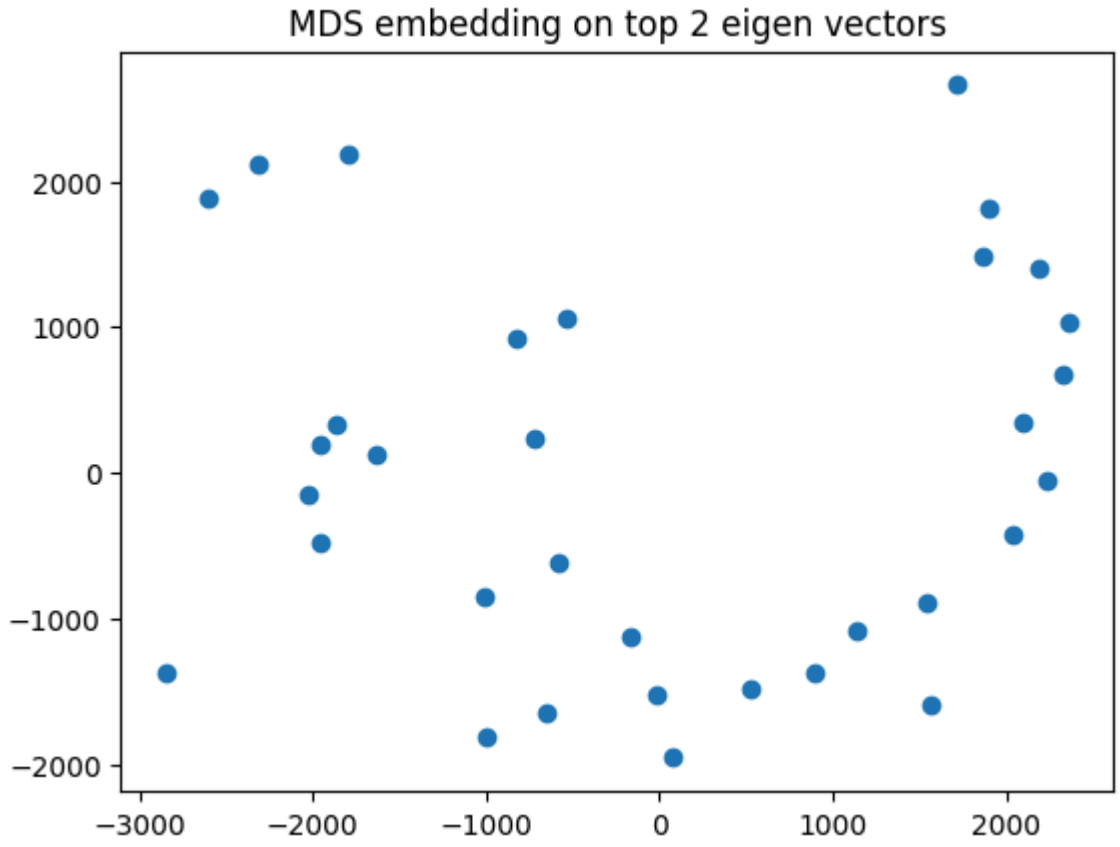
Question 1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding



# Load
data = loadmat("face.mat")
faces = data['Y'].reshape(10304, 33).T  # Reshape to (10304, 33)


# (MDS)
mds = MDS(n_components=2)
embedding = mds.fit_transform(faces)

plt.scatter(embedding[:,0], embedding[:,1])
plt.title('MDS embedding on top 2 eigen vectors')
plt.show()
```

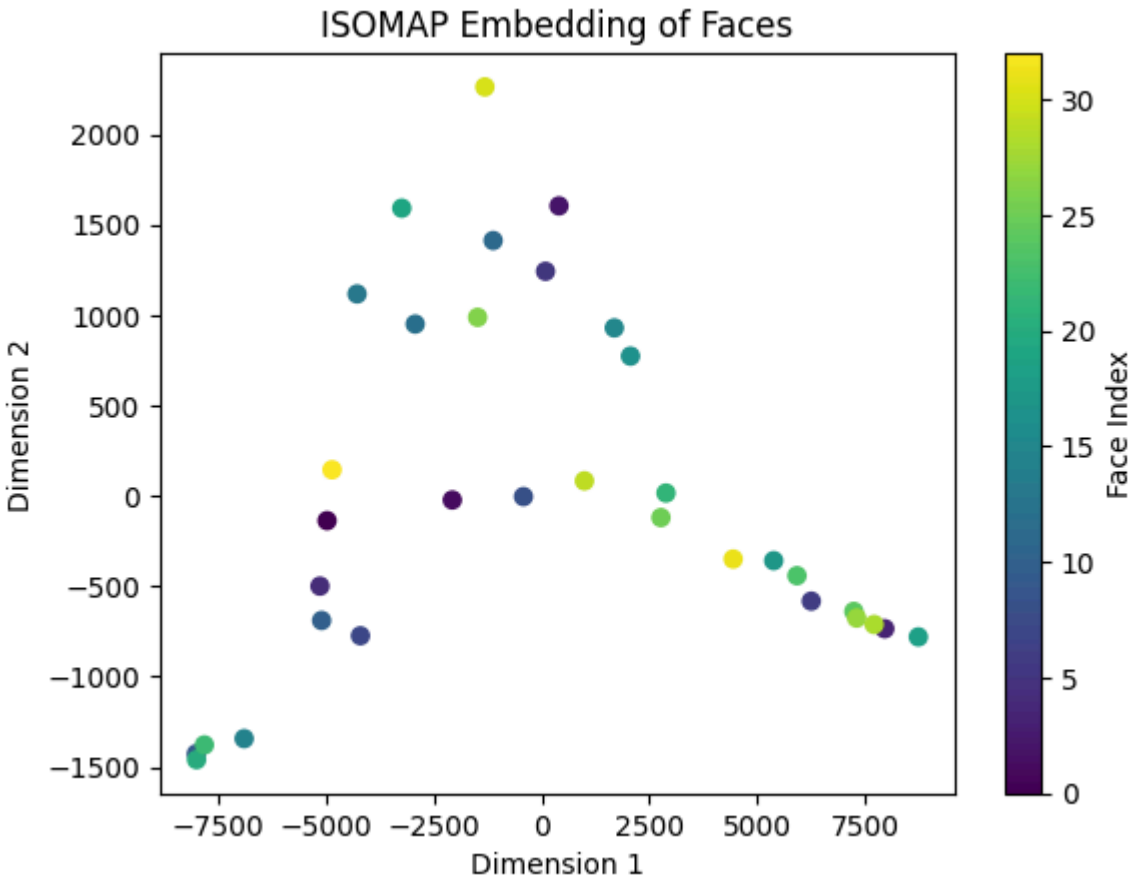
```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299: FutureWarning: Th
  warnings.warn(
```



```
# Get the first eigenvector values
eigenvector_1 = embedding[:, 0]

# Order the faces according to the first eigenvector
ordered_indices = np.argsort(eigenvector_1)

# Visualize the results
plt.figure(figsize=(10, 6))
for i, idx in enumerate(ordered_indices):
    plt.subplot(4, 9, i + 1)
    plt.imshow(faces[idx].reshape(112, 92), cmap='gray')
    plt.title(f"Face {i+1}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

MDS is based on Euclidean distances and is not able to capture the faces in the accurate direction

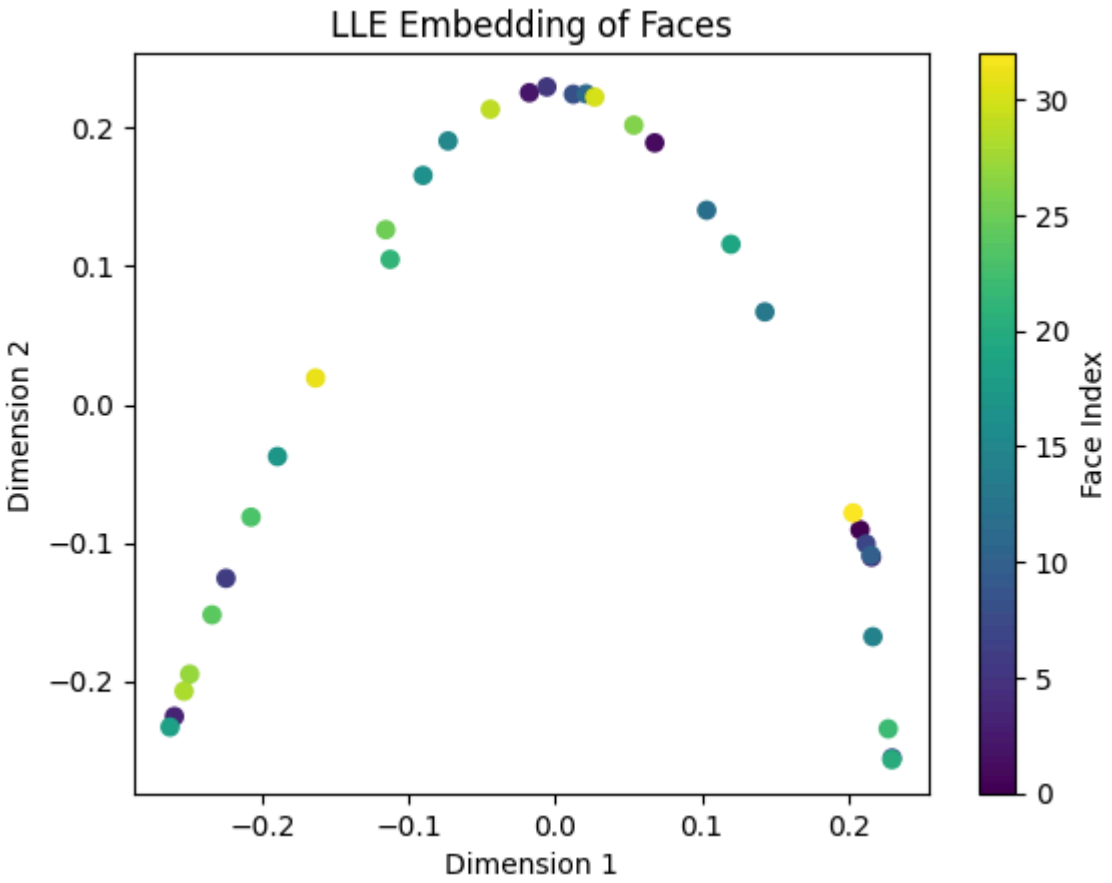
```
# isomap
n_neighbors = 5
n_components = 2
isomap = Isomap(n_neighbors=n_neighbors, n_components=n_components)
embedding_isomap = isomap.fit_transform(faces)

plt.figure()
plt.scatter(embedding_isomap[:, 0], embedding_isomap[:, 1], c=np.arange(33), cmap='viridis'
plt.colorbar(label='Face Index')
plt.title('ISOMAP Embedding of Faces')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.show()
```




```
isoindex = np.argsort(embedding_isomap[:, 0])

plt.figure(figsize=(10, 6))
for i, idx in enumerate(isoindex):
    plt.subplot(4, 9, i + 1)
    plt.imshow(faces[idx].reshape(112, 92), cmap='gray')
    plt.title(f"Face {i+1}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

Isomap preserves the order of faces as view changes from right to left. The geodeic distances used by isomap preserve the face angles compared to the euclidean distance used in MDS

```
# LLE
lle = LocallyLinearEmbedding(n_neighbors=n_neighbors, n_components=n_components)

embedding_lle = lle.fit_transform(faces)

plt.figure()
plt.scatter(embedding_lle[:, 0], embedding_lle[:, 1], c=np.arange(33), cmap='viridis')
plt.colorbar(label='Face Index')
plt.title('LLE Embedding of Faces')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.show()
```




```
lleindex = np.argsort(embedding_lle[:, 0])

plt.figure(figsize=(10, 6))
for i, idx in enumerate(lleindex):
    plt.subplot(4, 9, i + 1)
    plt.imshow(faces[idx].reshape(112, 92), cmap='gray')
    plt.title(f"Face {i+1}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

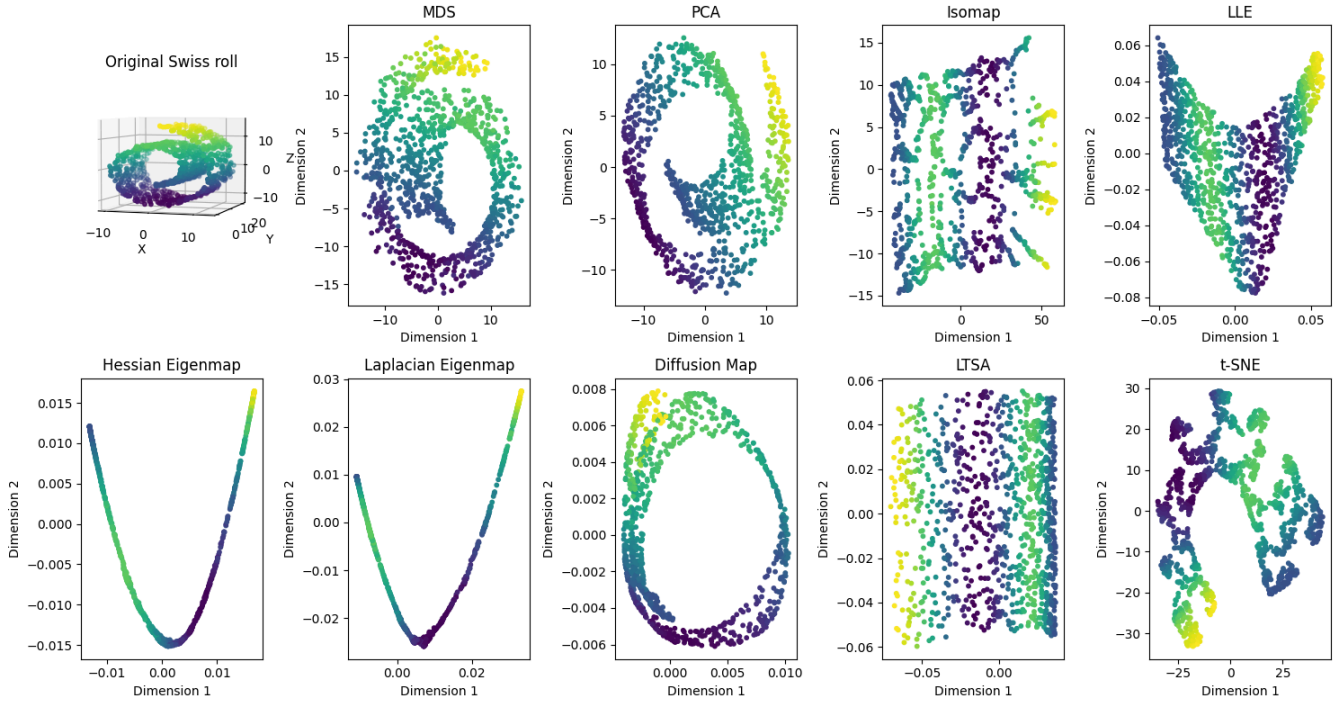
LLE preserves distances around the neighborhood, thus it orders the faces right to left, opposite to isomap. LLE and isomap are able to preserve the order of faces well while MDS is not able to capture tha angular data due to Euclidean distance

Question 2

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from sklearn.decomposition import PCA
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding, SpectralEmbedding, TSNE
from sklearn.metrics.pairwise import pairwise_distances
from mpl_toolkits.mplot3d import Axes3D

# Generate Swiss roll dataset
X, _ = make_swiss_roll(n_samples=1000, noise=0.2, random_state=42)

# Define the number of components for each method
n_components = 2

# Initialize manifold learning algorithms
methods = {
    'MDS': MDS(n_components=n_components),
    'PCA': PCA(n_components=n_components),
    'Isomap': Isomap(n_components=n_components),
    'LLE': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='standa
    'Hessian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='nearest_neig
    'Laplacian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='rbf'),
    'Diffusion Map': SpectralEmbedding(n_components=n_components, affinity='precomputed'),
    'LTSA': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='ltsa'
    't-SNE': TSNE(n_components=n_components, perplexity=30, init='pca',
                                  random_state=42)

}

# Precompute pairwise distances for Diffusion Map
pairwise_dist = pairwise_distances(X)

# Create subplots
fig, axs = plt.subplots(2,5, figsize=(15, 8))
axs = axs.ravel()
axs[0].axis('off')

ax = fig.add_subplot(2, 5, 1, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=X[:, 2], cmap=plt.cm.viridis, s=10)
ax.set_title('Original Swiss roll')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.view_init(4,-72)

# Plot the original Swiss roll dataset
#axs[0].scatter(X[:, 0], X[:, 1], c=X[:, 2], cmap=plt.cm.viridis, s=10)
#axs[0].set_title('Original Swiss Roll')
#axs[0].set_xlabel('X')
#axs[0].set_ylabel('Y')
```

```
# Loop through each method and plot the results
for i, (method_name, method) in enumerate(methods.items()):
    # Fit and transform the data
    if method_name == 'Diffusion Map':
        # Diffusion Map requires a precomputed affinity matrix
        affinity_matrix = np.exp(-pairwise_dist ** 2 / pairwise_dist.std() ** 2)
        transformed_data = method.fit_transform(affinity_matrix)
    else:
        transformed_data = method.fit_transform(X)

    # Plot the results
    axs[i+1].scatter(transformed_data[:, 0], transformed_data[:, 1], c=X[:, 2], cmap=plt.cm
    axs[i+1].set_title(method_name)
    axs[i+1].set_xlabel('Dimension 1')
    axs[i+1].set_ylabel('Dimension 2')

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299: FutureWarning: Th
  warnings.warn(
```

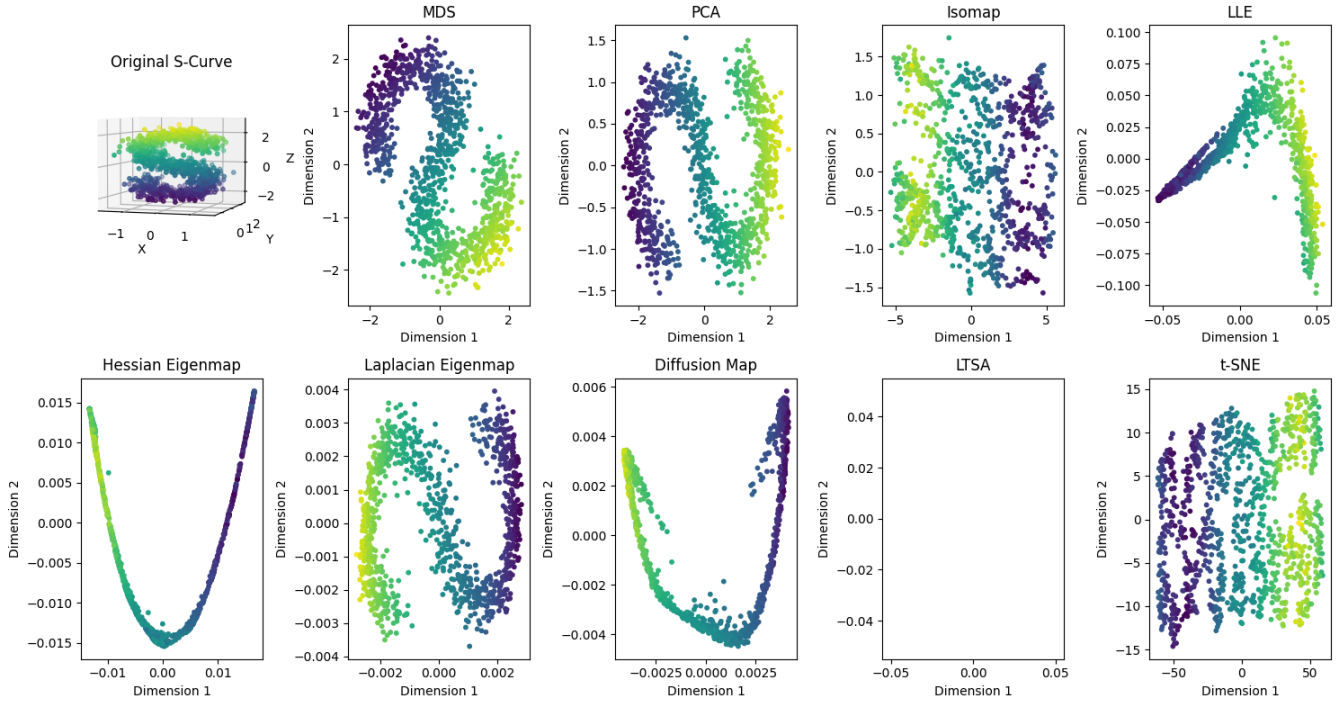


PCA and MDS cannot preserve the structure. LTSa performs well.tSNE could not represent the data effectively in low dimenions. Hessian, diffusion, isomaps are also not good.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding, SpectralEmbedding, TSNE
from sklearn.metrics.pairwise import pairwise_distances

X, _ = datasets.make_s_curve(n_samples=1000, noise=0.2, random_state=42)

# Initialize manifold learning algorithms
methods = {
    'MDS': MDS(n_components=n_components),
    'PCA': PCA(n_components=n_components),
    'Isomap': Isomap(n_components=n_components),
    'LLE': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='standa
    'Hessian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='nearest_neig
    'Laplacian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='rbf'),
    'Diffusion Map': SpectralEmbedding(n_components=n_components, affinity='precomputed'),
    'LTSA': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='ltsa'
    't-SNE': TSNE(n_components=n_components, perplexity=30, init='pca',
                                random_state=42)

}

# Precompute pairwise distances for Diffusion Map
pairwise_dist = pairwise_distances(X)

# Create subplots
fig, axs = plt.subplots(2,5, figsize=(15, 8))
axs = axs.ravel()

axs[0].axis('off')

ax = fig.add_subplot(2, 5, 1, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=X[:, 2], cmap=plt.cm.viridis, s=10)
ax.set_title('Original S-Curve')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.view_init(4,-72)

# Plot the original Swiss roll dataset
#axs[0].scatter(X[:, 0], X[:, 1], c=X[:, 2], cmap=plt.cm.viridis, s=10)
#axs[0].set_title('Original S curve')
#axs[0].set_xlabel('X')
#axs[0].set_ylabel('Y')

# Loop through each method and plot the results
for i, (method_name, method) in enumerate(methods.items()):
    # Fit and transform the data
    if method_name == 'Diffusion Map':
```

```
        # Diffusion Map requires a precomputed affinity matrix
        affinity_matrix = np.exp(-pairwise_dist ** 2 / pairwise_dist.std() ** 2)
        transformed_data = method.fit_transform(affinity_matrix)
    else:
        transformed_data = method.fit_transform(X)

    # Plot the results
    axs[i+1].scatter(transformed_data[:, 0], transformed_data[:, 1], c=X[:, 2], cmap=plt.cm
    axs[i+1].set_title(method_name)
    axs[i+1].set_xlabel('Dimension 1')
    axs[i+1].set_ylabel('Dimension 2')

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299: FutureWarning: Th
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/scipy/sparse/linalg/_eigen/arpack/arpack.py:939
  self.M_lu = lu_factor(M)
```



MDS and PCA work well. The manifold cant identify the boundaries. tSNE does could not represen effectively in low dimension

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.manifold import MDS, Isomap, LocallyLinearEmbedding, SpectralEmbedding, TSNE
from sklearn.metrics.pairwise import pairwise_distances

X, _ = datasets.make_blobs(n_samples=1000, n_features=3, centers=3, random_state=42)

# Initialize manifold learning algorithms
methods = {
    'MDS': MDS(n_components=n_components),
    'PCA': PCA(n_components=n_components),
    'Isomap': Isomap(n_components=n_components),
    'LLE': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='standa
    'Hessian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='nearest_neig
    'Laplacian Eigenmap': SpectralEmbedding(n_components=n_components, affinity='rbf'),
    'Diffusion Map': SpectralEmbedding(n_components=n_components, affinity='precomputed'),
    'LTSA': LocallyLinearEmbedding(n_neighbors=10, n_components=n_components, method='ltsa'
    't-SNE': TSNE(n_components=n_components, perplexity=30, init='pca',
                                random_state=42)

}

# Precompute pairwise distances for Diffusion Map
pairwise_dist = pairwise_distances(X)

# Create subplots
fig, axs = plt.subplots(2,5, figsize=(15, 8))
axs = axs.ravel()

axs[0].axis('off')

ax = fig.add_subplot(2, 5, 1, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=X[:, 2], cmap=plt.cm.viridis, s=10)
ax.set_title('3D clusters')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.view_init(4,-72)

# Plot the original Swiss roll dataset
#axs[0].scatter(X[:, 0], X[:, 1], c=X[:, 2], cmap=plt.cm.viridis, s=10)
#axs[0].set_title('Original S curve')
#axs[0].set_xlabel('X')
#axs[0].set_ylabel('Y')

# Loop through each method and plot the results
for i, (method_name, method) in enumerate(methods.items()):
    # Fit and transform the data
    if method_name == 'Diffusion Map':
```

```
        # Diffusion Map requires a precomputed affinity matrix
        affinity_matrix = np.exp(-pairwise_dist ** 2 / pairwise_dist.std() ** 2)
        transformed_data = method.fit_transform(affinity_matrix)
    else:
        transformed_data = method.fit_transform(X)

    # Plot the results
    axs[i+1].scatter(transformed_data[:, 0], transformed_data[:, 1], c=X[:, 2], cmap=plt.cm
    axs[i+1].set_title(method_name)
    axs[i+1].set_xlabel('Dimension 1')
    axs[i+1].set_ylabel('Dimension 2')

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

MDS and PCa work well, Euclidean distances are helpful. tSNE also identifies well. LLE, hessain Laplacian dont work seems like manifold structure is not preserved in 2D

```
/usr/local/lib/python3.10/dist-packages/scipy/sparse/_index.py:100: SparseEfficiencyWar
```

Question3

```
  self._set_intXint(row, col, x.flat[0])
```

```
import numpy as np
import scipy.io

# Load the Swiss-Roll dataset
data = scipy.io.loadmat('swiss_roll_data.mat')
X = data['X']  # 3D-data

# Define parameters
n = 100  # Number of random data points for block A
k = 2    # Number of desired eigenvectors

# Randomly select n data points
random_indices = np.random.choice(X.shape[0], n, replace=False)
block_A = X[random_indices]

# Compute the spectral decomposition of A
eigenvalues, eigenvectors = np.linalg.eigh(np.dot(block_A, block_A.T))
sorted_indices = np.argsort(eigenvalues)[::-1][:k]
U_k = eigenvectors[:, sorted_indices]

# Compute X1 and X2
X1 = np.dot(U_k, np.diag(np.sqrt(eigenvalues[sorted_indices])))
X2 = np.dot(np.dot(block_A.T, np.linalg.pinv(X1)), U_k)

# Compute the approximation ^K
A = np.dot(X1, X1.T)
B = np.dot(X1, X2.T)
C_hat = np.dot(X2, X2.T)
A_pseudo_inv = np.linalg.pinv(A)
K_hat = np.block([[A, B], [B.T, C_hat - np.dot(np.dot(B.T, A_pseudo_inv), B)]])

# Implement ISOMAP using the approximation ^K
# (You can use the Nyström method here to compute the eigenvectors of K_hat)

# Perform ISOMAP on K_hat
# (Compute eigenvalues and eigenvectors, and select top k eigenvectors)

# Embed the data into the low-dimensional space using the selected eigenvectors
# (This will give you the reduced-dimensional representation of the Swiss-Roll dataset)
```

Q.3 $K = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}$ $A \to n \times n$, $A = U \Lambda U^T$

$\Lambda = \text{diag}(\lambda_i)$ $U^T U = I$

$K = XX^T$, $X = [X_1; X_2] \in \mathbb{R}^{N \times k}$, $X_1 \in \mathbb{R}^{n \times k}$

Show that $X_1 = U_k \Lambda_k^{1/2}$ & $X_2 = B^T U_k \Lambda_k^{-1/2}$

$$K = XX^T = [X_1; X_2] \begin{bmatrix} X_1^T \\ X_2^T \end{bmatrix} = \begin{bmatrix} X_1 X_1^T & X_1 X_2^T \\ X_2 X_1^T & X_2 X_2^T \end{bmatrix}$$

$A = X_1 X_1^T$, $B = X_1 X_2^T$

Using EVD of A, $X_1 = U_k \Lambda_k^{1/2}$

$B = X_1 X_2^T \Rightarrow X_2^T = X_1^{-1} B$

$X_2 = B^T (X_1^{-1})^T$

$X_2 = B^T (\Lambda_k^{-1/2} U_k^{-1})^T$

$X_2 = B^T (\Lambda_k^{-1/2} U_k^T)$

$X_2 = B^T U_k \Lambda_k^{-1/2}$

b) $\hat{K} = \begin{bmatrix} x_1x_1^T & x_1x_2^T \\ x_2x_1^T & x_2x_2^T \end{bmatrix} \geq 0$

Let $\hat{K} = \begin{bmatrix} A & B \\ B^T & \hat{C} \end{bmatrix}$ $\quad A = x_1x_1^T \Rightarrow A \geq 0$

$\hat{C} - B^T A^{-1} B \geq 0$

$\|\hat{K} - K\|^2 = \|\hat{C} - B^T A^{\dagger} B\|^2$

If A is not invertible $\Rightarrow \hat{C} = x_2x_2^T = B^T A^{\dagger} B$

d) $\det(K) = \det(A) \cdot \det(K/A)$

Properties of determinant $\det(CD) = \det(C) \cdot \det(D)$

$\det(A) \neq 0$ Assuming A is invertible

$\det(A) \neq 0$

$A = x_1x_1^T \geq 0$

$\det(K/A) = \det(K) \times \det(A^{-1}) = \dfrac{\det(K)}{\det(A)}$

$\det(K) = \det(A) \times \dfrac{\det(K)}{\det(A)} = \det(K)$ Proved