Name (netid): Your Name (Your Netid)
CS 441 - HW 4: Dealing with Data

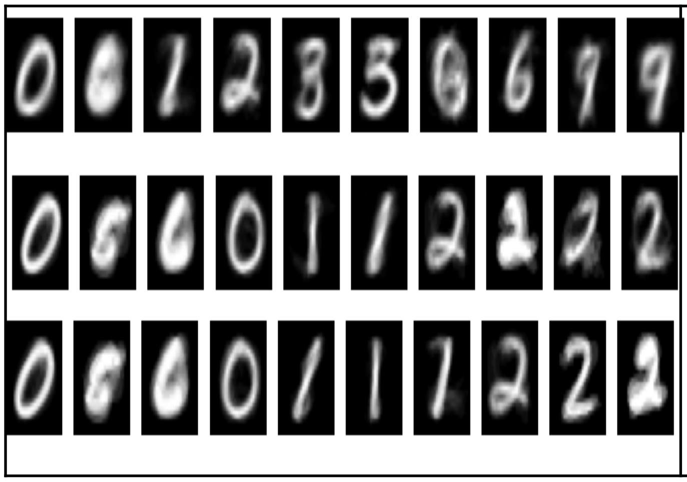
Complete the claimed points and sections below.

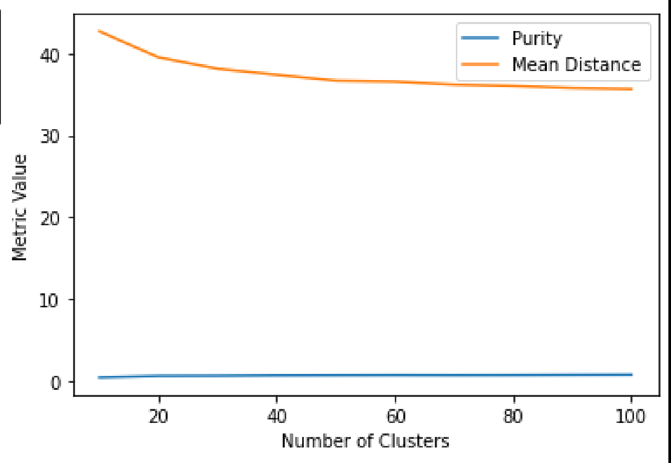
Total Points Claimed	**[ ] / 142**
1. Clustering and Fast Retrieval	
a. Test Kmeans Purity & Centroids	[ ] / 15
b. Questions	[ ] / 10
c. Fast 1-NN Retrieval	[ ] / 15
2. Estimating PDFs	
a. Histograms	[ ] / 10
b. Clustering	[ ] / 10
c. Gaussian Mixture Model	[ ] / 15
3. PCA and Data Compression	
a. Display Principal Components	[ ] / 5
b. Scatter Plot	[ ] / 5
c. Plot cumulative explained variance	[ ] / 5
d. Time & Accuracy	[ ] / 10
4. Stretch Goals	
a. Rotate Using PCA and comparison To original approach	[ ] / 15
b. Try Part 2 with your own images	[ ] / 10
c. Plot Using t-SNE and MDS	[ ] / 15
d. Completed HW3 survey by DATE	[ ] / 2

1. Clustering & Fast Retrieval

a. Test KMeans Purity
[15]

K Vs Purity Plot	K Vs Mean_Distance Plot





Paste images of centroids for K = 10, K = 20, and K=30 below (three rows).

b. Questions [10]

i. As you increase K, do you expect the purity to increase? Why or why not?

Increasing K in k-means clustering may or may not increase the purity of the clusters. It depends on the data and the structure of the clusters. If the clusters are well-separated and distinct, increasing K may improve the purity as it allows the algorithm to capture more subtle variations in the data. However, if the clusters are overlapping or poorly separated, increasing K may lead to fragmentation of the clusters and decrease the purity.

ii. In a given run, is the average distance of a sample to centroid guaranteed to monotonically decrease with each iteration (i.e. cannot increase)? Why or why not?

No, the average distance of a sample to the centroid is not guaranteed to monotonically decrease with each iteration, as k-means is sensitive to the initial centroid positions and can converge to local optima.

iii. If you do enough iterations, is Kmeans guaranteed to give you the optimal clustering that minimizes the sum of distances between each sample and its center? Why or why not?

No, Kmeans is not guaranteed to give the optimal clustering that minimizes the sum of distances between each sample and its center, even after enough iterations, because it can get stuck in local optima. The algorithm's final result depends on the initial choice of centroids and can converge to a suboptimal solution.

iv. Does improving the Kmeans objective (i.e. achieving lower mean squared error) necessarily improve expected purity ? Why or why not?

Improving the Kmeans objective does not necessarily improve the expected purity, because the objective only focuses on minimizing the sum of distances between each sample and its center, without taking into account the distribution of the samples across the clusters. Therefore, it is possible to have a low objective value, but with clusters that have a high degree of overlap and low purity.

c. Fast Retrievals

i. Brute Force: [5]

Test Error	Time to Add	Time to Search
0.0089	0.1147sZ	4.2405s

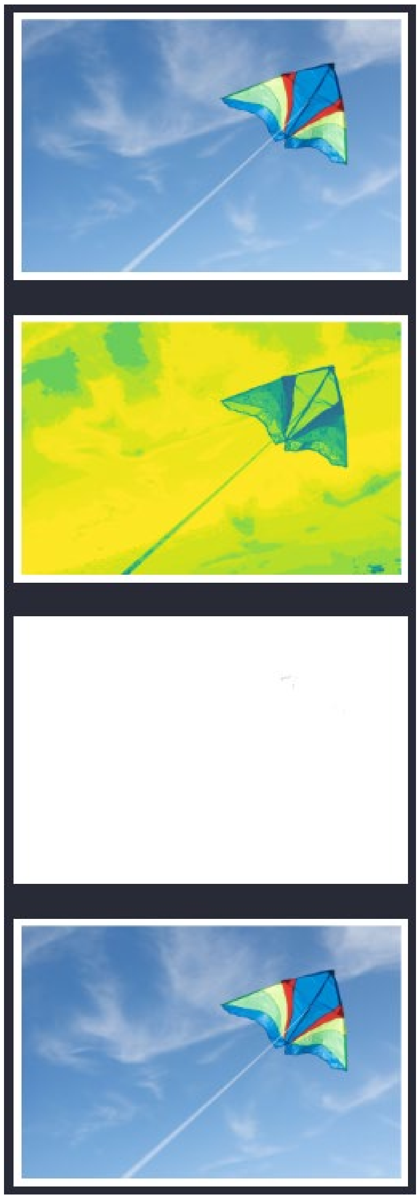
ii. LSH: [5]

Test Error	Time to Add	Time to Search	Nbits parameter
0.3943	0.1554s	0.5845s	8

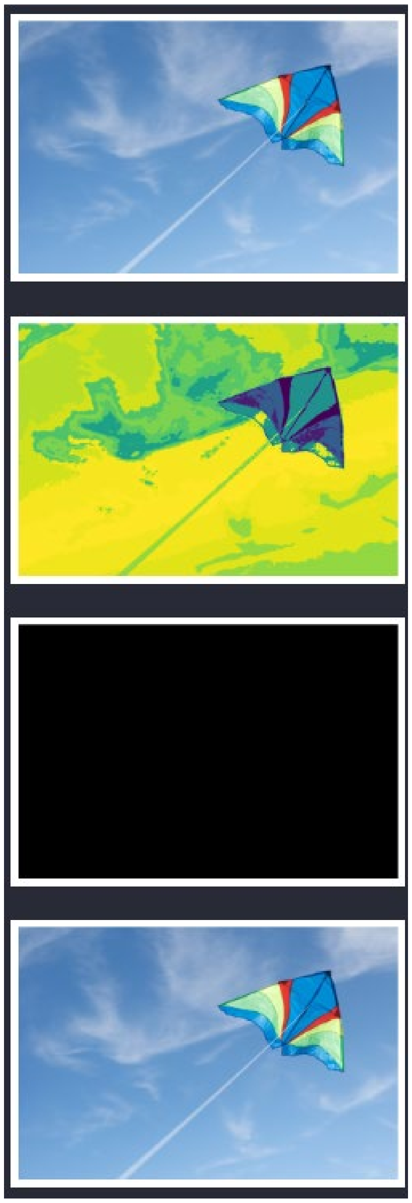
2. Estimating PDFs

Include the generated images (score map, thresholded score map, thresholded RGB) from the display code.

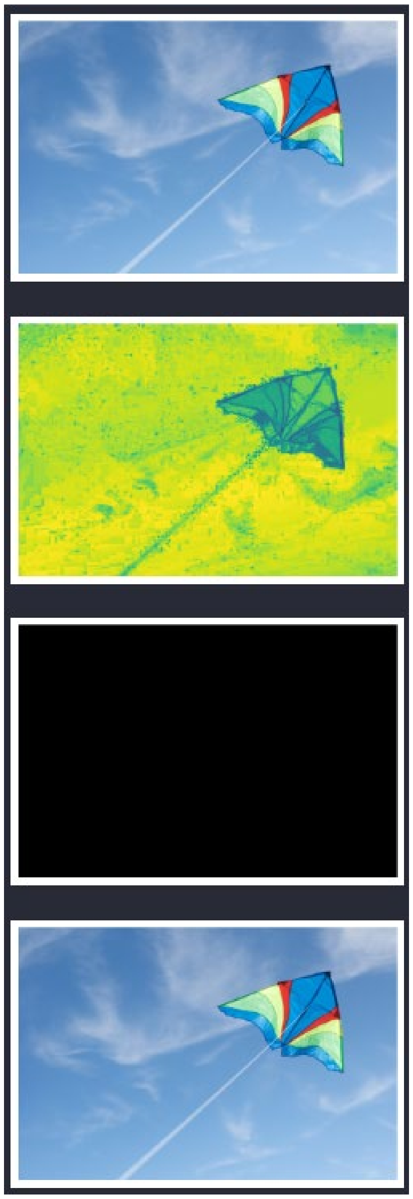
a. Histogram: [10]



b. Clustering: [10]

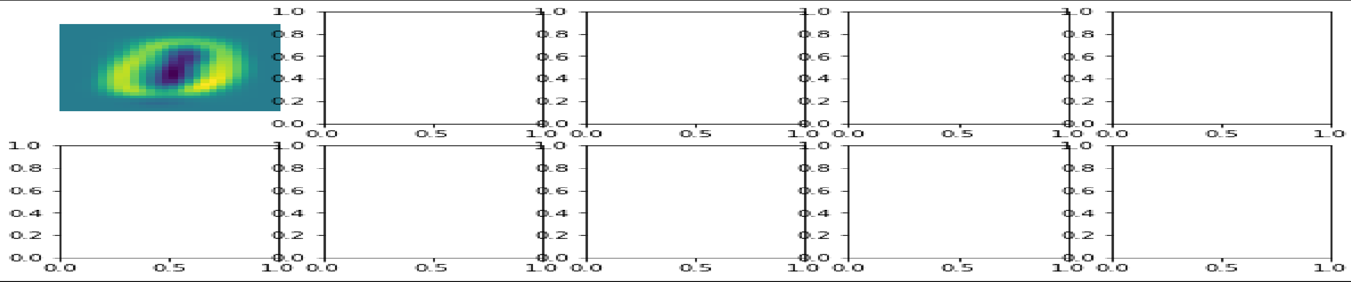


c. Gaussian mixture Model: [15]

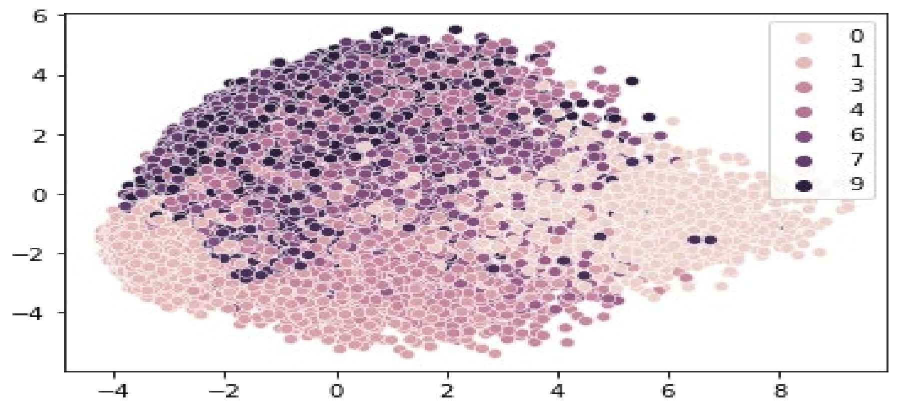


3. PCA and Data Compression

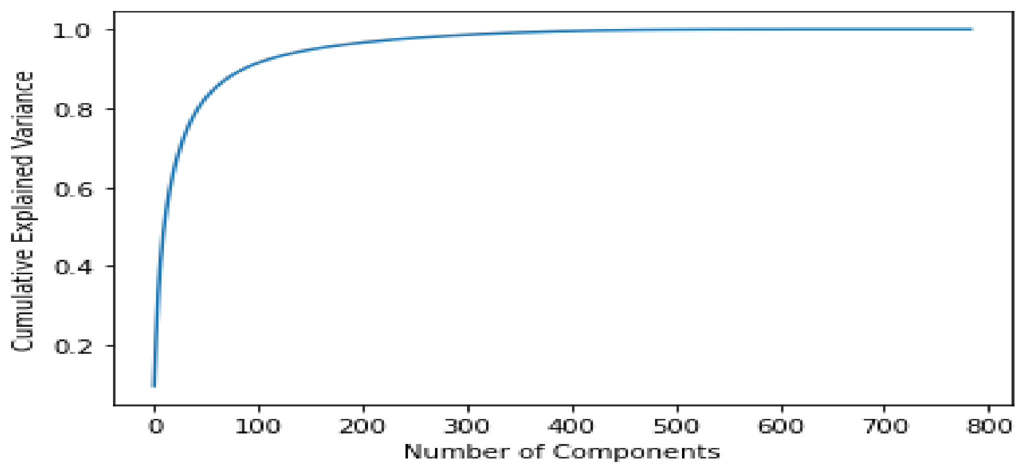
1. First 10 principal components [5]

Visualization


2. Scatterplot [5]

PLOT


3. Cumulative explained Variance [5]

PLOT


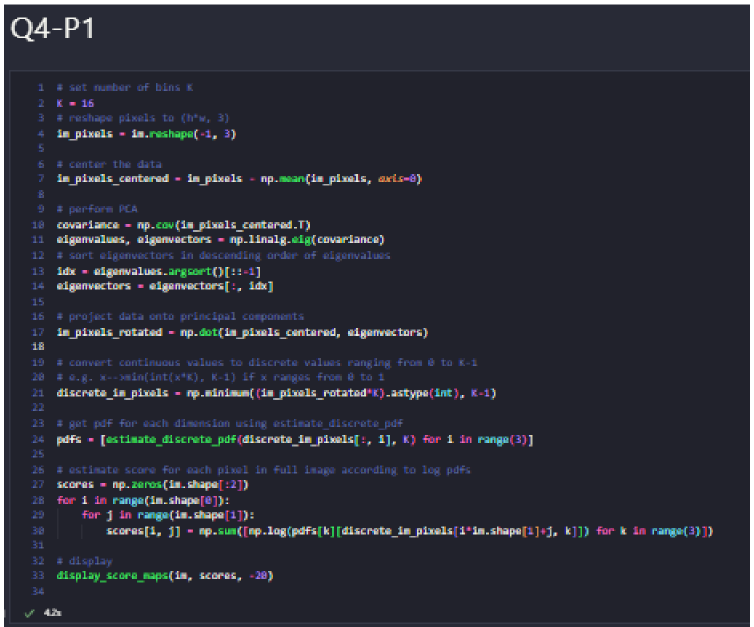
4. Faiss [10]

	Total Time	Test Error	Dimensions
Brute Force (PCA)	20.55	0.0331	784
Brute Force	0.1147	0.0089	784
LSH	0.1554	0.3943	784

Note: the last two rows are copied from 1.c for reference.

4. Stretch Goals

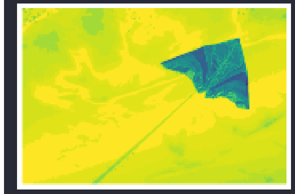
a. PDFS after using PCA to rotate your data [15]

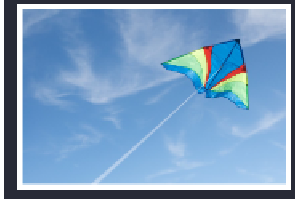


Q4-P1

```python
# set number of bins K
K = 16
# reshape pixels to (h*w, 3)
im_pixels = im.reshape(-1, 3)

# center the data
im_pixels_centered = im_pixels - np.mean(im_pixels, axis=0)

# perform PCA
covariance = np.cov(im_pixels_centered.T)
eigenvalues, eigenvectors = np.linalg.eig(covariance)
# sort eigenvectors in descending order of eigenvalues
idx = eigenvalues.argsort()[::-1]
eigenvectors = eigenvectors[:, idx]

# project data onto principal components
im_pixels_rotated = np.dot(im_pixels_centered, eigenvectors)

# convert continuous values to discrete values ranging from 0 to K-1
# e.g. x-->min(int(x*K), K-1) if x ranges from 0 to 1
discrete_im_pixels = np.minimum((im_pixels_rotated*K).astype(int), K-1)

# get pdf for each dimension using estimate_discrete_pdf
pdfs = [estimate_discrete_pdf(discrete_im_pixels[:, i], K) for i in range(3)]

# estimate score for each pixel in full image according to log pdfs
scores = np.zeros(im.shape[:2])
for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        scores[i, j] = np.sum([np.log(pdfs[k][discrete_im_pixels[i*im.shape[1]+j, k]]) for k in range(3)])

# display
display_score_maps(im, scores, -20)

```

✓ 4.2s



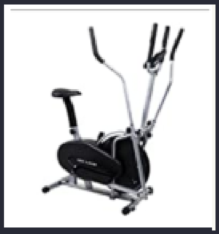


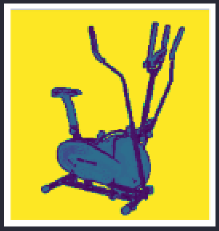
b. Apply Part 2 to your own choice of image, with the same deliverables [10]

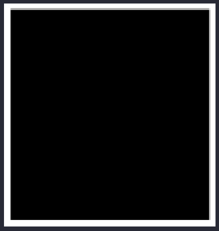
Method 1 (Per-dimension discrete pdf)

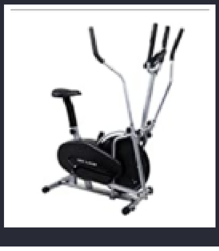
```python
# set number of bins K
K = 16
# reshape pixels to (h*w, 3)
im_pixels = im.reshape(-1, 3)
# convert continuous values to discrete values ranging from 0 to K-1
# e.g. x-->min(int(x*K), K-1) if x ranges from 0 to 1
discrete_im_pixels = np.minimum((im_pixels*K).astype(int), K-1)
# get pdf for each dimension using estimate_discrete_pdf
pdfs = [estimate_discrete_pdf(discrete_im_pixels[:, i], K) for i in range(3)]
# estimate score for each pixel in full image according to log pdfs
scores = np.zeros(im.shape[:2])
for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        scores[i, j] = np.sum([np.log(pdfs[k][discrete_im_pixels[i*im.shape[1]+j, k]]) for k in range(3)])

# display
display_score_maps(im, scores, -20)
```

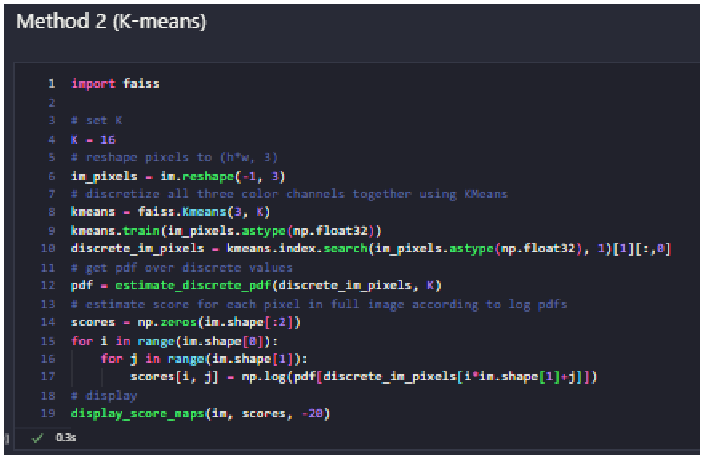
✓ 0.7s







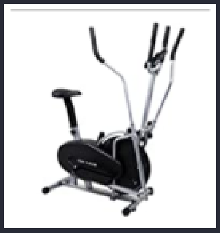


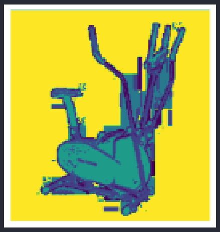


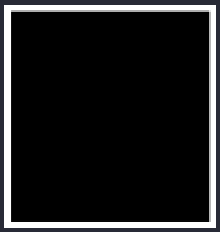
Method 2 (K-means)

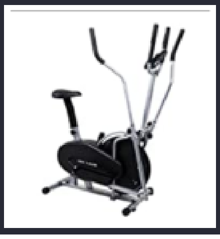
```python
import faiss

# set K
K = 16
# reshape pixels to (h*w, 3)
im_pixels = im.reshape(-1, 3)
# discretize all three color channels together using KMeans
kmeans = faiss.Kmeans(3, K)
kmeans.train(im_pixels.astype(np.float32))
discrete_im_pixels = kmeans.index.search(im_pixels.astype(np.float32), 1)[1][:,0]
# get pdf over discrete values
pdf = estimate_discrete_pdf(discrete_im_pixels, K)
# estimate score for each pixel in full image according to log pdfs
scores = np.zeros(im.shape[:2])
for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        scores[i, j] = np.log(pdf[discrete_im_pixels[i*im.shape[1]+j]])
# display
display_score_maps(im, scores, -20)
```

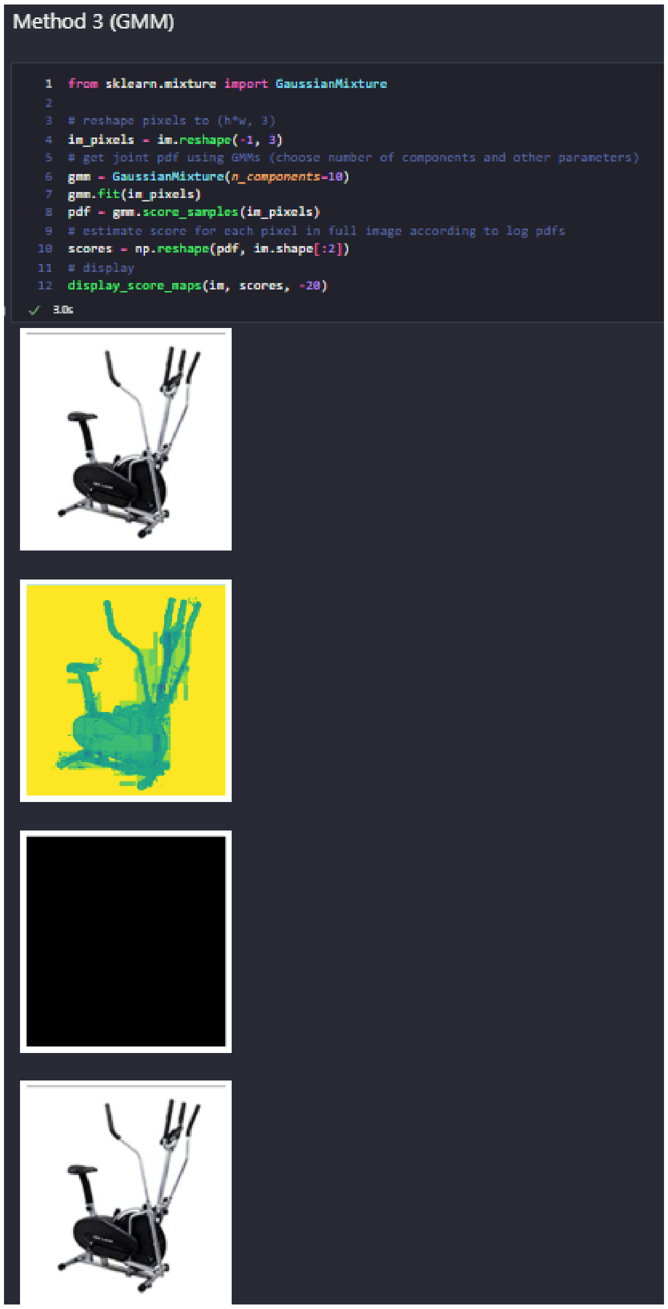
0.3s









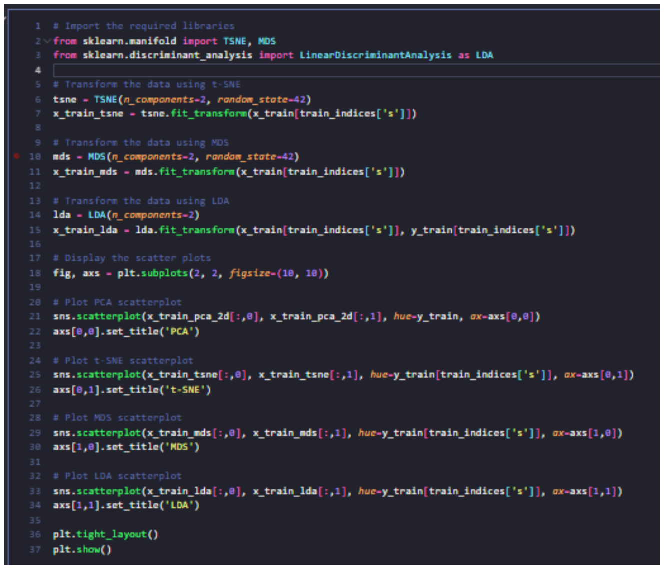


Method 3 (GMM)

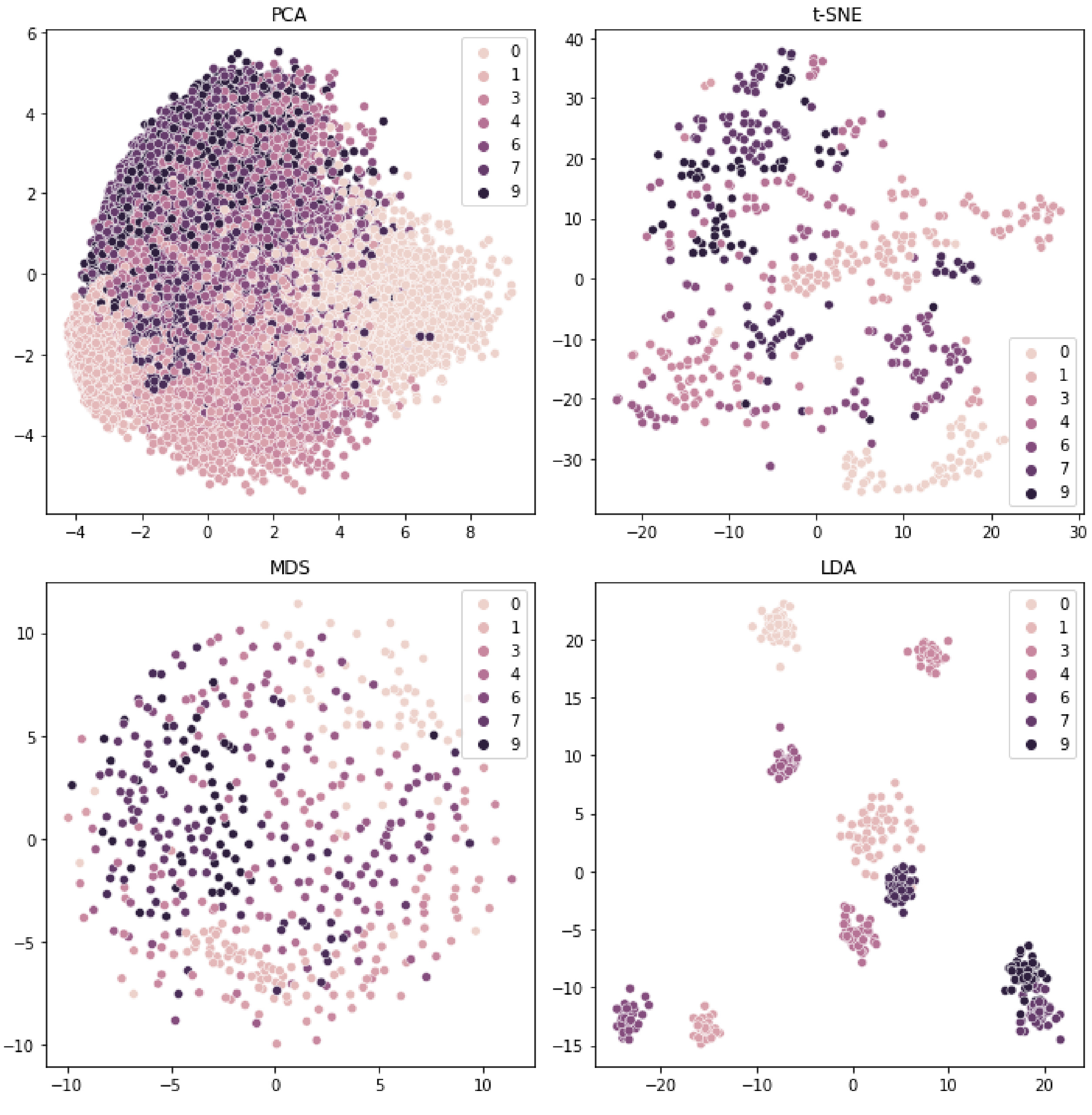
```python
from sklearn.mixture import GaussianMixture

# reshape pixels to (h*w, 3)
im_pixels = im.reshape(-1, 3)
# get joint pdf using GMMs (choose number of components and other parameters)
gmm = GaussianMixture(n_components=10)
gmm.fit(im_pixels)
pdf = gmm.score_samples(im_pixels)
# estimate score for each pixel in full image according to log pdfs
scores = np.reshape(pdf, im.shape[:2])
# display
display_score_maps(im, scores, -20)
```

✓ 3.0s

c. Scatterplots using at least two of t-SNE, MDS, and Linear Discriminant Analysis [15]



```python
# Import the required libraries
from sklearn.manifold import TSNE, MDS
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Transform the data using t-SNE
tsne = TSNE(n_components=2, random_state=42)
x_train_tsne = tsne.fit_transform(x_train[train_indices['s']])

# Transform the data using MDS
mds = MDS(n_components=2, random_state=42)
x_train_mds = mds.fit_transform(x_train[train_indices['s']])

# Transform the data using LDA
lda = LDA(n_components=2)
x_train_lda = lda.fit_transform(x_train[train_indices['s']], y_train[train_indices['s']])

# Display the scatter plots
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Plot PCA scatterplot
sns.scatterplot(x_train_pca_2d[:,0], x_train_pca_2d[:,1], hue=y_train, ax=axs[0,0])
axs[0,0].set_title('PCA')

# Plot t-SNE scatterplot
sns.scatterplot(x_train_tsne[:,0], x_train_tsne[:,1], hue=y_train[train_indices['s']], ax=axs[0,1])
axs[0,1].set_title('t-SNE')

# Plot MDS scatterplot
sns.scatterplot(x_train_mds[:,0], x_train_mds[:,1], hue=y_train[train_indices['s']], ax=axs[1,0])
axs[1,0].set_title('MDS')

# Plot LDA scatterplot
sns.scatterplot(x_train_lda[:,0], x_train_lda[:,1], hue=y_train[train_indices['s']], ax=axs[1,1])
axs[1,1].set_title('LDA')

plt.tight_layout()
plt.show()
```



Acknowledgments / Attribution

“None”.