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.

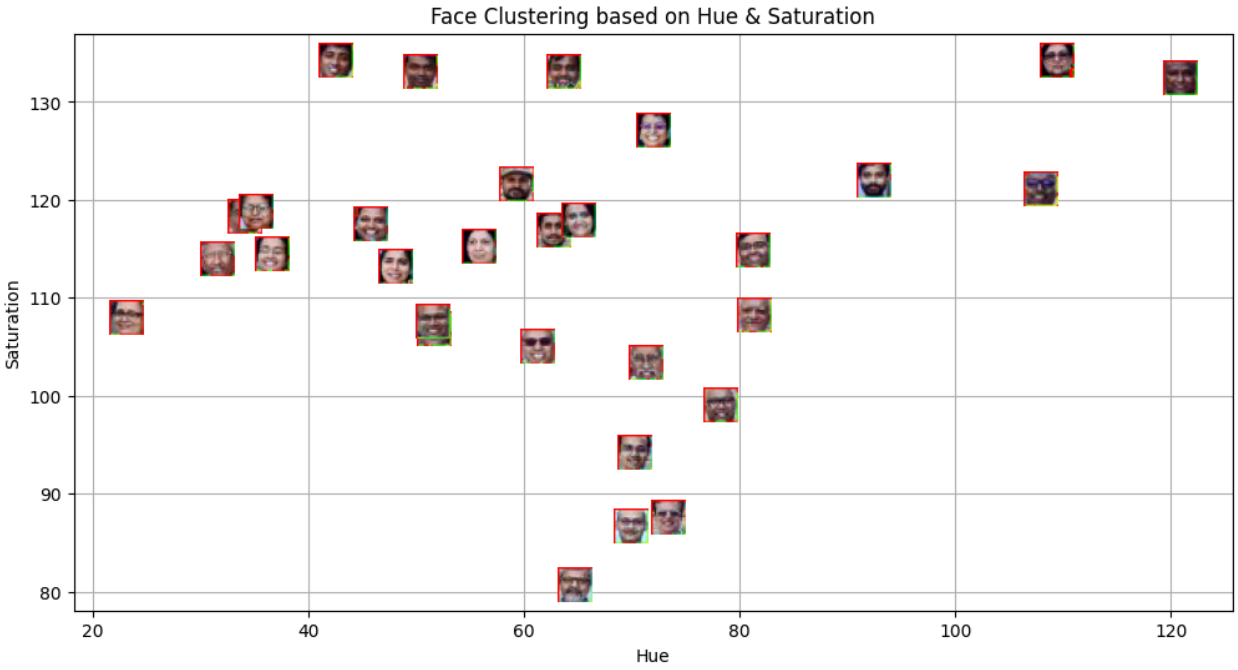
You would need to submit your GitHub repository link. Refer to the PDF document for the instructions and details.

In [8]:
```
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
```

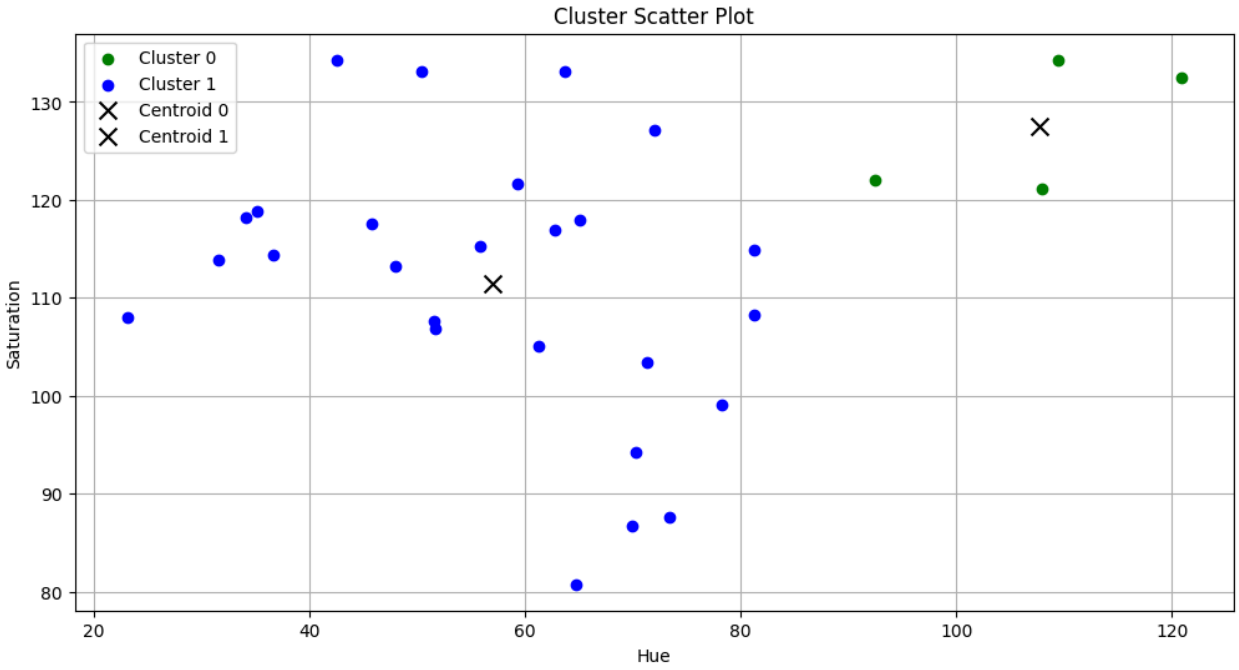
In [9]:
```
## Reading the image plaksha_Faculty.jpg
img = cv2.imread("Plaksha_Faculty.jpg")

## Convert the image to grayscale
img_g = cv2.cvtColor(img,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(img_g, 1.05, 4, minSize=(25,25), ma

# Define the text and font parameters
text = "Face"
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.5
font_color = (0, 0, 255)
font_thickness = 1

# 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)
    cv2.putText(img, text, (x, y-5), font, font_scale, font_color, font_thickn

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

In [10]:

```
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
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=2, random_state=42).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("Hue")
plt.ylabel("Saturation")
plt.title("Face Clustering based on Hue & Saturation")
plt.grid(True)
plt.show()
```



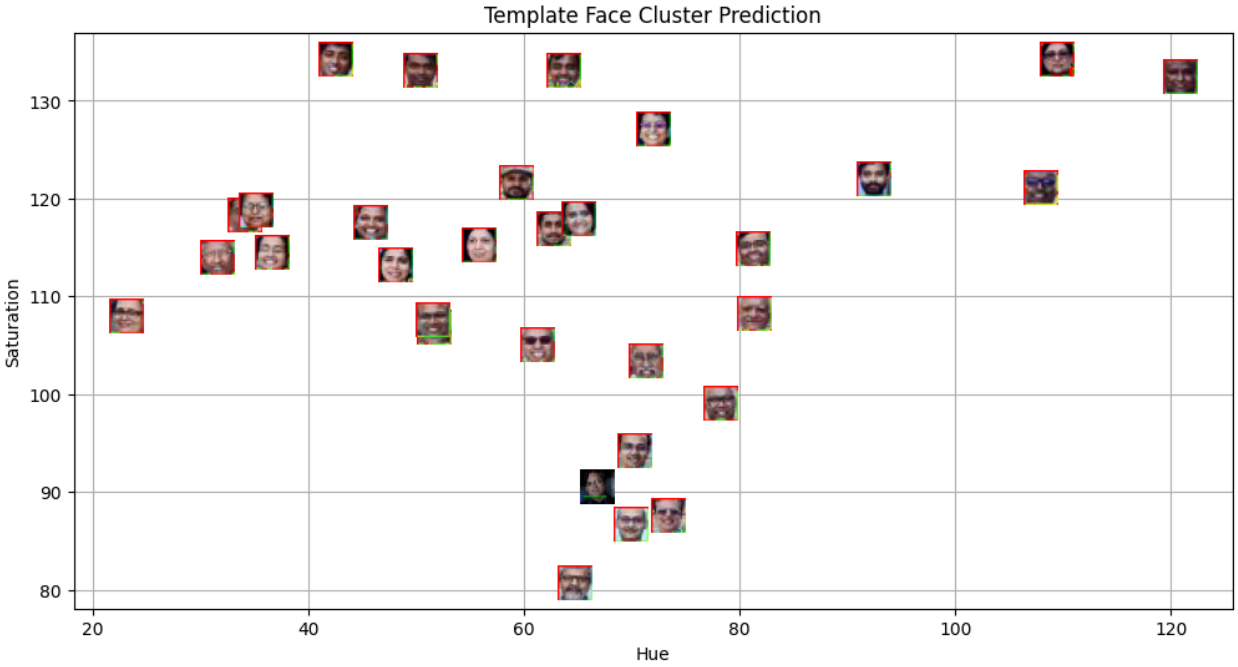
In [11]:
```
# Create an empty list to store legend labels
legend_labels = []

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

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)
plt.scatter(cluster_0_points[:,0], cluster_0_points[:,1], c='green', label='Cl

cluster_1_points = np.array(cluster_1_points)
plt.scatter(cluster_1_points[:,0], cluster_1_points[:,1], c='blue', label='Clu

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='black', marker='x', s=100, label=
plt.scatter(centroid_1[0], centroid_1[1], c='black', marker='x', s=100, label=

plt.xlabel("Hue")
plt.ylabel("Saturation")
plt.title("Cluster Scatter Plot")
plt.legend()
plt.grid(True)
plt.show()
```



In [12]:
```
## Read the class of the template image 'Dr_Shashi_Tharoor.jpg' using cv2 and
template_img = cv2.imread("Dr_Shashi_Tharoor.jpg")

template_gray = cv2.cvtColor(template_img, cv2.COLOR_BGR2GRAY)
template_faces = face_cascade.detectMultiScale(template_gray, 1.1, 4)

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 Face Detection", template_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [13]:
```
# 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
template_hue = np.mean(template_hsv[:, :, 0])
template_saturation = np.mean(template_hsv[:, :, 1])

# Predict the cluster label
template_label = kmeans.predict([[template_hue, template_saturation]])[0]

fig, ax = plt.subplots(figsize=(12, 6))

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

color = 'red' if template_label == 0 else 'blue'
```

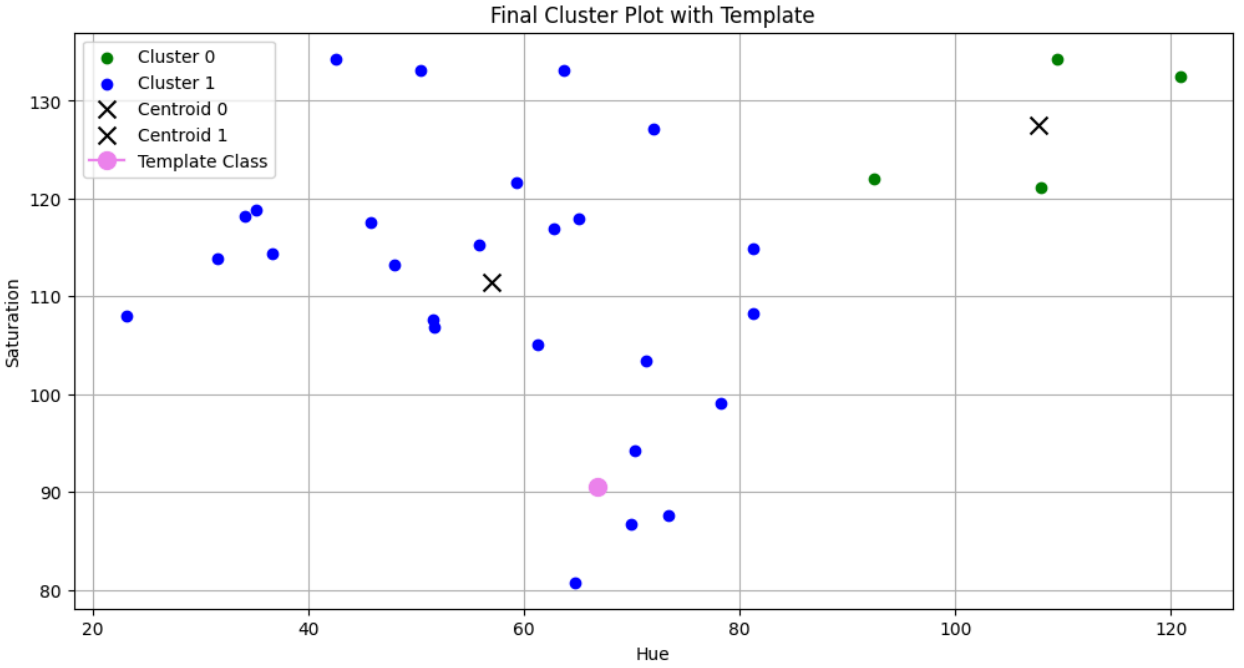
```
im = OffsetImage(cv2.cvtColor(cv2.resize(template_img, (20, 20)), cv2.COLOR_BG
ab = AnnotationBbox(im, (template_hue, template_saturation), frameon=False, pa
ax.add_artist(ab)

plt.xlabel("Hue")
plt.ylabel("Saturation")
plt.title("Template Face Cluster Prediction")
plt.grid(True)
plt.show()
```



In [14]:
```
# Create lists to store points for each cluster
cluster_0_points = []
cluster_1_points = []

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)
plt.scatter(cluster_0_points[:,0], cluster_0_points[:,1], c='green', label='Cl

cluster_1_points = np.array(cluster_1_points)
plt.scatter(cluster_1_points[:,0], cluster_1_points[:,1], c='blue', label='Clu

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='black', marker='x', s=100, label=
plt.scatter(centroid_1[0], centroid_1[1], c='black', marker='x', s=100, label=
```

```
plt.plot(template_hue, template_saturation, marker='o', c='violet', markersize
```

```
plt.xlabel("Hue")
plt.ylabel("Saturation")
plt.title("Final Cluster Plot with Template")
plt.legend()
plt.grid(True)
plt.show()
```



Report:

Answer the following questions within your report:

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

ANS: Euclidean distance Cosine similarity Manhattan distance chebyshev distance Minkowski distance

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

ANS: Face recognition — comparing facial features. Medical diagnosis — comparing patient data. Handwriting recognition — matching patterns in digits.

3. Explain various distance metrics.

ANS : Euclidean Distance--Straight-line distance between two points. Manhattan Distance--Distance measured along horizontal & vertical paths. Cosine Similarity--Measures angle between two vectors.

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

ANS : Cross-validation helps check if a model works well on new data.It helps prevents overfitting and gives a more reliable accuracy estimate.

5. Explain variance and bias in terms of KNN?

ANS : Bias--Error due to overly simple model. In KNN: Large K → High bias

Variance--Error due to sensitivity to data. In KNN: Small K → High variance