فاز ۱ : Feature Extraction

در این بخش ابتدا مقادیر فایل features.csv رابه یک فایل csv دیگر انتقال داده تا فیچر ها رنگ پوست و رنگ چشم را به آن اضفه بکنیم.

```
[ ] import pandas as pd

    # Paths to the source and destination CSV files
    datasetPath1 = '/content/features.csv'
    datasetPath2 = '/content/features2.csv'

    # Read the source CSV file into a DataFrame
    features_df = pd.read_csv(datasetPath1)

    # Write the DataFrame to the destination CSV file
    features_df.to_csv(datasetPath2, index=False)
```

در این بخش برای پیدا کردن تعداد مناسب پراسس بر اساس coreهای موجود که برابر ۲ برابر تعداد coreهاست، با استفاده از کتابخانه های multiprocessing و psutil تعداد coreهای در دسترس و میزان استفاده از آنها مشخص می شود.

```
[5] import multiprocessing
    num_cores = multiprocessing.cpu_count()
    print(f'Number of available CPU cores: {num_cores}')

    Number of available CPU cores: 2

    import psutil

    # Get the number of logical CPUs
    cpu_count = psutil.cpu_count()

    # Get per-core CPU usage
    per_core_usage = psutil.cpu_percent(interval=1, percpu=True)

    print(f"Number of logical CPUs: {cpu_count}")
    print("Per-core CPU usage:")
    for i, usage in enumerate(per_core_usage):
        print(f"Core {i}: {usage}%")

    Number of logical CPUs: 2
    Per-core CPU usage:
    Core 0: 5.0%
    Core 1: 5.0%
```

کتابخانه‌های مورد نیاز برای پردازش تصویر، محاسبات و چندرشته‌ای(multiprocessing)

```
import os
import cv2
import pandas as pd
import numpy as np
from mtcnn import MTCNN
from skimage.color import rgb2lab, lab2rgb
from multiprocessing import Pool, cpu_count
```

تابع تبدیل رنگ پوست به سایه‌های واقعی

این تابع یک رنگ RGB را گرفته و به رنگ واقعی پوست در مدل رنگی LAB تبدیل می‌کند:

```
# Function to map skin color shades to realistic colors
def map_skin_color_to_realistic_shade(average_rgb):
    lab_skin = rgb2lab(np.array([[average_rgb / 255]]))[0][0]
    L, a, b = lab_skin
    if L < 35:
        a, b = 18, 18
    elif L < 60:
        a, b = 15, 20
    else:
        a, b = 10, 25
    realistic_skin_lab = np.array([[L, a, b]])
    mapped_rgb = lab2rgb(realistic_skin_lab) * 255
    return mapped_rgb[0]
```

تابع مقیاس‌بندی RGB به یک مقدار واحد

این تابع مقدار RGB را به یک مقدار واحد بین ۰ و ۱ مقیاس‌بندی می‌کند:

```
# Function to scale RGB values to a single value between 0 and 1
def scale_rgb_to_single_value(rgb):
    return np.mean(rgb) / 255
```

تابع پردازش تصویر

این تابع تصویر را بارگذاری کرده، ابتدا با استفاده ازتابع MTCNN() صورت را دیتکت کرده و بر اساس مختصات آن نواحی چشم و گونه(برای تشخیص رنگ پوست) را شناسایی کرده و رنگ متوسط آن‌ها را محاسبه می‌کند:

```
def process_image(args):
    index, row, dataset_path = args
    image_name = row['image_id']
    detector = MTCNN()

    image_path = os.path.join(dataset_path, image_name)
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error: Could not read the image {image_name}.")
        return index, None, None

    rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    faces = detector.detect_faces(rgb_img)

    if faces:
        face = faces[0]
        x, y, width, height = face['box']

        # Calculate eye color
        left_eye_x_start, left_eye_y_start = x + int(width * 0.1), y + int(height * 0.3)
        eye_width, eye_height = width // 3, height // 5
        left_eye_roi = img[left_eye_y_start:left_eye_y_start + eye_height,
                           left_eye_x_start:left_eye_x_start + eye_width]
        right_eye_x_start, right_eye_y_start = x + int(width * 0.6), y + int(height * 0.3)
        right_eye_roi = img[right_eye_y_start:right_eye_y_start + eye_height,
                            right_eye_x_start:right_eye_x_start + eye_width]
        average_eye_rgb = (np.mean(left_eye_roi, axis=(0, 1)) + np.mean(right_eye_roi, axis=(0, 1))) / 2
        eye_color_single_value = scale_rgb_to_single_value(np.round(average_eye_rgb).astype(int))
```

```
        # Calculate skin color
        left_cheek_x_start, left_cheek_y_start = x + int(width * 0.2), y + int(height * 0.5)
        right_cheek_x_start, right_cheek_y_start = x + int(width * 0.6), y + int(height * 0.5)
        left_cheek_roi = img[left_cheek_y_start:left_cheek_y_start + int(height * 0.2),
                             left_cheek_x_start:left_cheek_x_start + int(width * 0.2)]
        right_cheek_roi = img[right_cheek_y_start:right_cheek_y_start + int(height * 0.2),
                              right_cheek_x_start:right_cheek_x_start + int(width * 0.2)]
        average_cheek_rgb = (np.mean(left_cheek_roi, axis=(0, 1)) + np.mean(right_cheek_roi, axis=(0, 1))) / 2
        mapped_cheek_rgb = map_skin_color_to_realistic_shade(average_cheek_rgb)
        skin_color_single_value = scale_rgb_to_single_value(np.round(mapped_cheek_rgb).astype(int))

        return index, eye_color_single_value, skin_color_single_value

    return index, None, None

def skincolor_eyecolor_to_dataset(dataset_path, csv_path, batch_size=500):
    # Load the existing dataset
    if os.path.exists(csv_path):
        features_df = pd.read_csv(csv_path)
        if 'eye_color' not in features_df.columns or 'skin_color' not in features_df.columns:
            features_df['eye_color'] = None
            features_df['skin_color'] = None
    else:
        print("CSV file not found.")
        return

    num_cores = cpu_count()
    num_processes = min(20, num_cores * 2)
    print(f"Using {num_processes} processes for multiprocessing.")
```

```
    num_images = len(features_df)
    num_epochs = (num_images + batch_size - 1) // batch_size

    for epoch in range(num_epochs):
        print(f"Starting epoch {epoch + 1}/{num_epochs}")

        start_idx = epoch * batch_size
        end_idx = min(start_idx + batch_size, num_images)
        batch_df = features_df.iloc[start_idx:end_idx]

        args = [(index, row, dataset_path) for index, row in batch_df.iterrows()]

        # Use multiprocessing to process images
        with Pool(processes=num_processes) as pool:
            results = pool.map(process_image, args)

        # Update DataFrame with results
        for index, eye_color, skin_color in results:
            if eye_color is not None and skin_color is not None:
                features_df.at[index, 'eye_color'] = eye_color
                features_df.at[index, 'skin_color'] = skin_color

        # Save updated DataFrame to CSV
        features_df.to_csv(csv_path, index=False)
        print(f"Epoch {epoch + 1} completed. Processed {end_idx - start_idx} images. CSV file updated.")

    print("All epochs completed. CSV file updated with eye color and skin color for all images.")

dataset_path = '/content/drive/MyDrive/dataset'
csv_path = '/content/drive/MyDrive/features2.csv'
```

```
dataset_path = '/content/drive/MyDrive/dataset'
csv_path = '/content/drive/MyDrive/features2.csv'
skincolor_eyecolor_to_dataset(dataset_path, csv_path, batch_size=500)
```

در این بخش دیتاست را به ایپاک های ۵۰۰ تایی تقسیم کرده و پس از پیدا کردن فیچر ها ، رنگ پوست و چشم آنها را به دیتاست اضافه میکنیم. برای تسریع انجام این کار به دلیل سنگینی محاسبات دیتاست ۵۰ هزارتایی از multiprocessingاستفاده می کنیم.

خروجی : فایل features.csv که برای هر عکس رنگ پوست و چشم محاسبه شده است. همچنین برای اطمینان ازپراسس شدن هر ایپاک لاگ گرفته شده است.

در این بخش روی یک فایل تست چشم ها، صورت، نواحی گونه که برای رنگ پوست پیدا شده اند تشخیص داده می شود. همچنین تشخیص چشم ها با harcascade نیز پیاده شده است تا مقایسه ای انجام شود که برای اکثر عکس ها تشخیص داده نشده است.

```
    # Get a list of all image files in the folder
    imageFiles = [f for f in os.listdir(folderPath) if f.endswith(('.png', '.jpg', '.jpeg'))]

    # List to store the names of images where no face is detected
    no_face_detected_images = []

    # Initialize MTCNN detector
    detector = MTCNN()

    # List to store the coordinates of detected faces for averaging
    all_faces_coordinates = []

    # Load Haar Cascade for eye detection
    eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')

    # Function to map skin color shades to realistic colors
    def map_skin_color_to_realistic_shade(average_rgb):
        lab_skin = rgb2lab(np.array([[average_rgb / 255]]))[0][0]
        L, a, b = lab_skin
        if L < 35:
            a, b = 18, 18
        elif L < 60:
            a, b = 15, 20
        else:
            a, b = 10, 25
        realistic_skin_lab = np.array([[L, a, b]])
        mapped_rgb = lab2rgb(realistic_skin_lab) * 255
        return mapped_rgb[0]
```

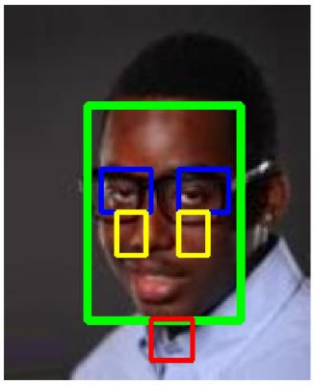
```
# Function to scale RGB values to a single value between 0 and 1
def scale_rgb_to_single_value(rgb):
    return np.mean(rgb) / 255

def skincolor_eyecolor_to_dataset(folderPath):
    if not imageFiles:
        print("Error: No images found in the folder.")
    else:
        for imageFile in imageFiles:
            imagePath = os.path.join(folderPath, imageFile)
            print(f"Processing image: {imageFile}")

            img = cv2.imread(imagePath)
            if img is None:
                print(f"Error: Could not read the image {imageFile}.")
                continue

            rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            faces = detector.detect_faces(rgb_img)

            if faces:
                for face in faces:
                    x, y, width, height = face['box']
                    cv2.rectangle(img, (x, y), (x + width, y + height), (0, 255, 0), 4)

                    # Manually specified eye regions
                    eye_width, eye_height = width // 3, height // 5
                    left_eye_x_start, left_eye_y_start = x + int(width * 0.1), y + int(height * 0.3)
                    cv2.rectangle(img, (left_eye_x_start, left_eye_y_start),
```

```
                left_cheek_x_end, left_cheek_y_end = left_cheek_x_start + int(width * 0.2), left_cheek_y_start + int(height * 0.2)
                cv2.rectangle(img, (left_cheek_x_start, left_cheek_y_start), (left_cheek_x_end, left_cheek_y_end), (0, 255, 255), 2)

                right_cheek_x_start, right_cheek_y_start = x + int(width * 0.6), y + int(height * 0.5)
                right_cheek_x_end, right_cheek_y_end = right_cheek_x_start + int(width * 0.2), right_cheek_y_start + int(height * 0.2)
                cv2.rectangle(img, (right_cheek_x_start, right_cheek_y_start), (right_cheek_x_end, right_cheek_y_end), (0, 255, 255), 2)

                left_cheek_roi = img[left_cheek_y_start:left_cheek_y_end, left_cheek_x_start:left_cheek_x_end]
                right_cheek_roi = img[right_cheek_y_start:right_cheek_y_end, right_cheek_x_start:right_cheek_x_end]

                average_left_cheek_rgb = np.mean(left_cheek_roi, axis=(0, 1))
                average_right_cheek_rgb = np.mean(right_cheek_roi, axis=(0, 1))
                average_cheek_rgb = (average_left_cheek_rgb + average_right_cheek_rgb) / 2
                mapped_cheek_rgb = map_skin_color_to_realistic_shade(average_cheek_rgb)

                print(f"Average eye color (R, G, B): {average_eye_rgb}")
                print(f"Realistic average cheek color (R, G, B): {mapped_cheek_rgb}")

                eye_color_single_value = scale_rgb_to_single_value(np.round(average_eye_rgb).astype(int))
                skin_color_single_value = scale_rgb_to_single_value(np.round(mapped_cheek_rgb).astype(int))
            else:
                no_face_detected_images.append(imageFile)

            img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            plt.figure(figsize=(20, 10))
            plt.imshow(img_rgb)
            plt.axis('off')
            plt.show()
```

```
                              (left_eye_x_start + eye_width, left_eye_y_start + eye_height),
                              (255, 0, 0), 2)

                right_eye_x_start, right_eye_y_start = x + int(width * 0.6), y + int(height * 0.3)
                cv2.rectangle(img, (right_eye_x_start, right_eye_y_start),
                              (right_eye_x_start + eye_width, right_eye_y_start + eye_height),
                              (255, 0, 0), 2)

                # Detect eyes with OpenCV's CascadeClassifier
                gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                detected_eyes = eye_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=5)

                # Draw rectangles around detected eyes
                for (ex, ey, ew, eh) in detected_eyes:
                    cv2.rectangle(img, (ex, ey), (ex + ew, ey + eh), (0, 0, 255), 2)

                left_eye_roi = img[left_eye_y_start:left_eye_y_start + eye_height,
                                   left_eye_x_start:left_eye_x_start + eye_width]
                right_eye_roi = img[right_eye_y_start:right_eye_y_start + eye_height,
                                    right_eye_x_start:right_eye_x_start + eye_width]

                average_left_eye_rgb = np.mean(left_eye_roi, axis=(0, 1))
                average_right_eye_rgb = np.mean(right_eye_roi, axis=(0, 1))
                average_eye_rgb = (average_left_eye_rgb + average_right_eye_rgb) / 2

                left_cheek_x_start, left_cheek_y_start = x + int(width * 0.2), y + int(height * 0.5)
                left_cheek_x_end, left_cheek_y_end = left_cheek_x_start + int(width * 0.2), left_cheek_y_start + int(height * 0.2)
                cv2.rectangle(img, (left_cheek_x_start, left_cheek_y_start), (left_cheek_x_end, left_cheek_y_end), (0, 255, 255), 2)
```

```
            plt.imshow(img_rgb)
            plt.axis('off')
            plt.show()

    if no_face_detected_images and all_faces_coordinates:
        avg_face_coordinates = np.mean(all_faces_coordinates, axis=0).astype(int)
        avg_x, avg_y, avg_width, avg_height = avg_face_coordinates
        print(f"Average face coordinates: {avg_face_coordinates}")

        for imageFile in no_face_detected_images:
            imagePath = os.path.join(folderPath, imageFile)
            img = cv2.imread(imagePath)
            if img is not None:
                cv2.rectangle(img, (avg_x, avg_y), (avg_x + avg_width, avg_y + avg_height), (255, 0, 0), 4)
                img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                plt.figure(figsize=(10, 5))
                plt.imshow(img_rgb)
                plt.axis('off')
                plt.show()

    if no_face_detected_images:
        print("Images with no faces detected:", no_face_detected_images)
    else:
        print("Faces detected in all images.")

skincolor_eyecolor_to_dataset(folderPath)
```

در عکس سمت راست چشم توسط کتابخانه اشتباه و عکس سمت به کل تشخیص نداده شده است.



فاز ۲: Feature Selection

۱. بارگزاری داده ها از فایل CSV:

هدف: این قسمت فایل features.csv را باز می‌کند و مقادیر آن را به‌صورت عددی در data ذخیره می‌کند.

جزئیات:

feature_names: نام ویژگی‌ها را از ردیف اول (سر ستون) استخراج کرده و ستون اول (که نام فایل‌هاست) را نادیده می‌گیرد.

data: داده‌های هر ردیف از فایل CSV را (بدون ستون نام فایل) به‌صورت لیست‌های عددی در data ذخیره می‌کند.

```
2. data = []
3. feature_names = []
4.
5. # Open the CSV file and extract feature names and data
6. with open('features.csv', mode='r') as file:
7.     reader = csv.reader(file)
8.     feature_names = next(reader)[1:]  # Get feature names from the
   header row, skipping the first column (filenames)
9.
10.          for row in reader:
11.              # Extract integer values from each row, skipping the
   filename
12.              int_values = [int(value) for value in row[1:]]  #
   Start from index 1 to skip the filename
13.              data.append(int_values)
```

۲. تبدیل داده ها برای دسترسی آسان به هر ویژگی:

```
data_transposed = list(zip(*data))
```

۳. محاسبه میانگین هر ویژگی:

```
means = [sum(feature) / len(feature) for feature in data_transposed]
```

۴. توابع کمکی برای محاسبه کوواریانس، واریانس و همبستگی پیرسون:

هدف :این توابع به‌منظور محاسبه آماری و همبستگی بین دو ویژگی استفاده می‌شوند.

- `covariance`: محاسبه کوواریانس بین دو ویژگی.
- `variance`: محاسبه واریانس یک ویژگی خاص.
- `correlation`: محاسبه همبستگی پیرسون بین دو ویژگی با استفاده از کوواریانس و انحراف استاندارد آن‌ها.

```
def covariance(X, Y, mean_X, mean_Y):
    return sum((X[i] - mean_X) * (Y[i] - mean_Y) for i in range(len(X))) /
len(X)

# Function to calculate variance of a feature
def variance(X, mean_X):
    return sum((x - mean_X) ** 2 for x in X) / len(X)

# Function to calculate Pearson correlation between two features
def correlation(X, Y, mean_X, mean_Y):
    cov = covariance(X, Y, mean_X, mean_Y)
    std_X = math.sqrt(variance(X, mean_X))
    std_Y = math.sqrt(variance(Y, mean_Y))
    return cov / (std_X * std_Y)
```

۵. محاسبه ماتریس correlation و چاپ آن:

```
correlation_matrix = []
for i in range(len(data_transposed)):
    row = []
    for j in range(len(data_transposed)):
        if i == j:
            row.append(1)  # Correlation with itself is 1
        else:
            correlation_value = correlation(data_transposed[i],
data_transposed[j], means[i], means[j])
            row.append(round(correlation_value, 2))
    correlation_matrix.append(row)
```

```
# Print the correlation matrix
print("Correlation Matrix:")
for row in correlation_matrix:
    print(row)
```

۶. محاسبه مجموع absolute correlation برای هر ویژگی:

```
absolute_correlation_sums = []
for i in range(len(correlation_matrix)):
    abs_sum = sum([abs(correlation_matrix[i][j]) for j in
range(len(correlation_matrix))])
    absolute_correlation_sums.append((i, abs_sum))
```

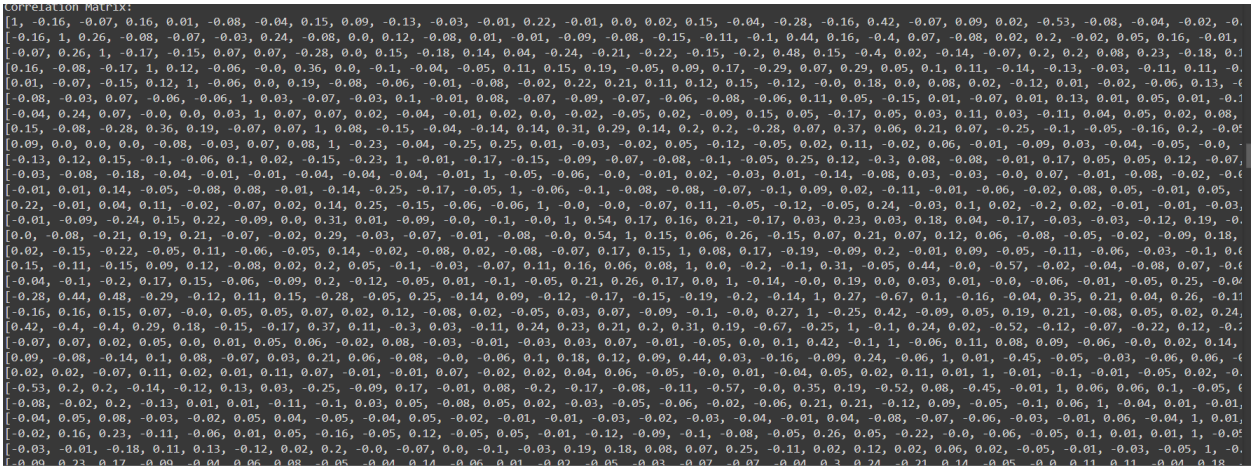
۷. انتخاب ویژگی‌ها با استفاده از thresholdهای مختلف:

- **هدف** :انتخاب مجموعه‌ای از ویژگی‌ها بر اساس آستانه‌های مختلف همبستگی مطلق.
- **جزئیات**:

  - برای هر آستانه، ویژگی‌هایی که مجموع همبستگی مطلق آن‌ها کمتر از مقدار آستانه است، انتخاب می‌شوند.
  - اگر تعداد ویژگی‌های انتخاب‌شده برابر یا بیشتر از ۶ باشد، آستانه بهینه پیدا شده و ویژگی‌ها به عنوان خروجی نمایش داده می‌شوند.

```
thresholds = [3, 3.5, 3.8, 4, 4.5, 5]
optimum_features = None

for threshold in thresholds:
    # Select features where the sum of absolute correlations is less than
the threshold
    selected_features_indices = [i for i, abs_sum in
absolute_correlation_sums if abs_sum < threshold]

    # Step 3: Print the feature names and their correlation sum
    selected_feature_names = [feature_names[index] for index in
selected_features_indices]

    print(f"\nFor threshold {threshold}, selected features
({len(selected_features_indices)}):")
    #print(selected_feature_names)
    if len(selected_features_indices) >= 6:
        optimum_features = selected_feature_names
        print(selected_feature_names)
        print("Optimum threshold found.")
        break
```

خروجی: correlation matrix



فاز ۳: Clustering

۱. بارگذاری و آماده‌سازی داده‌ها:

```
from sklearn.cluster import KMeans, DBSCAN, MeanShift
from sklearn.metrics import silhouette_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
import numpy as np
path = 'test_features.csv'
# Assuming data is already loaded into data1 and data2
# Example: scaled_data1, scaled_data2
    data1 = pd.read_csv(path)[optimum_features]
```

۲. استانداردسازی داده‌ها:

```
scaler = StandardScaler()
scaled_data1 = scaler.fit_transform(data1)
```

۳. آماده‌سازی داده‌ها و حذف ستون‌های غیرعددی:

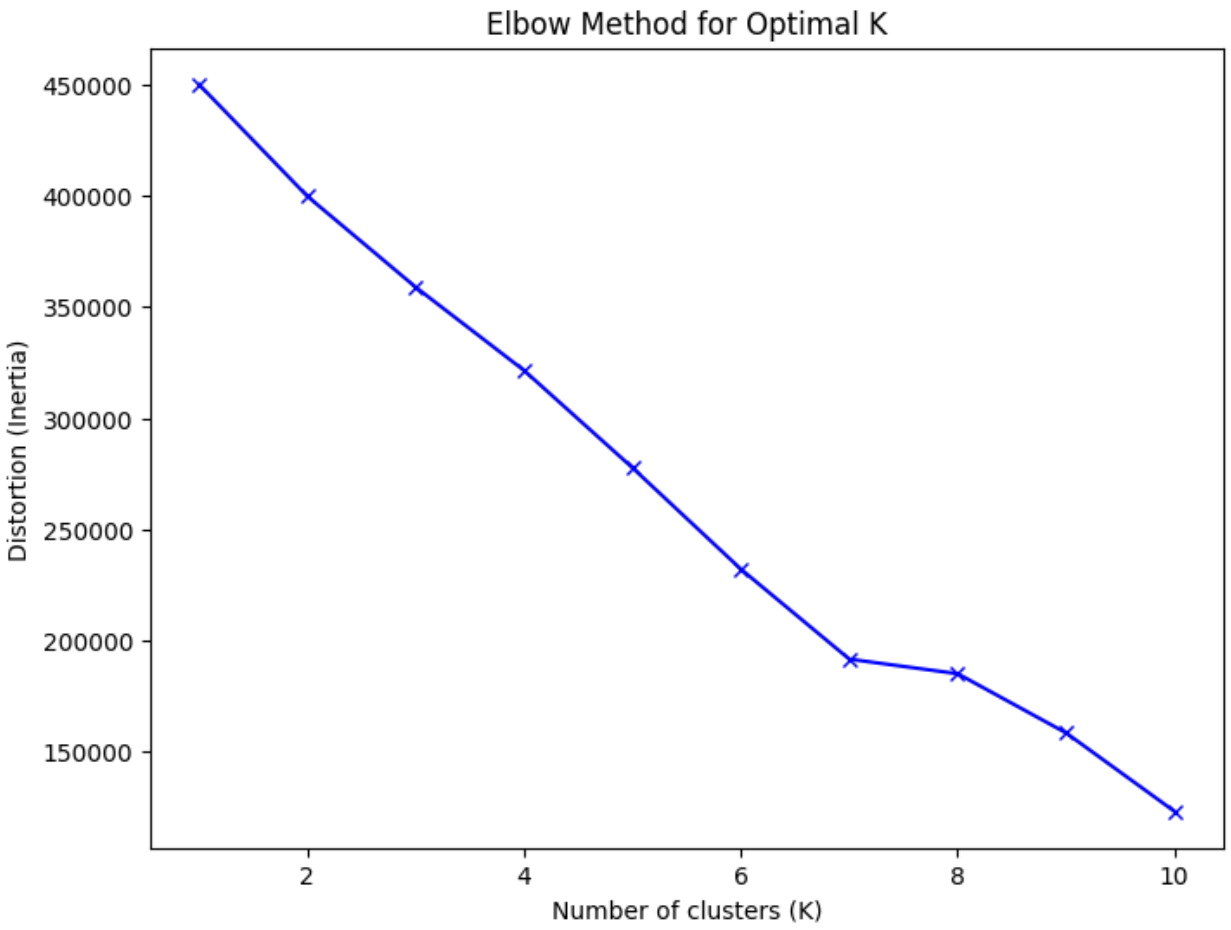
```
data =  data1

numeric_data = data.select_dtypes(include=[np.number])

# Check if any columns were dropped
dropped_columns = data.columns.difference(numeric_data.columns)
```

```
if len(dropped_columns) > 0:
    print(f"Dropped non-numeric columns: {list(dropped_columns)}")
```

۴. استفاده از Elbow Method برای یافتن تعداد بهینه خوشه ها:

- **هدف** :یافتن تعداد بهینه خوشه‌ها (K) با استفاده از روش آرنج.(Elbow Method)
- **جزئیات**:

- `find_optimal_k(data)`: این تابع از `KMeans` برای اعمال خوشه‌بندی بر روی داده‌ها با تعداد خوشه‌های مختلف استفاده کرده و میزان پراکندگی (یا همان `inertia`) برای هر تعداد خوشه را محاسبه می‌کند.
- `distortions.append(kmeans.inertia_)`: مقدار پراکندگی (inertia) برای هر مقدار `K` ذخیره می‌شود.
- نمودار منحنی آرنج رسم می‌شود که کاهش شدت پراکندگی را با افزایش تعداد خوشه‌ها نمایش می‌دهد و برای تعیین نقطه آرنج که تعداد خوشه بهینه است، به کار می‌رود.




```
def find_optimal_k(data):
    distortions = []
    K_range = range(1, 11)  # You can change this range for larger data
```

```
    for k in K_range:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(data)
        distortions.append(kmeans.inertia_)

    # Plot the elbow curve
    plt.figure(figsize=(8, 6))
    plt.plot(K_range, distortions, 'bx-')
    plt.xlabel('Number of clusters (K)')
    plt.ylabel('Distortion (Inertia)')
    plt.title('Elbow Method for Optimal K')
    plt.show()

    return distortions

distortions = find_optimal_k(scaled_data)
```

۵. تعیین تعداد خوشه‌ها (K) بر اساس الگوریتم قبل:

```
kb = 8
```

۶. اجرای الگوریتم K-means با مقدار بهینه K:

- **هدف** :اجرای الگوریتم K-means با تعداد خوشه‌های مشخص‌شده (`kb = 8`) و ذخیره برچسب خوشه هر نمونه در `cluster_labels_best`.
- **جزئیات**:

- `KMeans(n_clusters=kb, random_state=42)`: خوشه‌بندی K-means با ۸ خوشه اجرا می‌شود.
- `kmeans_best.fit(scaled_data)`: الگوریتم K-means بر روی داده‌های استاندارد شده اعمال می‌شود.
- `cluster_labels_best`: لیستی از برچسب‌های خوشه برای هر نمونه در داده.

```
kmeans_best = KMeans(n_clusters=kb, random_state=42)
kmeans_best.fit(scaled_data)
cluster_labels_best = kmeans_best.labels_
```

۷. کاهش ابعاد داده با استفاده از PCA:

- **هدف** :کاهش ابعاد داده به دو بُعد با استفاده از تحلیل مؤلفه‌های اصلی (PCA) به منظور بصری‌سازی.
- **جزئیات**:

- `PCA(n_components=2)`: انتخاب دو مؤلفه اصلی برای نگاشت داده‌ها به یک فضای دو بُعدی.
- `reduced_data`: داده‌های کاهش‌یافته به دو بُعد که برای نمایش خوشه‌بندی استفاده می‌شوند.

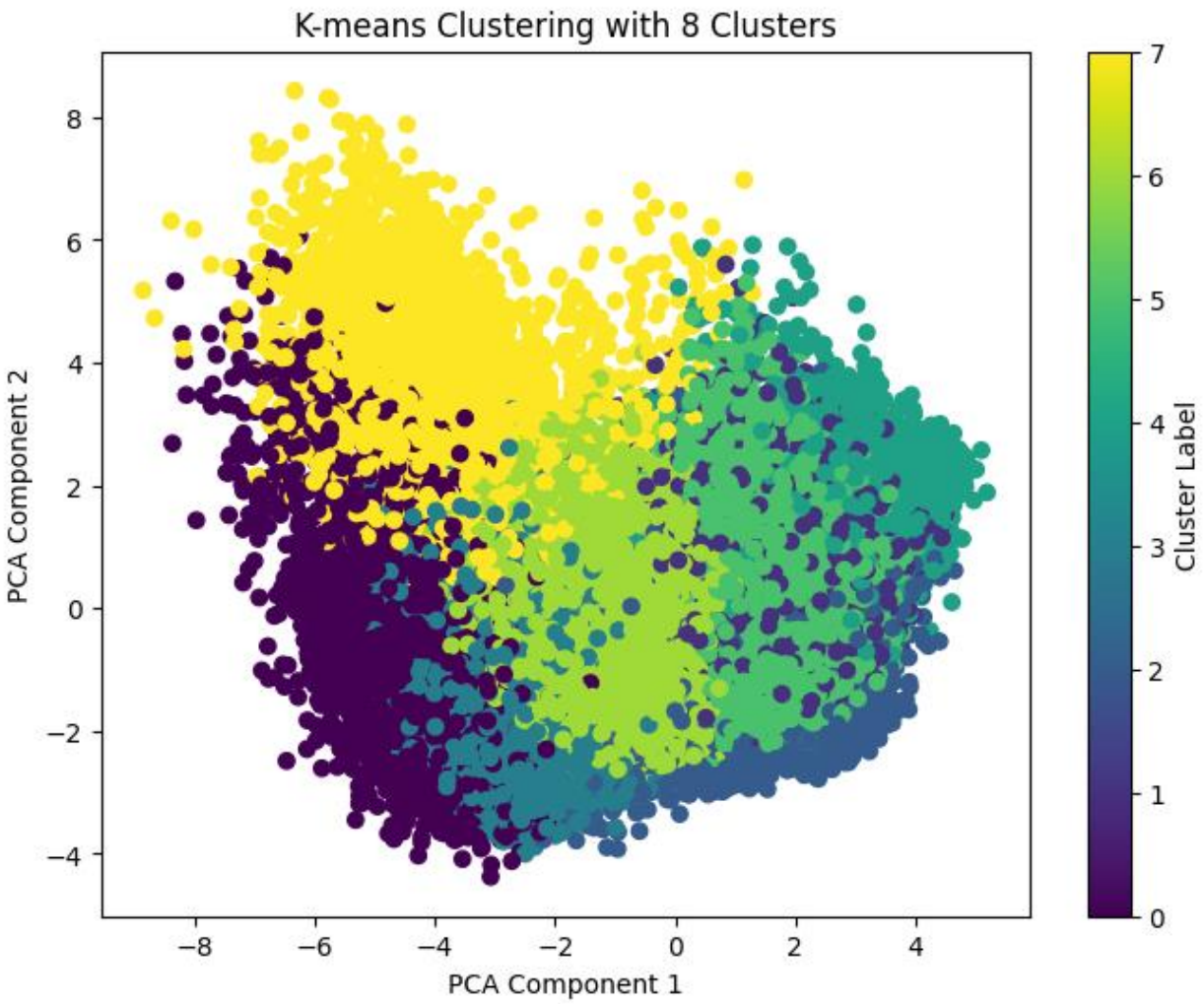
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data)
```

۸. محاسبه silhouette score برای ارزیابی کیفیت خوشه بندی:

```
sil_score = silhouette_score(scaled_data, cluster_labels_best)
print(f"Silhouette Score for K-means with {kb} clusters: {sil_score}")
```

۹. نمایش خوشه‌ها با استفاده از داده‌های کاهش‌یافته:

```
plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=cluster_labels_best,
cmap='viridis')
plt.title(f"K-means Clustering with {kb} Clusters")
plt.xlabel('PCA Component 1')

plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster Label')
plt.show()
```



۱۰. اجرای الگوریتم DBSCAN و بارگذاری و آمادهسازی دادهها:

```
data1 = pd.read_csv('test_features.csv')[optimum_features]
# Normalize the data
scaler = StandardScaler()
scaled_data1 = scaler.fit_transform(data1)
```

۱۱. تعریف تابع سفارشی برای silhouette score:

```
def silhouette_scorer(estimator, X):
    cluster_labels = estimator.fit_predict(X)

    # Check if there is more than 1 cluster to calculate silhouette score
    if len(set(cluster_labels)) > 1:
        score = silhouette_score(X, cluster_labels)
    else:
```

```
        score = -1  # If only one cluster or no valid clusters, return nan

    return score
```

۱۲. تنظیم پارامترهای شبکه (Grid) برای DBSCAN :

هدف :تعریف دامنه پارامترها `eps`) و (`min_samples` برای DBSCAN که با استفاده از Grid Search بررسی شوند.

جزئیات:

a. `eps`: فاصله همسایگی. مقادیر بین ۱ و ۲.۲۵ با گام ۰.۰۵ در نظر گرفته شده‌اند.
b. `min_samples`: حداقل تعداد نقاط برای تشکیل خوشه، که مقادیر بین ۲ تا ۱۰ را بررسی می‌کند.

```
param_grid_dbscan = {
    'eps': np.arange(1, 2.25, 0.05),  # Start from a small eps and go to a
larger value (2.0)
    'min_samples': [2, 3, 4, 5, 6, 10]  # Try lower values of min_samples
}
```

۱۳. یافتن بهترین پارامترها با GridSearchCV :

- **هدف** :اجرای Grid Search برای پیدا کردن بهترین پارامترهای DBSCAN بر اساس امتیاز سیلوئت.
- **جزئیات:**

- `GridSearchCV(...)`: جستجوی پارامترها با استفاده از ترکیب‌های مختلف `eps` و `min_samples`.
- `best_params_` و `best_score_`: بهترین پارامترها و بالاترین امتیاز سیلوئت را برمی‌گرداند.

```
grid_dbscan = GridSearchCV(DBSCAN(), param_grid_dbscan,
scoring=silhouette_scorer, cv=3)
grid_dbscan.fit(scaled_data1)

# Get the best parameters and score
best_dbscan_params = grid_dbscan.best_params_
best_dbscan_score = grid_dbscan.best_score_

# Output the best results
print("Best DBSCAN params for data1:", best_dbscan_params)
print("Best DBSCAN silhouette score for data1:", best_dbscan_score)
```

۱۴. اجرای DBSCAN با بهترین پارامترها و نمایش تعداد خوشه‌ها و نقاط نویز:

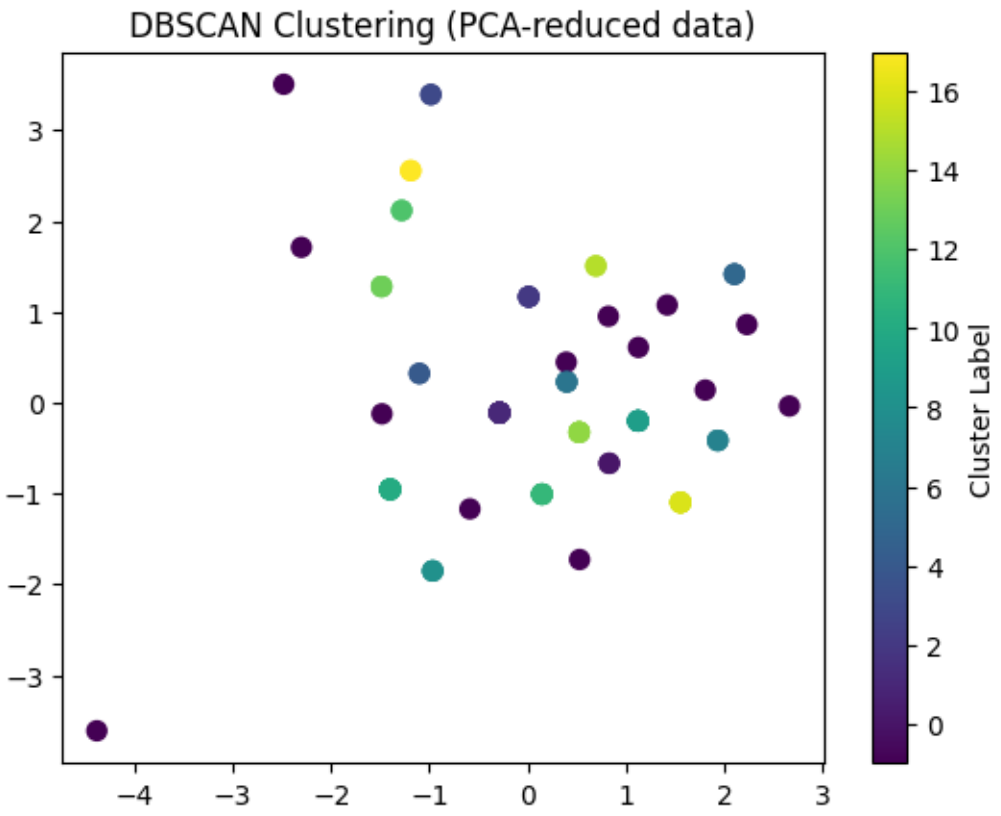
- **هدف**: اجرای DBSCAN با بهترین پارامترها و محاسبه تعداد خوشه‌ها و نقاط نویز.
- **جزئیات**:

  - `labels`: برچسب‌های خوشه برای هر نمونه. نقاط نویز با برچسب `-1` مشخص می‌شوند.
  - `n_noise` و `n_clusters`: تعداد نقاط نویز و تعداد خوشه‌ها را نشان می‌دهند.

```
best_dbscan = DBSCAN(eps=best_dbscan_params['eps'],
min_samples=best_dbscan_params['min_samples'])
labels = best_dbscan.fit_predict(scaled_data1)

# Diagnostic: Count noise points and number of clusters
n_noise = list(labels).count(-1)
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

print(f"Number of clusters: {n_clusters}")
print(f"Number of noise points: {n_noise}")

# Handle potential edge case: if no clusters are formed
if n_clusters == 0:
    print("Warning: No clusters were found. Try increasing 'eps' or
lowering 'min_samples'.")
```

۱۵. کاهش ابعاد داده‌ها با PCA و نمایش خوشه‌ها:

```
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data1)

# Plot the PCA-reduced data with cluster labels
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels,
cmap='viridis', s=50)
plt.title('DBSCAN Clustering (PCA-reduced data)')
plt.colorbar(label='Cluster Label')
plt.show()
```



۱۶. الگوریتم MeanShift:

برآورد Bandwidth برای Mean Shift

- **هدف:** تعیین مقدار پهنای باند (`bandwidth`) که یکی از مهم‌ترین پارامترها برای الگوریتم Mean Shift است.
- **جزئیات:**
  - `estimate_bandwidth(...)`: این تابع پهنای باند را بر اساس داده‌ها و پارامتر `quantile` برآورد می‌کند.
  - `quantile=0.2`: این پارامتر تعیین می‌کند که چند درصد از نزدیک‌ترین نمونه‌ها باید در یک خوشه قرار بگیرند. تنظیم این مقدار بر اساس داده‌ها و نیازهای خوشه‌بندی می‌تواند عملکرد Mean Shift را بهبود بخشد.

```
bandwidth = round(estimate_bandwidth(scaled_data, quantile=0.2,
n_samples=500), 2)  # Adjust the quantile and sample size if necessary
print(f'Estimated Bandwidth: {bandwidth}')
```

۱۷. اعمال الگوریتم Mean Shift با استفاده از bandwidth تخمینی

```
mean_shift = MeanShift(bandwidth=bandwidth)
mean_shift.fit(scaled_data)
```

۱۸. دریافت labelهای خوشه‌ها:

•**هدف** :اختصاص برچسب خوشه‌ها به هر نمونه در داده‌ها.
•**جزئیات**:

- `labels`: هر نمونه در داده‌ها به یک خوشه خاص اختصاص داده شده و شماره خوشه مربوط به آن در این متغیر ذخیره می‌شود.

```
labels = mean_shift.labels_
```

۱۹. محاسبه silhouette score:

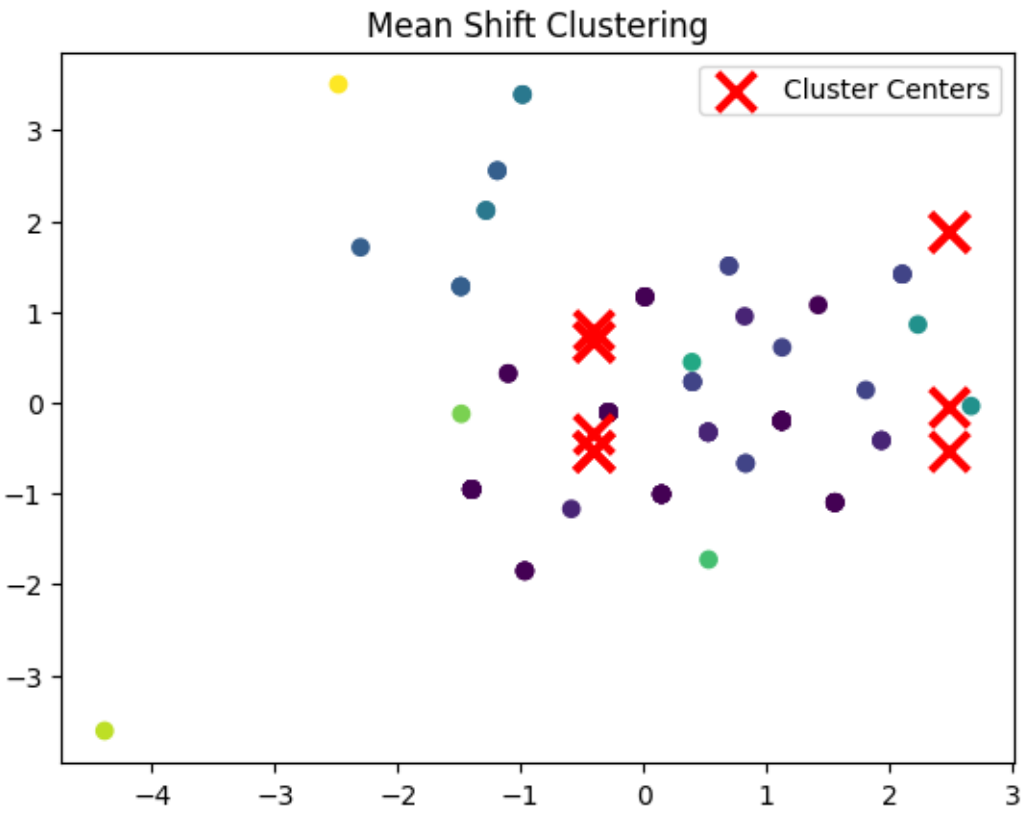
```
if len(np.unique(labels)) > 1:
    score = silhouette_score(scaled_data, labels)
    print(f'Silhouette Score: {score}')
else:
    print("Only one cluster found, silhouette score cannot be
calculated.")
```

۲۰. کاهش ابعاد داده‌ها با استفاده از PCA:

```
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data)
```

۲۱: نمایش نتایج خوشه‌بندی با نمودار پراکندگی:

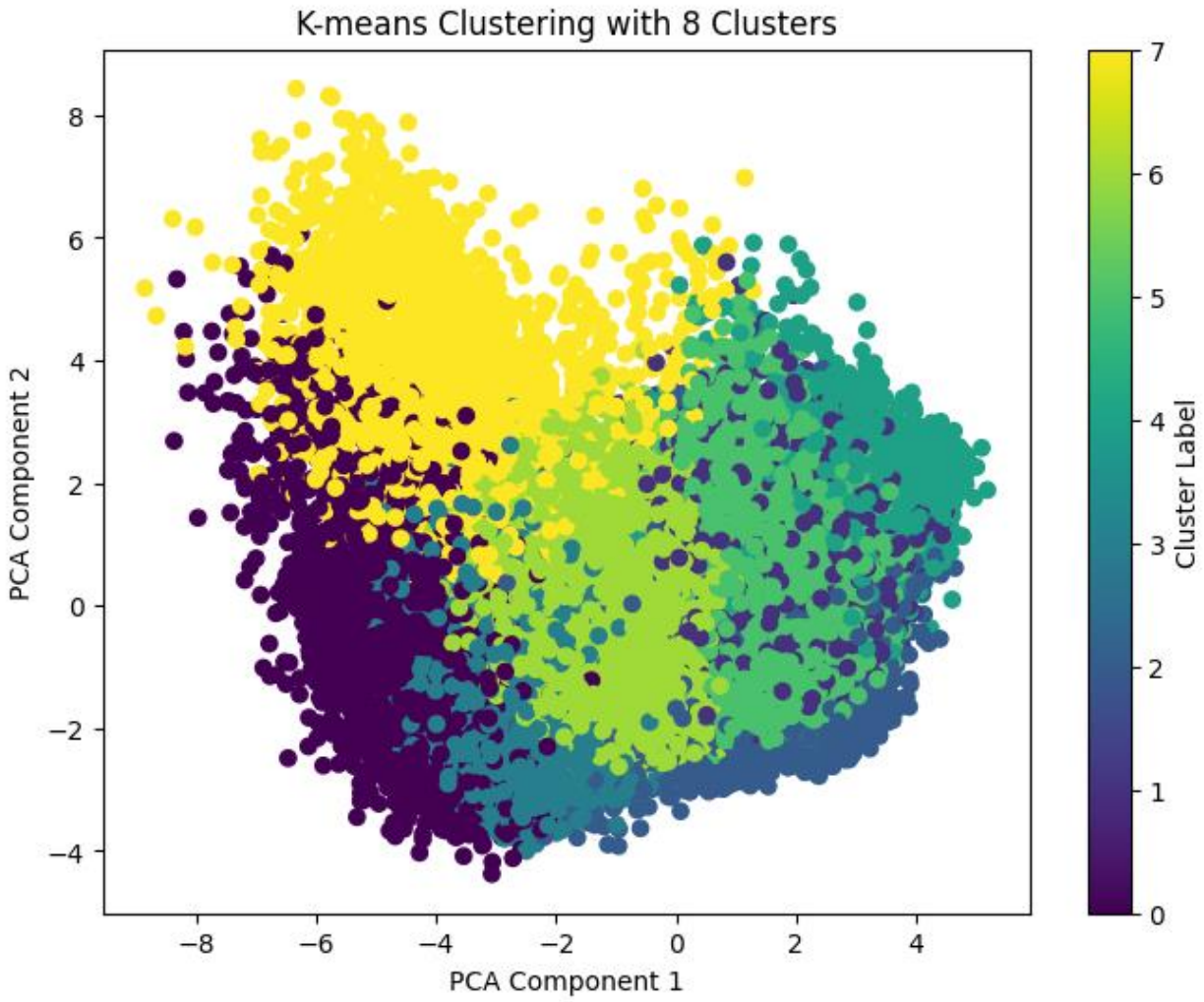
```
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels,
cmap='viridis', marker='o')
plt.scatter(mean_shift.cluster_centers_[:, 0],
mean_shift.cluster_centers_[:, 1],
            c='red', marker='x', s=200, linewidths=3, label='Cluster
Centers')
plt.title('Mean Shift Clustering')
plt.legend()
plt.show()
```



۲۱. اجرای مجدد K_Means برای بررسی فایل features2.csv:

```
data = data2
data = pd.read_csv('/content/features2.csv')


# Separate the image IDs
image_ids = data['image_id']

# Remove any non-numeric columns
numeric_data = data.select_dtypes(include=[np.number])

# Drop rows with missing values and update image_ids accordingly
numeric_data = numeric_data.dropna()
image_ids = image_ids[numeric_data.index]

# Choose K based on the elbow and evaluate using silhouette score
kb = 8

# Perform K-means clustering using the best K
kmeans_best = KMeans(n_clusters=kb, random_state=42)
kmeans_best.fit(scaled_data)

cluster_labels_best = kmeans_best.labels_

# Plot the clusters using PCA for visualization (if data is multi-dimensional)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_data)
```

```python
sil_score = silhouette_score(scaled_data, cluster_labels_best)

print(f"Silhouette Score for K-means with {kb} clusters: {sil_score}")

plt.figure(figsize=(8, 6))
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=cluster_labels_best, cmap='viridis')
plt.title(f"K-means Clustering with {kb} Clusters")
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster Label')
plt.show()

# Print the names of 10 pictures in each cluster
for cluster in range(kb):
    cluster_indices = np.where(cluster_labels_best == cluster)[0]
    cluster_image_ids = image_ids.iloc[cluster_indices].head(10).tolist()
    print(f"Cluster {cluster}: {cluster_image_ids}")
```



۲۲. مقایسه سه الگوریتم باهم و بررسی هر کدام توسط silhouette score:

مقداردهی اولیه مدل‌های خوشه‌بندی:

```
kmeans = KMeans(n_clusters=3)
dbscan = DBSCAN(eps=0.5, min_samples=5)
meanshift = MeanShift(bandwidth=2)
```

فراخوانی و اجرای الگوریتم ها:

```
kmeans_labels1 = kmeans.fit_predict(scaled_data1)
dbscan_labels1 = dbscan.fit_predict(scaled_data1)
meanshift_labels1 = meanshift.fit_predict(scaled_data1)
```

مقایسه silhouette score برای هر الگوریتم:

```
silhouette_kmeans1 = silhouette_score(scaled_data1, kmeans_labels1)
silhouette_dbscan1 = silhouette_score(scaled_data1, dbscan_labels1)
silhouette_meanshift1 = silhouette_score(scaled_data1, meanshift_labels1)

print(f"KMeans Silhouette Score: {silhouette_kmeans1}")
print(f"DBSCAN Silhouette Score: {silhouette_dbscan1}")
print(f"MeanShift Silhouette Score: {silhouette_meanshift1}")
```

```
KMeans Silhouette Score: 0.2536895120352182
DBSCAN Silhouette Score: 0.42537788017813255
MeanShift Silhouette Score: 0.8699999965750763
```

۲۳. مقایسه ویژگی های متمایز کننده هر خوشه و heatmap visualization:

بارگزاری و نرمال سازی داده ها

```
data1 = pd.read_csv('features.csv')[optimum_features]
scaler = StandardScaler()
scaled_data1 = scaler.fit_transform(data1)
```

۲۴. اجرای الگوریتم K_Means:

```
best_kmeans_model = KMeans(n_clusters=8, random_state=42)
labels_kmeans = best_kmeans_model.fit_predict(scaled_data1)
```

۲۵. اضافه کردن labelهای خوشه به داده ها:

```
data1['Cluster'] = labels_kmeans
```

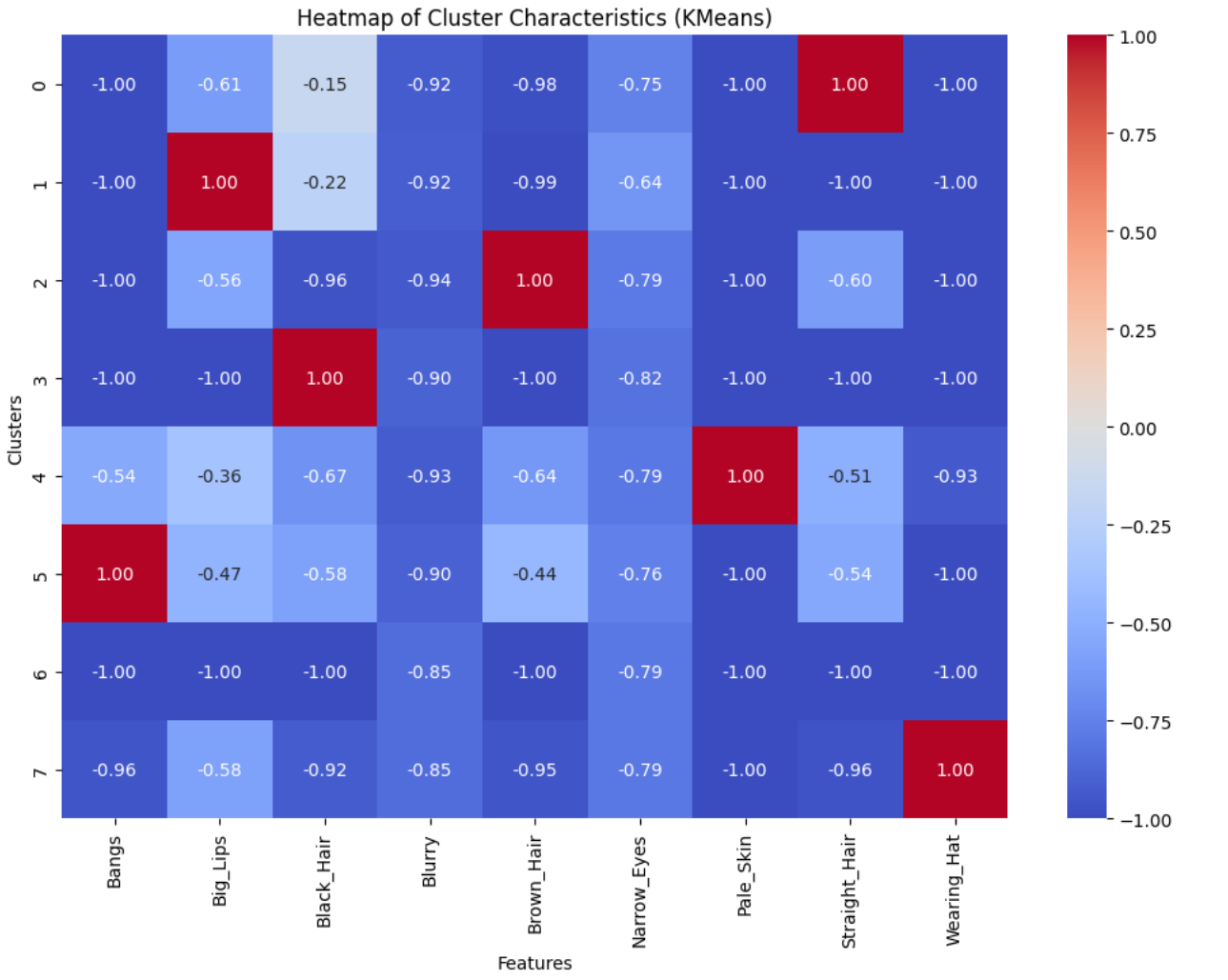
۲۶. محاسبه میانگین ویژگی ها برای هر خوشه:

```
cluster_means = data1.groupby('Cluster').mean()
```

۲۷. رسم heatmap:

- **رسم هیت‌مپ** :با استفاده از کتابخانه‌ی `seaborn`، هیت‌مپ میانگین ویژگی‌ها برای هر خوشه رسم می‌شود.
  - **cmap="coolwarm"**: از نقشه‌ی رنگی `coolwarm` استفاده می‌شود تا داده‌های بالا و پایین بهتر مشخص شوند.
  - **annot=True**: مقادیر میانگین‌ها بر روی هر سلول نمایش داده می‌شوند.
  - **fmt=".2f"**: نمایش مقادیر به صورت اعشاری تا دو رقم تنظیم شده است.
- **خروجی** :این هیت‌مپ به ما کمک می‌کند ویژگی‌های خاص هر خوشه را بهتر درک کنیم و تفاوت میان خوشه‌ها را از لحاظ مقادیر میانگین ویژگی‌ها مشاهده کنیم.

```
plt.figure(figsize=(12, 8))
sns.heatmap(cluster_means, annot=True, cmap="coolwarm", cbar=True,
fmt=".2f")
plt.title("Heatmap of Cluster Characteristics (KMeans)")
plt.xlabel("Features")
plt.ylabel("Clusters")
plt.show()
```



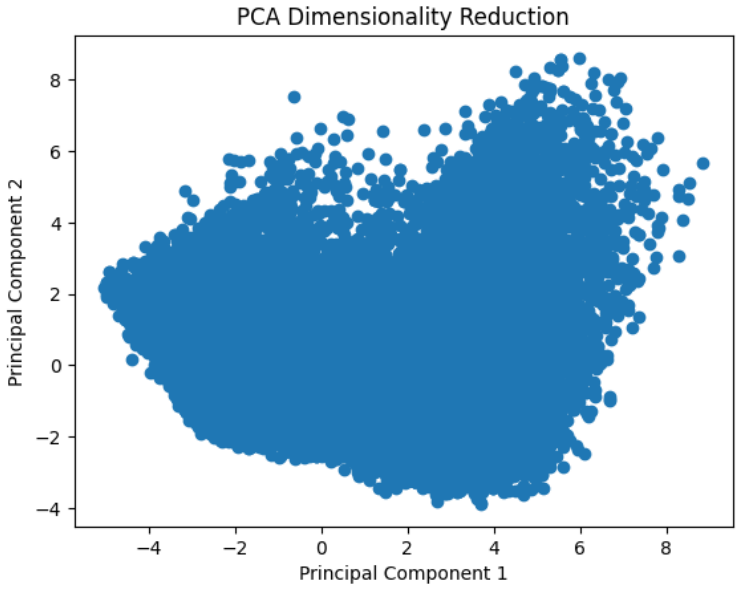
فاز ۴: Visualization

مجموعه داده از یک فایل CSV بارگذاری می‌شود داده‌های عددی با استفاده از StandardScaler استانداردسازی می‌شوند.
استانداردسازی به این معنی است که میانگین هر ویژگی به صفر و انحراف معیار آن به یک تبدیل می‌شود. این کار برای اطمینان از اینکه همه ویژگی‌ها در یک مقیاس مشابه قرار دارند انجام می‌شود.

PCA به منظور کاهش ابعاد داده‌ها استفاده می‌شود. اینجا تعداد مولفه‌ها (n_components) به ۲ تنظیم شده است، به این معنی که داده‌های اصلی به دو بعد کاهش می‌یابند. نتیجه این کاهش ابعاد در principal_components ذخیره می‌شود.

```
# Load the dataset
path = '/content/features.csv'
data = pd.read_csv(path)

# Exclude non-numeric columns (like 'image_id') and use only numeric features
features = [col for col in data.columns if data[col].dtype.name in ['int64', 'float64']]

# Select the numeric data
X = data[features]

# Standardize the features (important for PCA)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_data)

# Create a new DataFrame with the principal components
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# Visualization
plt.scatter(principal_df['PC1'], principal_df['PC2'])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Dimensionality Reduction')
plt.show()
```

خروجی:



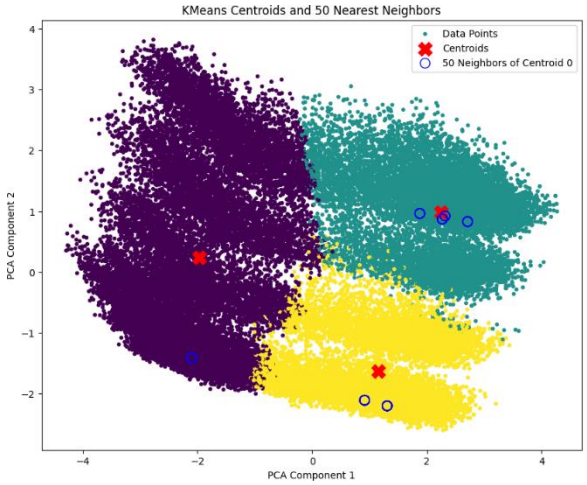
فاز ۵: K-Means و KNN

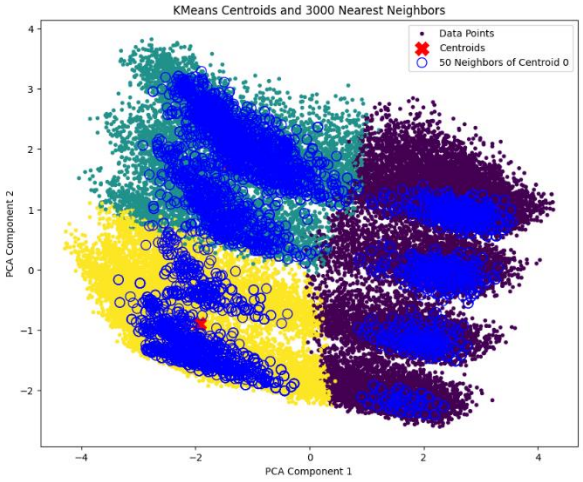
الگوریتم K-Means با ۳ خوشه (Cluster) روی داده‌ها اجرا می‌شود. K-Means با تعداد خوشه‌های n_clusters=3 ساخته و آموزش داده می‌شود. سپس، تعداد و ابعاد مراکز خوشه‌ها (Centroids) با استفاده از kme.cluster_centers_.shape چاپ می‌شود که نشان می‌دهد ۳ مرکز با ۱۰ ویژگی داریم.

دو مدل KNN با تعداد همسایگان مختلف (۵۰ و ۳۰۰۰) ایجاد و آموزش داده می‌شوند: در هر دو مدل از معیار فاصله کسینوسی (metric='cosine') برای محاسبه فاصله بین داده‌ها استفاده می‌شود.

```
path = '/content/features.csv'
dataset = pd.read_csv(path).drop('image_id',axis = 1)

# fit K-means with 3 clusters on our dataset.
#The numbers of
kme = KMeans(n_clusters=3)
kme.fit(dataset)

# we should have 3 vectors for 3 centroids.
print(kme.cluster_centers_.shape) # (3, 10)

# initialize NearestNeighbor with 50 neighbors and fit our dataset.
knn = NearestNeighbors(n_neighbors=50, metric='cosine')
knn.fit(dataset)

# initialize NearestNeighbor with 3000 neighbors and fit our dataset.
knn1 = NearestNeighbors(n_neighbors=3000, metric='cosine')
knn1.fit(dataset)

# Perform PCA to reduce dimensions to 2 for visualization
pca = PCA(n_components=2)
dataset_2d = pca.fit_transform(dataset)
centroids_2d = pca.transform(kme.cluster_centers_)

# Get the 50 nearest neighbors for each centroid
nearest_neighbors_50 = knn.kneighbors(kme.cluster_centers_, return_distance=False)

# Get the 3000 nearest neighbors for each centroid
#nearest_neighbors_3000 = knn1.kneighbors(kme.cluster_centers_, return_distance=False)
```

```
# Get the 3000 nearest neighbors for each centroid
#nearest_neighbors_3000 = knn1.kneighbors(kme.cluster_centers_, return_distance=False)

# Plot centroids and 50 nearest neighbors
plt.figure(figsize=(10, 8))

# Plot the dataset points
plt.scatter(dataset_2d[:, 0], dataset_2d[:, 1], c=kme.labels_, s=10, cmap='viridis', label='Data Points')

# Plot the centroids
plt.scatter(centroids_2d[:, 0], centroids_2d[:, 1], s=200, c='red', label='Centroids', marker='X')

# Highlight nearest neighbors (50 neighbors)
for i, neighbors in enumerate(nearest_neighbors_50):
    plt.scatter(dataset_2d[neighbors, 0], dataset_2d[neighbors, 1], edgecolors='blue', facecolors='none', s=100, label=f'50 Neighbors of Centroid {i}' if i == 0 else "")

plt.title("KMeans Centroids and 50 Nearest Neighbors")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend()
plt.show()
```





همین طور که مشاهده می شود وقتی ۳۰۰۰ داده نزدیک به مرکز کلاستر را بررسی می کنیم با بالا رفتن تعداد، شعاع همسایگی بالا رفته و باعث می شود که

فاز ۶: Prediction

۱. بارگزاری ویژگی های داده:

```
test_features_path = 'test_features.csv'
test_features = pd.read_csv(test_features_path)
```

۲. تعریف مدل Mean Shift و انجام خوشه‌بندی:

```
mean_shift_model = MeanShift(bandwidth=1.5)
X = test_features[optimum_features].values
mean_shift_model.fit(X)
cluster_centers = mean_shift_model.cluster_centers_
labels = mean_shift_model.labels_
```

۳. اختصاص خوشه ها به تصاویر:

```
test_folder_path = 'test'
image_files = [f for f in os.listdir(test_folder_path) if
f.endswith(('.png', '.jpg', '.jpeg'))]

# Update test_features dataframe with assigned cluster labels
test_features['Cluster_Label'] = labels
```

۴. دخیره تصاویر:

```
test_features.to_csv('test_features2.csv', index=False)
```

۵. انتخاب ۱۰ تصویر:

```
random_images = random.sample(image_files, 10)
```

۶. محاسبه شباهت:

```
def calculate_similarity(feature_vector, other_features):
    return np.linalg.norm(other_features - feature_vector, axis=1)
```

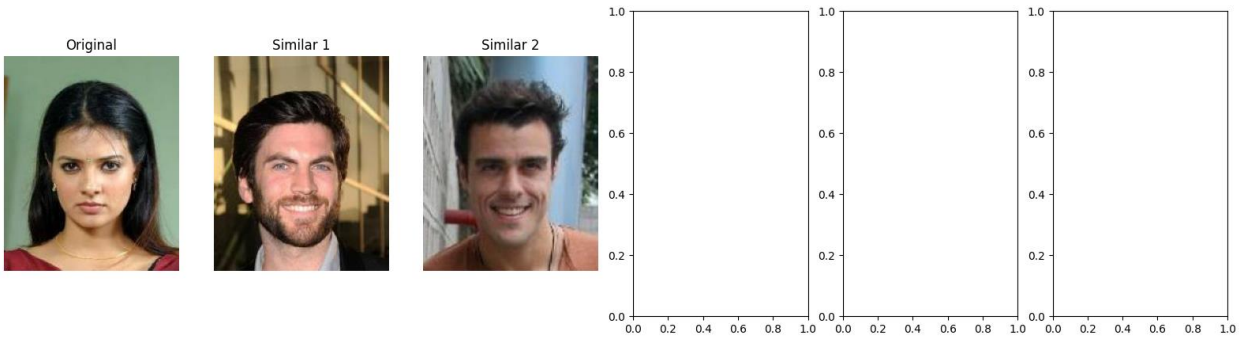
۷. نمایش تصویر و تصاویر مشابه:

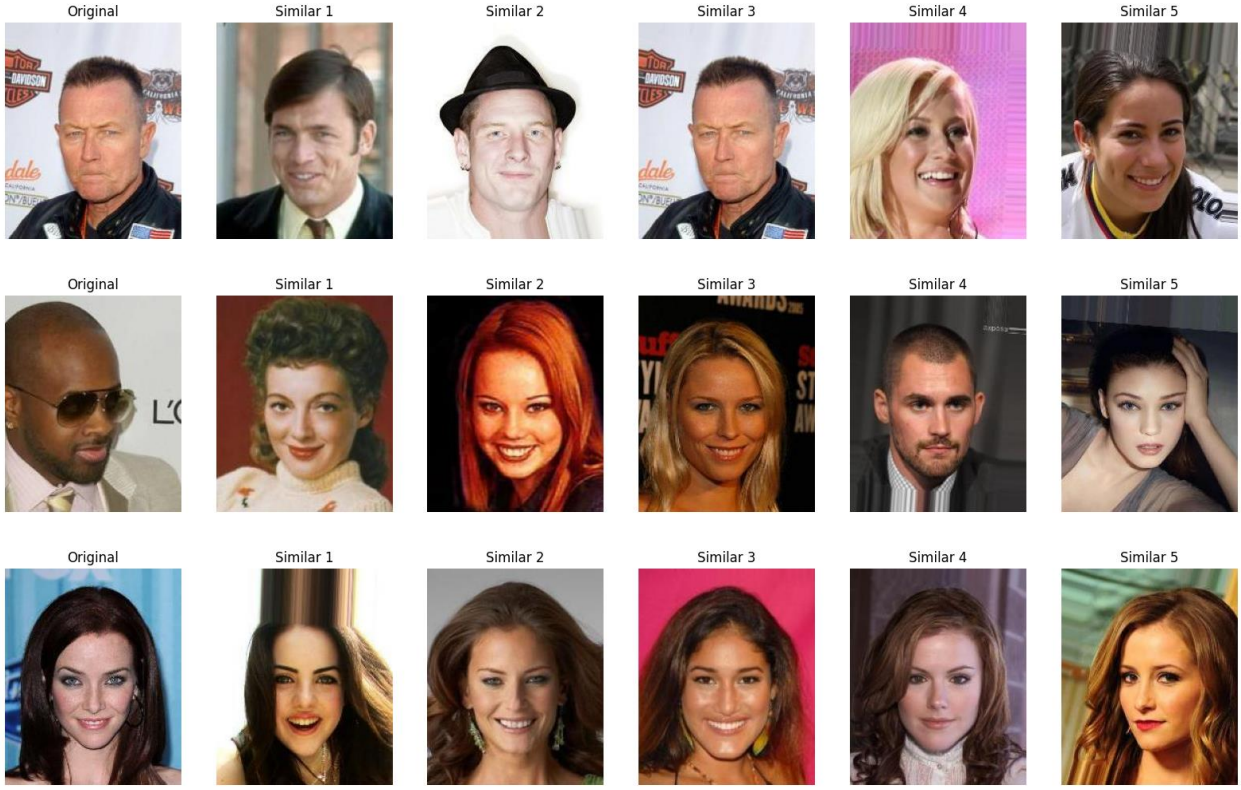
- **حلقه برای نمایش تصاویر:** برای هر تصویر انتخاب شده، ابتدا تصویر اصلی بارگذاری می‌شود.
- **جستجوی تصاویر مشابه**: تصاویری که در همان خوشه هستند پیدا می‌شوند و شباهت آنها به تصویر اصلی محاسبه می‌شود.
- **نمایش تصاویر**: تصویر اصلی و ۵ تصویر مشابه در یک پنجره نمایش داده می‌شوند.

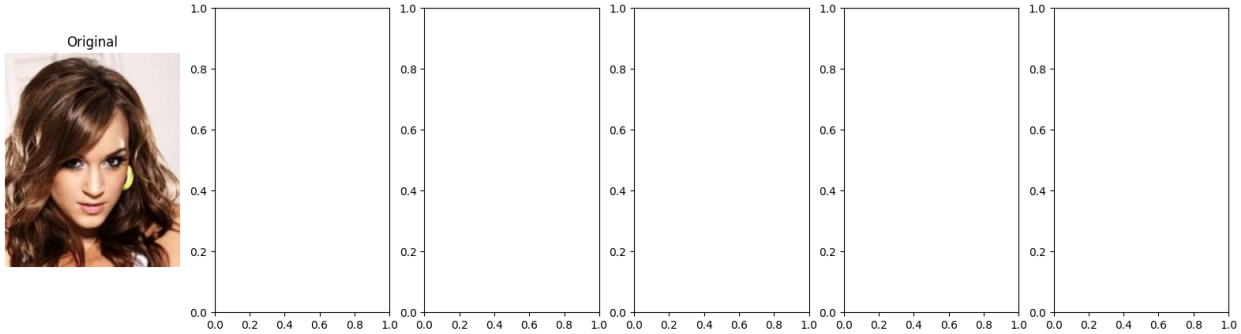
```
for image_file in random_images:
    image_path = os.path.join(test_folder_path, image_file)
    img = cv2.imread(image_path)
    cluster_label = test_features.loc[test_features['image_id'] ==
image_file, 'Cluster_Label'].values[0]

    # Get all images in the same cluster
    cluster_images = test_features[test_features['Cluster_Label'] ==
cluster_label]

    # Get feature vector of the selected image
    image_features = test_features.loc[test_features['image_id'] ==
image_file, optimum_features].values[0]

    # Calculate similarity and sort images in the same cluster by
similarity
    cluster_images['similarity'] = calculate_similarity(image_features,
cluster_images[optimum_features].values)
    similar_images = cluster_images.sort_values('similarity').head(6)

    # Display the images
    fig, axs = plt.subplots(1, 6, figsize=(20, 5))
    axs[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[0].set_title('Original')
    axs[0].axis('off')
```

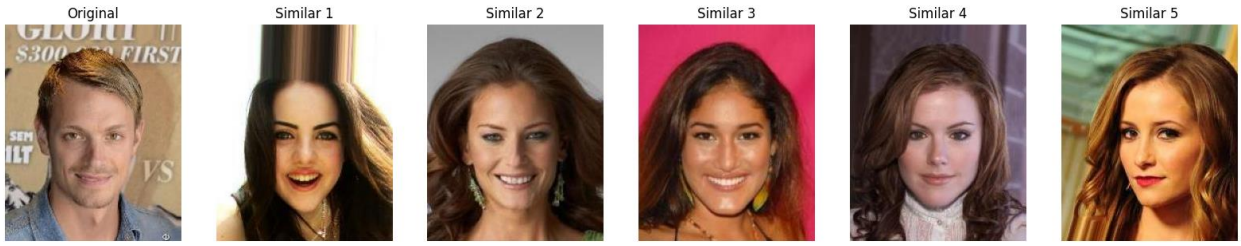
```
    for i, similar_image in enumerate(similar_images[1:].itertuples(),
start=1):
        similar_image_path = os.path.join(test_folder_path,
similar_image.image_id)
        similar_img = cv2.imread(similar_image_path)
        axs[i].imshow(cv2.cvtColor(similar_img, cv2.COLOR_BGR2RGB))
        axs[i].set_title(f'Similar {i}')
        axs[i].axis('off')

    plt.show()
```

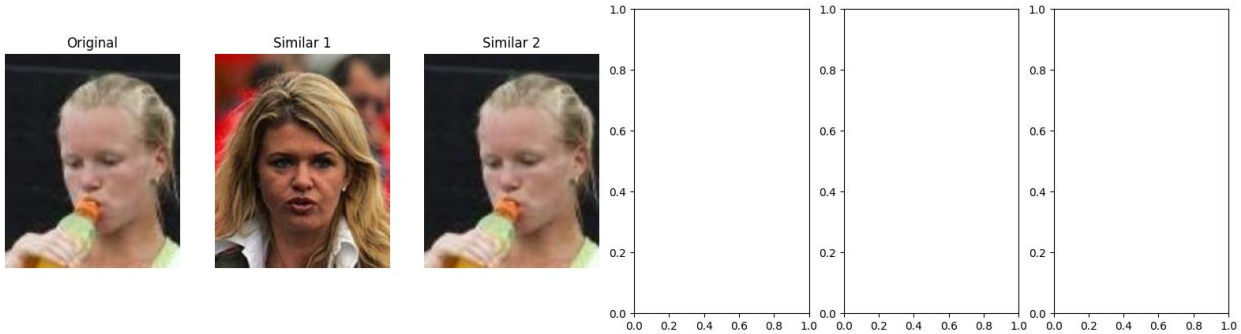
خروجی ها:

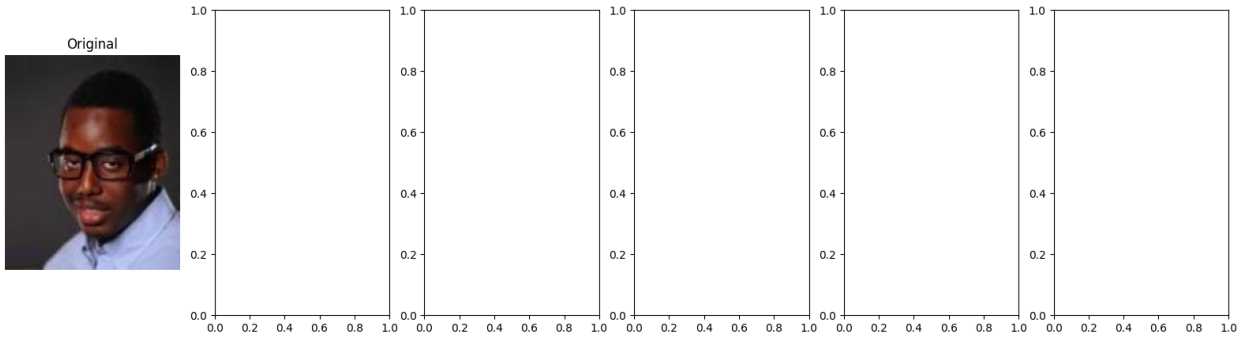





Original


Original
Similar 1
Similar 2
Similar 3
Similar 4
Similar 5


Original
Similar 1
Similar 2


Original


Original
Similar 1
Similar 2
Similar 3
Similar 4
Similar 5