Instructions

Follow the instructions given in comments prefixed with ## and write your code below that.

Also fill the partial code in given blanks.

Don't make any changes to the rest part of the codes

Answer the questions given at the end of this notebook within your report.

You would need to submit your GitHub repository link. Refer to the PDF document for the instructions and details.

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from scipy.spatial import distance
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

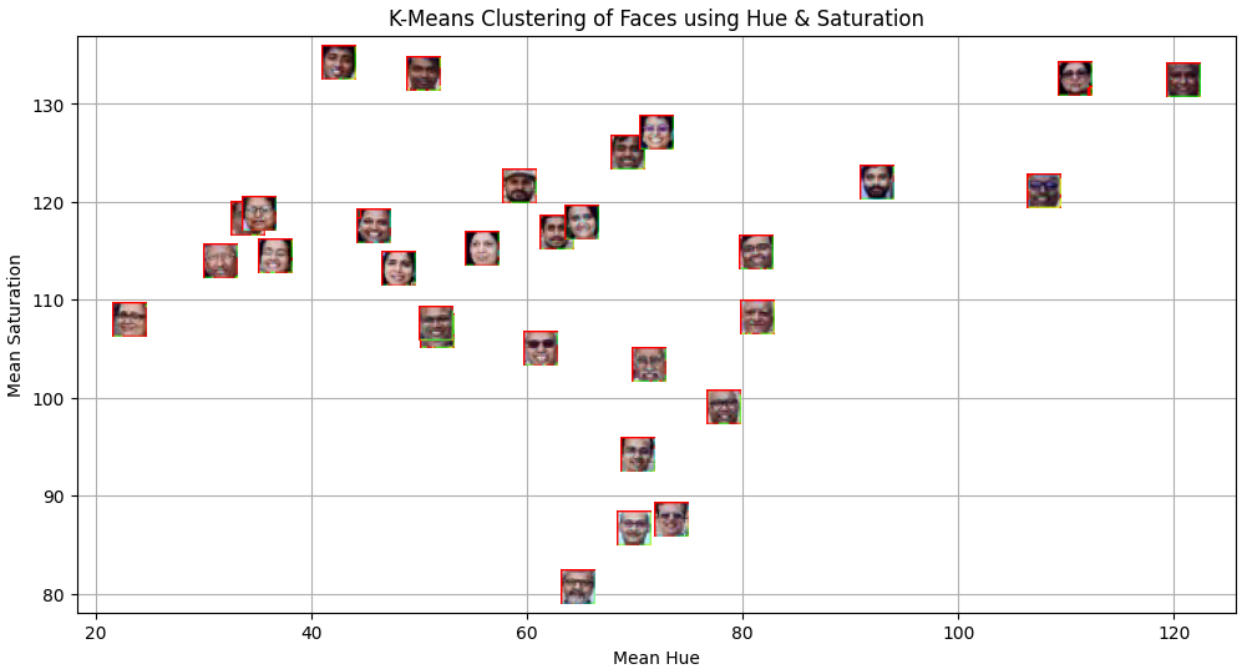
In [2]:
```
## Reading the image plaksha_Faculty.jpg
image = cv2.imread("Plaksha_Faculty.jpg")

## Convert the image to grayscale
gray_img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Loading the required haar-cascade xml classifier file
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_fron

# Applying the face detection method on the grayscale image.
## Change the parameters for better detection of faces in your case.
faces_rect = face_cascade.detectMultiScale(gray_img, 1.05, 4, minSize=(25,25),

# Define the text and font parameters
text = f"Total number of faces detected are {len(faces_rect)}" ## The text you
font = cv2.FONT_HERSHEY_SIMPLEX  ## Font type
font_scale = 1.0  ## Font scale factor
font_color = (0, 0, 255)  ## Text color in BGR format (here, it's red)
font_thickness = 2  ## Thickness of the text


# Iterating through rectangles of detected faces
for (x, y, w, h) in faces_rect:
    cv2.rectangle(image, (x, y), (x+w, y+h), (0, 0, 255), 2)
    # Use cv2.putText to add the text to the image, Use text, font, font_scale
    cv2.putText(image, text, (10, 30), font, font_scale, font_color, font_thic

## Display the image and window title should be "Total number of face detected
cv2.imshow("Total number of face detected are " + str(len(faces_rect)), image)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=waylan
d to run on Wayland anyway.
```

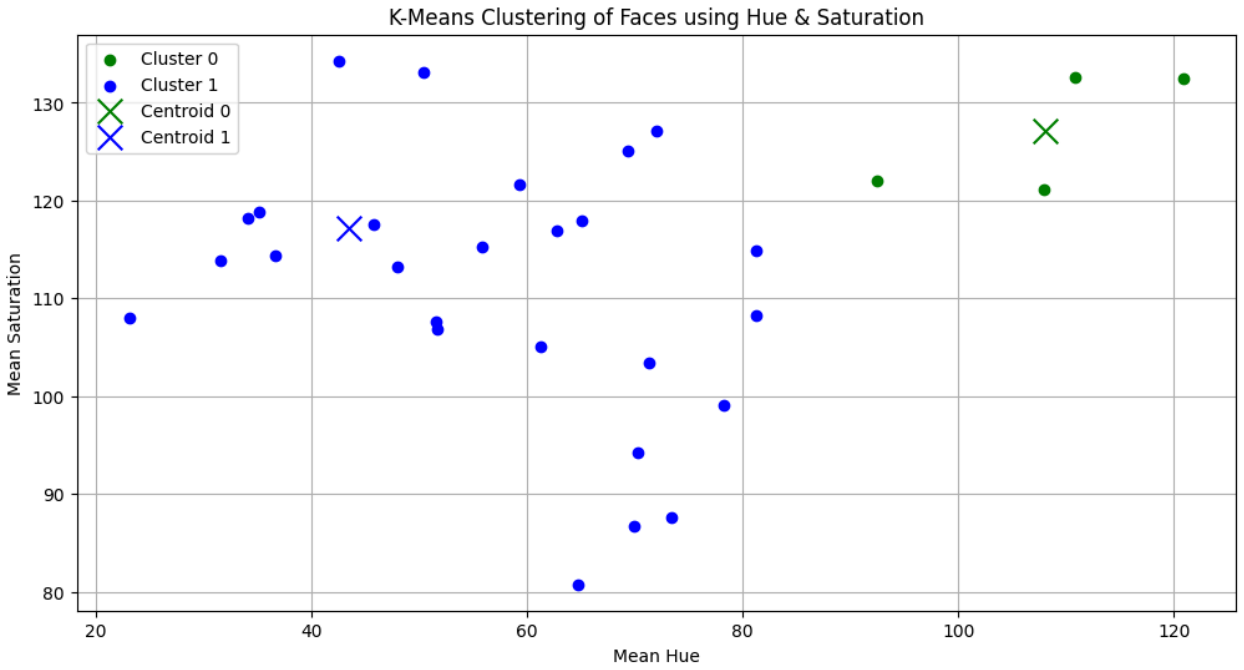
In [4]:

```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
# Extract face region features (Hue and Saturation)
img_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV) ## call the img and convert i
hue_saturation = []
face_images = []  # To store detected face images

for (x, y, w, h) in faces_rect:
    face = img_hsv[y:y + h, x:x + w]
    hue = np.mean(face[:, :, 0])
    saturation = np.mean(face[:, :, 1])
    hue_saturation.append((hue, saturation))
    face_images.append(face)

hue_saturation = np.array(hue_saturation)

## Perform k-Means clustering on hue_saturation and store in kmeans
kmeans = KMeans(n_clusters=3, random_state=0).fit(hue_saturation)
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Create a figure and axis
fig, ax = plt.subplots(figsize=(12, 6))

# Plot the clustered faces with custom markers
for i, (x,y,w,h ) in enumerate(faces_rect):
    im = OffsetImage(cv2.cvtColor(cv2.resize(face_images[i], (20, 20)), cv2.CO
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]), fram
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1])

plt.xlabel("Mean Hue")
plt.ylabel("Mean Saturation")
plt.title("K-Means Clustering of Faces using Hue & Saturation")
plt.grid(True)
plt.show()
```



In [6]:
```
# Create an empty list to store legend labels
legend_labels = []

# Create lists to store points for each cluster
cluster_0_points = []
cluster_1_points = []

# Your code for scatter plot goes here
fig, ax = plt.subplots(figsize=(12, 6))
for i, (x, y, w, h) in enumerate(faces_rect):
    if kmeans.labels_[i] == 0:
        cluster_0_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))


cluster_0_points = np.array(cluster_0_points)
# Plot points for cluster 0 in green
plt.scatter(cluster_0_points[:, 0], cluster_0_points[:, 1], c='green', label='


cluster_1_points = np.array(cluster_1_points)
# Plot points for cluster 1 in blue
plt.scatter(cluster_1_points[:, 0], cluster_1_points[:, 1], c='blue', label='C

# Calculate and plot centroids
centroid_0 = centroids[0]
centroid_1 = centroids[1]

# Plot both the centroid for cluster 0 and cluster 1
plt.scatter(centroid_0[0], centroid_0[1], c='green', marker='x', s=200, label=
plt.scatter(centroid_1[0], centroid_1[1], c='blue', marker='x', s=200, label='
```

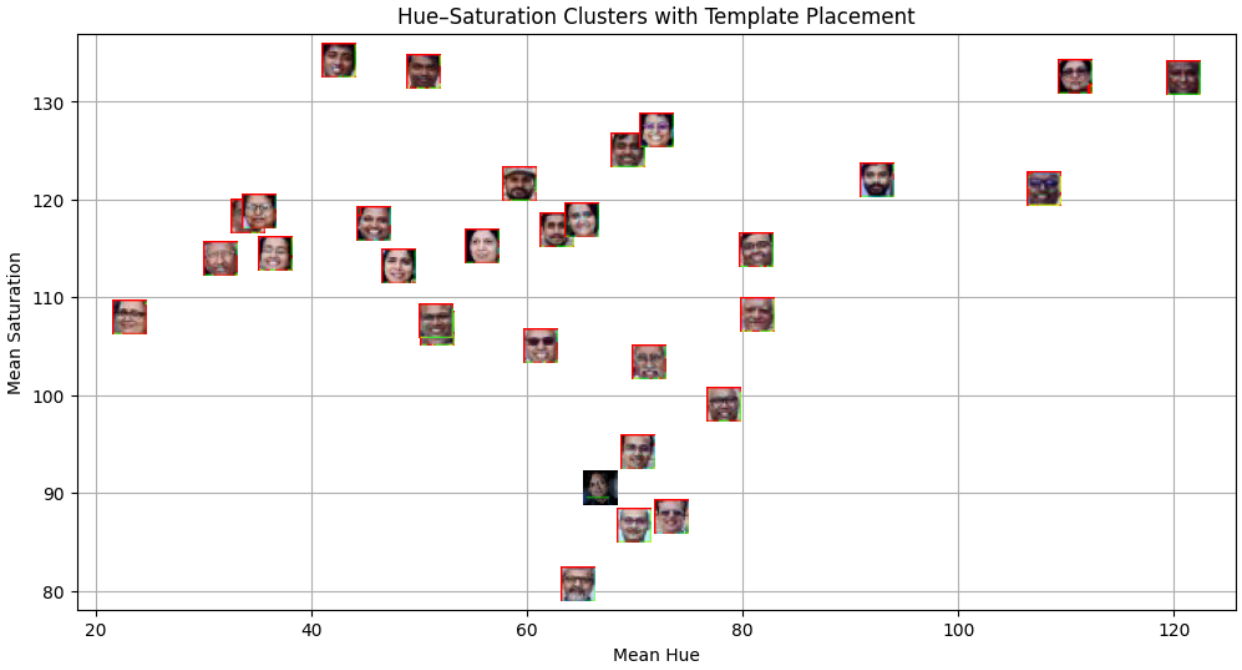
```
plt.xlabel("Mean Hue")
plt.ylabel("Mean Saturation")
plt.title("K-Means Clustering of Faces using Hue & Saturation")
plt.legend()
plt.grid(True)
plt.show()
```



In [7]:
```
## Read the class of the template image 'Dr_Shashi_Tharoor.jpg' using cv2 and
template_img = cv2.imread("Dr_Shashi_Tharoor.jpg")
# Detect face  in the template image after converting it to gray and store it
template_gray = cv2.cvtColor(template_img, cv2.COLOR_BGR2GRAY)
template_faces = face_cascade.detectMultiScale(template_gray, scaleFactor=1.1,
# Draw rectangles around the detected faces
for (x, y, w, h) in template_faces:
    cv2.rectangle(template_img, (x, y), (x + w, y + h), (0, 255, 0), 3)
cv2.imshow("Template Image with Detected Faces", template_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

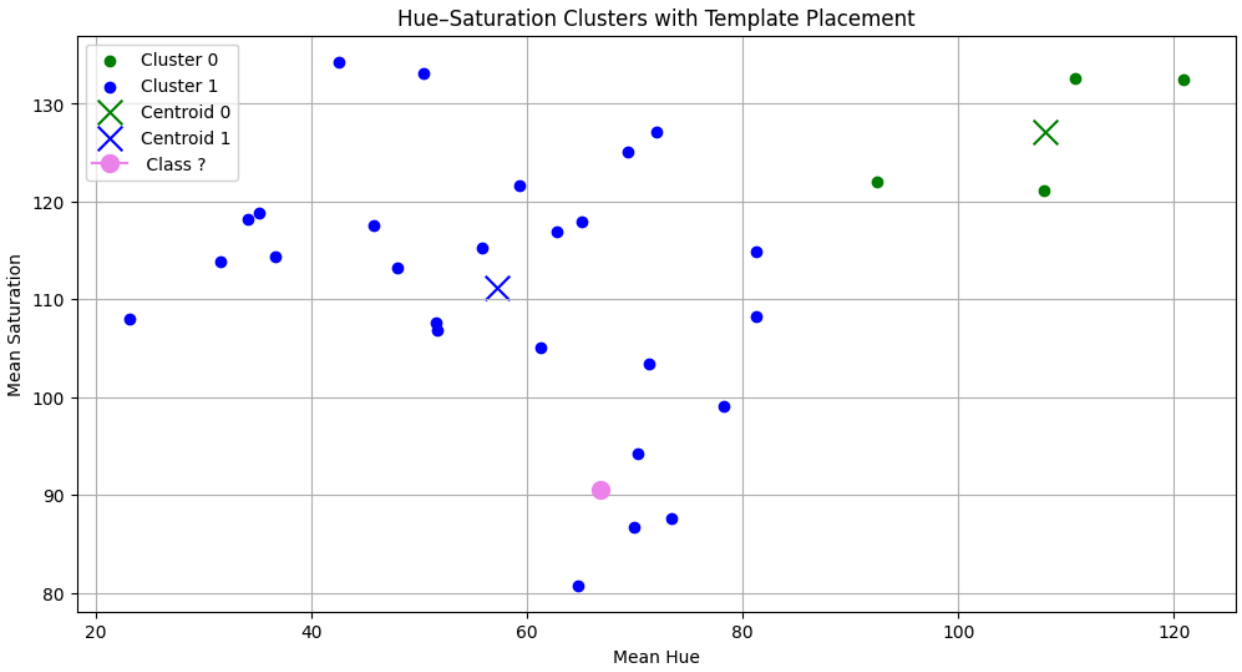
In [9]:
```
# Convert the template image to HSV color space and store it in template_hsv
template_hsv = cv2.cvtColor(template_img, cv2.COLOR_BGR2HSV)

# Extract hue and saturation features from the template image as we did it for
template_hue = np.mean(template_hsv[:, :, 0])
template_saturation = np.mean(template_hsv[:, :, 1])

# Predict the cluster label for the template image and store it in template_la
template_label = kmeans.predict([[template_hue, template_saturation]])[0]

# Create a figure and axis for visualization
fig, ax = plt.subplots(figsize=(12, 6))
```

```
# Plot the clustered faces with custom markers (similar to previous code)
for i, (x, y, w, h) in enumerate(faces_rect):
    color = 'red' if kmeans.labels_[i] == 0 else 'blue'
    im = OffsetImage(cv2.cvtColor(cv2.resize(face_images[i], (20, 20)), cv2.CO
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]), fram
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1], 'o', markersize=5, co

# Plot the template image in the respective cluster
if template_label == 0:
    color = 'red'
else:
    color = 'blue'
im = OffsetImage(cv2.cvtColor(cv2.resize(template_img, (20, 20)), cv2.COLOR_BG
ab = AnnotationBbox(im, (template_hue, template_saturation), frameon=False, pa
ax.add_artist(ab)

## Put x label
plt.xlabel("Mean Hue")
plt.ylabel("Mean Saturation")
plt.title("Hue–Saturation Clusters with Template Placement")
plt.grid(True)
plt.show()
```



In [11]:
```
# Create an empty list to store legend labels
legend_labels = []

# Create lists to store points for each cluster
cluster_0_points = []
cluster_1_points = []

# Your code for scatter plot goes here
fig, ax = plt.subplots(figsize=(12, 6))
```

```
for i, (x, y, w, h) in enumerate(faces_rect):
    if kmeans.labels_[i] == 0:
        cluster_0_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation[i, 1]))

# Plot points for cluster 0 in green
cluster_0_points = np.array(cluster_0_points)
plt.scatter(cluster_0_points[:, 0], cluster_0_points[:, 1], c='green', label='

# Plot points for cluster 1 in blue
cluster_1_points = np.array(cluster_1_points)
plt.scatter(cluster_1_points[:, 0], cluster_1_points[:, 1], c='blue', label='C

# Calculate and plot centroids for both the clusters
centroid_0 = np.mean(cluster_0_points, axis=0)
centroid_1 = np.mean(cluster_1_points, axis=0)
plt.scatter(centroid_0[0], centroid_0[1], c='green', marker='x', s=200, label=
plt.scatter(centroid_1[0], centroid_1[1], c='blue', marker='x', s=200, label='
plt.plot(template_hue, template_saturation, marker='o', c= 'violet',markersize



plt.xlabel("Mean Hue")
plt.ylabel("Mean Saturation")
plt.title("Hue-Saturation Clusters with Template Placement")
plt.legend()
plt.grid(True)
plt.show()
                                        ## End of the lab 5 ##
```



Report:

Answer the following questions within your report:

1. What are the common distance metrics used in distance-based classification algorithms?

A1 Manhattan, Euclidean , Minkowski , Chebyshev , Hamming distance etc

2. What are some real-world applications of distance-based classification algorithms?

A2. Used in pattern matching of pixel features, recommendation systems on social media platforms, spam filtering, healthcare - disease classfification based on data.

3. Explain various distance metrics.

A3. Euclidean distance calculates straight line distance using square root of squared differences.

Manhattan distance adds absolute differences and works well for grid-like movement or high dimensions.

Chebyshev distance takes the maximum difference among coordinates and is useful in chessboard-like movement.(King movement)

Minkowski distance is a general formula with parameter p where p=1 gives Manhattan and p=2 gives Euclidean.

4. What is the role of cross validation in model performance?

A4. Cross validationa evaluates how well a model generalizes to unseen data by splitting data into multiple training and testing sets.

It helps in selecting hyperparameters (e.g., value of k in KNN). It reduces overfitting by ensuring the model performs consistently across subsets.

Its a more reliable estimate of real-world performance and improves model selection and tuning.

5. Explain variance and bias in terms of KNN?

A5. In KNN, small values of k make the model sensitive to noise, resulting in low bias but high variance because it looks at nearby points. Large K values increase bias but reduce

variance. Thus we need to find the optimal k-value.