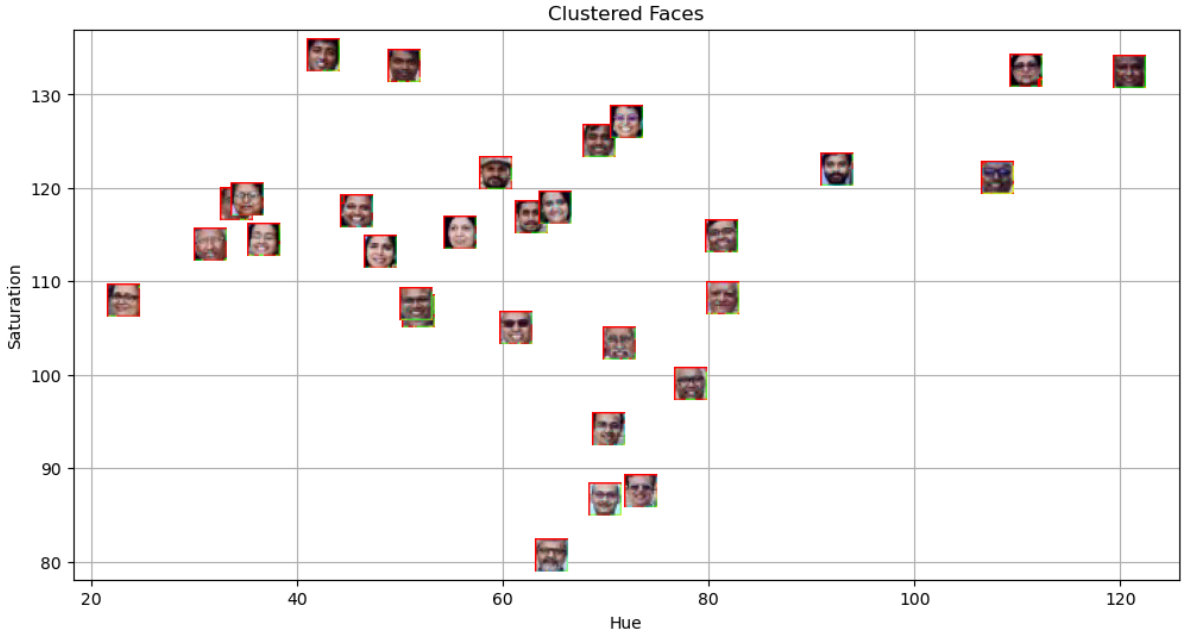
In [3]:
```
## import cv2
import cv2
## import numpy
import numpy as np
## import matplotlib pyplot
import matplotlib.pyplot as plt
## import KMeans cluster from sklearn
from sklearn.cluster import KMeans
## import distance from scipy.spatial
from scipy.spatial import distance
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

In [2]:
```
import cv2
## Reading the image plaksha_Faculty.jpg
img = cv2.imread("C:/Users/Sara Goyal/OneDrive/Desktop/SEM-4(DSEE

## 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 + 'haarcas

# 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=

# Define the text and font parameters
text = "" ## 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,
    cv2.putText(img, text, (x, y-10), font, font_scale, font_color, font

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

In [3]:
```
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
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 kmea
```

```
kmeans = KMeans(n_clusters=2).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)),
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i,
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1])


## Put x label
plt.xlabel('Hue')
## Put y label
plt.ylabel('Saturation')
## Put title
plt.title('Clustered Faces')
## Put grid
plt.grid(True)
## show the plot
plt.show()
```

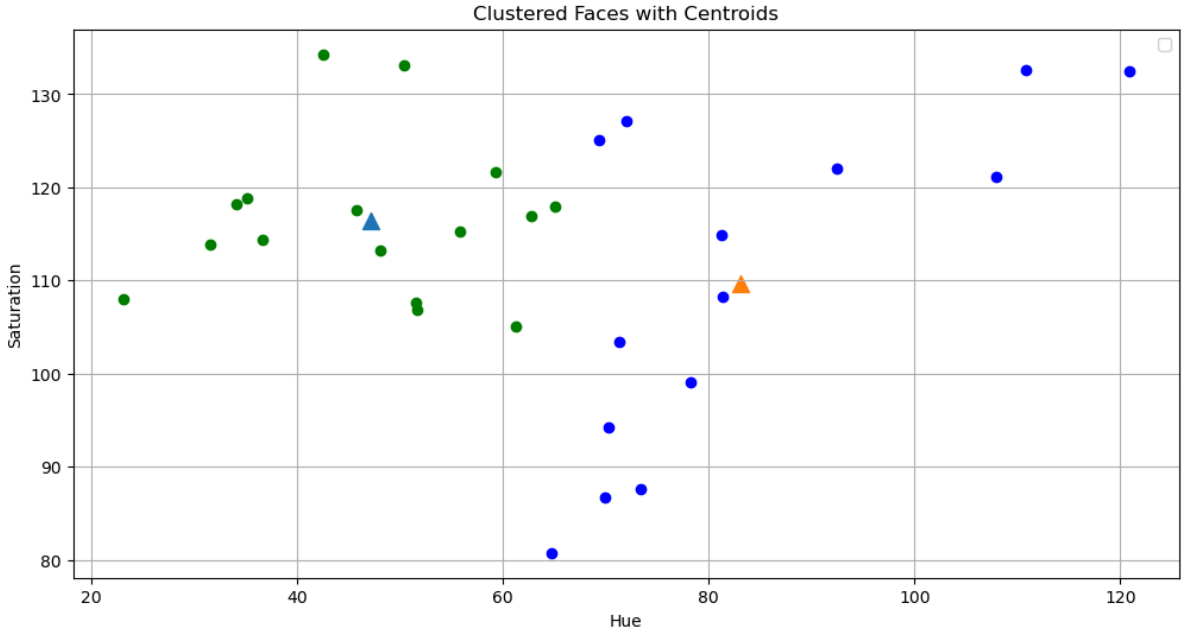
```
C:\Users\Sara Goyal\anaconda3\Lib\site-packages\sklearn\cluste
r\_kmeans.py:870: FutureWarning: The default value of 'n_init'
will change from 10 to 'auto' in 1.4. Set the value of 'n_init' ex
plicitly to suppress the warning
  warnings.warn(
C:\Users\Sara Goyal\anaconda3\Lib\site-packages\sklearn\cluste
r\_kmeans.py:1382: UserWarning: KMeans is known to have a m
emory leak on Windows with MKL, when there are less chunks t
han available threads. You can avoid it by setting the environm
ent variable OMP_NUM_THREADS=1.
  warnings.warn(
```



In [4]:
```
# 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_saturatior
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation


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')


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')

# 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], marker='^', s=100)
plt.scatter(centroid_1[0], centroid_1[1], marker='^', s=100)

## Put x label
plt.xlabel('Hue')
## Put y label
plt.ylabel('Saturation')
## Put title
plt.title('Clustered Faces with Centroids')
## Add a legend
plt.legend()
## Add grid
plt.grid(True)
## Show the plot
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```



In [ ]:
```
## Read the class of the template image 'Dr_Shashi_Tharoor.jpg' usi
template_img = cv2.imread("C:/Users/Sara Goyal/OneDrive/Desktop/SI
# Detect face  in the template image after converting it to gray an
template_gray = cv2.cvtColor(template_img, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcas
template_faces = face_cascade.detectMultiScale(template_gray, 1.1, 4)
# 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 Faces", template_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

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

# Extract hue and saturation features from the template image as v
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 t
template_label = kmeans.predict([[template_hue, template_saturation

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

# Plot the clustered faces with custom markers (similar to previous
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)),
    ab = AnnotationBbox(im, (hue_saturation[i, 0], hue_saturation[i,
    ax.add_artist(ab)
    plt.plot(hue_saturation[i, 0], hue_saturation[i, 1], 'o', markersize=

# 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.C
ab = AnnotationBbox(im, (template_hue, template_saturation), frameo
ax.add_artist(ab)

## Put x label
plt.xlabel('Hue')
## Put y label
plt.ylabel('Saturation')
## Put title
plt.title('Clustered Faces with Template Image')
## Add grid
plt.grid(True)
## show plot
plt.show()
```

In [ ]:
```
# 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_saturatior
    else:
        cluster_1_points.append((hue_saturation[i, 0], hue_saturation

# 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')

# 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')

# 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='red', marker='^', s=100
plt.scatter(centroid_1[0], centroid_1[1], color='purple', marker='^', s=1
plt.plot(template_hue, template_saturation, marker='o', c= 'violet',mar

## Put x label
plt.xlabel('Hue')
## Put y label
plt.ylabel('Saturation')
## Put title
plt.title('Clustered Faces with Centroids')
## Add a legend
plt.legend()
## Add grid
plt.grid(True)
## show the plot
plt.show()
                                                    ## End of the lab
```

1. What are the common distance metrics used in distance-based classification algorithms? Euclidean Distance: This measures the straight-line (shortest) distance between two points or vectors in a space, representing the most direct path. Manhattan Distance: This calculates the distance between two points by summing the absolute differences of their coordinates along the axes, as if moving only horizontally and vertically. Mahalanobis Distance: This quantifies how far a point is from the mean of a distribution, expressed in terms of standard deviations, accounting for the distribution's covariance.
2. What are some real-world applications of distance-based classification algorithms? Object Detection and Face Recognition: In computer vision, these methods classify images or image regions based on visual characteristics, such as identifying faces or objects in photos. Spam Classification: By transforming text into feature vectors, distance-based techniques can distinguish between spam and non-spam emails or documents effectively.
3. Explain various distance metrics. Euclidean Distance: It represents the shortest, straight-line distance between two vectors, often visualized as the hypotenuse in a right triangle formed by the points. Chebyshev Distance: Defined in a vector space, this metric measures the distance between two vectors as the maximum absolute difference across any of their coordinate dimensions. Formula: (\max(|x_a - x_b|, |y_a - y*b|)). *Minkowski Distance: This is a flexible, generalized distance metric that can adapt by varying the*

parameter (p). It calculates the distance between two points as ((\sum{i=1}^{n} |x_i - y_i|^p)^{1/p}), where (p) determines the specific form (e.g., (p=2) for Euclidean, (p=1) for Manhattan).

4. What is the role of cross validation in model performance? Preventing Overfitting: Cross-validation evaluates a model across multiple data splits, ensuring its performance isn't skewed by a single favorable partition, thus reducing the risk of overfitting. Choosing the Best Model: Cross-validation provides a standardized evaluation framework, enabling fair comparisons between different models or algorithms to identify the most robust and generalizable option
5. Explain variance and bias in terms of KNN? Bias: This refers to the tendency of a model to skew predictions, indicating oversimplification. A high bias occurs when the model is too basic, potentially underfitting the data. For instance, with a large (k) in KNN, the model averages over many neighbors, leading to overly smooth decision boundaries that might ignore local patterns. Variance: This describes the model's sensitivity to changes in the training data, reflecting the spread of predictions. High variance indicates that the model is overly responsive to noise, risking overfitting. For example, a small (k) in KNN makes the model highly dependent on nearby points, causing significant shifts in the decision boundary with minor data variations.

In [ ]: