**IS-733 LAB 04-21-2025 – UB01976**

**1.** Use **PCA** to reduce the dimensionality of a 2D dataset and visualize how much variance is captured.

1. **Plot the original data** to observe its 2D distribution.

*import numpy as np*

*import matplotlib.pyplot as plt*

*# Step 1: Generate a 2D dataset*

*np.random.seed(42)*

*X = np.random.multivariate\_normal(mean=[2, 5], cov=[[3, 2], [2, 2]], size=100)*

*# Step 2: Plot the original data*

*plt.scatter(X[:, 0], X[:, 1], alpha=0.7)*

*plt.title("Original 2D Data")*

*plt.xlabel("Feature 1")*

*plt.ylabel("Feature 2")*

*plt.axis("equal")*

*plt.grid(True)*

*plt.show()*

1. **Center the data** (subtract mean from each feature).
2. **Compute the covariance matrix**.
3. **Calculate the eigenvalues and eigenvectors**.
4. **Project the data onto the first principal component**.
5. **Plot the 1D projection** and compare it with the original.
6. **Plot the amount of variance explained** by each component.

**1 Ans: -** As the problem statement clearly mentions to use PCA to reduce the dimensionality of randomly generated 2D dataset and visualize the variance, we have the first 2 steps provided i.e. generating the 2D plot and plotting the plot, the remaining steps would be as follows;

2. Center the data (subtract mean from each feature)

**Python Code: -** *X\_centered = X - np.mean(X, axis=0)*

3. Compute the covariance matrix.

**Python Code: -** *cov\_matrix = np.cov(X\_centered, rowvar=False)*

4. Calculate the Eigenvalues and Eigenvectors

**Python Code: -** *eigenvalues, eigenvectors = np.linalg.eigh(cov\_matrix)*

5. Project the data onto the first principal component

**Python Code: -** *pc1 = eigenvectors[:, 0]*

*projected\_1d = X\_centered @ pc1*

*X\_projected\_2d = np.outer(projected\_1d, pc1) + np.mean(X, axis=0)*

6. Plot the 1D Projection and compare it with original

**Python Code: -** *plt.figure(figsize=(8, 6))*

*plt.scatter(X[:, 0], X[:, 1], label='Original Data', alpha=0.5)*

*plt.scatter(X\_projected\_2d[:, 0], X\_projected\_2d[:, 1], label='1D Projection', color='green', alpha=0.8)*

*plt.plot( [np.mean(X, axis=0)[0], np.mean(X, axis=0)[0] + pc1[0] \* 5], [np.mean(X, axis=0)[1], np.mean(X, axis=0)[1] + pc1[1] \* 5], color='red', label='PC1 (Direction)', linewidth=2 )*

*plt.legend()*

*plt.title("1D Projection on First Principal Component")*

*plt.xlabel("Feature 1")*

*plt.ylabel("Feature 2")*

*plt.axis("equal")*

*plt.grid(True)*

*plt.tight\_layout()*

*plt.show()*

7. Plot the amount of variance explained by each component

**Python Code: -** *explained\_variance\_ratio = eigenvalues / np.sum(eigenvalues)*

*plt.figure(figsize=(6, 4))*

*plt.bar([1, 2], explained\_variance\_ratio \* 100, color='skyblue')*

*plt.xticks([1, 2])*

*plt.ylabel("Variance Explained (%)")*

*plt.xlabel("Principal Component")*

*plt.title("Variance Explained by Each Component")*

*plt.grid(True)*

*plt.tight\_layout()*

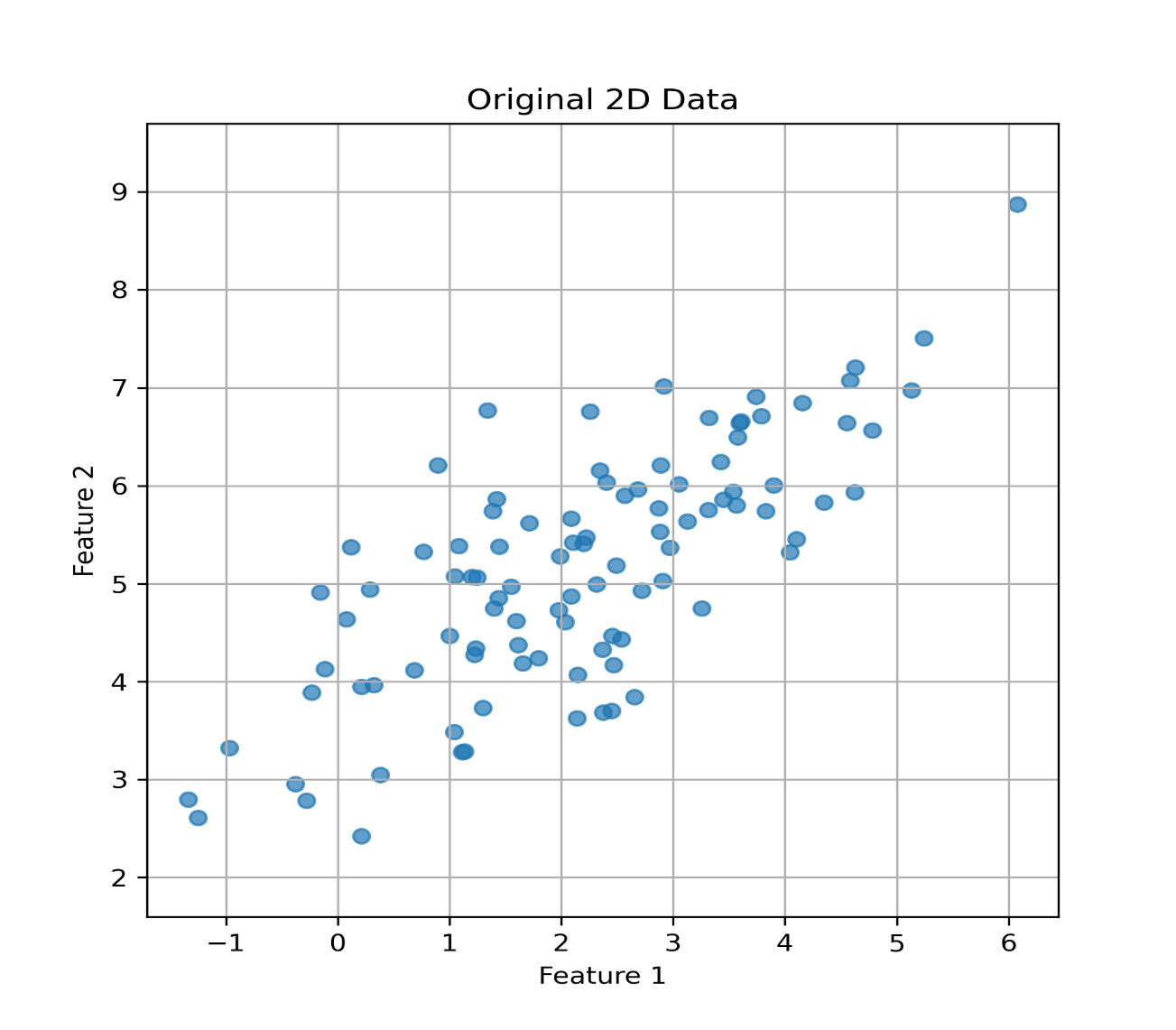
*plt.show()*

The complete code combining all the above steps from **Step1 to Step7** can be found at the following GitHub link provided, please run the code in the below link to get the plots;

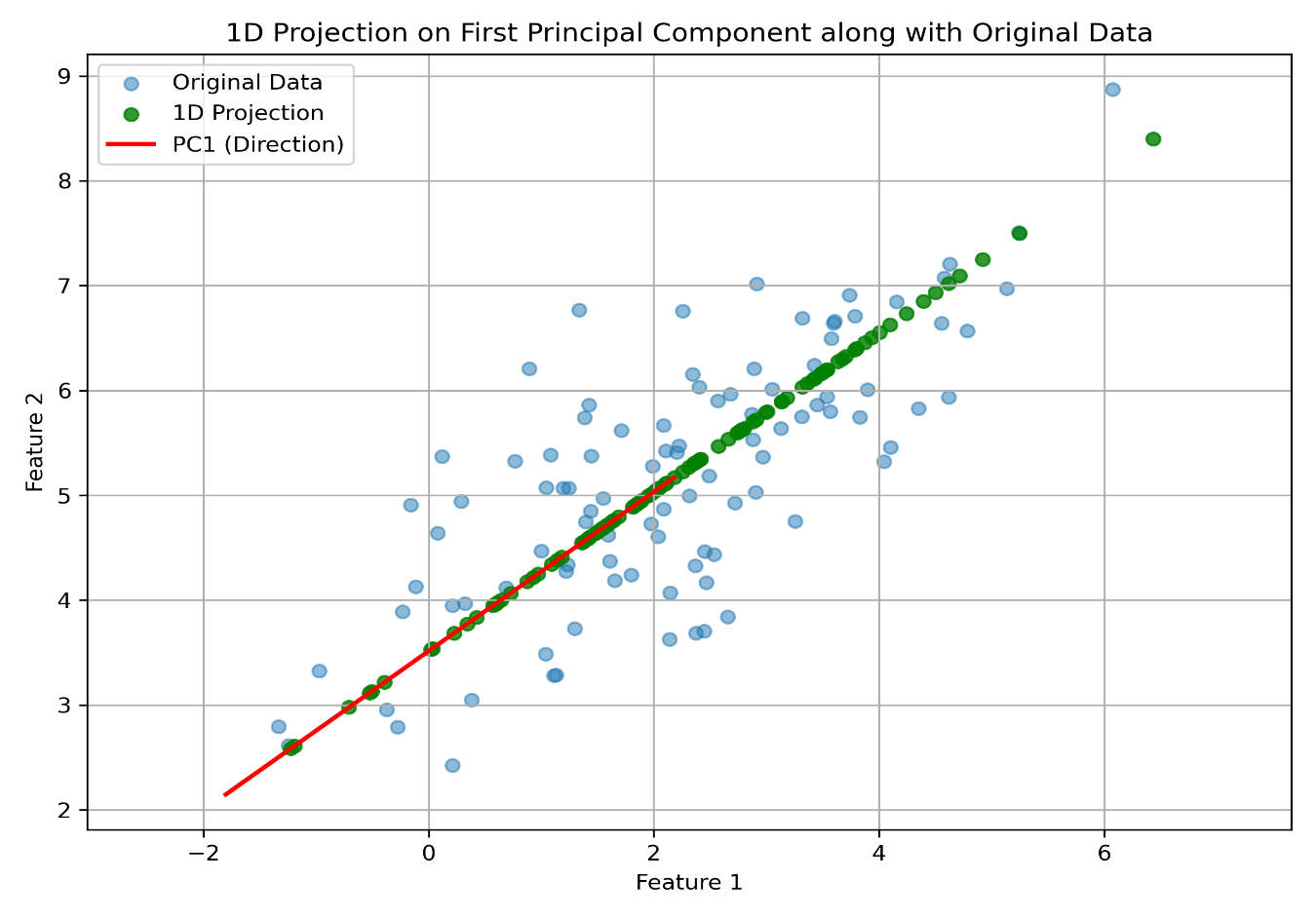
<https://github.com/UB01976/is7332025/blob/main/data-mining-project-repo/04212025_CW/Lab%2004-21-2025%20-%20UB01976.ipynb>

The plots for the above dataset which includes the original data with features, 1D projection on First principal component, and variance plot can be shown as below;

**Original Data Plot: -**



**The 1D Projection on first principal component (PC1) along with original data: -**

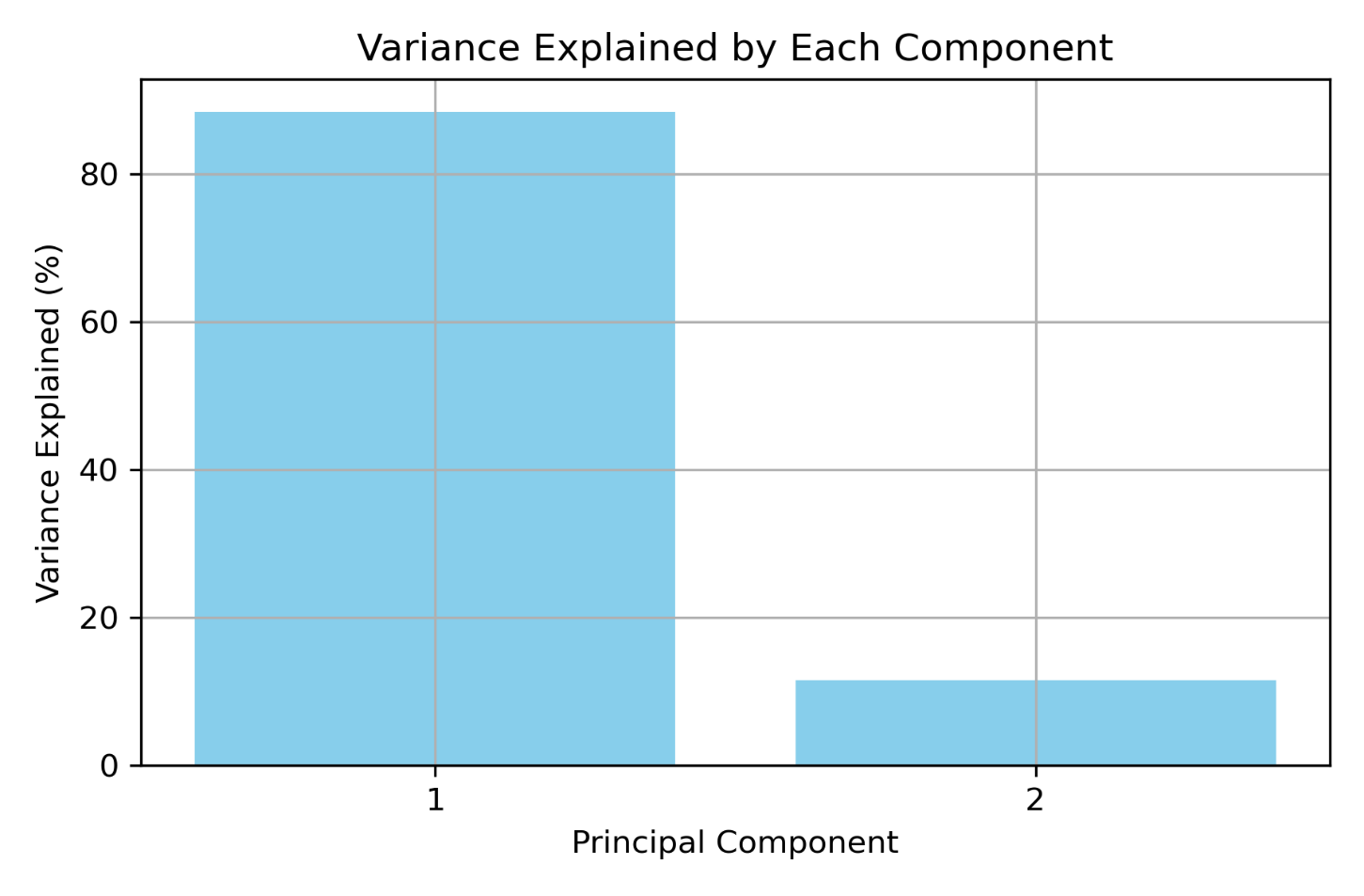


**The Eigenvalues and Eigenvectors are: -**

Eigenvalues are:  
 λ1 = 3.3454  
 λ2 = 0.4369  
  
Eigenvectors (columns corresponding to λ1, λ2):  
[[-0.79640776 0.60476002]  
 [-0.60476002 -0.79640776]]

**Variance Explained by each component plot: -**

***(Plot continued the next page...)***



**2.** Use **t-SNE** to visualize how it separates points from different classes in 2D space.

1. **Inspect the shape of X and y**. What do the 64 features represent?
2. **Apply TSNE from sklearn.manifold** with n\_components=2.
3. **Visualize the 2D t-SNE embedding** using matplotlib.pyplot.scatter, color-coded by digit labels.
4. Try different **perplexity values** (e.g., 5, 30, 50) and compare how clusters change.

from sklearn.datasets import load\_digits

digits = load\_digits()

X = digits.data # shape (1797, 64)

y = digits.target # labels: 0–9

**2 Ans: -** We will be using the t-SNE to visualize how it separates the data points from different classes in a 2D space. As we are already importing the digits from sklearn\_datasets library, we can move with the next of inspecting the shape of X and Y, i.e. X being the digits data with1797 samples and 64 features for each sample. Y would be labels to these 1797 samples in X with digit labels varying from class values of (0-9).

1. The **Python Code to Inspect the Shape of X and Y** would be as follows: -

*digits = load\_digits()*

*X = digits.data # Shape: (1797, 64)*

*y = digits.target # Labels: 0–9*

*print("Shape of X:", X.shape)*

*print("Shape of y:", y.shape)*

*print("Each sample has", X.shape[1], "features (representing 8x8 images).")*

The **64 features or attributes** represent the pixel intensities of the image, arranged in the one-dimensional vector (1D). Each sample in the data is an 8X8 pixel image of the digit. That means, each row in X (digits data) is an individual digit image, **which is represented by 64 values of brightness intensity or pixels** (which range between 0 to 16).

2. **Apply TSNE from sklearn.manifold** with n\_components=2.

Python Code: -

*for i, perp in enumerate(perplexities): tsne = TSNE(n\_components=2, perplexity=perp, random\_state=42) X\_embedded = tsne.fit\_transform(X)*

*plt.subplot(1, 3, i + 1)*  
*scatter = plt.scatter(X\_embedded[:, 0], X\_embedded[:, 1], c=y, cmap='tab10', s=10)*  
*plt.title(f't-SNE with Perplexity = {perp}')*  
*plt.xlabel("Component 1")*  
*plt.ylabel("Component 2")*  
*plt.grid(True)*

3. **Visualize the 2D t-SNE embedding** using matplotlib.pyplot.scatter, color-coded by digit labels.

Python Code: -

*plt.tight\_layout()*

*plt.suptitle("t-SNE Visualization with Different Perplexity Values", fontsize=16, y=1.05)*

*plt.colorbar(scatter, ax=plt.gcf().get\_axes(), label='Digit Label')*

*plt.show()*

4. Try different **perplexity values** (e.g., 5, 30, 50) and compare how clusters change.

Python Code: -

*perplexities = [5, 30, 50]*

*plt.figure(figsize=(18, 5))*

The complete code on using the t-SNE to visualize how it separates the data points from different classes in 2D space is found in the below GitHub link, please run the below code to get the output plot;

<https://github.com/UB01976/is7332025/blob/main/data-mining-project-repo/04212025_CW/Lab%2004-21-2025%20-%20UB01976.ipynb>

The plot showing t-SNE visualization on how it separates the data points in 2D is as shown below;

Shape of X: (1797, 64)  
Shape of y: (1797,)  
Each sample has 64 features (representing 8x8 images).

