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.

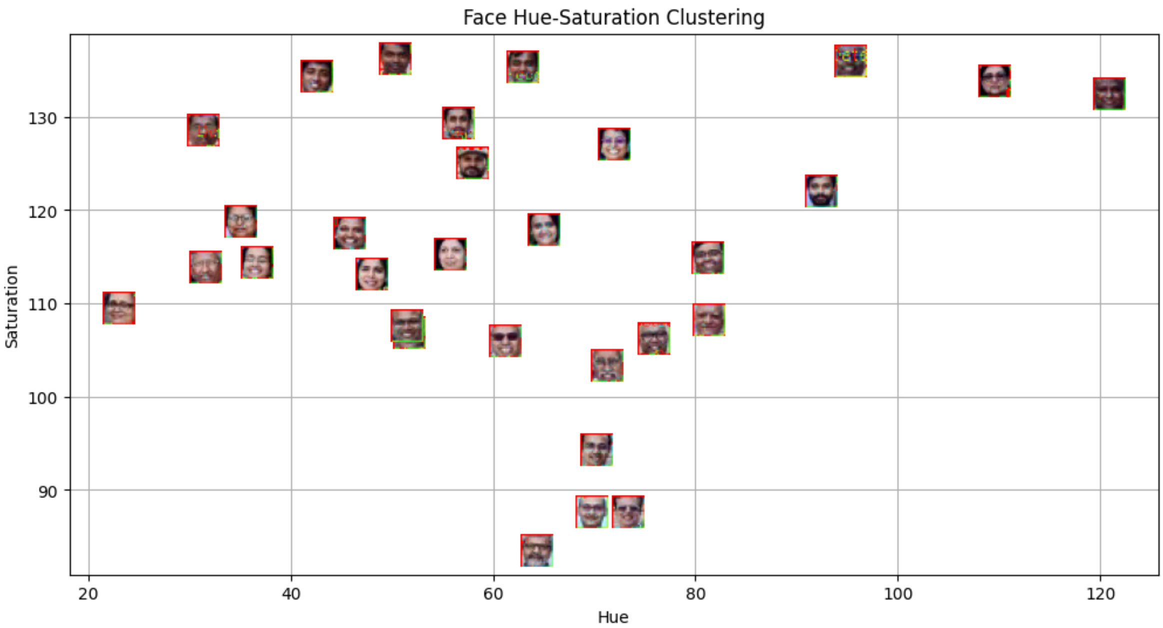
You would need to submit your GitHub repository link. Refer to the Section 6: Final Submission on the PDF document for the details.

In [1]:
```
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
```

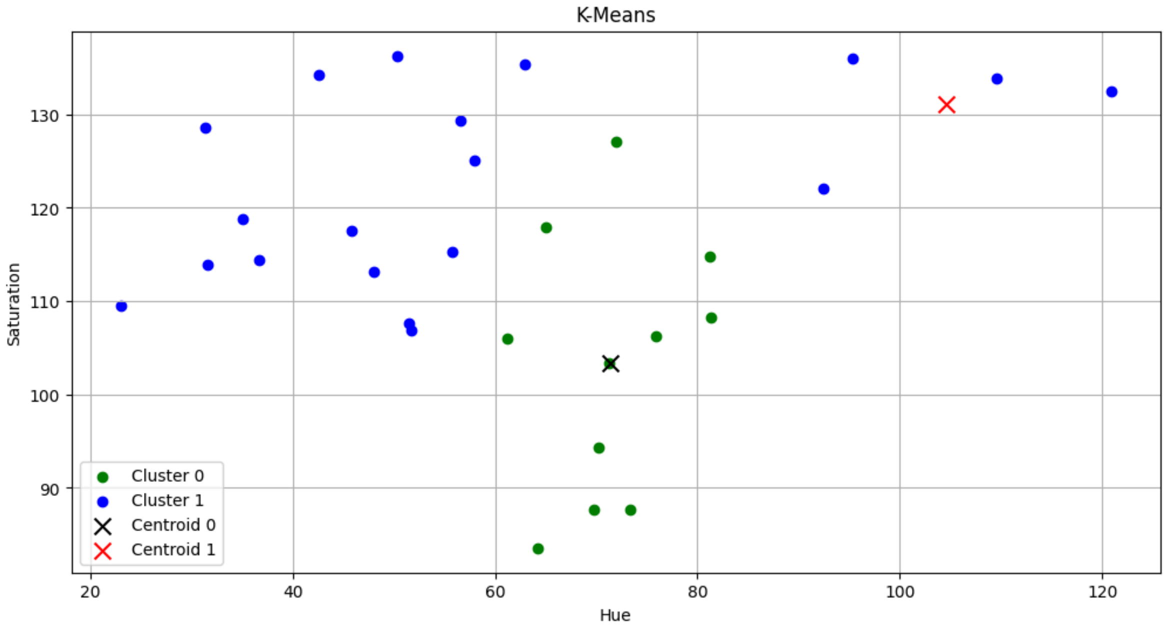
In [2]:
```
## Reading the image plaksha_Faculty.jpg
img = cv2.imread("Plaksha_Faculty.jpg")

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

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

# 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), m

# Define the text and font parameters
text = "Face Detected"  ## The text you want to write
font = cv2.FONT_HERSHEY_SIMPLEX  ## Font type
font_scale = 0.5  ## Font scale factor
font_color = (0, 0, 255)  ## Text color in BGR format (here, it's red)
font_thickness = 1  ## Thickness of the text

# Iterating through rectangles of detected faces
for (x, y, w, h) in faces_rect:
    cv2.rectangle(img, (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(img, text, (x, y - 10), font, font_scale, font_color, font_thick

## Display the image and window title should be "Total number of face detected a
cv2.imshow(f"Total number of faces detected are {len(faces_rect)}", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [4]:
```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox

# Extract face region features (Hue and Saturation)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)  ## call the img and convert it f
```

```
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=42).fit(hue_saturation)

# 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.COLC
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]), framec
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1], 'o', markersize=5)

## Put x label
plt.xlabel("Hue")

## Put y label
plt.ylabel("Saturation")

## Put title
plt.title("Face Hue-Saturation Clustering")

## Put grid
plt.grid(True)

## Show the plot
plt.show()
```



In [9]:
```
# 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], color='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], color='blue', label=

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

# Plot both the centroid for cluster 0 and cluster 1
plt.scatter(centroid_0[0], centroid_0[1], color='black', marker='x', s=100, labe
plt.scatter(centroid_1[0], centroid_1[1], color='red', marker='x', s=100, label=

## Put x label
plt.xlabel("Hue")

## Put y label
plt.ylabel("Saturation")

## Put title
plt.title("K-Means")

## Add a legend
plt.legend()

## Add grid
plt.grid(True)

## Show the plot
plt.show()
```



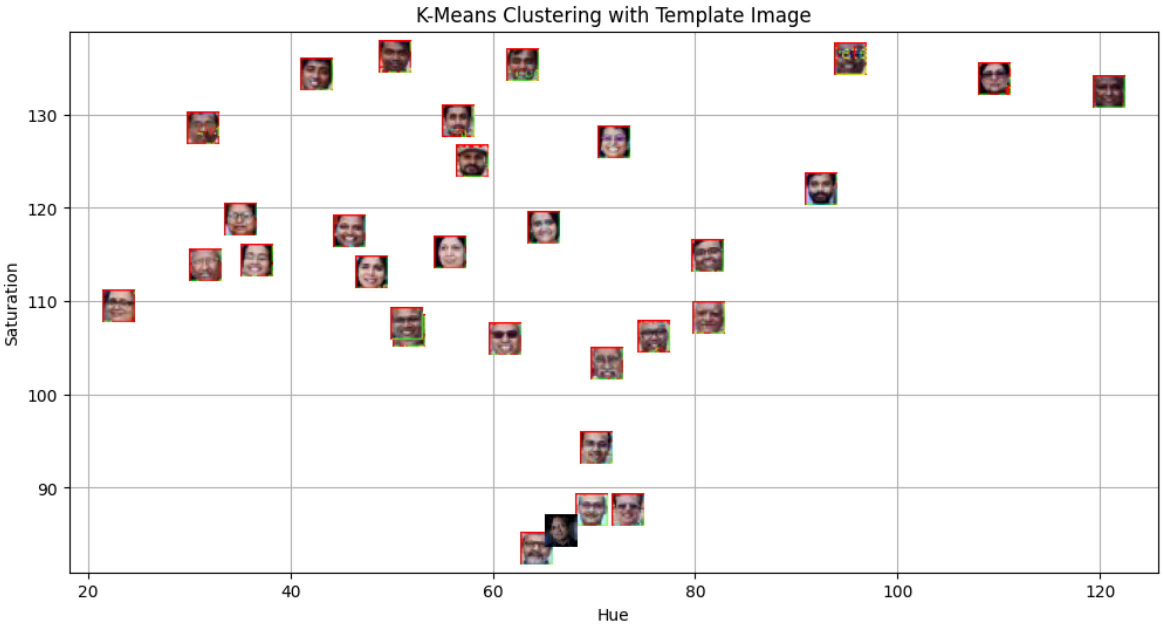
In [10]:

```
## Read the class of the template image 'Dr_Shashi_Tharoor.jpg' using cv2 and st
template_img = cv2.imread("Dr_Shashi_Tharoor.jpg")

# Detect face in the template image after converting it to gray and store it in
template_gray = cv2.cvtColor(template_img, cv2.COLOR_BGR2GRAY)
template_faces = face_cascade.detectMultiScale(template_gray, 1.05, 4, minSize=(

# 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("Detected Faces", template_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [11]:

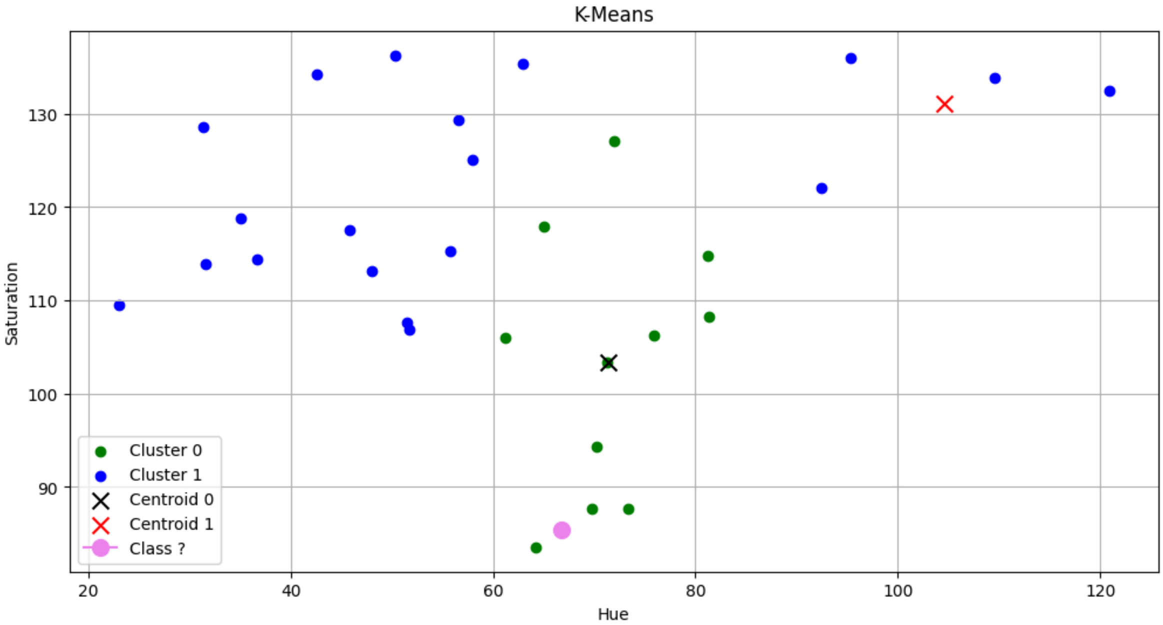
```
# Convert the template image to HSV color space and store it in template_hsv
template_hsv = cv2.cvtColor(template_img, cv2.COLOR_BGR2HSV)

# Extract hue and saturation features from the template image as we did it for d
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_labe
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.COLC
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i, 1]), framec
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1], 'o', markersize=5, colc

# Plot the template image in the respective cluster
color = 'red' if template_label == 0 else 'blue'
im = OffsetImage(cv2.cvtColor(cv2.resize(template_img, (20, 20)), cv2.COLOR_BGR2
ab = AnnotationBbox(im, (template_hue, template_saturation), frameon=False, pad=
```

```
ax.add_artist(ab)
plt.plot(template_hue, template_saturation, 'o', markersize=8, color=color, labe

## Put x label
plt.xlabel("Hue")
## Put y label
plt.ylabel("Saturation")
## Put title
plt.title("K-Means Clustering with Template Image")
## Add grid
plt.grid(True)
## Show plot
plt.show()
```



In [12]:
```
# 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], color='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], color='blue', label=

# Calculate and plot centroids for both the clusters
centroid_0 = kmeans.cluster_centers_[0]
centroid_1 = kmeans.cluster_centers_[1]
```

```
plt.scatter(centroid_0[0], centroid_0[1], color='black', marker='x', s=100, labe
plt.scatter(centroid_1[0], centroid_1[1], color='red', marker='x', s=100, label=

# Plot the template image's hue and saturation with a violet marker
plt.plot(template_hue, template_saturation, marker='o', c='violet', markersize=1

## Put x label
plt.xlabel("Hue")

## Put y label
plt.ylabel("Saturation")

## Put title
plt.title("K-Means")

## Add a legend
plt.legend()

## Add grid
plt.grid(True)

## Show the plot
plt.show()
```



Report:

Answer the following questions within your report:

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

Eucledian distance, Mahalnobis Distance, Hamming distance, cosine distance, manhattan distance, and chebychev distance.

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

Face identification, word simillarity, semantic search, sentiment analysis

3. Explain various distance metrics.

Euclidean Distance – It is the straight-line distance between two points in a multi-dimensional space. It is calculated using the Pythagorean theorem and is widely used in clustering and nearest neighbor algorithms.

Mahalanobis Distance – This distance accounts for correlations between variables and normalizes variations using the covariance matrix. It is useful for detecting outliers and measuring similarity in multivariate distributions.

Hamming Distance – It measures the number of positions where two strings (binary or categorical data) differ. It is commonly used in error detection, cryptography, and information theory.

Cosine Distance – It calculates the cosine of the angle between two vectors, measuring their directional similarity. It is frequently used in text mining and recommendation systems, where magnitude differences are less important.

Manhattan Distance – Also called "Taxicab" distance, it sums the absolute differences between corresponding coordinates. It is useful in grid-based pathfinding, like city block navigation.

Chebyshev Distance – It considers the maximum absolute difference in any coordinate dimension. This metric is useful in chess and industrial quality control, where movements are limited by constraints.

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

Cross-validation helps assess a model's performance by splitting the dataset into multiple subsets for training and testing, reducing overfitting and improving generalization. It provides a reliable estimate of model accuracy by averaging results from different train-test splits. The most common method, k-fold cross-validation, ensures the model performs well on unseen data.

5. Explain variance and bias in terms of KNN?

In KNN, bias is high when K is large, as the model oversimplifies patterns, leading to underfitting. Variance is high when K is small, as the model is overly sensitive to small fluctuations in the training data, leading to overfitting. Choosing an optimal K balances bias and variance for better generalization.

In [ ]:

```
```