<u>Assignment 1</u>

Name: Vaishnavi Gosavi **Roll No:** 71
Class: TY CSAI-A **Batch:** 2

Title: Perform PCA in dimension reduction of numerical data
1. Pre-process the data through standardization.
2. Perform PCA to reduce dimension.
3. Construct the scree plot.

Data visualization in lower dimensional representation.

Description:-

- **Principal Component Analysis (PCA)**
  Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of large datasets while retaining most of their original information. It transforms correlated variables into a smaller set of uncorrelated variables called principal components. PCA is widely used in machine learning and data preprocessing to address challenges such as multicollinearity, overfitting, and the "curse of dimensionality".

- **Key Features of PCA**
  - *Computational Efficiency*: Reducing the number of features leads to lower computational costs.
  - *Avoiding Overfitting:* By removing redundant features, PCA helps in preventing overfitting in models.
  - *Better Visualization:* High-dimensional data is difficult to visualize; PCA allows representation in 2D or 3D.
  - *Noise Reduction:* PCA captures the most important patterns in data while eliminating random noise.

- **Steps in PCA Implementation**

1. Data Pre-processing through Standardization

Since PCA is sensitive to the scale of data, standardization is a crucial step before applying PCA. This ensures that each feature contributes equally to the analysis.
The standardization formula is given by:

$$X' = \frac{X - \mu}{\sigma}$$

where,
- X is the original feature value,
- μ is the mean of the feature,
- σ is the standard deviation of the feature.

2. Compute the Covariance Matrix

To understand the relationships between different features, we compute the covariance matrix:

$$C = \frac{1}{n-1} X^T X$$

where,

- X is the standardized data matrix
- n is the number of observations

3. Compute Eigenvalues and Eigenvectors

The principal components are determined by computing the eigenvalues and eigenvectors of the covariance matrix:

$$Cv = \lambda v$$

where:

- λ represents the eigenvalues,
- v represents the eigenvectors (principal components).

4. Select Top k Principal Components

The principal components are sorted in descending order of their eigenvalues. The top k components that explain most of the variance are selected.

The proportion of variance explained (PVE) by each principal component is given by:

$$PVE = \frac{\lambda_i}{\sum \lambda}$$

where,

- λ_i is the eigenvalue of the i^{th} principal component.

Cumulative variance is often used to determine the number of components to retain:

$$CVE = \sum_{i=1}^{k} PVE_i$$

5. Transform Data to the New Lower Dimensional Space

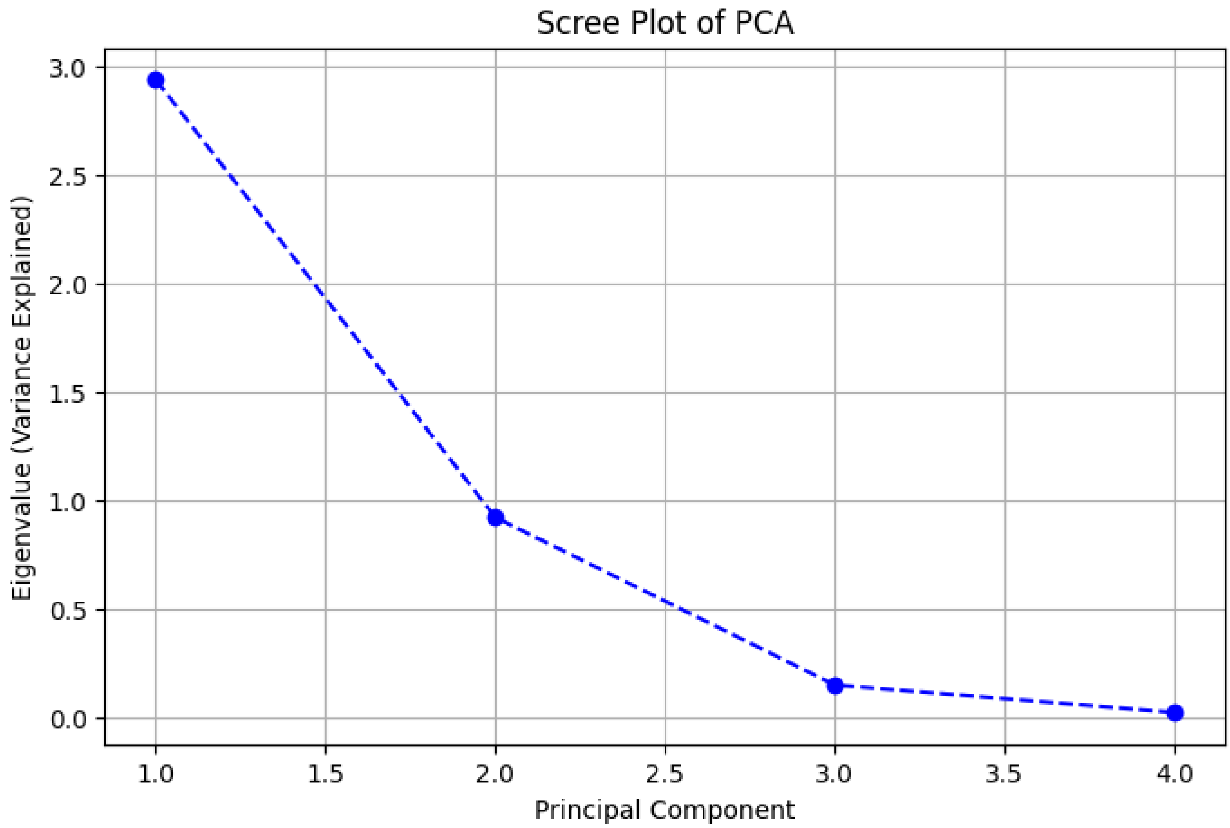
The reduced representation of the data is obtained by projecting it onto the selected principal components:

$$Z = XV_k$$

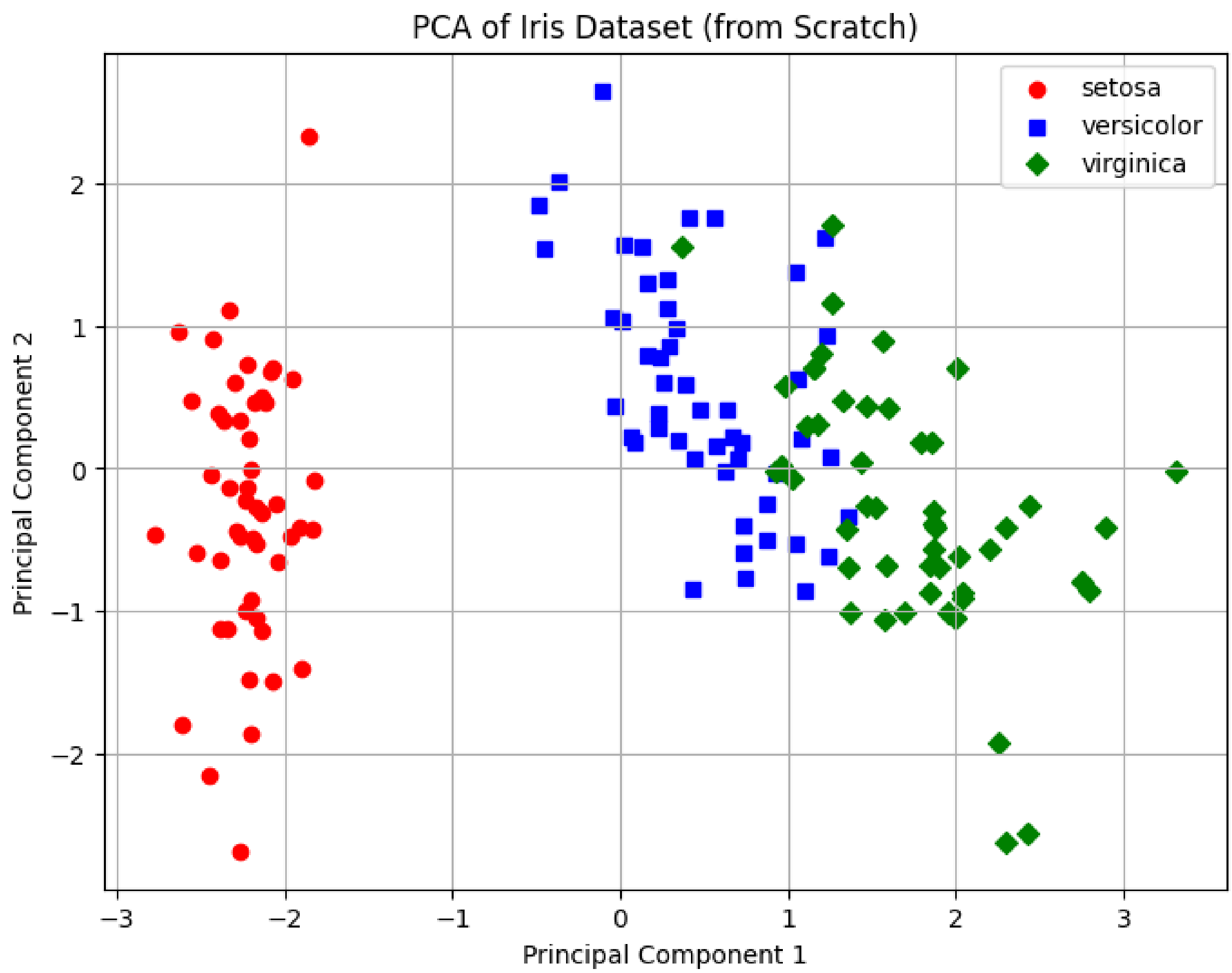
where:

- Z is the transformed data,
- V_k contains the top k eigenvectors.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = iris.data
y = iris.target

# Step 1: Standardization (Mean centering and scaling)
def standardize_data(X):
    mean = np.mean(X, axis=0)
    std_dev = np.std(X, axis=0)
    X_standardized = (X - mean) / std_dev
    return X_standardized

X_standardized = standardize_data(X)

# Step 2: Compute Covariance Matrix
def compute_covariance_matrix(X):
    return np.cov(X.T)  # Covariance matrix (features along columns)

cov_matrix = compute_covariance_matrix(X_standardized)

# Step 3: Compute Eigenvalues and Eigenvectors
def compute_eigenvalues_and_vectors(cov_matrix):
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)  # Eigen
decomposition
    return eigenvalues, eigenvectors

eigenvalues, eigenvectors =
compute_eigenvalues_and_vectors(cov_matrix)

# Step 4: Sort Eigenvalues and Corresponding Eigenvectors
sorted_indices = np.argsort(eigenvalues)[::-1]  # Sort in descending
order
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

# Step 5: Scree Plot (to decide how many components to keep)
plt.figure(figsize=(8, 5))
plt.plot(range(1, len(eigenvalues) + 1), eigenvalues, marker='o',
linestyle='--', color='b')
plt.xlabel("Principal Component")
plt.ylabel("Eigenvalue (Variance Explained)")
plt.title("Scree Plot of PCA")
plt.grid()
plt.show()
```




```
# Step 6: Select Top k Principal Components (Reduce Dimensions)
k = 2  # Reduce to 2 dimensions for visualization
top_eigenvectors = eigenvectors[:, :k]  # Take first k eigenvectors

# Step 7: Transform the Original Data
X_pca = X_standardized.dot(top_eigenvectors)

# Step 8: Data Visualization in 2D
plt.figure(figsize=(8, 6))
for label, marker, color in zip(range(3), ('o', 's', 'D'), ('red', 
'blue', 'green')):
    plt.scatter(X_pca[y == label, 0], X_pca[y == label, 1], 
marker=marker, color=color, label=iris.target_names[label])

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Iris Dataset (from Scratch)")
plt.legend()
plt.grid()
plt.show()
```



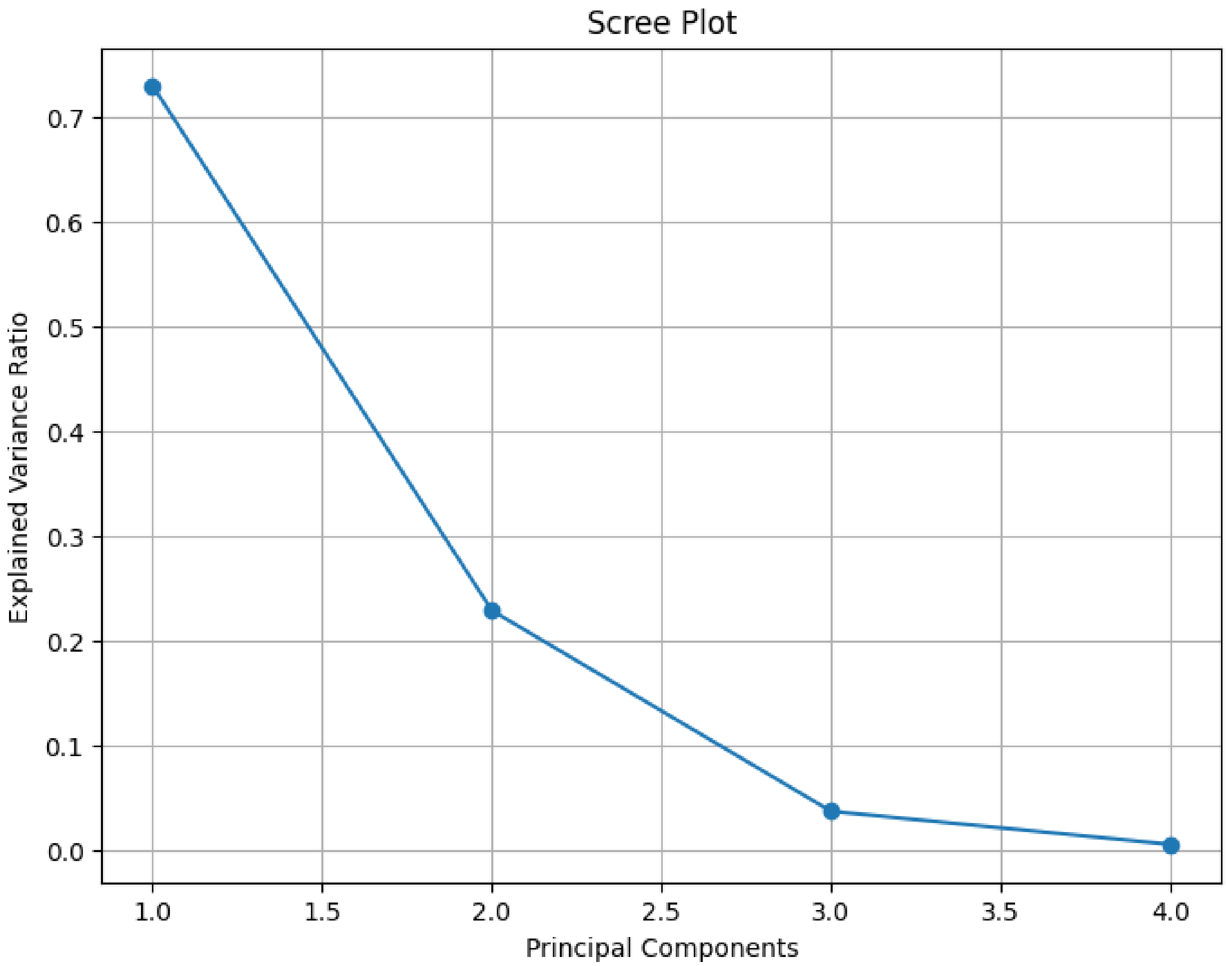

```
# Step 9: Train Logistic Regression on PCA-Reduced Data
X_train, X_test, y_train, y_test = train_test_split(X_pca, y,
test_size=0.2, random_state=42)

log_reg = LogisticRegression(multi_class="ovr", solver="lbfgs")
log_reg.fit(X_train, y_train)

# Step 10: Evaluate the Model
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Logistic Regression Accuracy after PCA: {accuracy * 100:.2f}
%")
```

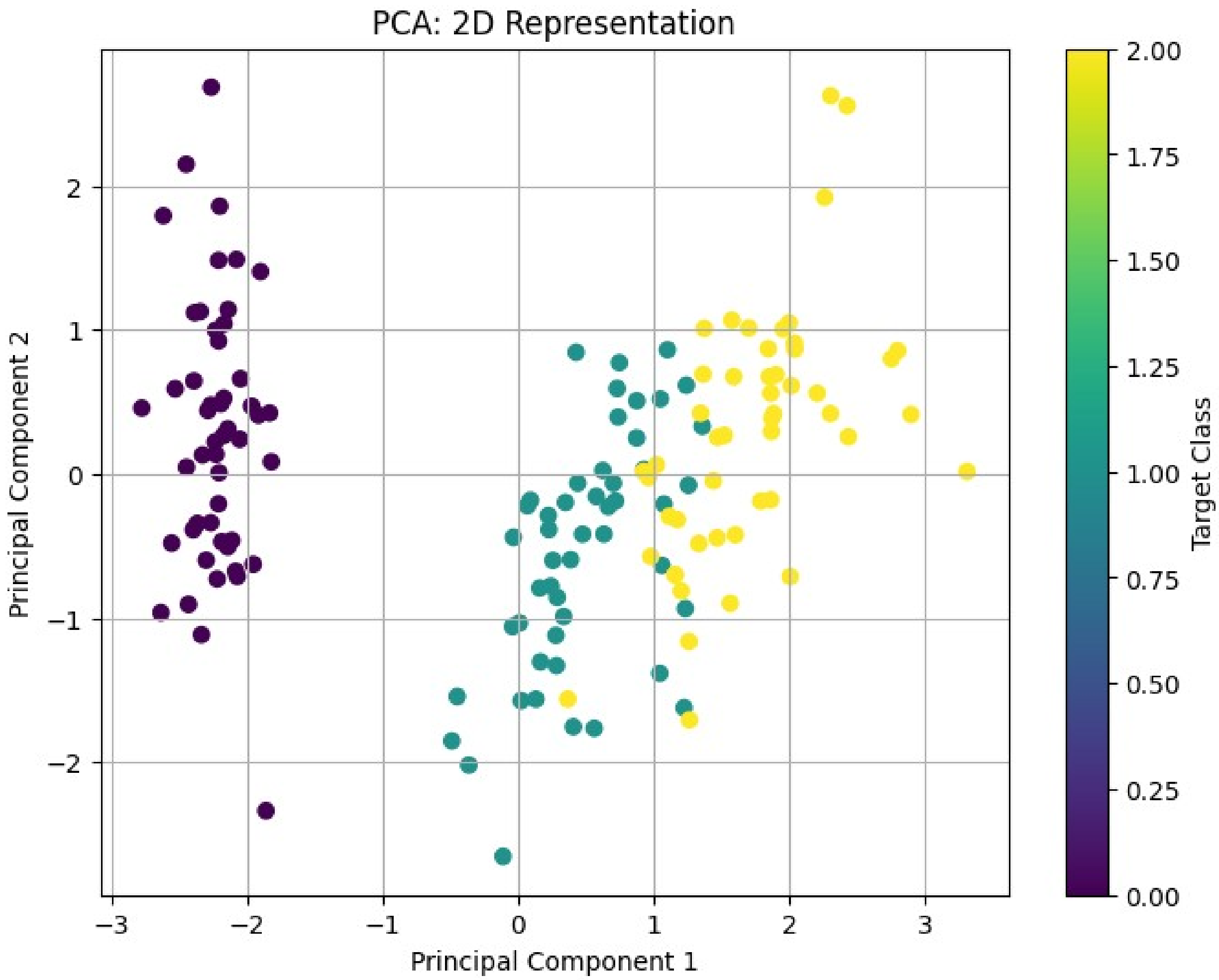
```
Logistic Regression Accuracy after PCA: 93.33%
```

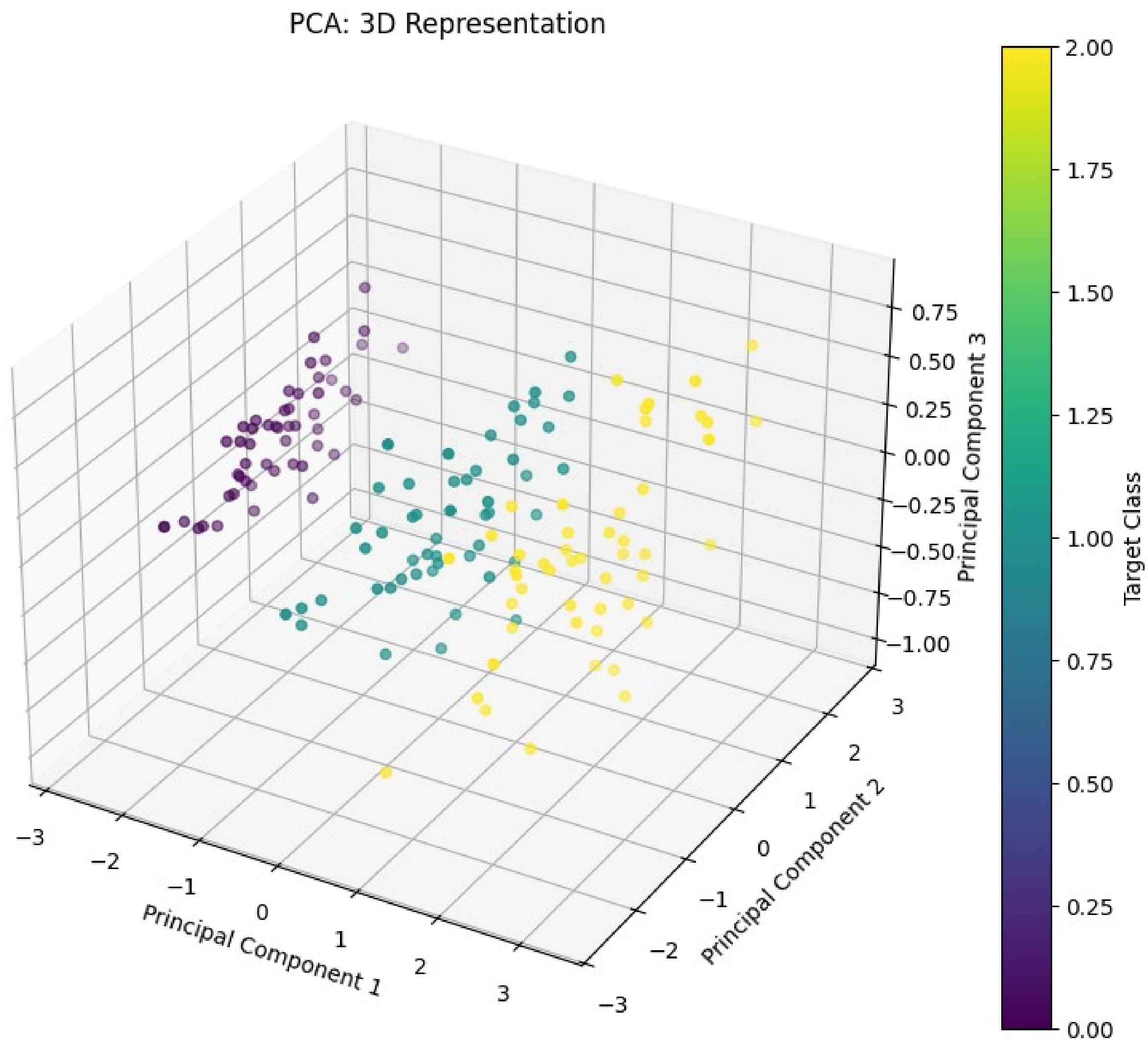
```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/
_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. Use
OneVsRestClassifier(LogisticRegression(..)) instead. Leave it to its
```

```
default value to avoid this warning.
  warnings.warn(
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
data = load_iris()
X = data.data
y = data.target
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variance = pca.explained_variance_ratio_
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(explained_variance) + 1), explained_variance,
marker='o')
plt.title('Scree Plot')
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.grid(True)
plt.show()
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('PCA: 2D Representation')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Target Class')
plt.grid(True)
plt.show()
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=y,
cmap='viridis')
ax.set_title('PCA: 3D Representation')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
fig.colorbar(scatter, label='Target Class')
plt.show()
```


Scree Plot
Explained Variance Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Principal Components


PCA: 2D Representation
Principal Component 1
Principal Component 2
Target Class
−3
−2
−1
0
1
2
3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



PCA: 3D Representation
Principal Component 1
Principal Component 2
Principal Component 3
Target Class
−3
−2
−1
0
1
2
3
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00