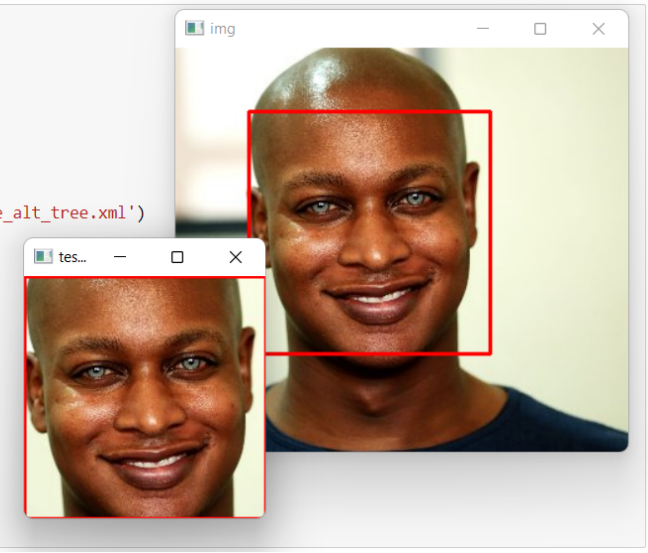
Project Report

This project is to find the dominant colors in face.

Step-1

1. We are detecting the face from an image and saving the extracted face in a folder.we are using the haarcascades face detection.we can also use any another face detection techniques here.

```python
import cv2

# Read the input image
img = cv2.imread('test-2.jpg')

# Convert into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Load the cascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_alt_tree.xml')

# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.1, 4)

# Draw rectangle around the faces and crop the faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 2)
    faces = img[y:y + h, x:x + w]
    cv2.imshow("test-6",faces)
    cv2.imwrite('test-6(test).jpg', faces)

# Display the output
cv2.imwrite('detected.jpg', img)
key = cv2.imshow('img', img)
cv2.waitKey()
```

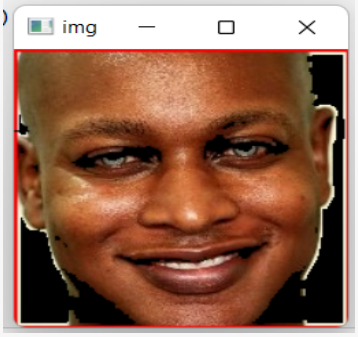


Step-2

Now we are passing the saved image of human face into a function which can extract the skin color from the face.The extractSkin function takes an 8 bit 3 channel image in the BGR color-space and returns the extracted image in same color-space.

The function works by using the HSV color-space and uses threshold to extracts pixel that corresponds to the skin color.
In the image we can see the all the other parts become black except the skin color

```python
def extractSkin(image):
  # Taking a copy of the image
  img =  image.copy()
  # Converting from BGR Colours Space to HSV
  img =  cv2.cvtColor(img,cv2.COLOR_BGR2HSV)

  # Defining HSV Threadholds
  lower_threshold = np.array([0, 48, 80], dtype=np.uint8)
  upper_threshold = np.array([20, 255, 255], dtype=np.uint8)

  # Single Channel mask,denoting presence of colours in the about threshold
  skinMask = cv2.inRange(img,lower_threshold,upper_threshold)

  # Cleaning up mask using Gaussian Filter
  skinMask = cv2.GaussianBlur(skinMask,(3,3),0)

  # Extracting skin from the threshold mask
  skin  =  cv2.bitwise_and(img,img,mask=skinMask)

  # Return the Skin image
  return cv2.cvtColor(skin,cv2.COLOR_HSV2BGR)
```



Step - 3

The removeBlack function is more sort of the utility function to remove out the black pixel from the skin extracted.This function is useful when threshold is used in the image.

```
def removeBlack(estimator_labels, estimator_cluster):

  # Check for black
  hasBlack = False

  # Get the total number of occurance for each color
  occurance_counter = Counter(estimator_labels)


  # Quick lambda function to compare to lists
  compare = lambda x, y: Counter(x) == Counter(y)

  # Loop through the most common occuring color
  for x in occurance_counter.most_common(len(estimator_cluster)):

    # Quick List comprehension to convert each of RBG Numbers to int
    color = [int(i) for i in estimator_cluster[x[0]].tolist() ]



    # Check if the color is [0,0,0] that if it is black
    if compare(color , [0,0,0]) == True:
      # delete the occurance
      del occurance_counter[x[0]]
      # remove the cluster
      hasBlack = True
      estimator_cluster = np.delete(estimator_cluster,x[0],0)
      break
  return (occurance_counter,estimator_cluster,hasBlack)
```

Step - 4

In step-4 or the function which we are using here is used for getting the information about the color from the image

The getColorInfomation function does all the heavy lifting to make sense of prediction that came from the clustering.

Taking the prediction labels (estimator_labels) and the cluster centroids (estimator_cluster) as the input and returns an array of dictionaries of the extracted colors.

The function also takes an optional parameter (hasThresholding) to indicate whether a mask was used. This passed from the extractDominantColor function

```python
def getColorInformation(estimator_labels, estimator_cluster,hasThresholding=False):
  # Variable to keep count of the occurance of each color predicted
  occurance_counter = None
  # Output list variable to return
  colorInformation = []
  #Check for Black
  hasBlack =False
  # If a mask has be applied, remove th black
  if hasThresholding == True:
    (occurance,cluster,black) = removeBlack(estimator_labels,estimator_cluster)
    occurance_counter =  occurance
    estimator_cluster = cluster
    hasBlack = black
  else:
    occurance_counter = Counter(estimator_labels)
  # Get the total sum of all the predicted occurances
  totalOccurance = sum(occurance_counter.values())
  # Loop through all the predicted colors
  for x in occurance_counter.most_common(len(estimator_cluster)):
    index = (int(x[0]))
    # Quick fix for index out of bound when there is no threshold
    index =  (index-1) if ((hasThresholding & hasBlack)& (int(index) !=0)) else index
    # Get the color number into a list
    color = estimator_cluster[index].tolist()
    # Get the percentage of each color
    color_percentage= (x[1]/totalOccurance)
    #make the dictionay of the information
    colorInfo = {"cluster_index":index , "color": color , "color_percentage" : color_percentage }
    # Add the dictionary to the list
    colorInformation.append(colorInfo)
  return colorInformation
```

Step - 4

We are using the An unsupervised clustering algorithm, KMeans Clustering is used to cluster the pixel data based on their RGB values.

We are using one function The extractDominantColor is the function that call the above function to output the information.The function take an 8 bit 3 channel BGR image as the input , the number of colors to be extracted. This does all the super heavy lifting by sparkling some magic power of machine learning.The function also takes an optional parameter (hasThresholding) to indicate whether a thresholding mask was used. This passed to the getColorInformation function

```
def extractDominantColor(image,number_of_colors=5,hasThresholding=False):

  # Quick Fix Increase cluster counter to neglect the black
  if hasThresholding == True:
    number_of_colors +=1

  # Taking Copy of the image
  img = image.copy()

  # Convert Image into RGB Colours Space
  img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

  # Reshape Image
  img = img.reshape((img.shape[0]*img.shape[1]) , 3)

  #Initiate KMeans Object
  estimator = KMeans(n_clusters=number_of_colors, random_state=0)

  # Fit the image
  estimator.fit(img)

  # Get Colour Information
  colorInformation = getColorInformation(estimator.labels_,estimator.cluster_centers_,hasThresholding)
  return colorInformation
```

Here we required the one parameter called number_of_colors in the extractDominatColor function that is called the K value(how many clusters are using) we need to find the find the number of k values.
We are using the Elbow-Method and the Silhouette score for k (clusters) for finding the number of k values.we are creating one dataset with the values between higher threshold and lower threshold, using those values we are checking the K value for the clusters.

```
import numpy as np
import pandas as pd
import sklearn.cluster as cluster
x1 = np.linspace(0,20,num = 215)
x2 = np.linspace(40,255,num = 215)
x3 = np.linspace(80,255,num = 215)
```

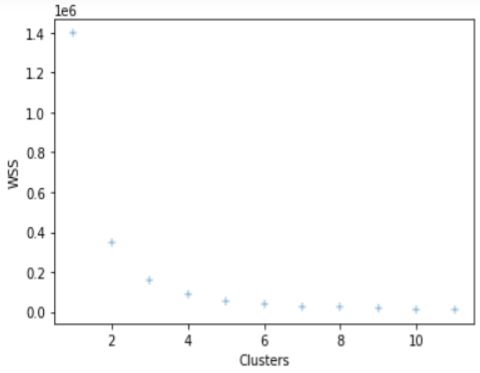
```
data = {'hue':x1,'saturation':x2,'value':x3}
data_chck = pd.DataFrame(data)
data_chck
```

Out[2]:

	hue	saturation	value
0	0.000000	40.000000	80.000000
1	0.093458	41.004673	80.817757
2	0.186916	42.009346	81.635514
3	0.280374	43.014019	82.453271
4	0.373832	44.018692	83.271028
...	...	...	...
210	19.626168	250.981308	251.728972
211	19.719626	251.985981	252.546729
212	19.813084	252.990654	253.364486
213	19.906542	253.995327	254.182243
214	20.000000	255.000000	255.000000

215 rows × 3 columns

using these elbow method and the Silhouette score we got the K value as 3



In [36]:
```
import sklearn.metrics as metrics
for i in range(3,13):
    labels=cluster.KMeans(n_clusters=i,init="k-means++",random_state=200).fit(df_short).labels_
    print ("Silhouette score for k(clusters) = "+str(i)+" is "
           +str(metrics.silhouette_score(df_short,labels,metric="euclidean",sample_size=1000,random_state=200)))
```

```
Silhouette score for k(clusters) = 3 is 0.5842640789538295
Silhouette score for k(clusters) = 4 is 0.5638824110106095
Silhouette score for k(clusters) = 5 is 0.55036448217279
Silhouette score for k(clusters) = 6 is 0.5412931552628185
Silhouette score for k(clusters) = 7 is 0.5324100082070229
Silhouette score for k(clusters) = 8 is 0.5269764103174758
Silhouette score for k(clusters) = 9 is 0.5221973056115494
Silhouette score for k(clusters) = 10 is 0.5180413618642947
Silhouette score for k(clusters) = 11 is 0.5123580129515818
Silhouette score for k(clusters) = 12 is 0.5094842290840257
```

Step - 5

This function we are using for making the visualization in a proper manner, In this we are getting the dominate colors of the person in a rectangle color bar format.
The plotColorBar function gives a visually representation of the extracted color information.

Taking the color information (colorInformation) as input and returns 500x100 8 bit 3 channel BGR color space image

```
def plotColorBar(colorInformation):
  #Create a 500x100 black image
  color_bar = np.zeros((100,500,3), dtype="uint8")

  top_x = 0
  for x in colorInformation:
    bottom_x = top_x + (x["color_percentage"] * color_bar.shape[1])

    color = tuple(map(int,(x['color'])))

    cv2.rectangle(color_bar , (int(top_x),0) , (int(bottom_x),color_bar.shape[0]) ,color , -1)
    top_x = bottom_x
  return color_bar
```

Color Bar

The function makes print out the color information in a readable manner

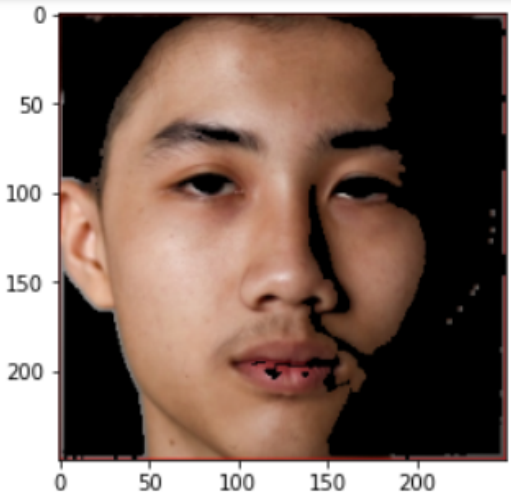
```
def prety_print_data(color_info):
  for x in color_info:
    print(pprint.pformat(x))
    print()
```

We are using all these and finally extracting the dominant color we are giving the input image that are detected and extracted by the harracascade face detector

```
/content/face.jpg (ctrl + click)
image = cv2.imread('/content/face.jpg')

# Resize image to a width of 250
image = imutils.resize(image,width=250)

plt.imshow(cv2.cvtColor(image,cv2.COLOR_BGR2RGB))
plt.show()

skin = extractSkin(image)

plt.imshow(cv2.cvtColor(skin,cv2.COLOR_BGR2RGB))
plt.show()

dominantColors = extractDominantColor(skin,hasThresholding=True)

#Show in the dominant color information
print("Color Information")
prety_print_data(dominantColors)

#Show in the dominant color as bar
print("Color Bar")
colour_bar = plotColorBar(dominantColors)
plt.axis("off")
plt.imshow(colour_bar)
plt.show()
```

The result of the face color extraction gives the cluster index (which cluster the color is under),The percentage of each color in the face image and the color bar also.

[29]




```
Color Information
{'cluster_index': 0,
 'color': [204.9854237288078, 160.84915254235875, 138.4328813559415],
 'color_percentage': 0.38811520810254785}

{'cluster_index': 2,
 'color': [162.53714991762254, 113.56943986820599, 90.28673805601056],
 'color_percentage': 0.3223347576093264}

{'cluster_index': 1,
 'color': [108.76068608766735, 61.013068336508695, 45.93366004174496],
 'color_percentage': 0.28955003428812576}

Color Bar
```