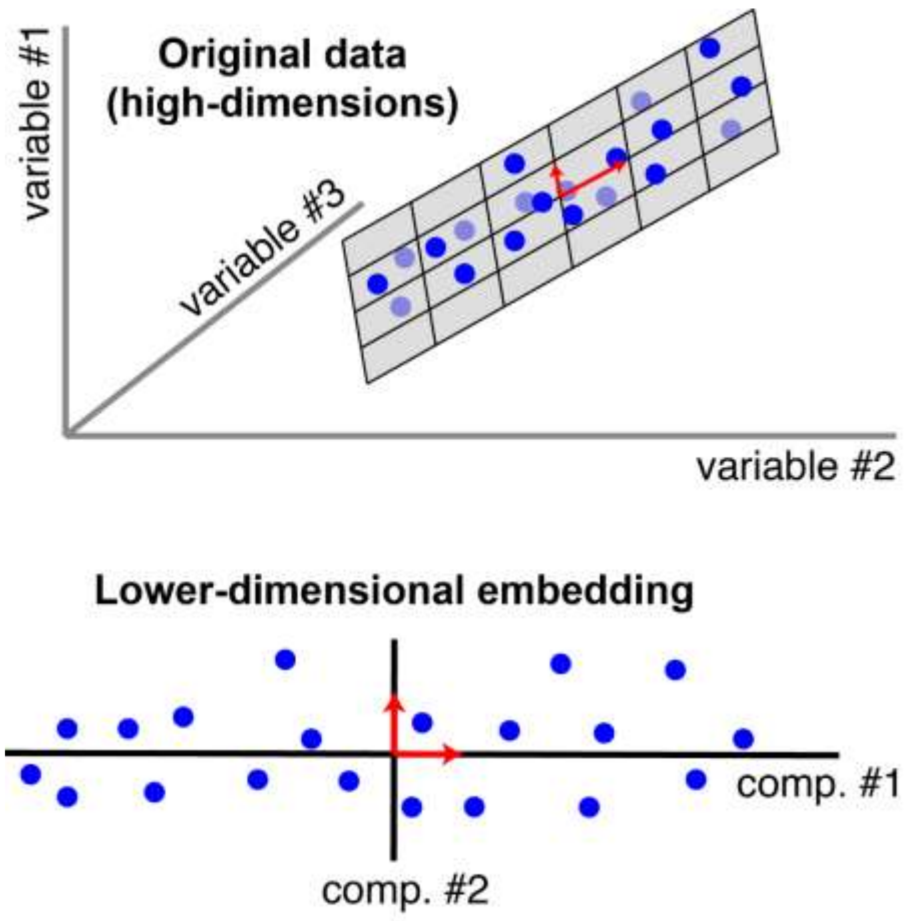
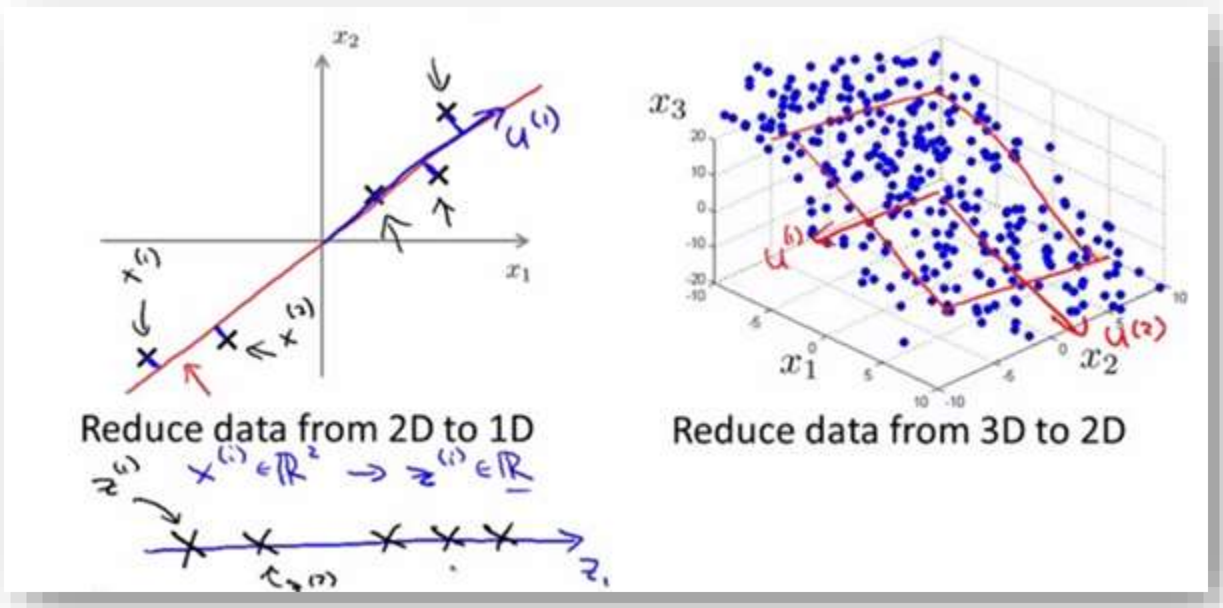
Principal Component Analysis (PCA)

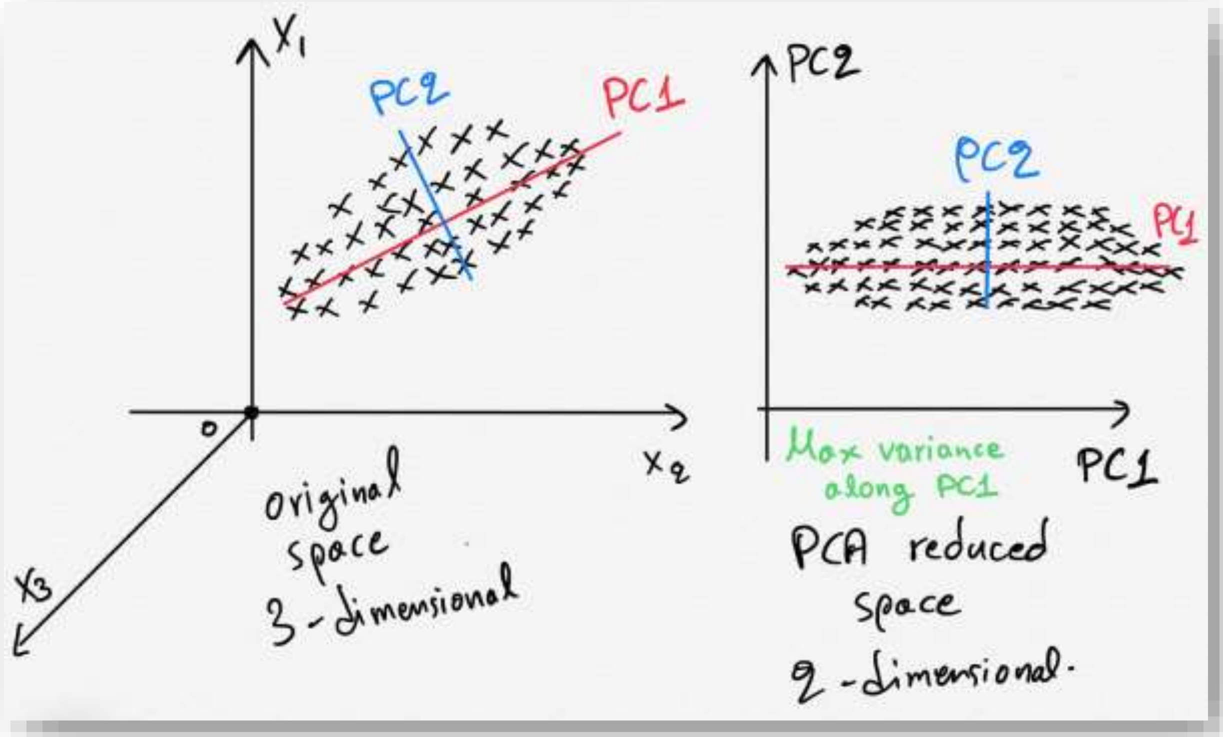
Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in data analysis, machine learning, and statistics. It is used to simplify complex datasets by reducing the number of variables while preserving the essential information. PCA achieves this by transforming the original variables into a new set of variables called principal components, which are linear combinations of the original variables.

Principal Component Analysis (PCA) is a dimensionality reduction technique invented by Karl Pearson in 1901, which is used for identification of a smaller number of uncorrelated variables known as Principal Components from a larger set of data. The following figures illustrates the same:





x_2
x_1
$u^{(1)}$
$x^{(1)}$
$x^{(2)}$
Reduce data from 2D to 1D
$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \mathbb{R}$
$z^{(1)}$
z_1
x_3
x_1
x_2
$u^{(1)}$
$u^{(2)}$
Reduce data from 3D to 2D



Dimension reduction offers several benefits such as:

- Compresses the data and thus reduces the storage space requirements
- Reduces the time required for computation since less dimensions require less computation
- Eliminates the redundant features
- Improves the model performance

Properties of Principal Component Analysis:

- Principal Component Analysis is a well-known dimension reduction technique
- It transforms the variables into a new set of variables called as principal components
- These principal components are linear combination of original variables and are orthogonal
- The first principal component accounts for most of the possible variation of original data
- The second principal component does its best to capture the variance in the data
- There can be only two principal components for a two-dimensional data set

Step-By-Step Explanation of PCA (Principal Component Analysis)

Principal Component Analysis (PCA) is a mathematical technique used for dimensionality reduction and data compression while preserving the most critical information in the data. It helps to reduce the complexity of high-dimensional data by transforming it into a lower-dimensional representation. Here's a step-by-step explanation of PCA:

Step 1: Data Preprocessing

Data Collection: Gather the dataset you want to analyse. Each row typically represents an observation, while each column corresponds to a variable (feature).

Standardization: If the variables are measured on different scales, standardize the data by subtracting the mean and dividing by the standard deviation for each variable.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable. This is often called as Z-score.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

Step 2: Compute the Covariance Matrix

Covariance Matrix: Calculate the covariance matrix of the standardized data. The covariance matrix, denoted as Σ (Sigma), describes the relationships and variances between variables. The diagonal elements of Σ represent the variances of individual variables, while off-diagonal elements represent covariances between pairs of variables.

$$\Sigma = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(x_i - \bar{x})^T$$

where n is the number of observations, x_i is a data point and $\bar{x}$ is the mean vector of the data.

Step 3: Compute Eigenvalues and Eigenvectors

Eigenvalue Decomposition: Perform eigenvalue decomposition on the covariance matrix ∑ to find its eigenvalues λ and corresponding eigenvectors v

$$\Sigma v = \lambda v$$

Where λ represents the eigenvalues and v represents the eigenvectors.

Step 4: Sort Eigenvalues

Sort Eigenvalues: Sort the eigenvalues in descending order. The eigenvalues indicate the amount of variance captured by each principal component (PC). By sorting them, you identify the most significant PCs.

Step 5: Select Principal Components

Select PCs: Decide how many principal components (PCs) to retain. You can choose a specific number of PCs or determine it based on the cumulative explained variance. A common approach is to retain enough PCs to explain a significant portion (e.g., 95%) of the total variance.

Step 6: Projection

Projection: Project the original data onto the selected principal components to obtain a lower-dimensional representation. This is done by multiplying the original data matrix (X) by the matrix of selected eigenvectors.

$$Y = X \cdot V_k$$

where Y is the reduced-dimensional data matrix, X is the original data matrix and V_k is a matrix containing the top k eigenvectors.

Step 7: Interpretation

Interpretation: The reduced-dimensional data in matrix Y now contains the information from the most important principal components. You can interpret these components to understand which variables or patterns contribute most to the variance in the data.

Step 8: Reconstruct Data (Optional)

Reconstruction (Optional): If needed, you can reverse the dimensionality reduction by multiplying Y by the transpose of Vk and adding back the mean (if standardization was performed). This gives you an approximation of the original data.

$$Xapprox = Y \cdot Vk^T + \bar{x}$$

PCA is a valuable tool for reducing dimensionality, visualizing data and simplifying complex datasets. It is widely used in various fields, including data science, machine learning and signal processing.

Python Code to Find PCA

To perform Principal Component Analysis (PCA) for the provided dataset in Python, you can follow these steps:

1. Create a NumPy array to represent the data.
2. Standardize the data.
3. Compute the covariance matrix.
4. Find the eigenvalues and eigenvectors.
5. Sort the eigenvalues and eigenvectors.
6. Choose the number of principal components to retain.
7. Project the data onto the selected principal components.
8. Visualize the results (optional).

Example

1. Find the Principal Component Analysis (PCA) for the following dataset:

x_1	**4**	**8**	**13**	**7**
x_2	**11**	**4**	**5**	**14**

Solution:

First, we need to standardize the data by subtracting the mean and dividing by the standard deviation for each variable.

Mean for x_1 and x_2 = [8, 8.5]

Standard deviation for x_1 and x_2 = [3.24037035, 4.15331193]

Standardized data is

x_1_std	-1.2344268	0.0	1.5430335	-0.3086067
x_2_std	0.60192927	-1.08347268	-0.84270097	1.32424438

Next, we calculate the covariance matrix ($\sum$) for the standardized data:

Covariance Matrix is

$$\sum = \begin{bmatrix} 1.33333333 & -0.81734138 \\ -0.81734138 & 1.33333333 \end{bmatrix}$$

Now, we find the eigenvalues (λ) and corresponding eigenvectors (ν) of the covariance matrix ($\sum$).

The eigenvalues are $\lambda_1 = 2.15067471$ and $\lambda_2 = 0.51599195$

The corresponding eigenvectors are

$$\begin{bmatrix} 0.70710678 & 0.70710678 \\ -0.70710678 & 0.70710678 \end{bmatrix}$$

Sort the eigenvalues in descending order and arrange the corresponding eigenvectors accordingly. In this case, $\lambda_1 > \lambda_2$, so:

Principal Component 1 (PC_1) corresponds to λ_1 and v_1.

Principal Component 2 (PC_2) corresponds to λ_2 and v_2.

Now, we need to choose the number of principal components to retain.

Let the number of principal components = 1

Now, Project the data onto the selected principal components.

This can be done by multiplying the original data matrix by the matrix of

selected eigenvectors.

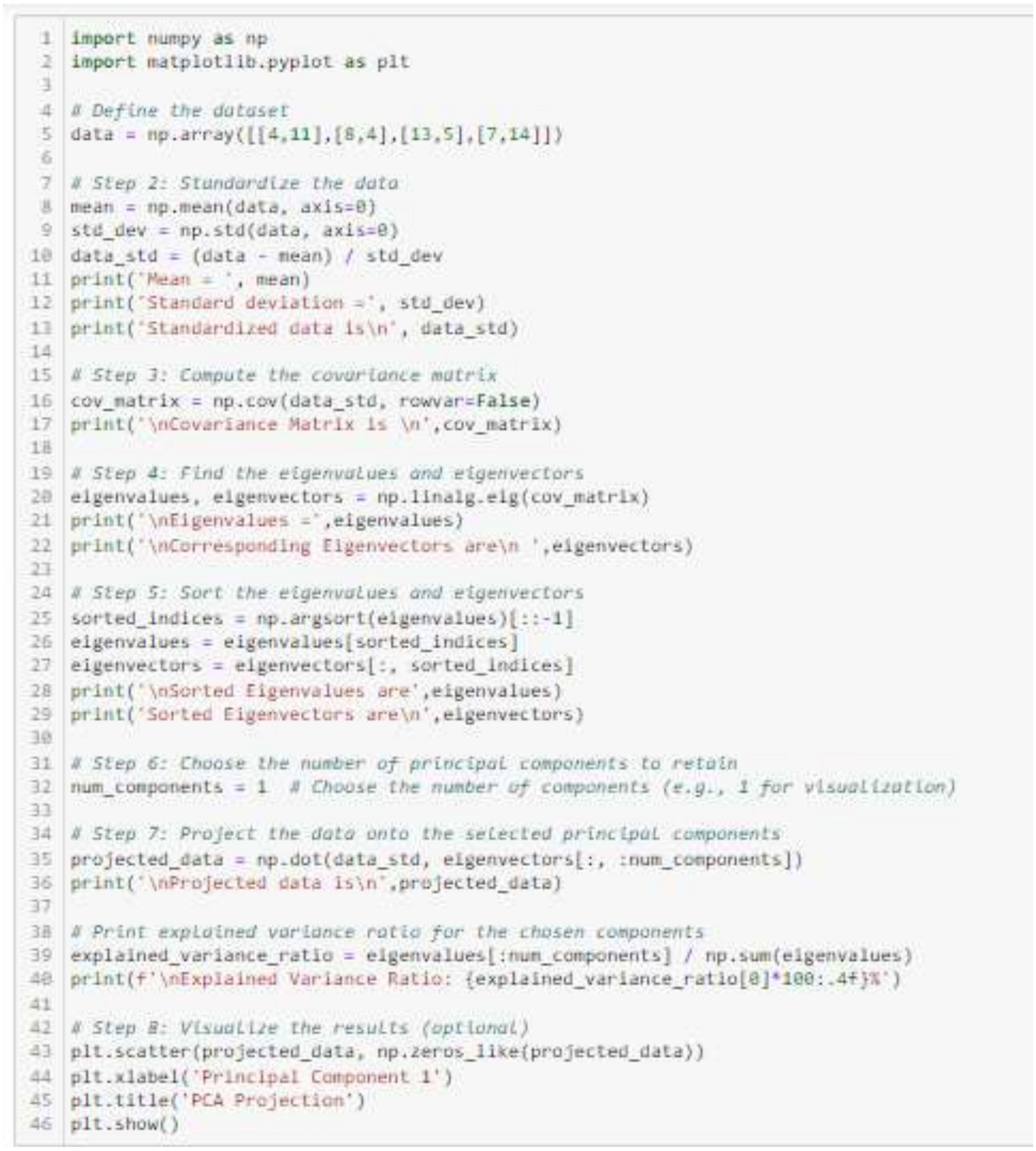
$$Y = X \cdot V_1 = \begin{bmatrix} -1.2344268 & 0.60192927 \\ 0. & -1.08347268 \\ 1.5430335 & -0.84270097 \\ -0.3086067 & 1.32424438 \end{bmatrix} \begin{bmatrix} 0.70710678 \\ -0.70710678 \end{bmatrix}$$

Projected data is $\begin{bmatrix} -1.29849983 \\ 0.76613088 \\ 1.68696902 \\ -1.15460007 \end{bmatrix}$

Explained variance ratio for the chosen component is

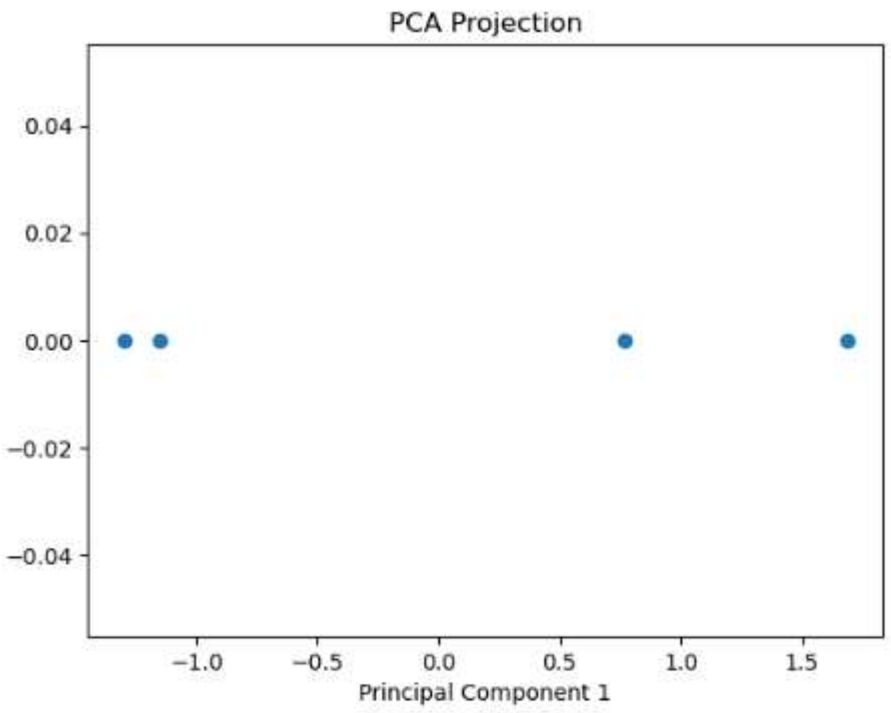
$$\frac{\lambda_1}{\lambda_1 + \lambda_2} \times 100 = \frac{2.15067471}{2.15067471 + 0.51599195} \times 100 = 80.65\%$$

Above example using Python Code



```python
import numpy as np
import matplotlib.pyplot as plt

# Define the dataset
data = np.array([[4,11],[8,4],[13,5],[7,14]])

# Step 2: Standardize the data
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)
data_std = (data - mean) / std_dev
print('Mean = ', mean)
print('Standard deviation =', std_dev)
print('Standardized data is\n', data_std)

# Step 3: Compute the covariance matrix
cov_matrix = np.cov(data_std, rowvar=False)
print('\nCovariance Matrix is \n',cov_matrix)

# Step 4: Find the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print('\nEigenvalues =',eigenvalues)
print('\nCorresponding Eigenvectors are\n ',eigenvectors)

# Step 5: Sort the eigenvalues and eigenvectors
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
print('\nSorted Eigenvalues are',eigenvalues)
print('Sorted Eigenvectors are\n',eigenvectors)

# Step 6: Choose the number of principal components to retain
num_components = 1  # Choose the number of components (e.g., 1 for visualization)

# Step 7: Project the data onto the selected principal components
projected_data = np.dot(data_std, eigenvectors[:, :num_components])
print('\nProjected data is\n',projected_data)

# Print explained variance ratio for the chosen components
explained_variance_ratio = eigenvalues[:num_components] / np.sum(eigenvalues)
print(f'\nExplained Variance Ratio: {explained_variance_ratio[0]*100:.4f}%')

# Step 8: Visualize the results (optional)
plt.scatter(projected_data, np.zeros_like(projected_data))
plt.xlabel('Principal Component 1')
plt.title('PCA Projection')
plt.show()
```

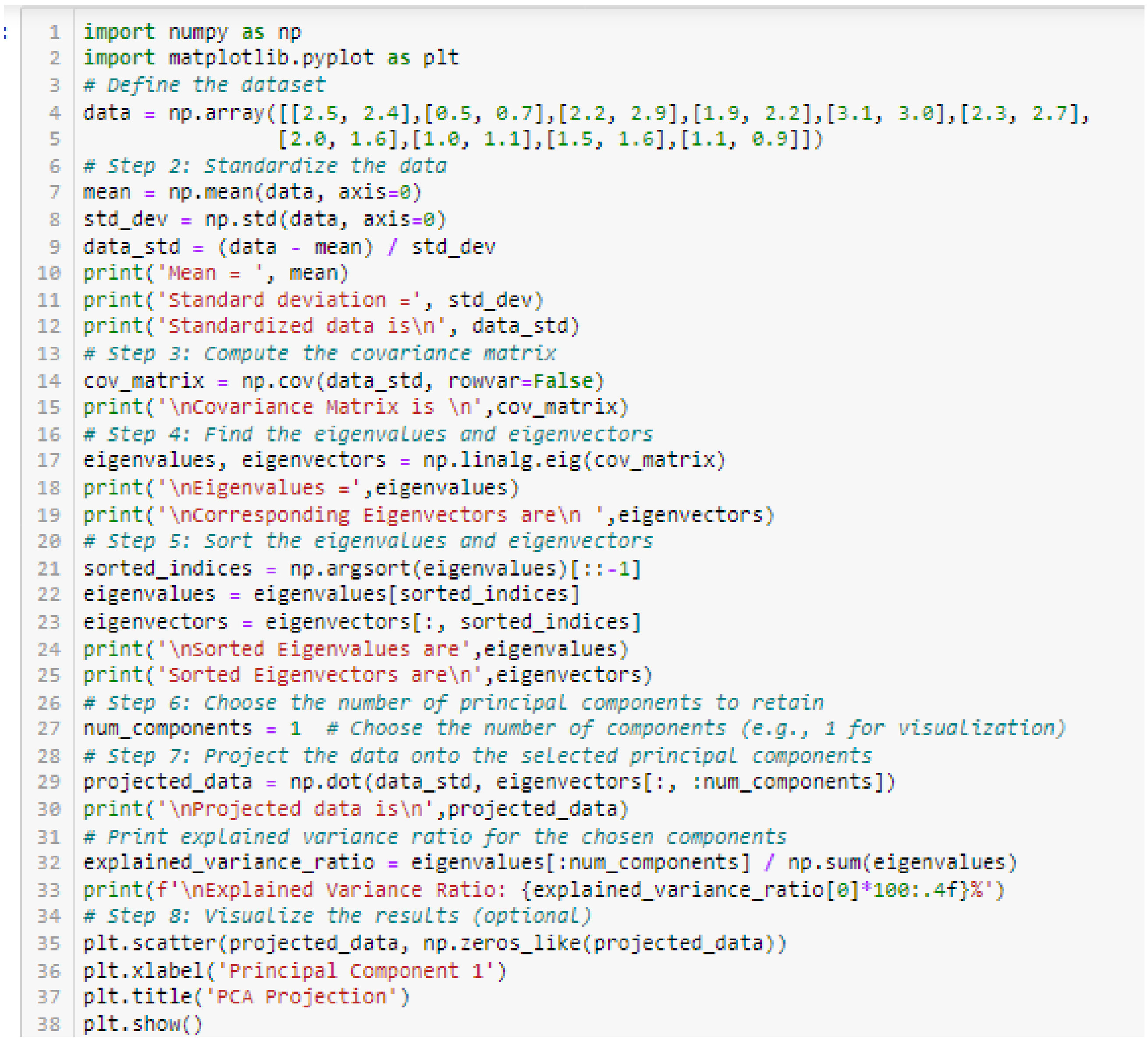
```
Mean =  [8.  8.5]
Standard deviation = [3.24037035 4.15331193]
Standardized data is
 [[-1.2344268   0.60192927]
 [ 0.         -1.08347268]
 [ 1.5430335  -0.84270097]
 [-0.3086067   1.32424438]]

Covariance Matrix is
 [[ 1.33333333 -0.81734138]
 [-0.81734138  1.33333333]]

Eigenvalues = [2.15067471 0.51599195]

Corresponding Eigenvectors are
  [[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]

Sorted Eigenvalues are [2.15067471 0.51599195]
Sorted Eigenvectors are
 [[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]

Projected data is
 [[-1.29849983]
 [ 0.76613088]
 [ 1.68696902]
 [-1.15460007]]

Explained Variance Ratio: 80.6503%
```



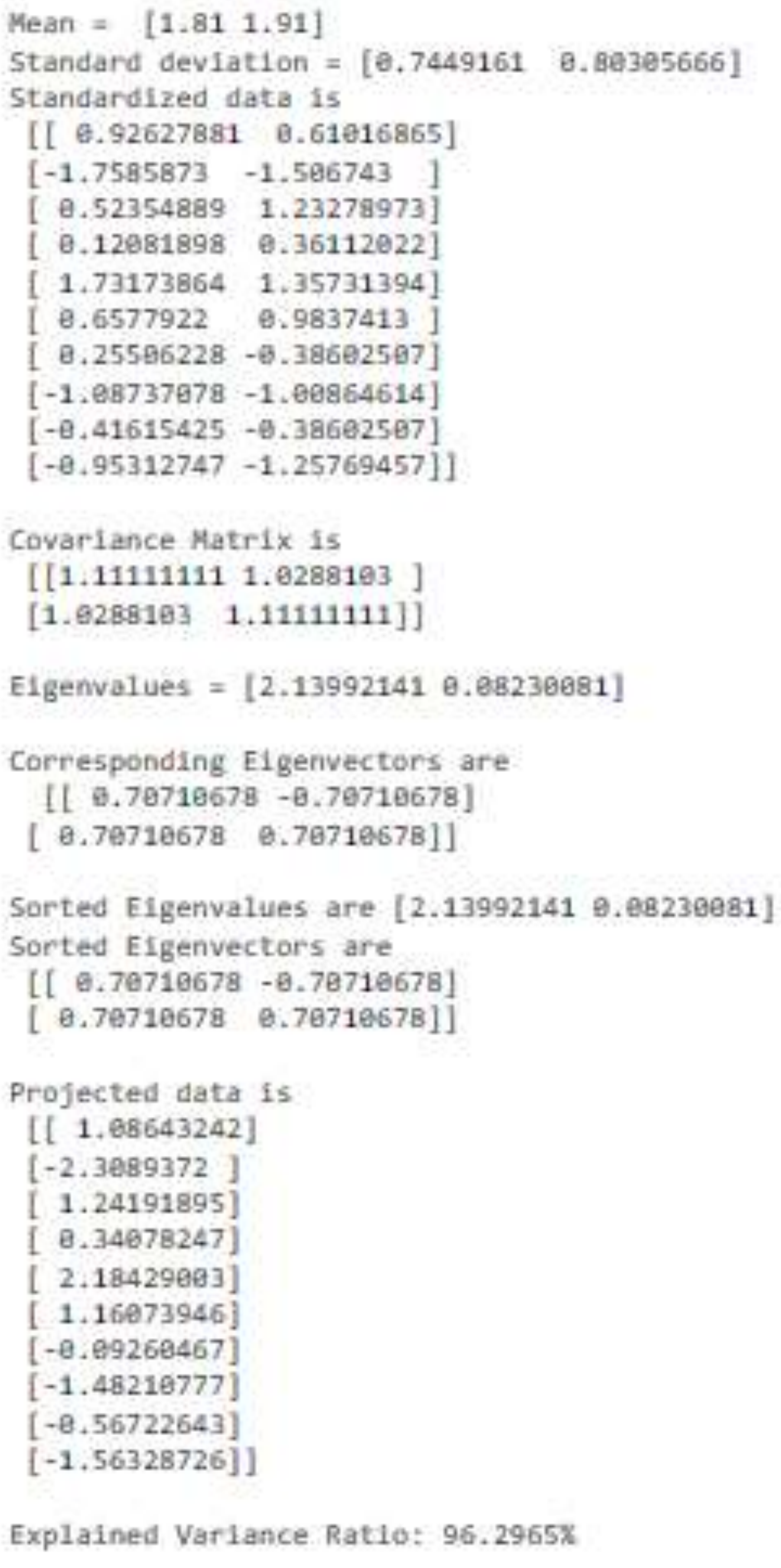
2. Find the Principal Component Analysis (PCA) for the following dataset:

x_1	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1
x_2	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9

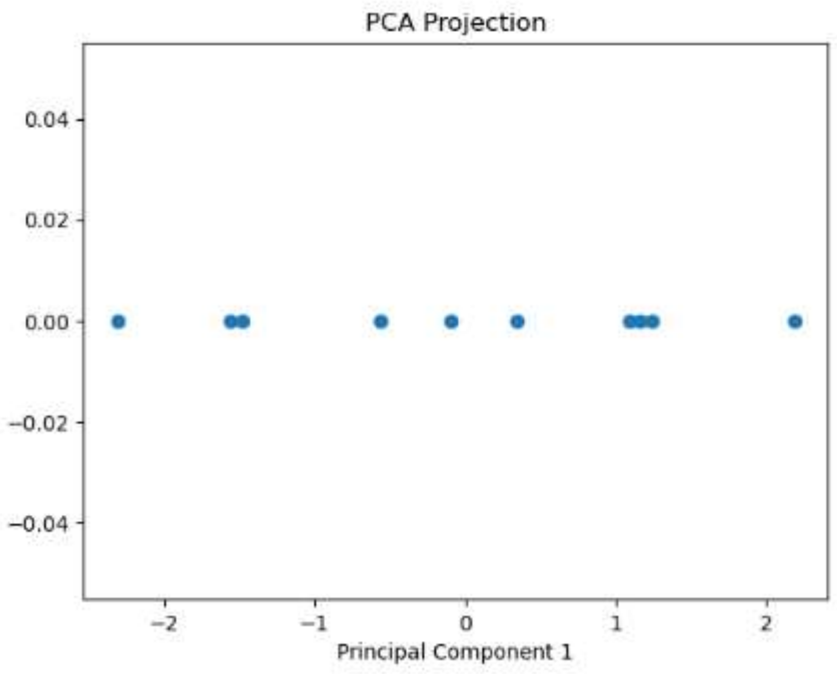
Using Python Code



```python
import numpy as np
import matplotlib.pyplot as plt
# Define the dataset
data = np.array([[2.5, 2.4],[0.5, 0.7],[2.2, 2.9],[1.9, 2.2],[3.1, 3.0],[2.3, 2.7],
                [2.0, 1.6],[1.0, 1.1],[1.5, 1.6],[1.1, 0.9]])
# Step 2: Standardize the data
mean = np.mean(data, axis=0)
std_dev = np.std(data, axis=0)
data_std = (data - mean) / std_dev
print('Mean = ', mean)
print('Standard deviation =', std_dev)
print('Standardized data is\n', data_std)
# Step 3: Compute the covariance matrix
cov_matrix = np.cov(data_std, rowvar=False)
print('\nCovariance Matrix is \n',cov_matrix)
# Step 4: Find the eigenvalues and eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print('\nEigenvalues =',eigenvalues)
print('\nCorresponding Eigenvectors are\n ',eigenvectors)
# Step 5: Sort the eigenvalues and eigenvectors
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
print('\nSorted Eigenvalues are',eigenvalues)
print('Sorted Eigenvectors are\n',eigenvectors)
# Step 6: Choose the number of principal components to retain
num_components = 1  # Choose the number of components (e.g., 1 for visualization)
# Step 7: Project the data onto the selected principal components
projected_data = np.dot(data_std, eigenvectors[:, :num_components])
print('\nProjected data is\n',projected_data)
# Print explained variance ratio for the chosen components
explained_variance_ratio = eigenvalues[:num_components] / np.sum(eigenvalues)
print(f'\nExplained Variance Ratio: {explained_variance_ratio[0]*100:.4f}%')
# Step 8: Visualize the results (optional)
plt.scatter(projected_data, np.zeros_like(projected_data))
plt.xlabel('Principal Component 1')
plt.title('PCA Projection')
plt.show()
```



```
Mean =  [1.81 1.91]
Standard deviation = [0.7449161  0.80305666]
Standardized data is
 [[ 0.92627881  0.61016865]
 [-1.7585873  -1.506743  ]
 [ 0.52354889  1.23278973]
 [ 0.12081898  0.36112022]
 [ 1.73173864  1.35731394]
 [ 0.6577922   0.9837413 ]
 [ 0.25506228 -0.38602507]
 [-1.08737078 -1.00864614]
 [-0.41615425 -0.38602507]
 [-0.95312747 -1.25769457]]

Covariance Matrix is
 [[1.11111111 1.0288103 ]
 [1.0288103  1.11111111]]

Eigenvalues = [2.13992141 0.08230081]

Corresponding Eigenvectors are
  [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

Sorted Eigenvalues are [2.13992141 0.08230081]
Sorted Eigenvectors are
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

Projected data is
 [[ 1.08643242]
 [-2.3089372 ]
 [ 1.24191895]
 [ 0.34078247]
 [ 2.18429003]
 [ 1.16073946]
 [-0.09260467]
 [-1.48210777]
 [-0.56722643]
 [-1.56328726]]

Explained Variance Ratio: 96.2965%
```



To perform Principal Component Analysis (PCA) in Python using the sklearn library:

In this code:

- We first import the necessary libraries.
- We assume `X` is your data.
- We standardize the data using `StandardScaler` from `sklearn.preprocessing`. This subtracts the mean and divides by the standard deviation for each variable.
- We perform PCA using `PCA` from `sklearn.decomposition`. The `n_components` parameter specifies the number of components to keep. In this case, we are keeping all components.
- The transformed data is stored in `principalComponents`.

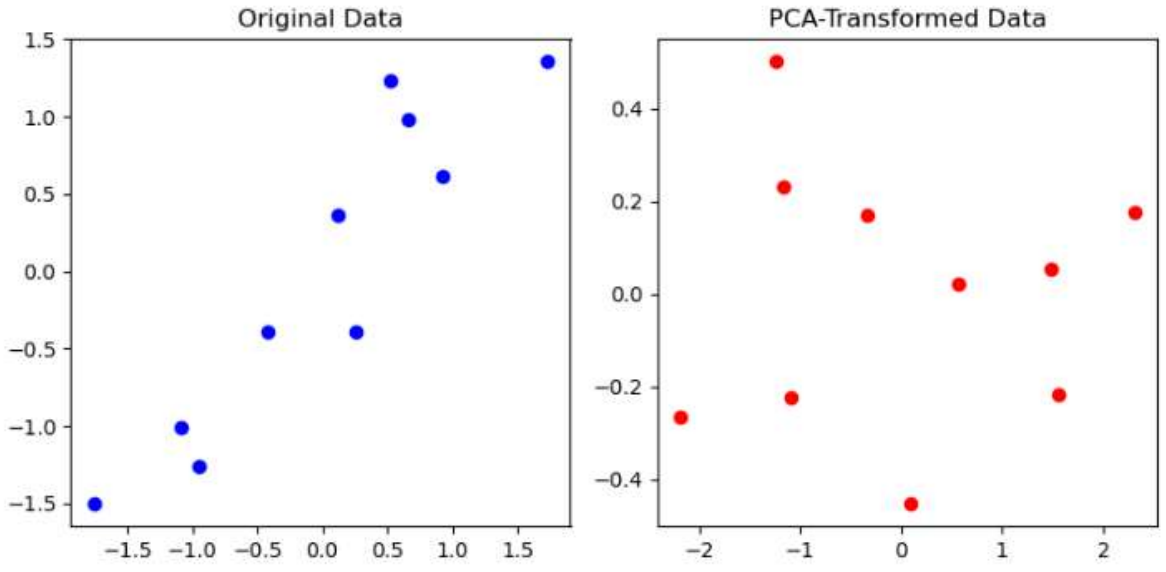
Examples

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np

# Assume X is your data
X = np.array([[1, 2], [3, 4], [5, 6]])

# Standardize the data
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Perform PCA
pca = PCA(n_components=1)
principalComponents = pca.fit_transform(X_std)

print(principalComponents)

```

```
[[ 1.73205081]
 [-0.        ]
 [-1.73205081]]
```

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Create a sample dataset (you can replace this with your dataset)
data = np.array([[2.5, 2.4],[0.5, 0.7],[2.2, 2.9],[1.9, 2.2],[3.1, 3.0],
        [2.3, 2.7],[2.0, 1.6],[1.0, 1.1],[1.5, 1.6],[1.1, 0.9]])
# Step 3: Standardize the data
scaler = StandardScaler()
data_std = scaler.fit_transform(data)
print('Standardized data is:\n',data_std)
# Step 4: Initialize PCA
pca = PCA(n_components=2)  # Choose the number of components you want
print(pca)
# Step 5: Fit the PCA model to the standardized data
pca.fit(data_std)
# Step 6: Transform the data into the PCA space
data_pca = pca.transform(data_std)
print("Principal component Analysis is \n",data_pca)
# Step 7: Visualize the results (optional)
plt.figure(figsize=(8, 4))
# Original data
plt.subplot(1, 2, 1)
plt.scatter(data_std[:, 0], data_std[:, 1], c='blue')
plt.title('Original Data')
# PCA-transformed data
plt.subplot(1, 2, 2)
plt.scatter(data_pca[:, 0], data_pca[:, 1], c='red')
plt.title('PCA-Transformed Data')
plt.tight_layout()
plt.show()
# Print explained variance ratios
explained_variance_ratio = pca.explained_variance_ratio_
print(f'\nExplained Variance Ratios: {explained_variance_ratio[0]*100:.3f}% and {explained_variance_ratio[1]*100:.3f}%')
```

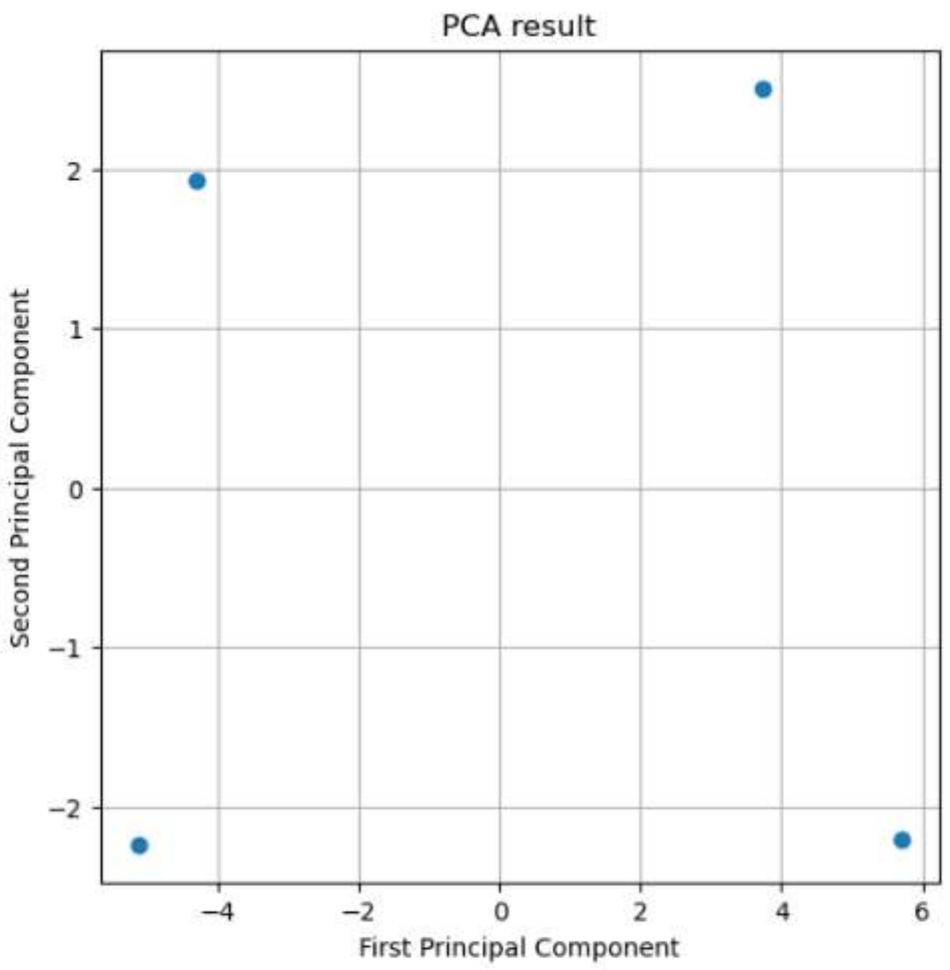
```
Standardized data is:
 [[ 0.92627881  0.61016865]
 [-1.7585873  -1.506743  ]
 [ 0.52354889  1.23278973]
 [ 0.12081898  0.36112022]
 [ 1.73173864  1.35731394]
 [ 0.6577922   0.9837413 ]
 [ 0.25506228 -0.38602507]
 [-1.08737078 -1.00864614]
 [-0.41615425 -0.38602507]
 [-0.95312747 -1.25769457]]
PCA(n_components=2)
Principal component Analysis i
 [[-1.08643242 -0.22352364]
 [ 2.3089372   0.17808082]
 [-1.24191895  0.501509  ]
 [-0.34078247  0.16991864]
 [-2.18429003 -0.26475825]
 [-1.16073946  0.23048082]
 [ 0.09260467 -0.45331721]
 [ 1.48210777  0.05566672]
 [ 0.56722643  0.02130455]
 [ 1.56328726 -0.21536146]]
```



```
Explained Variance Ratios: 96.296% and 3.704%
```

```python
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# Given data
data = np.array([[4, 11], [8, 4], [13, 5], [7, 14]])
# Initialize PCA
pca = PCA(n_components=2)
# Fit the PCA model to your data
pca.fit(data)
# The principal components
print("Principal components:")
print(pca.components_)
# The explained variance
print("Explained variance:")
print(pca.explained_variance_)
# Transform the data to the first principal components
transformed_data = pca.transform(data)
print("Transformed data:")
print(transformed_data)
# Create a scatter plot of the transformed data
plt.figure(figsize=(6,6))
plt.scatter(transformed_data[:, 0], transformed_data[:, 1])
plt.title('PCA result')
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.grid(True)
plt.show()
```

```
Principal components:
[[ 0.55738997 -0.83025082]
 [-0.83025082 -0.55738997]]
Explained variance:
[30.38486432  6.61513568]
Transformed data:
[[-4.30518692  1.92752836]
 [ 3.73612869  2.50825486]
 [ 5.69282771 -2.20038921]
 [-5.12376947 -2.23539401]]
```



PCA has several applications:

Dimensionality Reduction: It reduces the number of variables in a dataset, making it easier to analyse and visualize while retaining most of the important information.

Feature Engineering: In machine learning, PCA can be used for feature selection or feature extraction to improve model performance.

Noise Reduction: PCA can help remove noise from data by focusing on the principal components with the highest variance.

Data Visualization: PCA can be used to visualize high-dimensional data in a lower-dimensional space, allowing for easier data exploration and pattern recognition.

NOTE

Keep in mind that while PCA is a powerful technique, it's not always suitable for every dataset. The choice of the number of principal components to retain should be made carefully, and the interpretation of the reduced dimensions might not always be straightforward. PCA is also sensitive to outliers, so data preprocessing and outlier handling are essential.