```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from skimage import io,color,transform
import os
from google.colab import drive
drive.mount('/content/drive')
folder_path = "/content/drive/MyDrive/AML/HW3/Q1/360"
%cd {folder_path}
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/AML/HW3/Q1/360
```

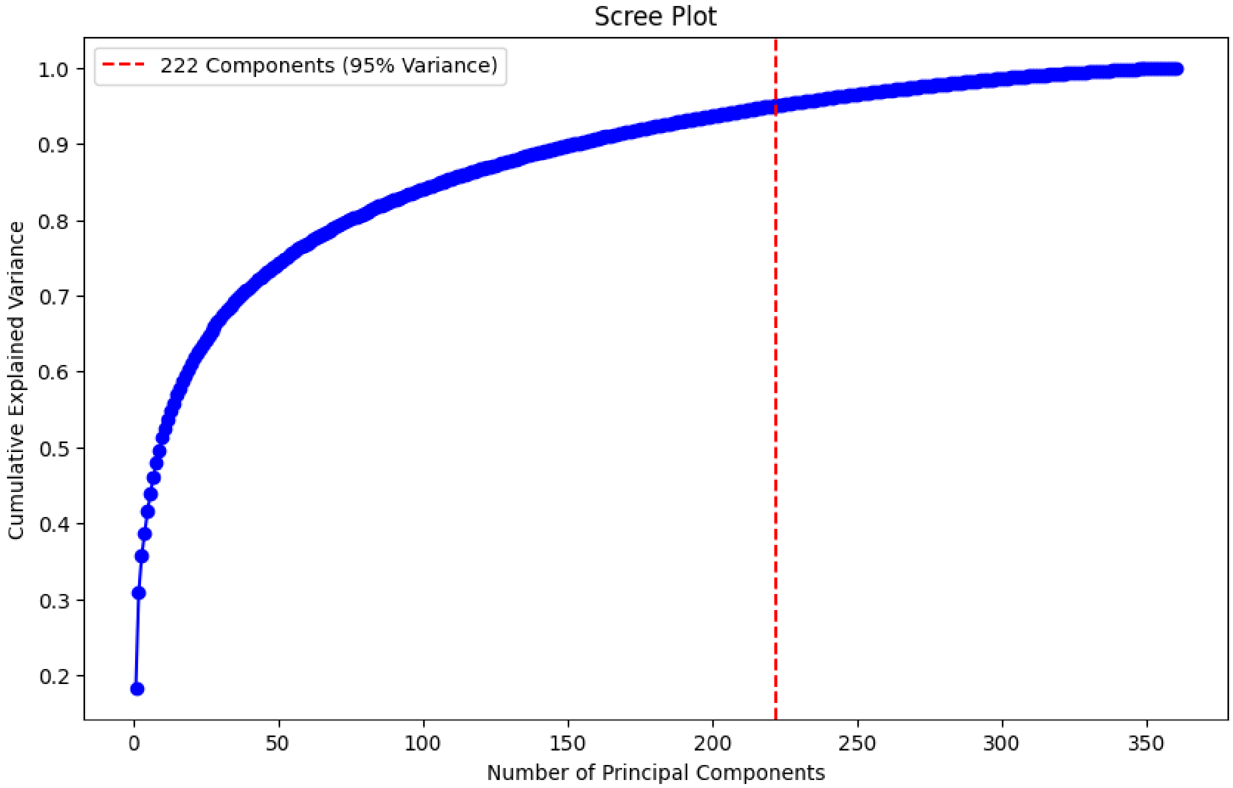
Question 1:

Apply PCA to the images from folder '360 Rocks'. How many components do you need to preserve 95% of the variance? [3 points]

```
%cd {folder_path}
# Load images from the folder
image_files = [f for f in os.listdir(folder_path) if
f.endswith(('.jpg', '.jpeg', '.png'))]

# Load the images into a numpy array and convert to grayscale
images = []
for image_file in image_files:
    img = io.imread(os.path.join(folder_path, image_file))
    img_gray = color.rgb2gray(img)  # Convert to grayscale
    images.append(img_gray)

# Convert the list of images to a 2D numpy array (num_images,
num_pixels)
images_array = np.array(images)
print(images_array.shape)
# Flatten the array to 1D (num_images, num_pixels)
num_images, height, width = images_array.shape
images_1d = images_array.reshape(num_images, height * width)

# Standardize the data (mean=0, variance=1)
scaler = StandardScaler()
images_standardized = scaler.fit_transform(images_1d)
```

```
/content/drive/MyDrive/AML/HW3/Q1/360
(360, 800, 800)
```

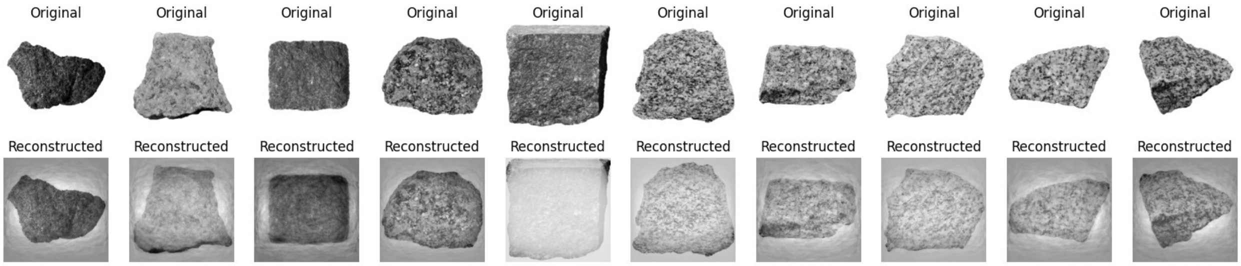
```
# Apply PCA without specifying the number of components
pca = PCA()
pca.fit(images_standardized)

# Calculate cumulative explained variance
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

# Find the number of components to preserve 95% variance
n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1

# Plot the scree plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,
marker='o', linestyle='-', color='b')
plt.title('Scree Plot')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance')
plt.axvline(x=n_components_95, color='r', linestyle='--',
label=f'{n_components_95} Components (95% Variance)')
plt.legend()
plt.show()

print("\n\n")
print(f"Number of components to preserve 95% variance:
{n_components_95}")
```



```
Number of components to preserve 95% variance: 222
```

Question 2 : Plot 10 images of your choice in the original form (without PCA) and then plot their reconstruction (projection in the original space) after you kept 95% of variance using PCA. [3 points]

```
# Apply PCA with 222 components
n_components = 222
pca = PCA(n_components=n_components)
images_pca = pca.fit_transform(images_standardized)

# Reconstruct the images from the PCA components
images_reconstructed = pca.inverse_transform(images_pca)
images_reconstructed = images_reconstructed.reshape(num_images,
height, width)

# Display 10 original and reconstructed images
num_displayed = 10
fig, axes = plt.subplots(2, num_displayed, figsize=(2 * num_displayed,
4))

for i in range(num_displayed):
    axes[0, i].imshow(images[i], cmap='gray')
    axes[0, i].axis('off')
    axes[0, i].set_title('Original')

    axes[1, i].imshow(images_reconstructed[i], cmap='gray')
    axes[1, i].axis('off')
    axes[1, i].set_title('Reconstructed')

plt.show()
```



Thanks to PCA, images look a little blurred out, but we are only considering around 222 features instead of the original

Note : for the following part of the quesions, i have resized the image from 800x800 to 28x28 and converted it to grayscale as anything more than that was making our colab sessions freeze and terminate.

I did ask the professor about it, and he said it should be fine.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from skimage import io,color,transform
import os
from google.colab import drive
drive.mount('/content/drive')
folder_path = "/content/drive/MyDrive/AML/360 Rocks"
%cd {folder_path}
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/AML/360 Rocks
```

```
%cd {folder_path}
# Load images from the folder
image_files = [f for f in os.listdir(folder_path) if
f.endswith(('.jpg', '.jpeg', '.png'))]

# Load the images into a numpy array and convert to grayscale
images = []
categories = []
for image_file in image_files:
    img = io.imread(os.path.join(folder_path, image_file))
    img_gray = color.rgb2gray(img)  # Convert to grayscale
    img_resized = transform.resize(img_gray, (28,28),
anti_aliasing=True)  #resized
    images.append(img_resized)
    # Extract category from the first letter in the filename
    category = image_file[0]
    categories.append(category)

# Convert the list of images to a 2D numpy array (num_images,
num_pixels)
X = np.array(images)

# Flatten the array to 1D (num_images, num_pixels)
num_images, height, width = X.shape
X.shape
```

```
/content/drive/MyDrive/AML/HW3/Q1/360
```

```
(360, 28, 28)
```

```
#split into dependent and independent features.
X_data = X.reshape(num_images,height*width)
y = categories
y_sample = ['red' if i=="I"  else 'green' if i == "M" else "blue" for
i in y] # add colors for the plot
```

Question 3A :

PCA with 2 dimensions (1) Amount of Variance preserved with these 2 component(1)

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X_data)
cumulative_variance_explained = np.sum(pca.explained_variance_ratio_)
print(f"Variance explained by the first two principal components:
{cumulative_variance_explained * 100:.2f}%")

Variance explained by the first two principal components: 45.71%
```

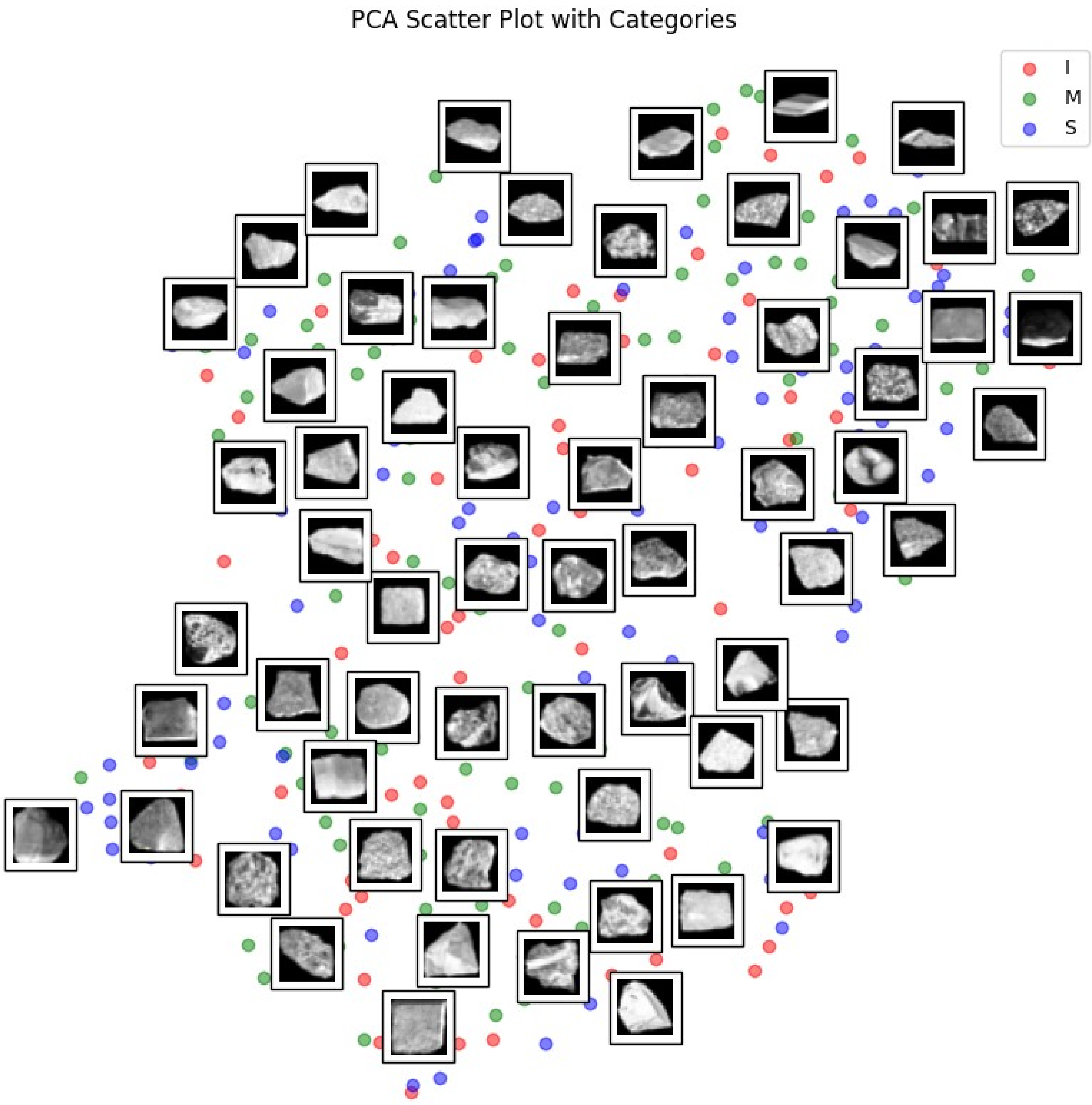
Question-3 B

Scatter Plots of Components with some Rock Images.

```
from sklearn.preprocessing import MinMaxScaler
from matplotlib.offsetbox import AnnotationBbox, OffsetImage

def plot_digits(X, y, min_distance=0.08, images=None, figsize=(13,
10)):
    # Let's scale the input features so that they range from 0 to 1
    X_normalized = MinMaxScaler().fit_transform(X)
    # Now we create the list of coordinates of the digits plotted so
far.
    # We pretend that one is already plotted far away at the start, to
    # avoid `if` statements in the loop below
    neighbors = np.array([[10., 10.]])
    # The rest should be self-explanatory
    plt.figure(figsize=figsize)
    cmap = plt.cm.jet
    digits = np.unique(y_sample)
    import pandas as pd
    df = pd.DataFrame(X_normalized, columns=['PC1', 'PC2'])
    df['Category'] = categories
    colors = {'I': 'red', 'M': 'green', 'S': 'blue'}
    for category, colour in colors.items():
      subset = df[df['Category'] == category]
      plt.scatter(subset['PC1'], subset['PC2'], c=colour,
label=category, alpha=0.5)
    plt.axis("off")
    ax = plt.gca()  # get current axes
    for index, image_coord in enumerate(X_normalized):
      closest_distance = np.linalg.norm(neighbors - image_coord,
axis=1).min()
      if closest_distance > min_distance:
          neighbors = np.r_[neighbors, [image_coord]]
          if images is None:
              plt.text(image_coord[0], image_coord[1],
str((y_sample[index])),
                        color=color,
                        fontdict={"weight": "bold", "size": 16})
          else:
              image = images[index].reshape(28, 28)
              imagebox = AnnotationBbox(OffsetImage(image,
cmap="binary"),image_coord)
              ax.add_artist(imagebox)
    # Add labels and title
    plt.xlabel('Principal Component 1 (PC1)')
    plt.ylabel('Principal Component 2 (PC2)')
    plt.title('Scatter Plot with Categories')

    # Show legend
    plt.legend()

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

Question 3B : PCA :

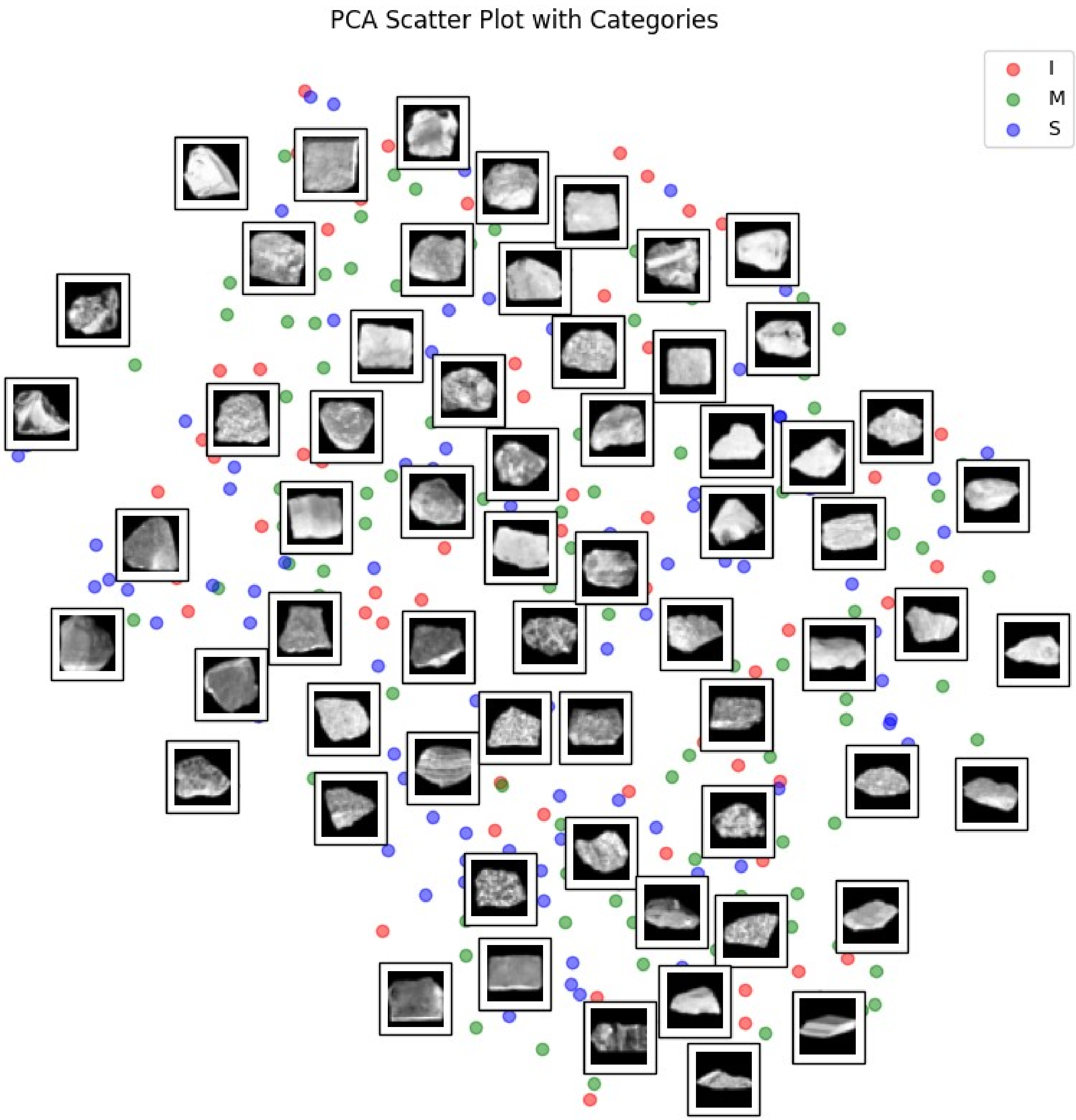
```
plot_digits(X_reduced, y_sample, images=X_data, figsize=(10,10))
```



Question 3B:

A : T-SNE

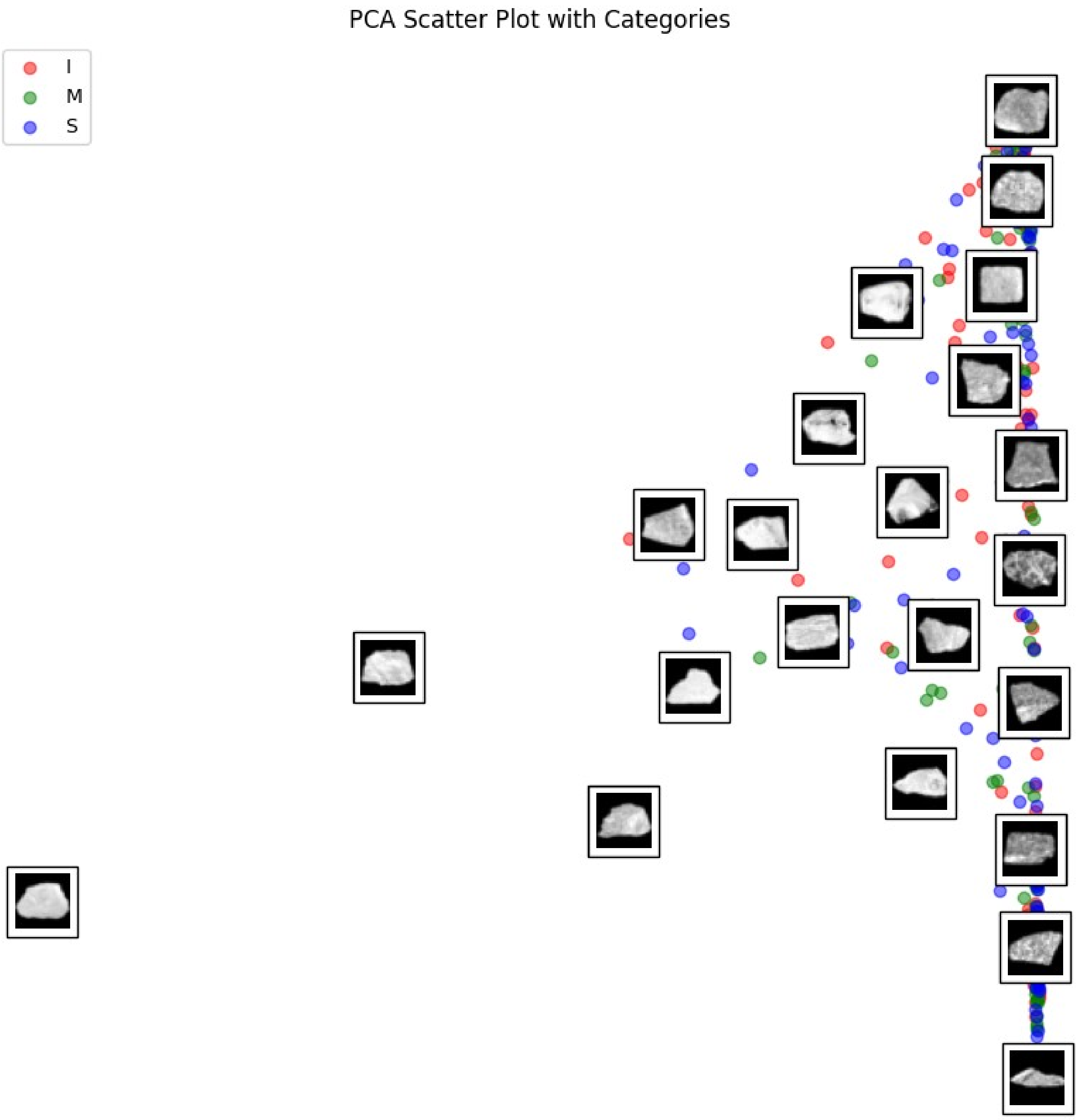
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
X_reduced_tsne = tsne.fit_transform(X_data)
plt.show()

plot_digits(X_reduced_tsne, y_sample, images=X_data, figsize=(10,10))
```



Question 3B:

B : LLE

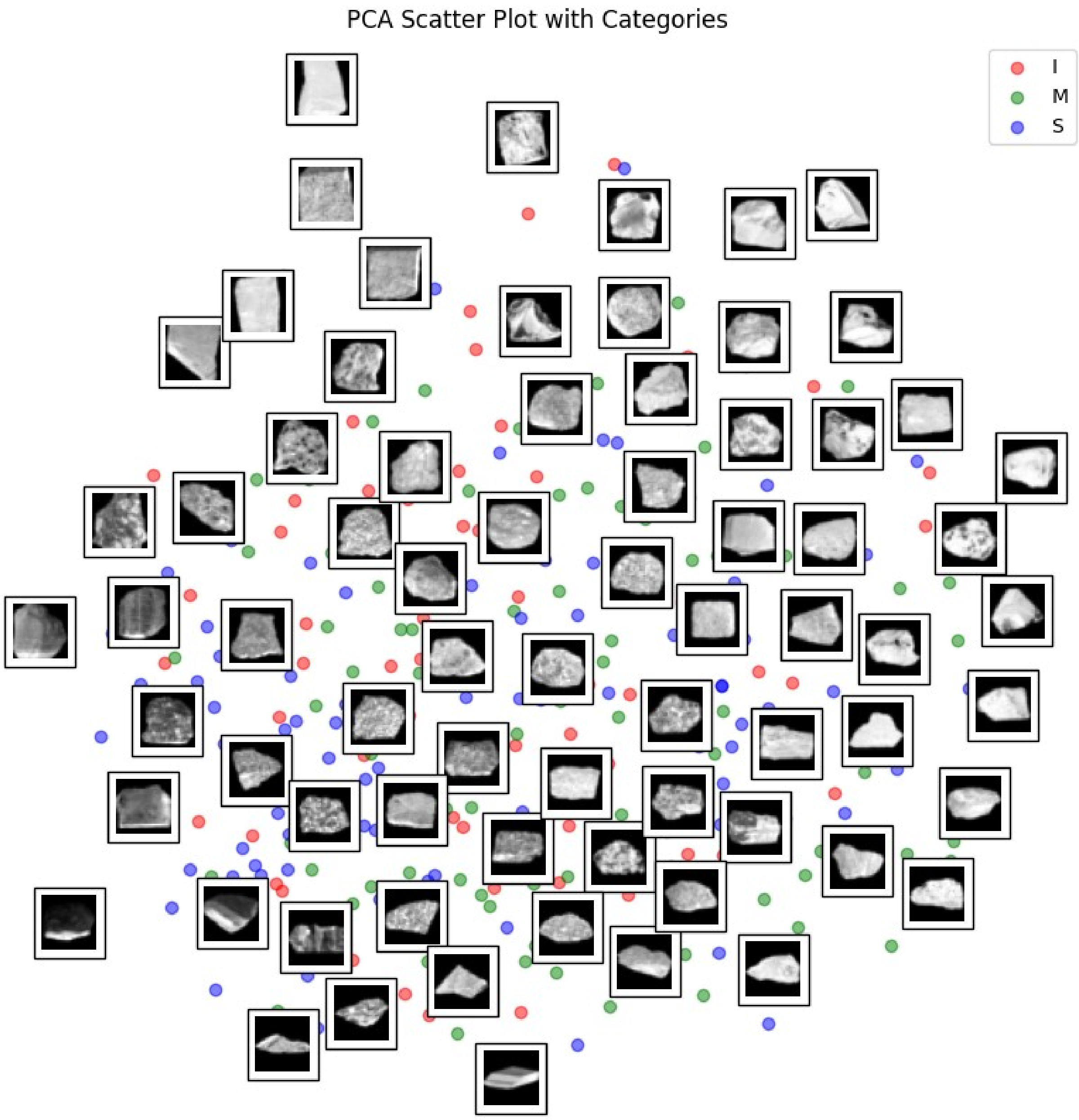
```
from sklearn.manifold import LocallyLinearEmbedding
lle = LocallyLinearEmbedding(n_components=2)
X_lle_reduced = lle.fit_transform(X_data)
plot_digits(X_lle_reduced, y_sample, images=X_data, figsize=(10,10))
```



Question 3B:

C : MDS

```
from sklearn.manifold import MDS
X_mds_reduced =
MDS(n_components=2,normalized_stress=False).fit_transform(X_data)
plot_digits(X_mds_reduced, y_sample,images=X_data, figsize=(10,10))
plt.show()
```



: Question 3 C - Discussion on the visualizations (preferred or not) (1)

All of the plots do have a lot of overlap. Perhaps more data would have helped us to identify these clusters.

PCA looks better when compared to other plots.

```
#I dont prefer any visialization as everything is overlapped
```

4th : Now let's see if these dimensionality reduction techniques can give us similar features to those that humans use to judge the images. File mds_360.txt contains 8 features for each of the images (rankings are in the same order as the images in '360 Rocks' folder. Run PCA, t-SNE, LLE and MDS to reduce the dimensionality of the images to 8. Then, compare those image embeddings with the ones from humans that are in the mds_360.txt file.

Note : for the following part of the quesions, i have resized the image from 800x800 to 128x128 to fit the computations on my colab.

```
%cd {folder_path}
# Load images from the folder
image_files = [f for f in os.listdir(folder_path) if
f.endswith(('.jpg', '.jpeg', '.png'))]

# Load the images into a numpy array and convert to grayscale
images = []
categories = []
for image_file in image_files:
    img = io.imread(os.path.join(folder_path, image_file))
    #img_gray = color.rgb2gray(img)  # Convert to grayscale
    #img_resized = transform.resize(img_gray, (512,512),
anti_aliasing=True)
    img_resized = transform.resize(img, (128,128), anti_aliasing=True)
    images.append(img_resized)
    # Extract category from the first letter in the filename
    category = image_file[0]
    categories.append(category)

# Convert the list of images to a 2D numpy array (num_images,
num_pixels)
X = np.array(images)
print(X.shape)
# Flatten the array to 2D (num_images, num_pixels)
num_images, height, width, channels = X.shape
images_2d = X.reshape(num_images, height * width * channels)
scaler = StandardScaler()
images_standardized = scaler.fit_transform(images_2d)

X_data =X.reshape(num_images,height*width*channels)
```

Question 4:

PCA with 8 components:

```
#reduce to 8 components:
pca = PCA(n_components=8)
X_pca_reduced = pca.fit_transform(images_standardized)
matrix_with_pca_embeddings_data = X_pca_reduced

file_path = "/content/drive/MyDrive/AML/HW3/Q1/mds_360.txt"
matrix_with_human_data = np.loadtxt(file_path)

#perform the procrustes test
from scipy.spatial import procrustes
mtx1, mtx2, pca_disparity = procrustes(matrix_with_human_data,
matrix_with_pca_embeddings_data)
correlation_matrix_pca = np.corrcoef(mtx1, mtx2)
pca_disparity
```

```
0.9792274808609364
```

```
correlation_matrix = np.corrcoef(mtx1, mtx2, rowvar=False)
# Extract the correlation coefficients
correlation_coefficients = correlation_matrix[:mtx1.shape[1],
mtx1.shape[1]:]
# Display the result
print("Correlation Coefficients between corresponding dimensions:")
import pandas as pd
pd.DataFrame(correlation_coefficients)
```

```
Correlation Coefficients between corresponding dimensions:

          0         1         2         3         4         5
6  \
0  0.153999 -0.012381  0.046431 -0.076552 -0.000744  0.021783 -
0.062219
1 -0.010415  0.047970 -0.036824  0.038975 -0.010876 -0.012018
0.006813
2  0.044701 -0.042145  0.095105 -0.050245 -0.019611  0.065091
0.025408
3 -0.075519  0.045708 -0.051486  0.205653 -0.021435 -0.060186
0.048386
4 -0.000675 -0.011723 -0.018471 -0.019702  0.188745 -0.024215
0.020093
5  0.027148 -0.017806  0.084264 -0.076037 -0.033284  0.137219
0.053382
6 -0.076828  0.010002  0.032588  0.060564  0.027362  0.052888
0.130636
7 -0.029464  0.014282 -0.021991  0.019170  0.031297  0.055102
0.062375

          7
0 -0.030703
1  0.012519
2 -0.022062
```

```
3  0.019707
4  0.029572
5  0.071563
6  0.080260
7  0.221382
```

Question 4:

t-SNE

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=8,method='exact')
matrix_with_tsne_embeddings_data =
tsne.fit_transform(images_standardized)
mtx1, mtx2, tsne_disparity = procrustes(matrix_with_human_data,
matrix_with_tsne_embeddings_data)
tsne_disparity
```

```
0.9836681058554037
```

```
correlation_matrix = np.corrcoef(mtx1, mtx2, rowvar=False)
# Extract the correlation coefficients
correlation_coefficient = correlation_matrix[:mtx1.shape[1],
mtx1.shape[1]:]
# Display the result
print("Correlation Coefficients between corresponding dimensions: t-
SNE")
pd.DataFrame(correlation_coefficient)
```

```
Correlation Coefficients between corresponding dimensions: t-SNE
```

```
          0         1         2         3         4         5
6  \
0  0.153999 -0.012381  0.046431 -0.076552 -0.000744  0.021783 -
0.062219
1 -0.010415  0.047970 -0.036824  0.038975 -0.010876 -0.012018
0.006813
2  0.044701 -0.042145  0.095105 -0.050245 -0.019611  0.065091
0.025408
3 -0.075519  0.045708 -0.051486  0.205653 -0.021435 -0.060186
0.048386
4 -0.000675 -0.011723 -0.018471 -0.019702  0.188745 -0.024215
0.020093
5  0.027148 -0.017806  0.084264 -0.076037 -0.033284  0.137219
0.053382
6 -0.076828  0.010002  0.032588  0.060564  0.027362  0.052888
0.130636
7 -0.029464  0.014282 -0.021991  0.019170  0.031297  0.055102
0.062375
```

```
          7
0 -0.030703
1  0.012519
2 -0.022062
3  0.019707
4  0.029572
5  0.071563
6  0.080260
7  0.221382
```

Question 4:

LLE:

```
from sklearn.manifold import LocallyLinearEmbedding
lle = LocallyLinearEmbedding(n_components=8)
matrix_with_lle_embeddings_data =
lle.fit_transform(images_standardized)
mtx1, mtx2, lle_disparity = procrustes(matrix_with_human_data,
matrix_with_lle_embeddings_data)
lle_disparity

0.9801696615862322

correlation_matrix = np.corrcoef(mtx1, mtx2, rowvar=False)
# Extract the correlation coefficients
correlation_coefficient_lle = correlation_matrix[:mtx1.shape[1],
mtx1.shape[1]:]
# Display the result
print("Correlation Coefficients between corresponding dimensions: t-
SNE")
pd.DataFrame(correlation_coefficient_lle)

Correlation Coefficients between corresponding dimensions: t-SNE

          0         1         2         3         4         5
6  \
0  0.142101  0.013764  0.031324 -0.044334  0.041790 -0.010191 -
0.009199
1  0.014823  0.107643 -0.010799  0.028362  0.040605  0.032150
0.018183
2  0.042105 -0.013479  0.088020 -0.047428  0.017336  0.049692
0.021854
3 -0.048481  0.028798 -0.038584  0.152423 -0.015590  0.002433 -
0.011964
4  0.055339  0.049928  0.017079 -0.018879  0.188806 -0.023902 -
0.019499
5 -0.010696  0.031333  0.038802  0.002335 -0.018945  0.156760 -
0.026211
6 -0.012957  0.023779  0.022899 -0.015410 -0.020739 -0.035172
```

```
0.145838
7 -0.046705 -0.031084 -0.011987  0.004082  0.042461  0.019563
0.023707

          7
0 -0.032587
1 -0.023357
2 -0.011242
3  0.003115
4  0.039231
5  0.014327
6  0.023297
7  0.158035
```

Question 4:

MDS:

```
from sklearn.manifold import MDS
matrix_with_mds_embeddings_data =
MDS(n_components=8,normalized_stress=False).fit_transform(X_data)
mtx1, mtx2, mds_disparity = procrustes(matrix_with_human_data,
matrix_with_mds_embeddings_data)
mds_disparity

0.9789384991615805

correlation_matrix = np.corrcoef(mtx1, mtx2, rowvar=False)
# Extract the correlation coefficients
correlation_coefficient = correlation_matrix[:mtx1.shape[1],
mtx1.shape[1]:]
# Display the result
print("Correlation Coefficients between corresponding dimensions: t-
SNE")
pd.DataFrame(correlation_coefficient)

Correlation Coefficients between corresponding dimensions: t-SNE

          0         1         2         3         4         5
6  \
0  0.160685  0.020231  0.013026 -0.073962 -0.007740  0.009602 -
0.052312
1  0.018659  0.099399 -0.000136  0.060899 -0.010155 -0.007225
0.025085
2  0.015866 -0.000180  0.100867 -0.040101 -0.026450  0.073564
0.011264
3 -0.091684  0.081850 -0.040811  0.148299 -0.010180 -0.055236
0.020534
4 -0.009276 -0.013194 -0.026022 -0.009841  0.222574 -0.019807 -
0.009596
```

```
5  0.013010 -0.010613  0.081831 -0.060374 -0.022395  0.137645
0.057066
6 -0.082164  0.042718  0.014525  0.026018 -0.012578  0.066151
0.138906
7 -0.032388 -0.003812 -0.037101  0.014320  0.049636  0.045061
0.030582

          7
0 -0.022705
1 -0.002465
2 -0.031681
3  0.012444
4  0.041698
5  0.042802
6  0.033673
7  0.176675
```

Final Disparity Table for all techniques:

```
disparity = [[pca_disparity,"PCA"],[tsne_disparity,"TSNE"],
[lle_disparity,"LLE"],[mds_disparity,"MDS"]]
import pandas as pd
disparity_df = pd.DataFrame(disparity,columns=["Disparity
scores","Method"])
disparity_df

   Disparity scores Method
0          0.979227    PCA
1          0.983668   TSNE
2          0.980170    LLE
3          0.978938    MDS
```

Question 5a:

To speed up the algorithm, use PCA to reduce the dimensionality of the dataset to two. Determine the number of clusters using one of the techniques we discussed in class.

```
%cd {folder_path}
# Load images from the folder
image_files = [f for f in os.listdir(folder_path) if
f.endswith(('.jpg', '.jpeg', '.png'))]

# Load the images into a numpy array and convert to grayscale
images = []
categories = []
```

```
labels=[]
for image_file in image_files:
    img = io.imread(os.path.join(folder_path, image_file))
    img_gray = color.rgb2gray(img)  # Convert to grayscale
    images.append(img_gray)
    category = image_file[0]
    categories.append(category)
    category = image_file[0]
    if category == 'I':
      labels.append(0)
    elif category == 'M':
      labels.append(1)
    elif category == 'S':
       labels.append(2)
# Convert the list of images to a 2D numpy array (num_images,
num_pixels)
images_array = np.array(images)

# Flatten the array to 1D (num_images, num_pixels)
num_images, height, width = images_array.shape
images_1d = images_array.reshape(num_images, height * width)

# Standardize the data (mean=0, variance=1)
scaler = StandardScaler()
images_standardized = scaler.fit_transform(images_1d)
```

```
/content/drive/MyDrive/AML/360 Rocks
```

```
# Apply PCA with 2 components
n_components = 2
pca = PCA(n_components=n_components)
images_pca = pca.fit_transform(images_standardized)

# Print the cumulative explained variance
cumulative_variance_explained = np.sum(pca.explained_variance_ratio_)
print(f"Variance explained by the first two principal components:
{cumulative_variance_explained * 100:.2f}%")
```

```
Variance explained by the first two principal components: 30.85%
```

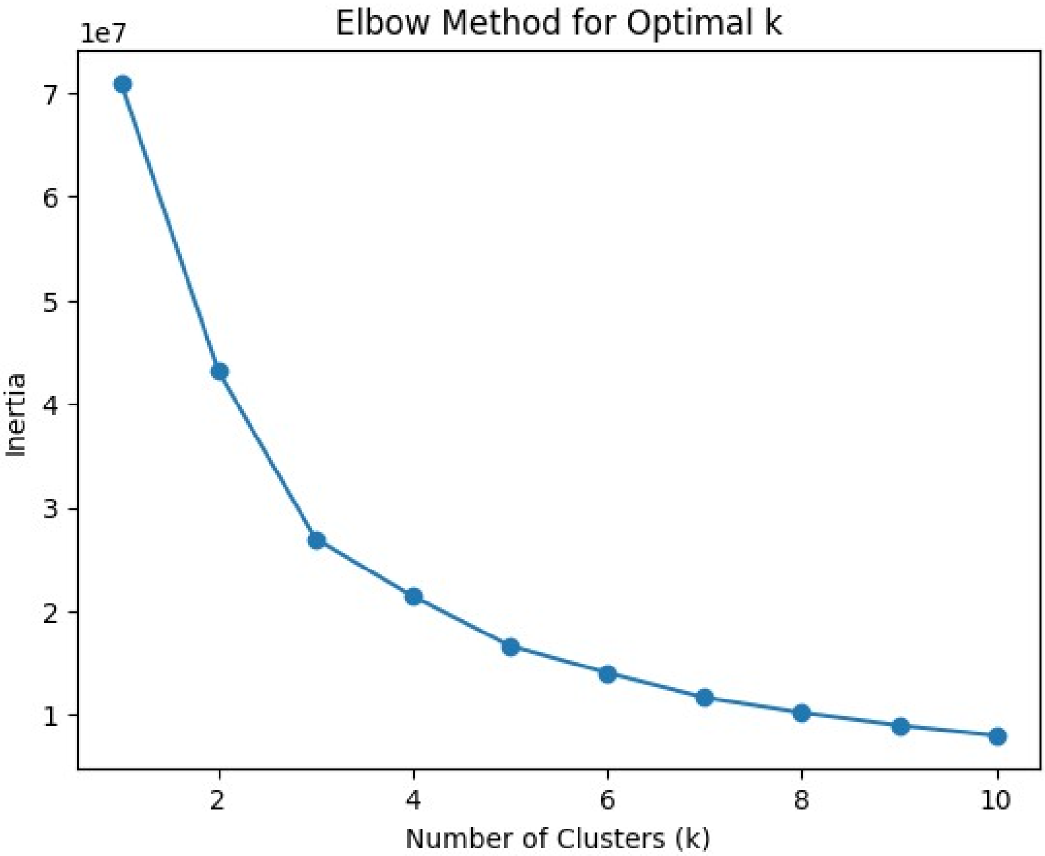
Find number of cluster using Elbow method

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Try different values of k
k_values = range(1, 11)
inertias = []

for k in k_values:
```

```python
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(images_pca)
    inertias.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(k_values, inertias, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_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` explicitly
to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
```

```
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_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` explicitly to suppress the
warning
  warnings.warn(
```



As we can see for k=3 we will mostly get good results in clustering

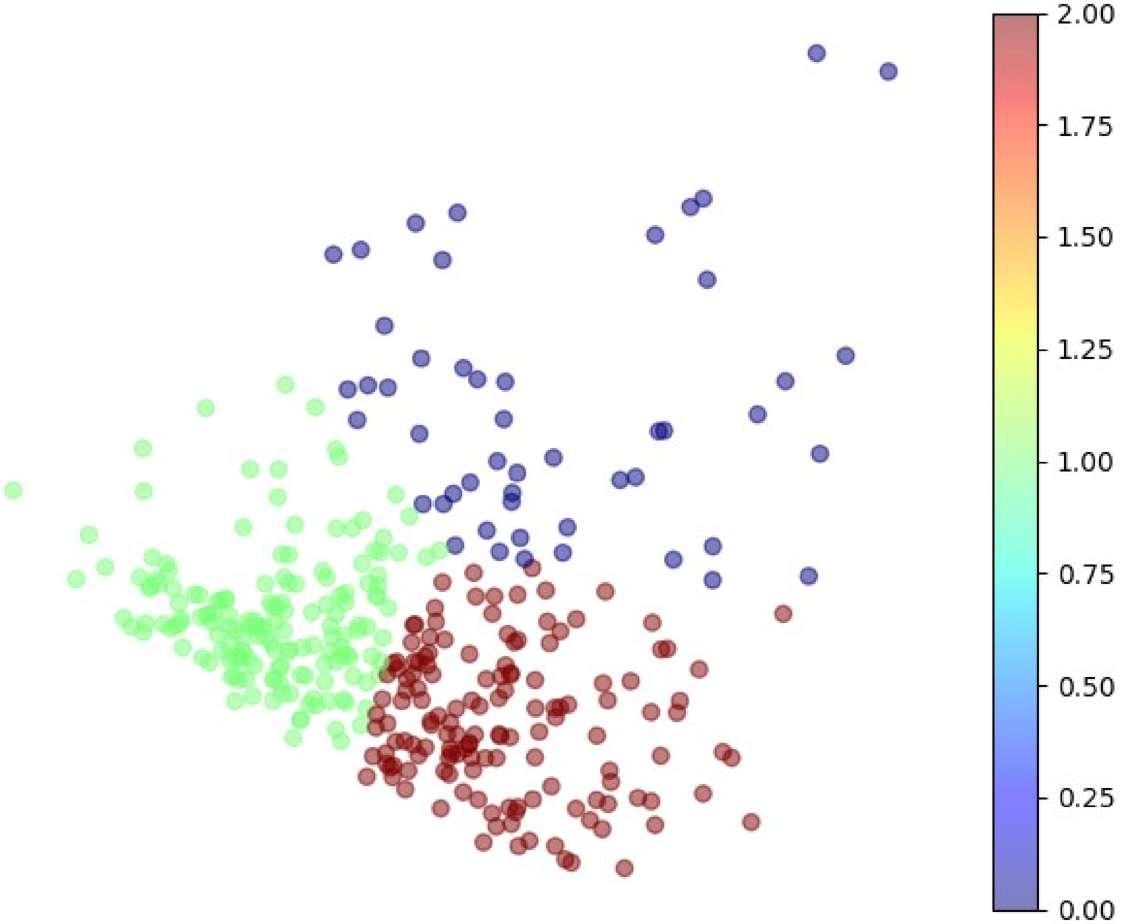
Q 5b) Visualize the clusters in a similar way to the visualization in the ipynb file with points of each cluster uniquely labelled

```
optimal_k = 3 # the optimal number of clusters from the elbow method
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
clusters = kmeans.fit_predict(images_pca)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
```

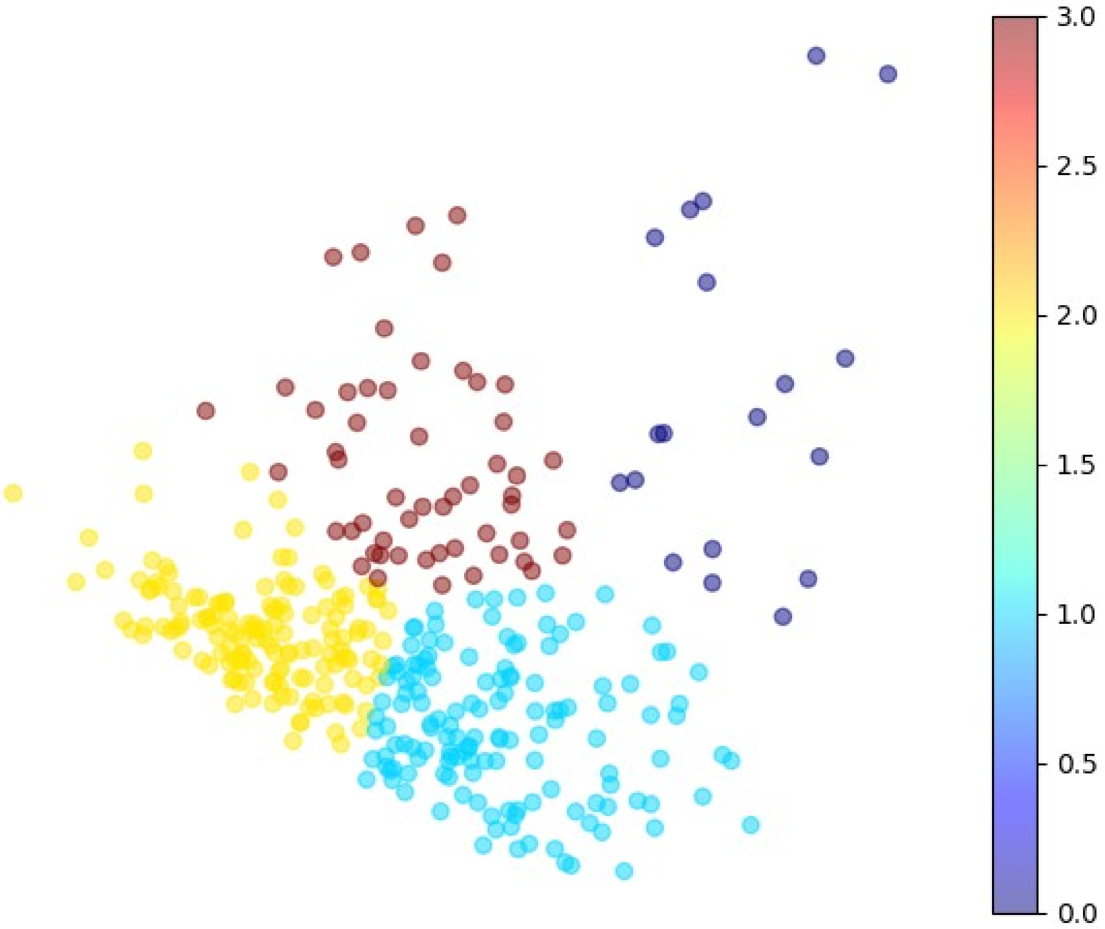
```
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```

```
plt.figure(figsize=(8, 6))
plt.scatter(images_pca[:,0], images_pca[:, 1],
            c=clusters, cmap="jet", alpha=0.5)
plt.axis('off')
plt.colorbar()
plt.show()
```



```
#plot a 2d cluster
optimal_k = 4
kmeans2 = KMeans(n_clusters=optimal_k, init="random", random_state=42)
clusters = kmeans2.fit_predict(images_pca)
plt.figure(figsize=(8, 6))
plt.scatter(images_pca[:,0], images_pca[:, 1],
            c=clusters, cmap="jet", alpha=0.5)
plt.axis('off')
plt.colorbar()
plt.show()
```

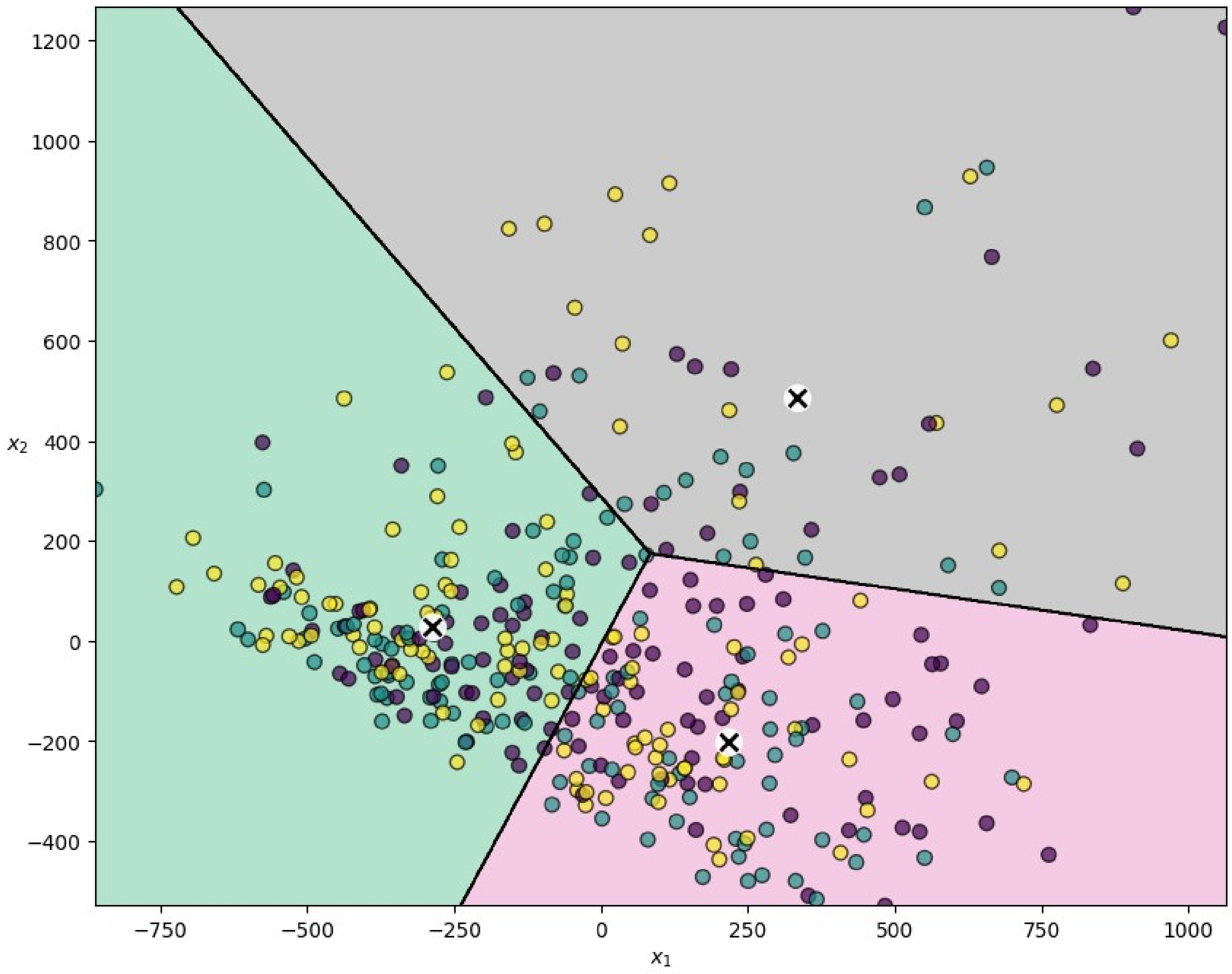
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



Q 5b) Visualize the clusters in a similar way to the visualization in the ipynb file with points of each cluster uniquely labelled

```
def plot_data(X):
  plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)
def plot_centroids(centroids, weights=None, circle_color='w',
cross_color='k'):
  if weights is not None:
    centroids = centroids[weights > weights.max() / 10]
  plt.scatter(centroids[:, 0], centroids[:, 1],
              marker='o', s=35, linewidths=8,
                color=circle_color, zorder=10, alpha=0.9)
  plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=2, linewidths=12,
                color=cross_color, zorder=11, alpha=1)
def plot_decision_boundaries(clusterer, labels, X, resolution=1000,
show_centroids=True,
                             show_xlabels=True, show_ylabels=True):
  mins = X.min(axis=0) - 0.1
  maxs = X.max(axis=0) + 0.1
  xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
  Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
  Z = Z.reshape(xx.shape)
  plt.contourf(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                cmap="Pastel2")
  plt.contour(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                linewidths=1, colors='k')
  plt.scatter(X[:, 0], X[:, 1], c=[label for label in labels],
              cmap='viridis', edgecolors='k', marker='o', s=50,
alpha=0.7)
  if show_centroids:
    plot_centroids(clusterer.cluster_centers_)
  if show_xlabels:
    plt.xlabel("$x_1$")
  else:
    plt.tick_params(labelbottom=False)
  if show_ylabels:
    plt.ylabel("$x_2$", rotation=0)
  else:
    plt.tick_params(labelleft=False)

optimal_k = 3
kmeans3 = KMeans(n_clusters=optimal_k, init="random", random_state=42)
clusters = kmeans3.fit_predict(images_pca)
plt.figure(figsize=(10, 8))
plot_decision_boundaries(kmeans3, labels, images_pca)
plt.show()
```

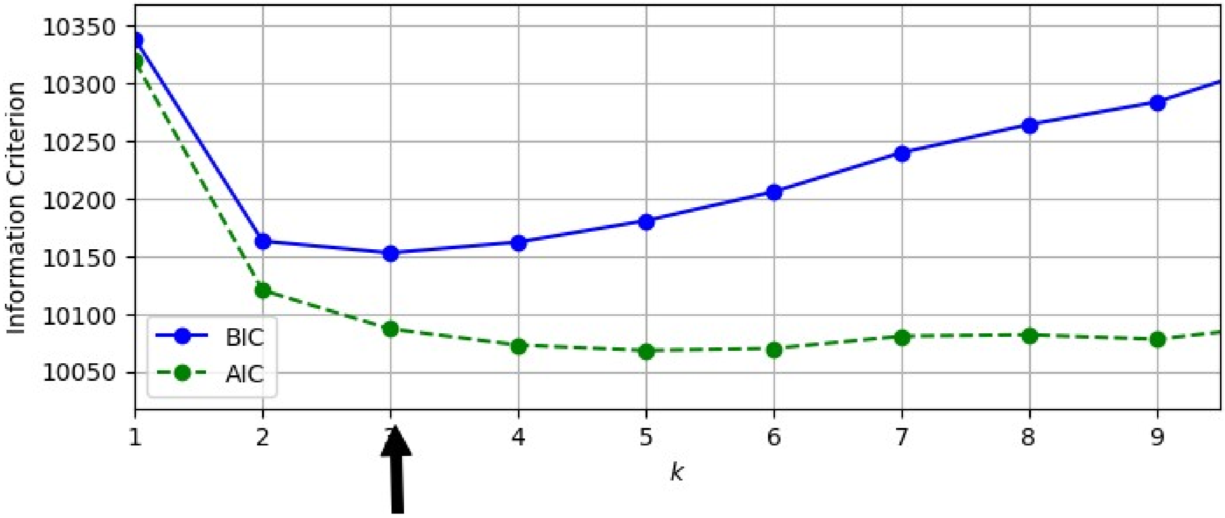
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly
to suppress the warning
  warnings.warn(
```



Q6:

A: Number of clusters and EM implementation

```
from sklearn.mixture import GaussianMixture

gms_per_k = [GaussianMixture(n_components=k, n_init=10,
random_state=42).fit(images_pca)
             for k in range(1, 11)]
bics = [model.bic(images_pca) for model in gms_per_k]
aics = [model.aic(images_pca) for model in gms_per_k]

plt.figure(figsize=(8, 3))
plt.plot(range(1, 11), bics, "bo-", label="BIC")
plt.plot(range(1, 11), aics, "go--", label="AIC")
plt.xlabel("$k$")
plt.ylabel("Information Criterion")
plt.axis([1, 9.5, min(aics) - 50, max(aics) + 50])
plt.annotate("", xy=(3, bics[2]), xytext=(3.4, 8650),
             arrowprops=dict(facecolor='black', shrink=0.1))
plt.legend()
plt.grid()
plt.show()
```


Information Criterion
10350
10300
10250
10200
10150
10100
10050
1
2
3
4
5
6
7
8
9
k
BIC
AIC

Upon applying the Bayesian Information Criterion (BIC) to the Expectation-Maximization (EM) algorithm with Principal Component Analysis (PCA) dimensionality reduction, we observed that the BIC scores were minimized when the number of clusters = 3. This implies that, based on the balance between model fit and complexity, the most suitable configuration for clustering was achieved with three clusters.

```
gm = GaussianMixture(n_components=3, n_init=10, random_state=42)
y_pred=gm.fit(images_pca)
gm.weights_
```

```
array([0.20934661, 0.46931359, 0.3213398 ])
```

```
gm.means_
```

```
array([[ 55.51656056, 100.74265172],
       [ 34.55990284, -31.3653715 ],
       [-86.6423501 , -19.82305863]])
```

```
#Did the algorithm actually converge?
gm.converged_
```

```
True
```

```
#How many iterations did it take?
gm.n_iter_
```

```
21
```

Let's predict which cluster each instance belongs to (hard clustering) or the probabilities that it came from each cluster.

```
gm.predict(images_pca)
```

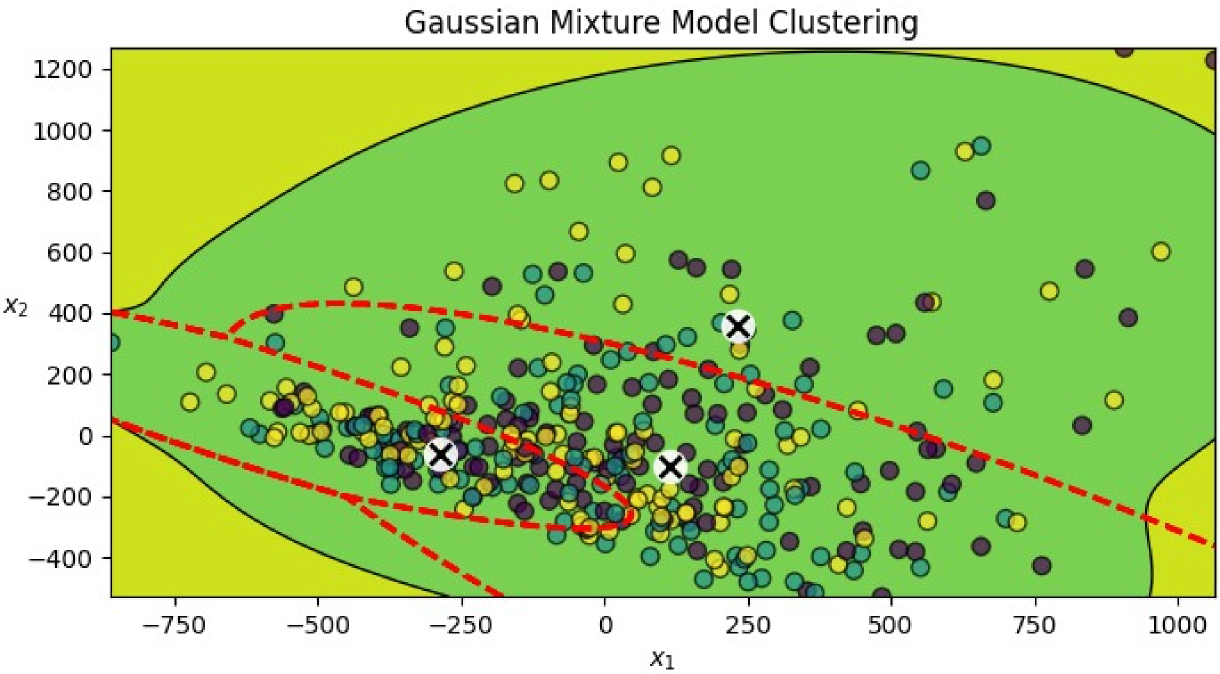
```
array([1, 1, 1, 0, 1, 2, 1, 0, 0, 1, 2, 2, 1, 2, 1, 0, 1, 1, 1, 1, 1,
1,
       2, 0, 1, 1, 1, 1, 0, 1, 1, 1, 2, 2, 0, 1, 1, 2, 1, 0, 1, 2, 2,
2,
       1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 2, 2, 1, 0, 2, 2, 1, 1, 1, 1, 1,
1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 0, 0, 2, 1,
0,
       1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 0, 2, 0, 2, 2, 2, 2, 2,
0,
       1, 2, 2, 2, 2, 2, 1, 0, 2, 2, 1, 1, 2, 2, 1, 2, 0, 1, 2, 1, 2,
1,
       2, 1, 1, 2, 2, 2, 0, 0, 2, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 2, 1,
1,
       2, 1, 2, 1, 2, 1, 1, 0, 1, 0, 1, 0, 1, 1, 2, 1, 1, 1, 0, 1, 1,
1,
       2, 1, 2, 1, 2, 1, 2, 2, 2, 1, 0, 2, 0, 0, 1, 2, 1, 1, 0, 1, 1,
2,
       1, 2, 1, 1, 2, 1, 0, 2, 2, 1, 1, 1, 1, 0, 1, 2, 2, 1, 0, 1, 2,
2,
       1, 1, 2, 0, 2, 1, 1, 1, 1, 2, 1, 1, 2, 0, 1, 1, 0, 2, 1, 1, 1,
0,
       1, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 0, 1, 0, 2, 2, 1, 2,
2,
       1, 1, 1, 2, 1, 1, 1, 0, 1, 1, 0, 2, 2, 1, 0, 2, 2, 2, 0, 2, 2,
0,
       1, 0, 2, 1, 2, 2, 1, 2, 2, 2, 2, 0, 0, 1, 2, 1, 1, 0, 2, 1, 1,
1,
       2, 2, 2, 2, 1, 2, 1, 1, 1, 2, 0, 2, 2, 0, 1, 0, 1, 0, 2, 0, 2,
1,
       2, 1, 0, 1, 1, 0, 2, 1, 2, 2, 0, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1,
0,
       1, 0, 2, 1, 2, 2, 2, 1])
```

```
gm.predict_proba(images_pca).round(3)
```

```
array([[0.03 , 0.841, 0.13 ],
       [0.045, 0.917, 0.038],
       [0.064, 0.932, 0.004],
       ...,
       [0.034, 0.047, 0.919],
       [0.041, 0.03 , 0.929],
       [0.102, 0.893, 0.005]])
```

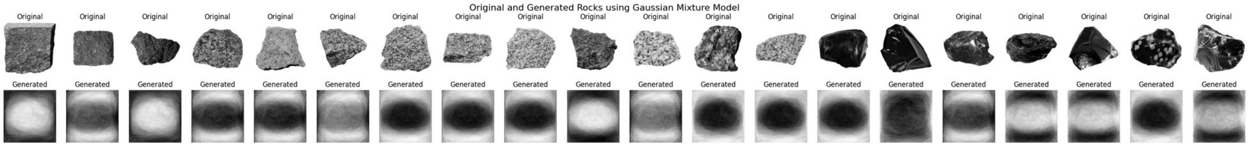
6 B)Visualize the clusters

```
optimal_components = 3 #define optimal component
gmm1 = GaussianMixture(n_components=optimal_components,
random_state=42)
clusters = gmm1.fit_predict(images_pca)

def plot_centroids(centroids, weights=None, circle_color='w',
cross_color='k'):
  if weights is not None:
    centroids = centroids[weights > weights.max() / 10]
  plt.scatter(centroids[:, 0], centroids[:, 1],
              marker='o', s=35, linewidths=8,
              color=circle_color, zorder=10, alpha=0.9)
  plt.scatter(centroids[:, 0], centroids[:, 1],
              marker='x', s=2, linewidths=12,
              color=cross_color, zorder=11, alpha=1)

from matplotlib.colors import LogNorm
import matplotlib.pyplot as plt

def plot_gaussian_mixture(clusterer, X, labels, resolution=1000,
show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = -clusterer.score_samples(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z,
                 norm=LogNorm(vmin=1.0, vmax=30.0),
                 levels=np.logspace(0, 2, 12))
    plt.contour(xx, yy, Z,
                norm=LogNorm(vmin=1.0, vmax=30.0),
                levels=np.logspace(0, 2, 12),
                linewidths=1, colors='k')

    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contour(xx, yy, Z,
                linewidths=2, colors='r', linestyles='dashed')

    plt.scatter(X[:, 0], X[:, 1], c=labels,
                cmap='viridis', edgecolors='k', marker='o', s=50,
alpha=0.7)

    plot_centroids(clusterer.means_, clusterer.weights_)

    plt.xlabel("$x_1$")
    if show_ylabels:
        plt.ylabel("$x_2$", rotation=0)
    else:
        plt.tick_params(labelleft=False)


# Assuming y_pred_gm contains the cluster labels
gm = GaussianMixture(n_components=3, n_init=10, random_state=42)
y_pred_gm = gm.fit_predict(images_pca)

plt.figure(figsize=(8, 4))
plot_gaussian_mixture(gm, images_pca, labels, resolution=1000)
plt.title("Gaussian Mixture Model Clustering")
plt.show()
```



C) Use the model to generate 20 new rocks

```
from sklearn.mixture import GaussianMixture

gm = GaussianMixture(n_components=3, random_state=42)
y_pred = gm.fit_predict(images_pca)
n_gen_rocks = 20
gen_rocks_reduced, y_gen_rocks = gm.sample(n_samples=n_gen_rocks)
gen_rocks = pca.inverse_transform(gen_rocks_reduced)
gen_rocks = gen_rocks.reshape(20,800,800)


fig, axes = plt.subplots(2, n_gen_rocks, figsize=(2 * n_gen_rocks, 4))

for i in range(n_gen_rocks):
    # Plot Original Rocks
    axes[0, i].imshow(images[i], cmap='gray')
    axes[0, i].axis('off')
    axes[0, i].set_title('Original')

    # Plot Generated Rocks
    axes[1, i].imshow(gen_rocks[i], cmap='gray')
    axes[1, i].axis('off')
    axes[1, i].set_title('Generated')

plt.suptitle('Original and Generated Rocks using Gaussian Mixture 
Model', fontsize=16)
plt.show()
```


Original and Generated Rocks using Gaussian Mixture Model
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Original
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated
Generated

4syhnea02

November 18, 2023

Question 7:

[1]:

```python
#lets import packages
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import pathlib, os, random
import numpy as np
import pandas as pd
import tensorflow as tf
import cv2
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Activation, BatchNormalization, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import EarlyStopping,ModelCheckpoint
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
folder_path = "/content/drive/MyDrive/AML/HW3/Q1/360"
%cd {folder_path}
train_path = "/content/drive/MyDrive/AML/HW3/Q1/360/"
valid_path = "/content/drive/MyDrive/AML/HW3/Q1/120/"
from keras import regularizers
from keras import models
from sklearn.preprocessing import StandardScaler
import time
```

```
Mounted at /content/drive
/content/drive/MyDrive/AML/HW3/Q1
```

[2]:

```python
height,width,channels=400,400,1
```

[2]:

```python
#load images
def load_images(folder_path):
    images = []
    labels = []
    for filename in os.listdir(folder_path):
      if filename.endswith(".jpg"):
        img = cv2.imread(os.path.join(folder_path, filename))
        label = filename[0] # Extract first character which helps␣
        img = cv2.resize(img, (400,400))
        images.append(img)
        labels.append(label)
    return np.array(images), np.array(labels)
```

[74]:

```
#split our data
X_train, y_train = load_images(train_path)
X_val, y_val = load_images(valid_path)


# scaling it btw 0-1, we divide it by 255 as its the max value for a pixel
X_train = X_train / 255.0
X_val = X_val / 255.0
```

[75]:

```
# convert images to grayscale
def to_gray(images):
  gray_images = [cv2.cvtColor(cv2.convertScaleAbs(img), cv2.COLOR_BGR2GRAY) for␣
 ↪img in images]
  return gray_images


# standardize the data
def standardize(images):
  #flatten the data
  flattened_images = [img.flatten() for img in images]
  X = np.array(flattened_images)
  #scale the data using standard scaler
  scaler = StandardScaler()
  X_standardized = scaler.fit_transform(X)
  return X_standardized
```

[76]:

```
# Convert images to grayscale
X_train = to_gray(X_train)
# Standardize the grayscale images

X_train = standardize(X_train)
# Convert images to grayscale
X_val = to_gray(X_val)
# Standardize the grayscale images
X_val = standardize(X_val)
```

[42]:

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
```

```
#label encoding
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_val_encoded = label_encoder.fit_transform(y_val)

# convert to one_hot
y_train_one_hot = to_categorical(y_train_encoded, num_classes=3)
y_val_one_hot = to_categorical(y_val_encoded, num_classes=3)

y_train, y_val = y_train.flatten(), y_val.flatten()
```

[156]:
```
# define our model

height,width,channels=400,400,1


model = models.Sequential()
model.add(layers.Flatten(input_shape=X_train[0].shape))
model.add(layers.Dense(100, activation='relu',kernel_regularizer=regularizers.
 ↪l2(0.05)))
model.add(layers.Dense(64, activation='relu')),
model.add(layers.Dense(8, activation='relu',kernel_regularizer=regularizers.
 ↪l2(0.05)))   # 8 neurons in last-1 layer
model.add(layers.Dense(3, activation='softmax'))# 3 neurons in last layer
```

[157]:
```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.
 ↪001),loss='categorical_crossentropy',metrics=['accuracy'])
```

[158]:
```
import tensorflow as tf
tf.config.run_functions_eagerly(True)
start_time = time.time()
print("Trainign Begins")
history = model.fit(X_train, y_train_one_hot,validation_data=(X_val,␣
 ↪y_val_one_hot),epochs=30, batch_size=32)

# Record the end time
print("Training ends/n")
end_time = time.time()
print("Total time" ,(end_time -start_time))
```

```
Trainign Begins
Epoch 1/30
12/12 [==============================] - 9s 739ms/step - loss: 4.2580 -
accuracy: 0.3361 - val_loss: 2.9405 - val_accuracy: 0.3083
Epoch 2/30
```

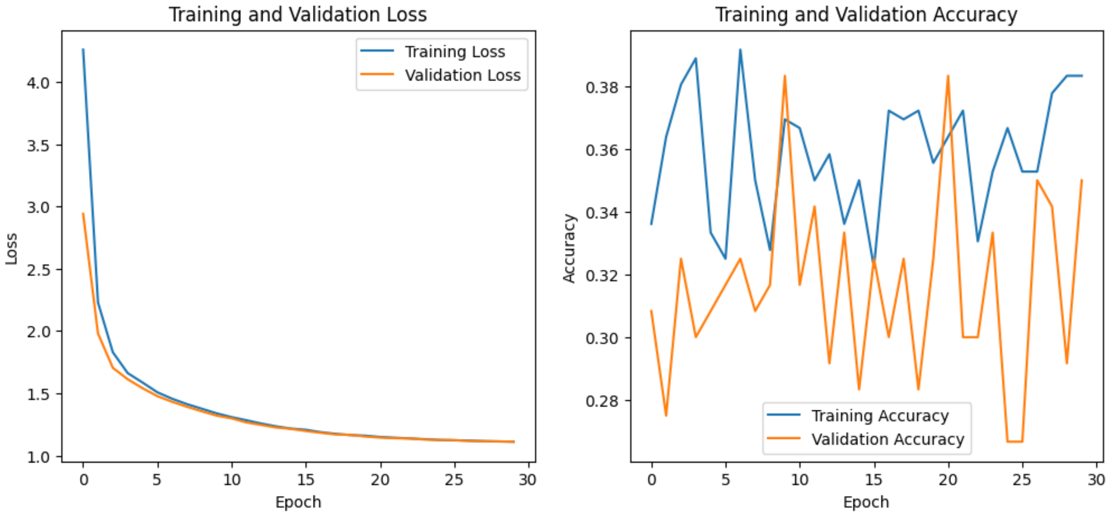
```
12/12 [==============================] - 10s 834ms/step - loss: 2.2285 -
accuracy: 0.3639 - val_loss: 1.9798 - val_accuracy: 0.2750
Epoch 3/30
12/12 [==============================] - 8s 623ms/step - loss: 1.8293 -
accuracy: 0.3806 - val_loss: 1.7041 - val_accuracy: 0.3250
Epoch 4/30
12/12 [==============================] - 11s 892ms/step - loss: 1.6618 -
accuracy: 0.3889 - val_loss: 1.6125 - val_accuracy: 0.3000
Epoch 5/30
12/12 [==============================] - 9s 773ms/step - loss: 1.5863 -
accuracy: 0.3333 - val_loss: 1.5405 - val_accuracy: 0.3083
Epoch 6/30
12/12 [==============================] - 10s 800ms/step - loss: 1.5077 -
accuracy: 0.3250 - val_loss: 1.4771 - val_accuracy: 0.3167
Epoch 7/30
12/12 [==============================] - 9s 716ms/step - loss: 1.4557 -
accuracy: 0.3917 - val_loss: 1.4313 - val_accuracy: 0.3250
Epoch 8/30
12/12 [==============================] - 9s 750ms/step - loss: 1.4123 -
accuracy: 0.3500 - val_loss: 1.3914 - val_accuracy: 0.3083
Epoch 9/30
12/12 [==============================] - 10s 844ms/step - loss: 1.3755 -
accuracy: 0.3278 - val_loss: 1.3546 - val_accuracy: 0.3167
Epoch 10/30
12/12 [==============================] - 11s 918ms/step - loss: 1.3385 -
accuracy: 0.3694 - val_loss: 1.3204 - val_accuracy: 0.3833
Epoch 11/30
12/12 [==============================] - 10s 893ms/step - loss: 1.3084 -
accuracy: 0.3667 - val_loss: 1.2984 - val_accuracy: 0.3167
Epoch 12/30
12/12 [==============================] - 8s 678ms/step - loss: 1.2833 -
accuracy: 0.3500 - val_loss: 1.2657 - val_accuracy: 0.3417
Epoch 13/30
12/12 [==============================] - 11s 901ms/step - loss: 1.2571 -
accuracy: 0.3583 - val_loss: 1.2455 - val_accuracy: 0.2917
Epoch 14/30
12/12 [==============================] - 9s 735ms/step - loss: 1.2341 -
accuracy: 0.3361 - val_loss: 1.2251 - val_accuracy: 0.3333
Epoch 15/30
12/12 [==============================] - 13s 1s/step - loss: 1.2165 - accuracy:
0.3500 - val_loss: 1.2142 - val_accuracy: 0.2833
Epoch 16/30
12/12 [==============================] - 15s 1s/step - loss: 1.2068 - accuracy:
0.3222 - val_loss: 1.1969 - val_accuracy: 0.3250
Epoch 17/30
12/12 [==============================] - 9s 788ms/step - loss: 1.1877 -
accuracy: 0.3722 - val_loss: 1.1826 - val_accuracy: 0.3000
Epoch 18/30
```

```
12/12 [==============================] - 10s 820ms/step - loss: 1.1742 -
accuracy: 0.3694 - val_loss: 1.1696 - val_accuracy: 0.3250
Epoch 19/30
12/12 [==============================] - 11s 906ms/step - loss: 1.1640 -
accuracy: 0.3722 - val_loss: 1.1663 - val_accuracy: 0.2833
Epoch 20/30
12/12 [==============================] - 9s 729ms/step - loss: 1.1595 -
accuracy: 0.3556 - val_loss: 1.1524 - val_accuracy: 0.3250
Epoch 21/30
12/12 [==============================] - 10s 881ms/step - loss: 1.1491 -
accuracy: 0.3639 - val_loss: 1.1441 - val_accuracy: 0.3833
Epoch 22/30
12/12 [==============================] - 8s 631ms/step - loss: 1.1428 -
accuracy: 0.3722 - val_loss: 1.1391 - val_accuracy: 0.3000
Epoch 23/30
12/12 [==============================] - 10s 838ms/step - loss: 1.1375 -
accuracy: 0.3306 - val_loss: 1.1368 - val_accuracy: 0.3000
Epoch 24/30
12/12 [==============================] - 8s 649ms/step - loss: 1.1313 -
accuracy: 0.3528 - val_loss: 1.1284 - val_accuracy: 0.3333
Epoch 25/30
12/12 [==============================] - 10s 876ms/step - loss: 1.1249 -
accuracy: 0.3667 - val_loss: 1.1255 - val_accuracy: 0.2667
Epoch 26/30
12/12 [==============================] - 8s 654ms/step - loss: 1.1225 -
accuracy: 0.3528 - val_loss: 1.1239 - val_accuracy: 0.2667
Epoch 27/30
12/12 [==============================] - 10s 883ms/step - loss: 1.1195 -
accuracy: 0.3528 - val_loss: 1.1171 - val_accuracy: 0.3500
Epoch 28/30
12/12 [==============================] - 9s 728ms/step - loss: 1.1158 -
accuracy: 0.3778 - val_loss: 1.1146 - val_accuracy: 0.3417
Epoch 29/30
12/12 [==============================] - 11s 916ms/step - loss: 1.1130 -
accuracy: 0.3833 - val_loss: 1.1139 - val_accuracy: 0.2917
Epoch 30/30
12/12 [==============================] - 9s 800ms/step - loss: 1.1110 -
accuracy: 0.3833 - val_loss: 1.1112 - val_accuracy: 0.3500
Training ends/n
Total time 322.49855637550354
```

Question 7 A: Total time 1243.2983515262604

[159]:

```python
# Plot training and validation loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



The training and validation accuracy plot does seem a little wierd, but because we have built a model that is fairly simpler with data that has only around 360 images, there are chances of model not capturing the entire variance in the data. Perhaps a better model and access to GPU with lots of rock images might help us train a better model.

[160]:
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_8 (Flatten)         (None, 160000)            0

 dense_37 (Dense)            (None, 100)               16000100

 dense_38 (Dense)            (None, 64)                6464
 dense_39 (Dense)            (None, 8)                 520


 dense_40 (Dense)            (None, 3)                 27

=================================================================
Total params: 16007111 (61.06 MB)
Trainable params: 16007111 (61.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Total parameters: 16007111

Trainable Parameters: 16007111

Bias params : 175

[161]:
```
#extract the last -1 model weights and predict the values
intermediate_layer_model = models.Model(inputs=model.input, outputs=model.
  ↪layers[-2].output)
intermediate_output = intermediate_layer_model.predict(X_train)
```

```
12/12 [==============================] - 2s 119ms/step
```

[162]:
```
intermediate_output
```

[162]:
```
array([[0.        , 0.03961407, 0.02701724, …, 0.01970244, 0.00707483,
         0.        ],
        [0.        , 0.03885668, 0.02765423, …, 0.01913757, 0.00715571,
         0.        ],
        [0.        , 0.03585506, 0.02581708, …, 0.01945174, 0.00665932,
         0.        ],
        …,
        [0.        , 0.03335616, 0.02172653, …, 0.02114686, 0.00556673,
         0.        ],
        [0.        , 0.0347882 , 0.02229385, …, 0.02125981, 0.00576035,
         0.        ],
        [0.        , 0.03928898, 0.02848026, …, 0.01894047, 0.00796087,
         0.        ]], dtype=float32)
```

[163]:
```
#read the given data
human_data_360 = np.loadtxt("/content/mds_360.txt")
human_data_120 = np.loadtxt("/content/mds_120.txt")
```

[164]:
```
human_data_360
```

[164]:
```
array([[-3.743, -1.204,  2.001, …, -1.992,  4.95 ,  1.695],
        [ 2.332,  1.625,  0.985, …,  0.093,  6.724,  0.708],
        [ 0.346,  1.49 , -3.795, …, -3.786,  0.706, -2.854],
        …,
        [-3.475, -3.431, -2.184, …, -2.265,  1.129, -1.201],
        [-0.051, -2.358,  1.994, …,  7.268, -0.593, -1.432],
        [ 1.134, -4.9  ,  0.983, …,  4.695,  0.624, -1.195]])
```

```
[165]: np.unique(intermediate_output)
```

```
[165]: array([0.        , 0.00116988, 0.00165255, …, 0.0438983 , 0.04653567,
              0.04708417], dtype=float32)
```

```
[176]: from scipy.spatial import procrustes
       train_mtx1, train_mtx2, train_disparity = procrustes(␣
         ↪human_data_360,intermediate_output)
       train_disparity
```

```
[176]: 0.9942420689377564
```

```
[177]: correlation_matrix = np.corrcoef(train_mtx1, train_mtx2, rowvar=False)
       # Extract the correlation coefficients
       correlation_coefficient = correlation_matrix[:train_mtx1.shape[1], train_mtx1.
         ↪shape[1]:]
       # Display the result
       print("Correlation Coefficients")
       pd.DataFrame(correlation_coefficient)
```

```
      Correlation Coefficients
```

```
[177]:           0         1         2         3         4         5         6  \
       0  0.068537  0.062101 -0.054877  0.051291 -0.058181 -0.053973  0.061802
       1  0.058438  0.061345 -0.051852  0.044545 -0.054332 -0.043723  0.061538
       2 -0.018101 -0.018176  0.073481 -0.013048  0.016338  0.010146 -0.017130
       3  0.080602  0.074389 -0.062162  0.110924 -0.091120 -0.105165  0.073801
       4 -0.031490 -0.031250  0.026808 -0.031383  0.031938  0.032002 -0.030576
       5 -0.065858 -0.056695  0.037534 -0.081657  0.072147  0.096604 -0.050219
       6  0.072839  0.077076 -0.061207  0.055350 -0.066582 -0.048507  0.079607
       7  0.006976  0.016772 -0.025153  0.013786 -0.014337 -0.004733  0.018302

                 7
       0  0.026361
       1  0.059643
       2 -0.031352
       3  0.081870
       4 -0.029323
       5 -0.021822
       6  0.081515
       7  0.063186
```

[178]:

```
intermediate_output_val = intermediate_layer_model.predict(X_val)
```

```
1/4 [======>…] - ETA: 0s
```

```
/usr/local/lib/python3.10/dist-
packages/tensorflow/python/data/ops/structured_function.py:258: UserWarning:
Even though the `tf.config.experimental_run_functions_eagerly` option is set,
this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```

```
4/4 [==============================] - 0s 45ms/step
```

[179]:

```
val_mtx1, val_mtx2, val_disparity = procrustes(human_data_120,␣
 ↪intermediate_output_val)
val_disparity
```

[179]:

```
0.9907676903243698
```

[180]:

```
correlation_matrix = np.corrcoef(val_mtx1, val_mtx2, rowvar=False)
# Extract the correlation coefficients
correlation_coefficient = correlation_matrix[:val_mtx1.shape[1], val_mtx1.
 ↪shape[1]:]
# Display the result
print("Correlation Coefficients between corresponding dimensions: t-SNE")
pd.DataFrame(correlation_coefficient)
```

```
Correlation Coefficients between corresponding dimensions: t-SNE
```

[180]:

```
          0         1         2         3         4         5         6  \
0  0.126377 -0.131379 -0.112503  0.109645 -0.105722 -0.034767  0.063237
1 -0.053632  0.071768  0.041707 -0.039592  0.042136  0.020306 -0.078402
2 -0.137876  0.125210  0.149034 -0.123189  0.147282  0.071028 -0.060089
3  0.093885 -0.083046 -0.086070  0.103300 -0.083796 -0.022000 -0.023299
4 -0.090885  0.088732  0.103312 -0.084128  0.111598  0.052943 -0.068036
5 -0.004442  0.006355  0.007405 -0.003283  0.007869  0.039888 -0.022451
6  0.012414 -0.037703 -0.009625 -0.005342 -0.015537 -0.034495  0.111549
7  0.031577 -0.030258 -0.023839  0.042086 -0.025698  0.004679 -0.020433

          7
0  0.103313
1 -0.040413
2 -0.095588
3  0.117905
4 -0.072278
5  0.001956
6 -0.013123
7  0.061983
```