K Mean Clustering

Import the libraries

In [38]:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display, HTML
```

Sapmle Input Data vizualization

In [2]:

```
image_filenames = ['1.jpg', '2.jpg', '3.jpg']

# Open and display each image
for filename in image_filenames:
    # Open the image using PIL
    image = Image.open(filename)

    # Define the desired width for the resized image
    desired_width = 250

    # Calculate the new height while maintaining the aspect ratio
    aspect_ratio = float(image.size[1]) / float(image.size[0])
    desired_height = int(aspect_ratio * desired_width)

    # Resize the image
    resized_image = image.resize((desired_width, desired_height))

    # Display the resized image
    display(resized_image)
```

Major
LEAGUE

Read the images

In [3]:

```
#________________This code is for getting information of images(No editing needed)_______________________



input_img_1_info = cv.imread('1.jpg')
input_img_2_info = cv.imread('2.jpg')
input_img_3_info = cv.imread('3.jpg')

#Converting BGR to RGB(OpenCV uses BGR format by default)__You can skip this step if you are using PIL to load images
input_img_1 = cv.cvtColor(input_img_1_info, cv.COLOR_BGR2RGB)
input_img_2 = cv.cvtColor(input_img_2_info, cv.COLOR_BGR2RGB)
input_img_3 = cv.cvtColor(input_img_3_info, cv.COLOR_BGR2RGB)


#______________RESIZE IMAGES HERE IF NEEDED_______________________
scaling_factor = 0.3  # Adjust this value to change the size of the shrunk images
#your code here depending on the library you are using



#______________RESIZE IMAGES HERE IF NEEDED_______________________

#Information of images
print("shapes of images:")
print('Image 1: ', input_img_1_info.shape)
print('Image 2: ', input_img_2_info.shape)
print('Image 3: ', input_img_3_info.shape)

#__________You can use PIL to load images which use RGB format by default_____________________
# BUT you wont be able to get information of images as given by cv2

# input_img_1 = Image.open('1.jpg')
# input_img_2 = Image.open('2.jpg')
# input_img_3 = Image.open('3.jpg')
# print("sizes of images:")
# print('Image 1: ', input_img_1.size)
# print('Image 2: ', input_img_2.size)
# print('Image 3: ', input_img_3.size)
```

```
shapes of images:
Image 1:  (744, 750, 3)
Image 2:  (533, 800, 3)
Image 3:  (393, 700, 3)
```

Expexted Output:
shapes of images:
Image 1: (744, 750, 3)
Image 2: (533, 800, 3)
Image 3: (393, 700, 3)

Understanding the data

In [4]:

```
#printing the first row of the image
print("Image 1 : ROW1:")
print(input_img_1_info[0].shape)
print("-------------")
print(input_img_1_info[0])
print("-------------")
type(input_img_1_info)
```

```
Image 1 : ROW1:
(750, 3)
-------------
[[145 167 178]
 [145 167 178]
 [145 167 178]
 ...
 [ 96 115  98]
 [ 96 115  98]
 [ 96 115  98]]
-------------
```

Out[4]:

```
numpy.ndarray
```

Preprocessing the data

In [5]:

```
#_____________You will need this piece of code if you are using PIL to load im
ages_________________________
# Cuz Cv2 make use of numpy array to store images and PIL uses Image object to
store images

input_img_1 = np.array(input_img_1)
print(f"input_img_1.shape: {input_img_1.shape}")

input_img_2 = np.array(input_img_2)
print(f"input_img_2.shape: {input_img_2.shape}")

input_img_3 = np.array(input_img_3)
print(f"input_img_3.shape: {input_img_3.shape}")
```

```
input_img_1.shape: (744, 750, 3)
input_img_2.shape: (533, 800, 3)
input_img_3.shape: (393, 700, 3)
```

In [6]:

```python
#_________________This code will need your attention to run properly____________

# Convert the image 1 to 2D array
input_img_1_2d = input_img_1.reshape(-1, input_img_1.shape[-1])
print("Image 1 : 2D array:")
print(input_img_1_2d.shape)

# Convert the image 2 to 2D array
input_img_2_2d = input_img_2.reshape(-1, input_img_2.shape[-1])
print("Image 2 : 2D array:")
print(input_img_2_2d.shape)

# Convert the image 3 to 2D array
input_img_3_2d = input_img_3.reshape(-1, input_img_3.shape[-1])
print("Image 3 : 2D array:")
print(input_img_3_2d.shape)
```

```
Image 1 : 2D array:
(558000, 3)
Image 2 : 2D array:
(426400, 3)
Image 3 : 2D array:
(275100, 3)
```

Expected Output

Image 1 : 2D array: (558000, 3)
Image 2 : 2D array: (426400, 3)
Image 3 : 2D array: (275100, 3)

Implimentation of K-Mean Clustering

Helper Funtions

In [7]:

```python
#your code here
def random_centroids_initializer(data, k):
    random_indices = np.random.choice(data.shape[0], k, replace=False)
    centroids = data[random_indices]
    return centroids
```

In [8]:

```python
#your code here
def assign_data_points_to_nearest_centroid(data, centroids):
    distances = np.abs(data[:, np.newaxis, :] - centroids)
    distances = np.sum(distances, axis=2)
    labels = np.argmin(distances, axis=1)
    return labels
```

In [9]:

```
#your code here

def update_centroids_by_manhattan_distance(data, labels, k):
    centroids = np.zeros((k, data.shape[1]))
    for i in range(k):
        cluster_data = data[labels == i]
        if len(cluster_data) > 0:
            centroids[i] = np.mean(cluster_data, axis=0)
    return centroids
```

Mian Function

In [10]:

```
#_________________This code will need your attention to run properly__________
__
#your code here

def kmeans_clustering(data, k, max_iterations=100):

    # Randomly initialize centroids
    #call the helper function here of random_centroids_initializer
    centroids = random_centroids_initializer(data, k)
    tol=1e-5
    for _ in range(max_iterations):
        # Assign each data point to the nearest centroid
        labels = assign_data_points_to_nearest_centroid(data, centroids)

        # Update centroids by taking the mean of the assigned data points
        new_centroids = update_centroids_by_manhattan_distance(data, labels,
k)

        # Check convergence
        if np.allclose(centroids, new_centroids, atol=tol):
            break

        centroids = new_centroids


    #Upadte centroids to the latest centroids

    return labels, centroids
```

Verification of the output

In [17]:

```
print("For input_image_1: \n")
image = input_img_1_2d
k = 2
labels, centroids = kmeans_clustering(image, k)
print('k', k)
print('unique labels', np.unique(labels))
print('Labels shape: ', labels.shape)
print('Centroids shape: ', centroids.shape)
print('centroids', centroids)
print("--------------------------")
```

```
For input_image_1: 

k 2
unique labels [0 1]
Labels shape:  (558000,)
Centroids shape:  (2, 3)
centroids [[ 98.15321625  72.44336854  63.15206393]
 [182.32910336 168.23132632 150.00980018]]
--------------------------
```

In [18]:

```
print("\nFor input_image_2: ")
image = input_img_2_2d
k = 2
labels, centroids = kmeans_clustering(image, k)
print('k', k)
print('unique labels', np.unique(labels))
print('Labels shape: ', labels.shape)
print('Centroids shape: ', centroids.shape)
print('centroids', centroids)
```

```
For input_image_2: 
k 2
unique labels [0 1]
Labels shape:  (426400,)
Centroids shape:  (2, 3)
centroids [[226.34647018 223.46249381 220.90380275]
 [193.99718075 161.61216588 146.87529588]]
```

In [19]:

```
print("\nFor input_image_3: ")
image = input_img_3_2d
k = 2
labels, centroids = kmeans_clustering(image, k)
print('k', k)
print('unique labels', np.unique(labels))
print('Labels shape: ', labels.shape)
print('Centroids shape: ', centroids.shape)
print('centroids', centroids)
```

```
For input_image_3:
k 2
unique labels [0 1]
Labels shape:  (275100,)
Centroids shape:  (2, 3)
centroids [[118.32229891 158.51120919 238.83084683]
 [ 79.00552162 107.03481755  28.29886674]]
```

Expected Output

Note : The output will vary if you use diffrent distance metric or reduced size of the image.

For input_image_1:

- k 2
- unique labels [0 1]
- Labels shape: (558000,)
- Centroids shape: (2, 3)

```
[[ 98.15321625 72.44336854 63.15206393]
[182.32910336 168.23132632 150.00980018]]
```

For input_image_2:

- k 2
- unique labels [0 1]
- Labels shape: (426400,)
- Centroids shape: (2, 3)

```
centroids [[226.34647018 223.46249381 220.90380275]
[193.99718075 161.61216588 146.87529588]]
```

For input_image_1:

- k 2
- unique labels [0 1]
- Labels shape: (275100,)
- Centroids shape: (2, 3)

```
centroids [[118.32229891 158.51120919 238.83084683]
[ 79.00552162 107.03481755 28.29886674]]
```

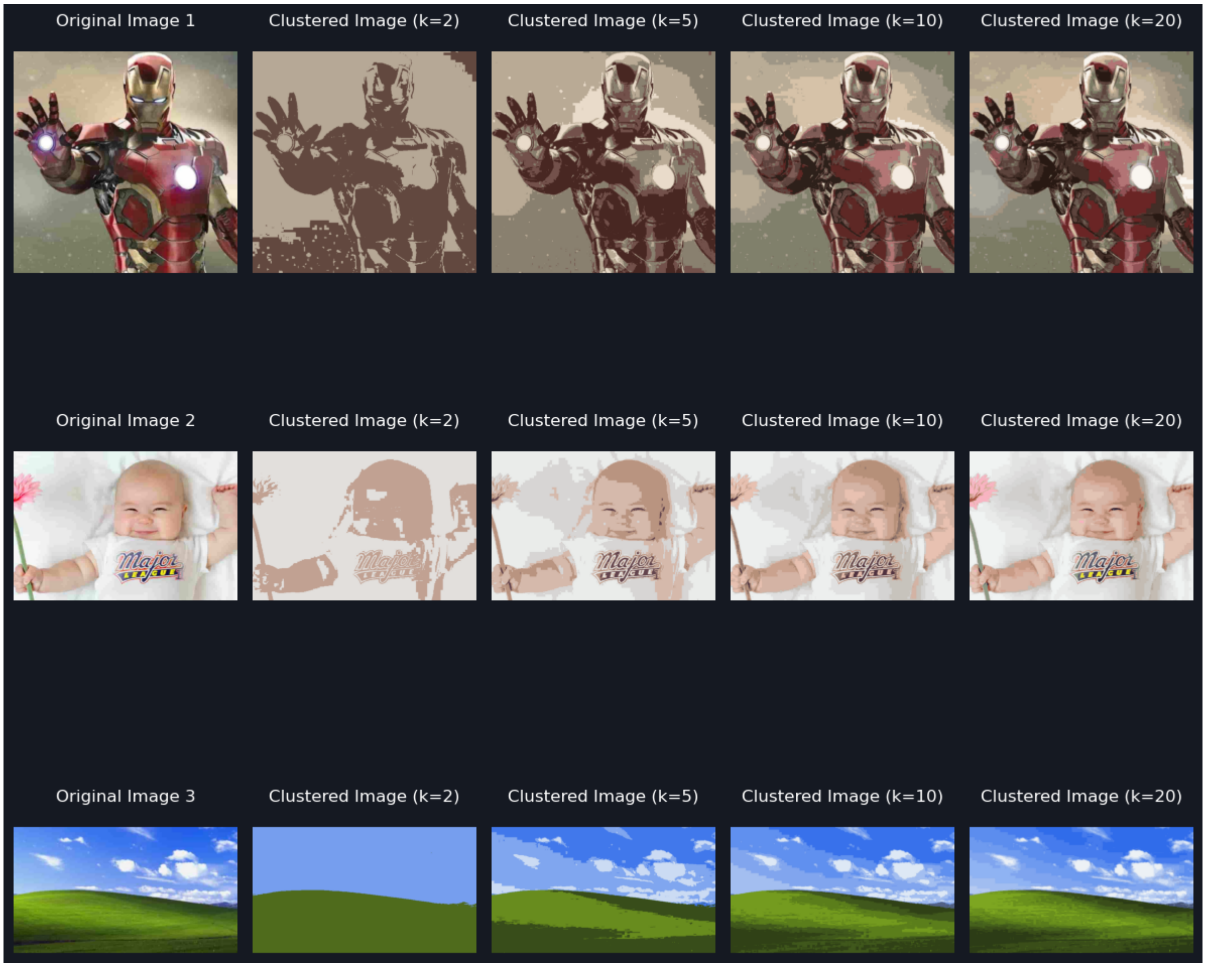
BATCH RUN

K-Mean Clustering on the all images

NOTE: IF YOUR CODE IS TAKING TOO MUCH TIME TO RUN, TRY RESIZE THE IMAGE TO SMALLER SIZE

In [36]:

```
k_values = [2, 5, 10, 20]
images = [input_img_1_2d, input_img_2_2d, input_img_3_2d]
org_images = [input_img_1, input_img_2, input_img_3]

num_rows = len(images)
num_cols = len(k_values) + 1

fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 12))
fig.set_facecolor('#151922')  # Set dark navy background for the figure

for i, (image, org_image) in enumerate(zip(images, org_images)):
    org_height, org_width, _ = org_image.shape
    org_size_inches = (org_width / 100, org_height / 100)  # Convert to inches

    axs[i, 0].imshow(org_image)
    axs[i, 0].set_title(f'Original Image {i+1}\n', color='white')
    axs[i, 0].axis('off')

    for j, k in enumerate(k_values):
        labels, centroids = kmeans_clustering(image, k)
        # Replace pixel values with centroid values
        new_image_data = centroids[labels].reshape(org_image.shape)
        # Convert the data back to image format
        new_image = Image.fromarray(new_image_data.astype(np.uint8))
        # Resize the image to match the original size
        resized_image = new_image.resize((org_width, org_height))
        axs[i, j+1].imshow(resized_image)
        axs[i, j+1].set_title(f'Clustered Image (k={k})\n', color='white')
        axs[i, j+1].axis('off')

plt.tight_layout()
plt.show()
```



Expected Output

Example Image

Example Image

Example Image