    dist = np.square(x1 - x2) + np.square(y1 - y2)
    dist = np.sqrt(dist)

    return dist

def k_means(points, means, clusters):

    # the number of iterations
    iterations = 10
    m, n = points.shape

    # these are the index values that
    # correspond to the cluster to
```



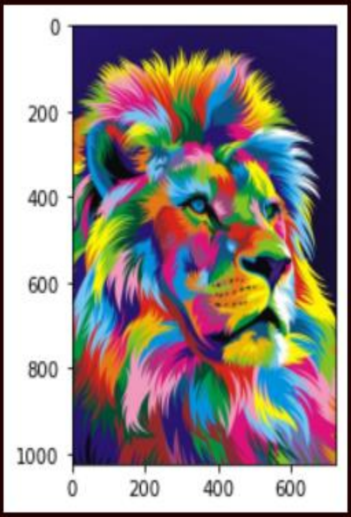
```
def k_means(points, means, clusters):

    # the number of iterations
    iterations = 10
    m, n = points.shape

    # these are the index values that
    # correspond to the cluster to
    # which each pixel belongs to.
    index = np.zeros(m)

    # k-means algorithm.
    while(iterations > 0):
```

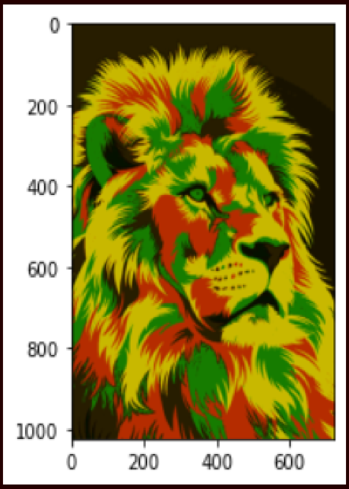

```
        for j in range(len(points)):

            # initialize minimum value to a large value
            minv = 1000
            temp = None

            for k in range(clusters):

                x1 = points[j, 0]
                y1 = points[j, 1]
                x2 = means[k, 0]
                y2 = means[k, 1]

                if(distance(x1, y1, x2, y2) < minv):
                    minv = distance(x1, y1, x2, y2)
                    temp = k
                    index[j] = k

        for k in range(clusters):

                if(distance(x1, y1, x2, y2) < minv):
                    minv = distance(x1, y1, x2, y2)
                    temp = k
                    index[j] = k

        for k in range(clusters):

            sumx = 0
            sumy = 0
            count = 0

            for j in range(len(points)):

                if(index[j] == k):
                    sumx += points[j, 0]
                    sumy += points[j, 1]
                    count += 1

            if(count == 0):
                count = 1

            means[k, 0] = float(sumx / count)
            means[k, 1] = float(sumy / count)

        iterations -= 1

    return means, index

def compress_image(means, index, img):

    # recovering the compressed image by
    # assigning each pixel to its corresponding centroid.
    centroid = np.array(means)
    recovered = centroid[index.astype(int), :]
```



```python
    return means, index

def compress_image(means, index, img):

    # recovering the compressed image by
    # assigning each pixel to its corresponding centroid.
    centroid = np.array(means)
    recovered = centroid[index.astype(int), :]

    # getting back the 3d matrix (row, col, rgb(3))
    recovered = np.reshape(recovered, (img.shape[0], img.shape[1],
                                       img.shape[2]))

    # plotting the compressed image.
    plt.imshow(recovered)
    plt.show()

    # saving the compressed image.
    imageio.imwrite('compressed_' + str(clusters) +
                         '_colors.png', recovered)

# Driver Code
if __name__ == '__main__':

    img = read_image()

    clusters = 16
    clusters = int(input('Enter the number of colors / clusters in the
compressed image. Default = 16. Higher number means more clarity. \n'))

    points, means = initialize_means(img, clusters)
    means, index = k_means(points, means, clusters)
    compress_image(means, index, img)
```

...





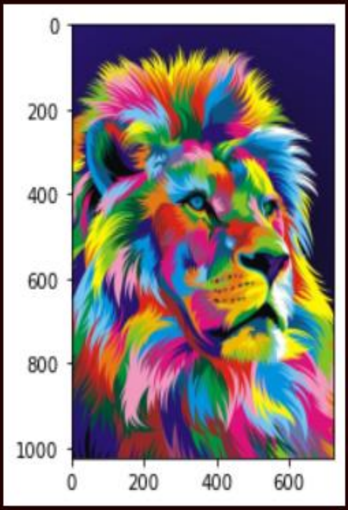
```
Lossy conversion from float64 to uint8. Range [0, 1]. Convert image to uint8 prior
to saving to suppress this warning.
```

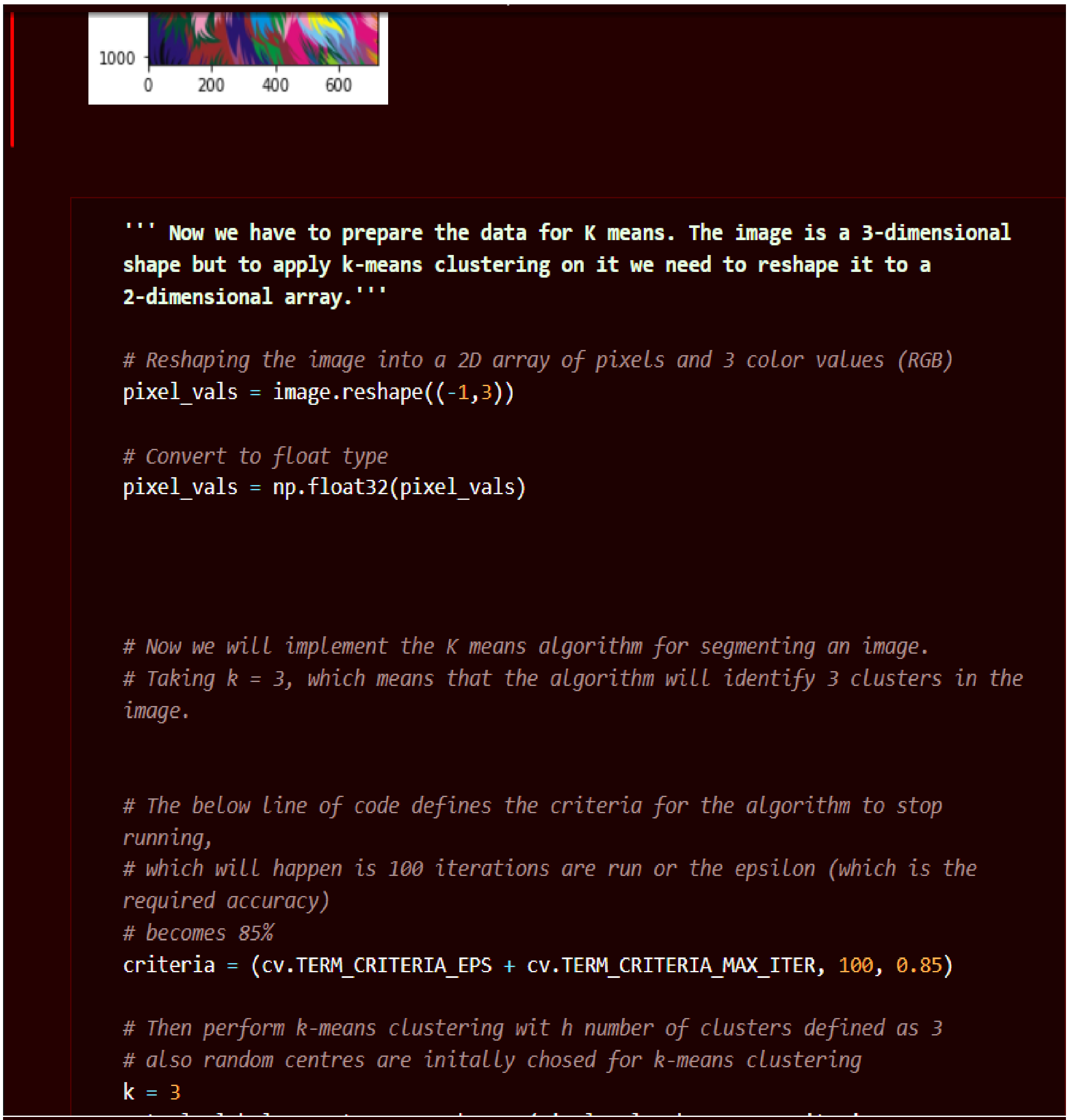
Image Segmentation

```python
# Image Segmentation


image = cv.imread('lion1.jpg') # Reading the image

# Change color to RGB (from BGR)
image = cv.cvtColor(image, cv.COLOR_BGR2RGB)

plt.imshow(image)
```

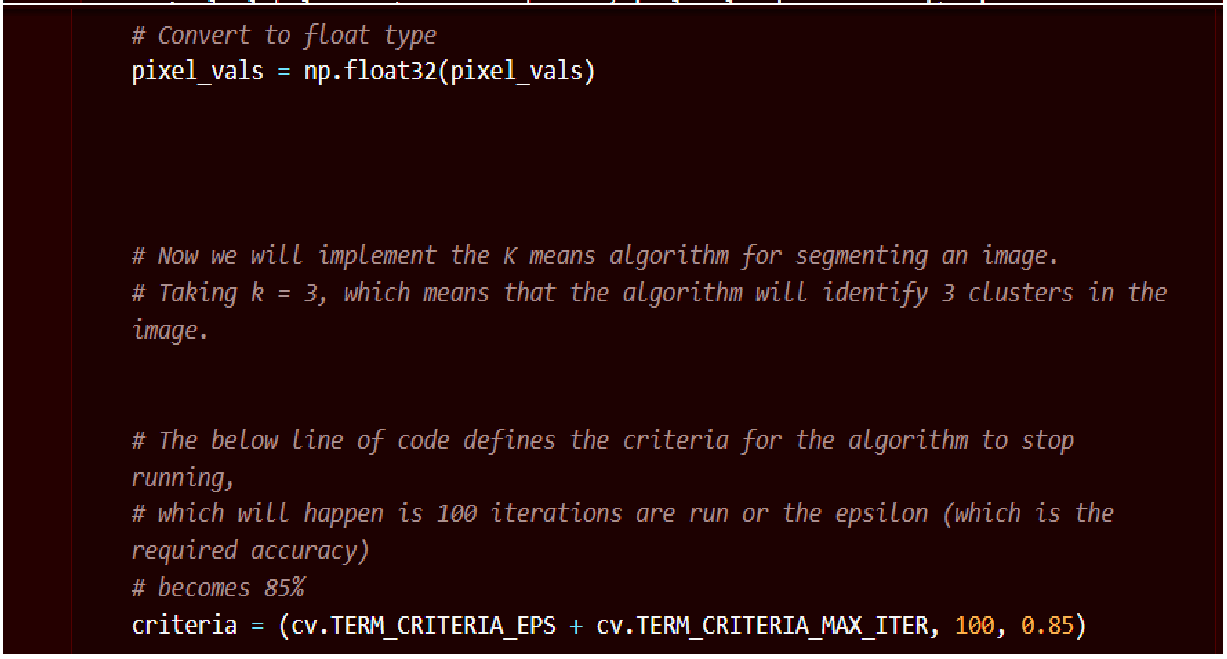
Python

```
<matplotlib.image.AxesImage at 0x1f7c11a0b80>
```

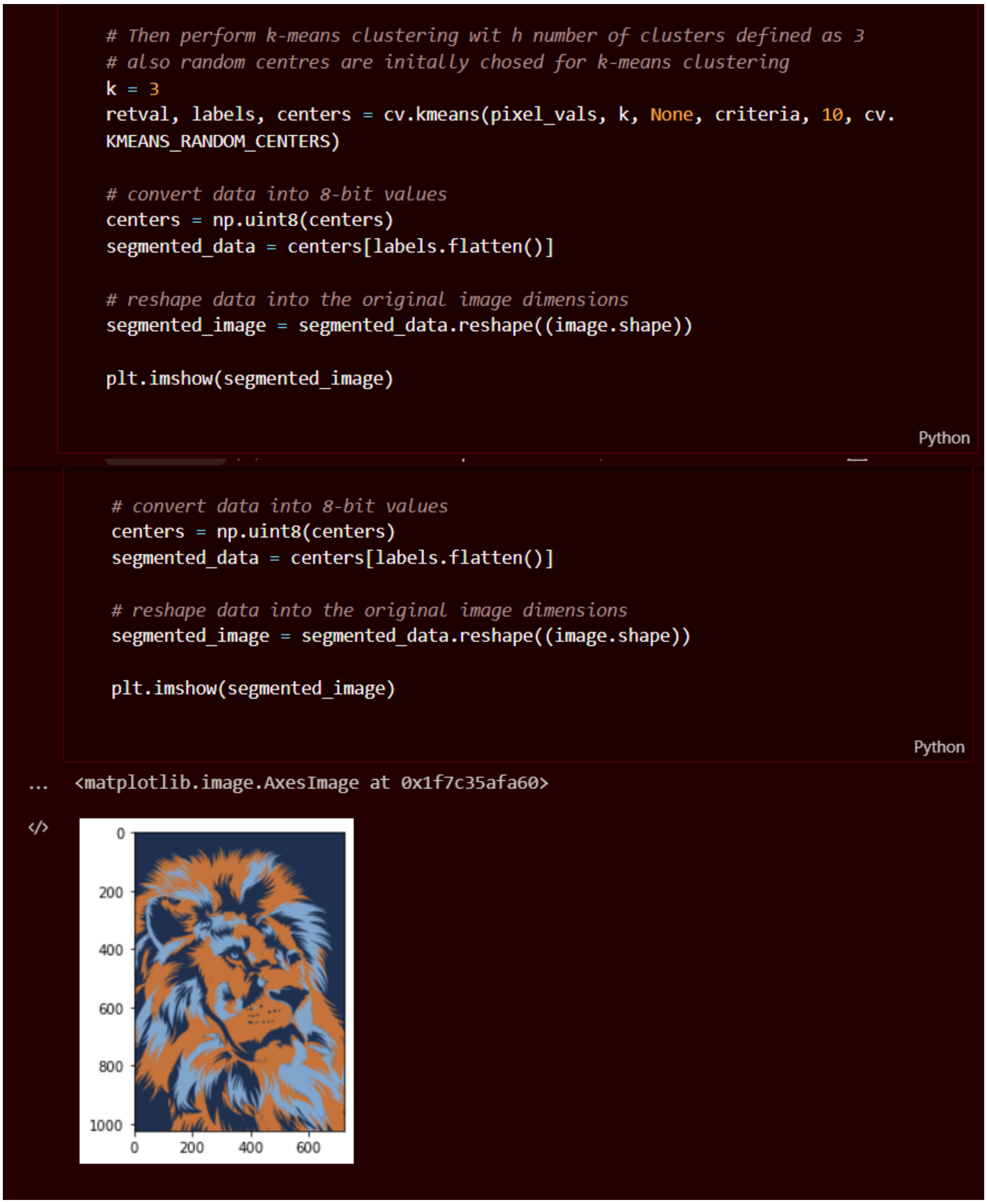




```
''' Now we have to prepare the data for K means. The image is a 3-dimensional
shape but to apply k-means clustering on it we need to reshape it to a
2-dimensional array.'''

# Reshaping the image into a 2D array of pixels and 3 color values (RGB)
pixel_vals = image.reshape((-1,3))

# Convert to float type
pixel_vals = np.float32(pixel_vals)




# Now we will implement the K means algorithm for segmenting an image.
# Taking k = 3, which means that the algorithm will identify 3 clusters in the
image.


# The below line of code defines the criteria for the algorithm to stop
running,
# which will happen is 100 iterations are run or the epsilon (which is the
required accuracy)
# becomes 85%
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 0.85)

# Then perform k-means clustering wit h number of clusters defined as 3
# also random centres are initally chosed for k-means clustering
k = 3
```



```
# Convert to float type
pixel_vals = np.float32(pixel_vals)




# Now we will implement the K means algorithm for segmenting an image.
# Taking k = 3, which means that the algorithm will identify 3 clusters in the
image.


# The below line of code defines the criteria for the algorithm to stop
running,
# which will happen is 100 iterations are run or the epsilon (which is the
required accuracy)
# becomes 85%
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 100, 0.85)
```



```
# Then perform k-means clustering wit h number of clusters defined as 3
# also random centres are initally chosed for k-means clustering
k = 3
retval, labels, centers = cv.kmeans(pixel_vals, k, None, criteria, 10, cv.
KMEANS_RANDOM_CENTERS)

# convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

plt.imshow(segmented_image)
```

```
# convert data into 8-bit values
centers = np.uint8(centers)
segmented_data = centers[labels.flatten()]

# reshape data into the original image dimensions
segmented_image = segmented_data.reshape((image.shape))

plt.imshow(segmented_image)
```

```
<matplotlib.image.AxesImage at 0x1f7c35afa60>
```

Fig. 7.1 Code Implemented with Image Outputs

CONCLUSION AND FUTURE SCOPE

The future scope for image processing involves searching outer space for various intelligent life forms using cameras and telescopes, followed by image processing algorithms and filtration methods. Newly discovered Intelligent species are also formed entirely from scientists in different parts of the globe are going to move towards advances in the field of object detection and image processing uses / applications. [13]

The image processing process can extract some information from image, or get enhanced version of an image. But in order to optimize your workflow and avoid waste of time, it is very important to capture the image and process it afterwards, followed by image detection as a post-processing step. [14]

FUTURE APPLICATIONS

- Improvements in Augmented reality
- Traffic data collection

- Autonomous cars use image segmentation.
- Morphological image processing
- Military applications
- Sophisticated optical sorting
- Medical imaging
- Improvements in Stereography
- Rise in industrial applications
- Space exploration
- Fingerprint and retina recognition

REFERENCES

[1] A. Munshi et al., “Image compression using K-mean clustering algorithm,” IJCSNS International Journal of Computer Science and Network Security, vol. 21, no. 9, p. 275, 2021, doi: 10.22937/IJCSNS.2021.21.9.36.

[2] N. Dhanachandra, K. Manglem, and Y. J. Chanu, “Image Segmentation Using K-means Clustering Algorithm and Subtractive Clustering Algorithm,” Procedia Computer Science, vol. 54, pp. 764–771, 2015, doi: 10.1016/J.PROCS.2015.06.090.

[3] F. G. Febrinanto, C. Dewi, and A. Triwiratno, “IOP Conference Series: The Implementation of K-Means Algorithm as Image Segmenting Method in Identifying the Citrus Leaves Disease”, doi: 10.1088/1755-1315/243/1/012024.

[4] “Introduction to K-means Clustering: Implementation and Image Compression. | by Chingis Oinar | Towards Data Science.” https://towardsdatascience.com/introduction-to-k-means-clustering-implementation-and-image-compression-8c59c439d1b (accessed Dec. 31, 2021).

[5] “Image Compression using K-Means Clustering | by Satyam Kumar | Towards Data Science.” https://towardsdatascience.com/image-compression-using-k-means-clustering-aa0c91bb0eeb (accessed Dec. 31, 2021).

[6] “Image Compression using Seam Carving and Clustering – Aditya Sharma – Data Science | AI | Deep Learning.” https://adityashrm21.github.io/Image-Compression/ (accessed Jan. 04, 2022).

[7] “Image Processing in Python: Algorithms, Tools, and Methods You Should Know - neptune.ai.” https://neptune.ai/blog/image-processing-in-python-algorithms-tools-and-methods-you-should-know (accessed Dec. 31, 2021).

[8] “Understanding Lossy Compression and When to Use It | WordPressio.” https://www.wordpressio.com/understanding-lossy-compression-and-when-to-use-it/ (accessed Jan. 04, 2022).

[9] “Introduction to K-means Clustering: Implementation and Image Compression. | by Chingis Oinar | Towards Data Science.” https://towardsdatascience.com/introduction-to-k-means-clustering-implementation-and-image-compression-8c59c439d1b (accessed Dec. 31, 2021).

[10] “Image Segmentation using K Means Clustering - GeeksforGeeks.” https://www.geeksforgeeks.org/image-segmentation-using-k-means-clustering/ (accessed Dec. 31, 2021).

[11] “Image Segmentation - K-means Clustering.” https://pixelsciences.blogspot.com/2017/07/image-segmentation-k-means-clustering.html (accessed Jan. 04, 2022).

[12] “Image Processing for Data Science | by Anuj Khandelwal | Coherent Thoughts | Medium.” https://medium.com/coherent-thoughts/image-processing-for-data-science-4b280464a39 (accessed Jan. 04, 2022).

[13] “Top 10 applications of Image processing.” https://createbytes.com/insights/Top-10-applications-of-Image-processing/ (accessed Jan. 04, 2022).

[14] “Image Processing Projects using MATLAB, Python & Android.” https://www.elprocus.com/image-processing-projects-for-engineering-students/ (accessed Jan. 04, 2022).