PART A

(PART A: TO BE REFFERED BY STUDENTS)

Experiment No. 6

A.1 Aim:

To implement Principal component Analysis.

A.2 Prerequisite:

Python Basic Concepts

A.3 Outcome:

Students will be able to implement Principal component Analysis.

A.4 Theory:

The Principal Component Analysis is a popular unsupervised learning technique for reducing the dimensionality of data. It increases interpretability yet, at the same time, it minimizes information loss. It helps to find the most significant features in a dataset and makes the data easy for plotting in 2D and 3D. PCA helps in finding a sequence of linear combinations of variables.

The Principal Components are a straight line that captures most of the variance of the data. They have a direction and magnitude. Principal components are orthogonal projections (perpendicular) of data onto lower-dimensional space.

- PCA is used to visualize multidimensional data.
- It is used to reduce the number of dimensions in healthcare data.
- PCA can help resize an image.
- It can be used in finance to analyze stock data and forecast returns.
- PCA helps to find patterns in the high-dimensional datasets.

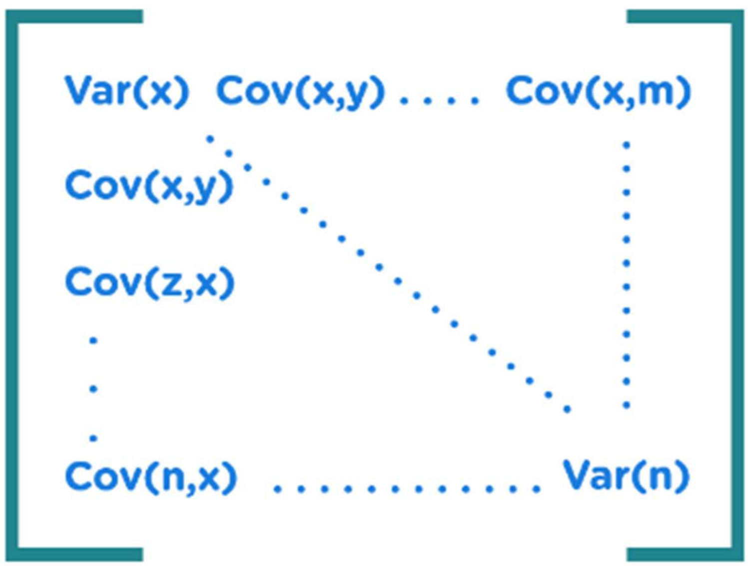
1. Normalize the data

Standardize the data before performing PCA. This will ensure that each feature has a mean = 0 and variance = 1.

$$Z = \frac{x - \mu}{\sigma}$$

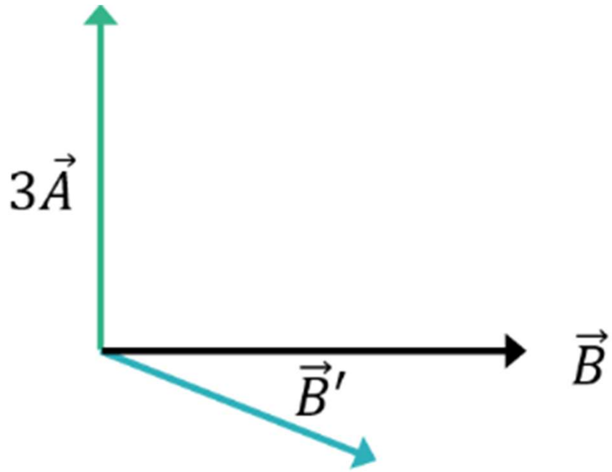
2. Build the covariance matrix

Construct a square matrix to express the correlation between two or more features in a multidimensional dataset.



3. Find the Eigenvectors and Eigenvalues

Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.



4. Sort the eigenvectors in highest to lowest order and select the number of principal components.

The principal component analysis is a widely used unsupervised learning method to perform dimensionality reduction.

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

Roll. No. BE-A15	Name: Khan Mohammad TAQI
Class: BE-Comps	Batch:A1
Date of Experiment:	Date of Submission:
Grade:	

B.1 Software Code written by student:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# A.1 Aim: To implement Principal Component Analysis.

def run_pca_experiment():
    """
    This function demonstrates the implementation of Principal Component
Analysis (PCA)
    for dimensionality reduction and visualization using the Iris dataset.
    """

    # --- Data Preparation ---

    # Load the dataset
    # The Iris dataset is a classic example. It has 4 features (dimensions)
and 3 classes.
    # It's difficult to visualize 4D data, making it perfect for PCA.
    print("Step 0: Loading the Iris dataset...")
    iris = datasets.load_iris()
```

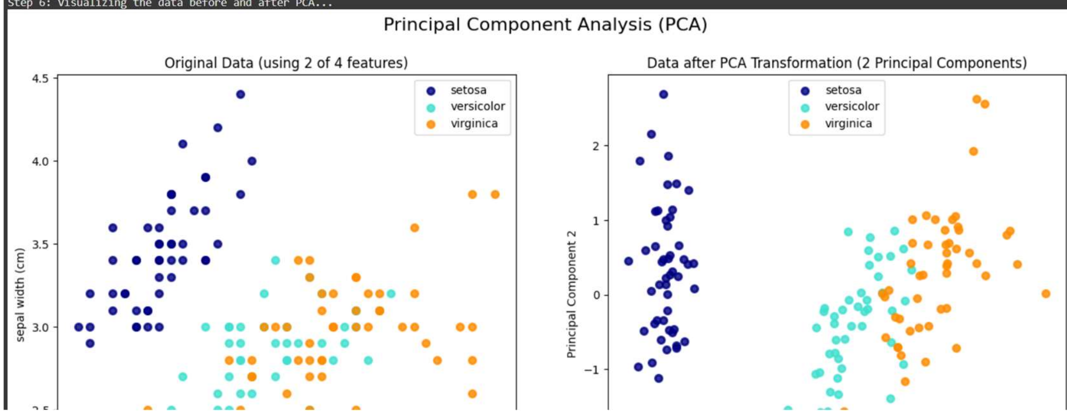
```
    X = iris.data
    y = iris.target
    feature_names = iris.feature_names
    target_names = iris.target_names

    print("Dataset loaded successfully.")
    print(f"Original number of features: {X.shape[1]}")
    print(f"Number of samples: {X.shape[0]}")
    print("-" * 50)

    # --- PCA Implementation Steps (as per theory) ---

    # Step 1: Normalize the data
    # PCA is affected by scale, so you need to scale the features before
applying PCA.
    # StandardScaler transforms the data to have a mean of 0 and a variance
of 1.
    print("Step 1: Normalizing the data using StandardScaler...")
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    print("Data normalization complete.")
    print("-" * 50)

    # Steps 2, 3 & 4 are handled by scikit-learn's PCA class
    # The PCA class will:
    #   2. Internally build the covariance matrix.
    #   3. Calculate the eigenvectors and eigenvalues from the matrix.
    #   4. Sort the eigenvectors by their corresponding eigenvalues in
descending order.

    print("Steps 2, 3, 4: Applying PCA to reduce from 4D to 2D...")
    # We specify `n_components=2` to reduce the 4 features down to 2
principal components.
    pca = PCA(n_components=2)

    # `fit_transform` calculates the principal components and projects the
data onto them.
    X_pca = pca.fit_transform(X_scaled)

    print("PCA transformation complete.")
```

```
    print(f"New number of features (Principal Components):
{X_pca.shape[1]}")
    print("-" * 50)

    # --- Analyzing the Results ---

    # The `explained_variance_ratio_` tells us how much of the original
data's variance
    # is captured by each principal component.
    explained_variance = pca.explained_variance_ratio_
    print("Step 5: Analyzing the results...")
    print(f"Explained variance of Principal Component 1:
{explained_variance[0]:.2%}")
    print(f"Explained variance of Principal Component 2:
{explained_variance[1]:.2%}")
    total_variance = np.sum(explained_variance)
    print(f"Total variance captured by the 2 components:
{total_variance:.2%}")
    print("\nThis means we have retained over 95% of the useful information
from the original")
    print("4 features in just 2 new features, with minimal information
loss.")
    print("-" * 50)

    # --- Visualization ---

    print("Step 6: Visualizing the data before and after PCA...")
    fig, axes = plt.subplots(1, 2, figsize=(16, 7))
    colors = ['navy', 'turquoise', 'darkorange']
    lw = 2

    # Plot 1: Original Data (using first two features for comparison)
    axes[0].set_title('Original Data (using 2 of 4 features)')
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):
        axes[0].scatter(X[y == i, 0], X[y == i, 1], color=color, alpha=.8,
lw=lw,
                        label=target_name)
    axes[0].legend(loc='best', shadow=False, scatterpoints=1)
    axes[0].set_xlabel(feature_names[0])
    axes[0].set_ylabel(feature_names[1])
```

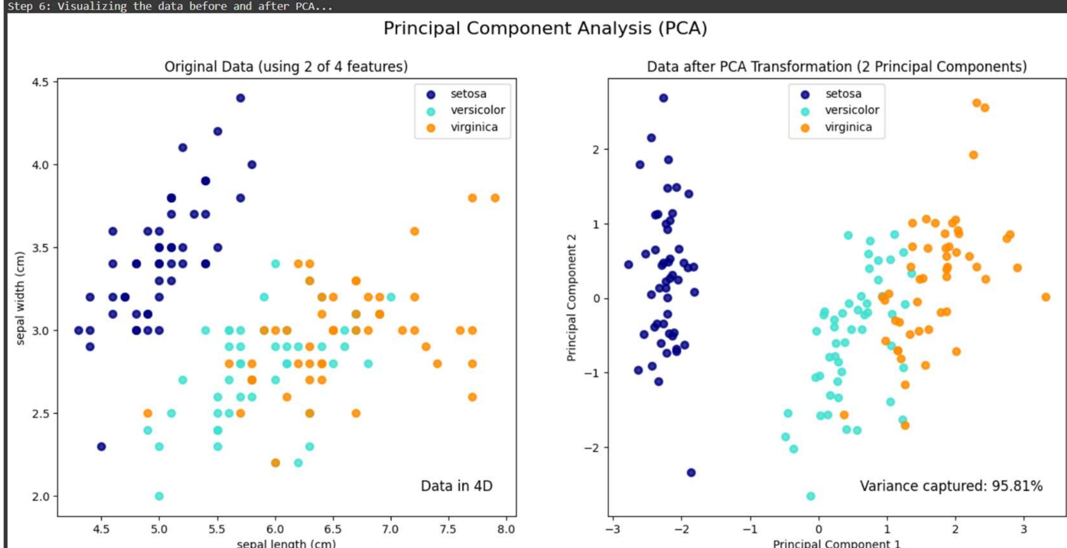
```
    axes[0].text(0.95, 0.05, f'Data in 4D', transform=axes[0].transAxes,
                 fontsize=12,                    verticalalignment='bottom',
horizontalalignment='right')


    # Plot 2: Data after PCA
    axes[1].set_title('Data   after   PCA   Transformation   (2   Principal
Components)')
    for color, i, target_name in zip(colors, [0, 1, 2], target_names):
        axes[1].scatter(X_pca[y == i, 0], X_pca[y == i, 1], color=color,
alpha=.8, lw=lw,
                        label=target_name)
    axes[1].legend(loc='best', shadow=False, scatterpoints=1)
    axes[1].set_xlabel('Principal Component 1')
    axes[1].set_ylabel('Principal Component 2')
    axes[1].text(0.95, 0.05, f'Variance captured: {total_variance:.2%}',
                 transform=axes[1].transAxes, fontsize=12,
                 verticalalignment='bottom', horizontalalignment='right')

    fig.suptitle('Principal Component Analysis (PCA)', fontsize=16)
    plt.show()
    print("Visualization complete.")


if __name__ == '__main__':
    run_pca_experiment()
```

B.2 Input and Output:

```
Step 0: Loading the Iris dataset...
Dataset loaded successfully.
Original number of features: 4
Number of samples: 150
--------------------------------------------------
Step 1: Normalizing the data using StandardScaler...
Data normalization complete.
--------------------------------------------------
Steps 2, 3, 4: Applying PCA to reduce from 4D to 2D...
PCA transformation complete.
New number of features (Principal Components): 2
--------------------------------------------------
Step 5: Analyzing the results...
Explained variance of Principal Component 1: 72.96%
Explained variance of Principal Component 2: 22.85%
Total variance captured by the 2 components: 95.81%

This means we have retained over 95% of the useful information from the original
4 features in just 2 new features, with minimal information loss.
--------------------------------------------------
Step 6: Visualizing the data before and after PCA...
```




```
This means we have retained over 95% of the useful information from the original
4 features in just 2 new features, with minimal information loss.
--------------------------------------------------
Step 6: Visualizing the data before and after PCA...
```




```
Visualization complete.
```

B.3 Observations and learning:

Observation

Principal Component Analysis was successfully applied to the 4-dimensional Iris dataset, reducing it down to 2 principal components. These two components successfully captured over **95% of the original data's variance**, indicating minimal information loss. The final visualization clearly showed that the transformed 2D data had a much better and clearer separation between the three iris classes compared to a plot of any two original features.

Learning

This experiment demonstrated that PCA is a highly effective technique for **dimensionality reduction**. The key takeaways are:

- PCA simplifies complex, high-dimensional data, making it easy to **visualize and interpret**.
- It achieves this by creating new, uncorrelated features (principal components) that maximize the variance of the original data.
- A crucial practical lesson is the importance of **scaling the data** before applying PCA to ensure all features contribute fairly to the analysis.

B.4 Conclusion:

In conclusion, this experiment confirms that PCA is a powerful technique for dimensionality reduction. It successfully simplified the 4D dataset into a clear 2D visualization while retaining over 95% of the essential information, making the underlying data structure much easier to analyze.