NAME: - Medhansh K PRN: - 122A8035 BATCH: - B2

EXPERIMENT 10

**AIM: -** Principal Component Analysis

# THEROY: -

Here's a breakdown of the steps involved in Principal Component Analysis (PCA):

# Data Preparation:

* 1. Gather your dataset, which should consist of numerical features.
  2. Organize the data into a matrix, where rows represent samples and columns represent features.

# Standardization (Scaling):

* 1. This is a crucial step. Standardize the features to have zero mean and unit variance.
  2. This ensures that features with different scales contribute equally to the PCA.
  3. Common methods include z-score standardization.

# Covariance Matrix Calculation:

* 1. Compute the covariance matrix of the standardized data.
  2. The covariance matrix shows how the features vary together.

# Eigenvalue and Eigenvector Decomposition:

* 1. Calculate the eigenvalues and eigenvectors of the covariance matrix.
  2. Eigenvectors represent the directions of maximum variance.
  3. Eigenvalues represent the magnitude of the variance along those directions.

# Eigenvector Sorting:

* 1. Sort the eigenvectors in descending order based on their corresponding eigenvalues.
  2. This arranges the principal components in order of importance (variance explained).

# Principal Component Selection:

* 1. Choose the top 'k' eigenvectors, where 'k' is the desired number of dimensions for the reduced dataset.
  2. The selection of k is often determined by the desired amount of explained variance.
  3. The explained variance ratio helps to make this decision.

# Data Projection:

* 1. Multiply the standardized data by the selected eigenvectors (the principal components).
  2. This transforms the original data into the new, lower-dimensional space defined by the principal components.

# Interpretation:

* 1. Analyze the transformed data.
  2. Visualize the data in the reduced dimensions.
  3. Interpret the principal components in terms of the original features.
  4. Examine the explained variance ratio to understand how much information is retained.

# Principal Component Analysis Algorithm Step by Step Implementation

# Import necessary libraries import numpy as np

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

# Set up plotting style plt.style.use('ggplot')

sns.set\_style('whitegrid')

%matplotlib inline

# Generate random data np.random.seed(42) # For reproducibility

# Create 3 clusters with different means but correlated features n\_samples = 200 # Samples per cluster

n\_features = 5 # Number of features

# Generate correlated features - create a covariance matrix with off-diagonal elements cov\_matrix = np.eye(n\_features) # Start with identity matrix

# Add correlation between features for i in range(n\_features):

for j in range(n\_features): if i != j:

# Higher correlation between neighboring features cov\_matrix[i, j] = 0.8 \*\* abs(i-j)

# Create 3 clusters cluster\_centers = [

np.array([0, 0, 0, 0, 0]), # Cluster 1 centered at origin

np.array([3, 4, 2, 3, 2]), # Cluster 2

np.array([-3, -2, -4, -1, -2]) # Cluster 3

]

# Generate data for each cluster with the correlated features X = np.vstack([

np.random.multivariate\_normal(center, cov\_matrix, n\_samples) for center in cluster\_centers

])

# Create labels for the clusters y = np.hstack([

np.full(n\_samples, i) for i in range(len(cluster\_centers))

])

# Create feature names

feature\_names = [f'Feature\_{i+1}' for i in range(n\_features)] target\_names = [f'Cluster\_{i}' for i in range(len(cluster\_centers))]

# Create a DataFrame for better visualization

df = pd.DataFrame(X, columns=feature\_names) df['target'] = y

df['cluster'] = [target\_names[i] for i in y]

# Display information about the data print("Random Dataset Information:") print(f"- Number of samples: {X.shape[0]}") print(f"- Number of features: {X.shape[1]}")

print(f"- Number of clusters: {len(np.unique(y))}") print("\nFeature correlation matrix:")

correlation = np.corrcoef(X.T) plt.figure(figsize=(10, 8))

sns.heatmap(correlation, annot=True, cmap='coolwarm', xticklabels=feature\_names, yticklabels=feature\_names)

plt.title('Correlation Matrix of Features') plt.tight\_layout()

plt.show()

# 1. BEFORE PCA: Pairwise scatter plots of original features (showing just a subset) plt.figure(figsize=(15, 6))

plt.suptitle('BEFORE PCA: Pairwise Plots of Original Features (Sample)', fontsize=16, y=1.02)

# Create a grid of pairwise scatter plots for a sample of feature combinations feature\_pairs = [(0, 1), (0, 2), (1, 2)]

plot\_positions = [1, 2, 3]

for pair, position in zip(feature\_pairs, plot\_positions): plt.subplot(1, 3, position)

for cluster\_id, cluster\_name in enumerate(target\_names): plt.scatter(

df[df['target'] == cluster\_id][feature\_names[pair[0]]], df[df['target'] == cluster\_id][feature\_names[pair[1]]], label=cluster\_name,

alpha=0.7

)

plt.xlabel(feature\_names[pair[0]]) plt.ylabel(feature\_names[pair[1]])

if position == 3: # Only show legend on the last plot plt.legend()

plt.tight\_layout(rect=[0, 0, 1, 0.96]) plt.show()

# 2. Apply PCA

# First standardize the data scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Perform PCA pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

# Get explained variance information explained\_variance = pca.explained\_variance\_ratio\_

cumulative\_variance = np.cumsum(pca.explained\_variance\_ratio\_)

# Create a DataFrame for the PCA-transformed data

pca\_df = pd.DataFrame(data=X\_pca, columns=[f'PC{i+1}' for i in range(X\_pca.shape[1])]) pca\_df['target'] = y

pca\_df['cluster'] = [target\_names[i] for i in y]

# 3. AFTER PCA: Visualize the PCA results plt.figure(figsize=(15, 6))

plt.suptitle('AFTER PCA: Data in Principal Component Space', fontsize=16, y=1.02)

# Plot PC1 vs PC2 plt.subplot(1, 3, 1)

for cluster\_id, cluster\_name in enumerate(target\_names): plt.scatter(

pca\_df[pca\_df['target'] == cluster\_id]['PC1'], pca\_df[pca\_df['target'] == cluster\_id]['PC2'], label=cluster\_name,

alpha=0.7

)

plt.xlabel(f'PC1 ({explained\_variance[0]:.2%} variance)') plt.ylabel(f'PC2 ({explained\_variance[1]:.2%} variance)') plt.title('PC1 vs PC2')

# Plot PC1 vs PC3 plt.subplot(1, 3, 2)

for cluster\_id, cluster\_name in enumerate(target\_names): plt.scatter(

pca\_df[pca\_df['target'] == cluster\_id]['PC1'], pca\_df[pca\_df['target'] == cluster\_id]['PC3'], label=cluster\_name,

alpha=0.7

)

plt.xlabel(f'PC1 ({explained\_variance[0]:.2%} variance)') plt.ylabel(f'PC3 ({explained\_variance[2]:.2%} variance)') plt.title('PC1 vs PC3')

# Plot explained variance plt.subplot(1, 3, 3)

plt.bar(range(1, len(explained\_variance) + 1), explained\_variance, alpha=0.7, label='Individual') plt.step(range(1, len(explained\_variance) + 1), cumulative\_variance, where='mid', label='Cumulative')

plt.axhline(y=0.95, linestyle='--', color='r', label='95% variance') plt.xlabel('Principal Component')

plt.ylabel('Explained Variance Ratio')

plt.title('Explained Variance') plt.legend()

plt.xticks(range(1, len(explained\_variance) + 1))

plt.tight\_layout(rect=[0, 0, 1, 0.96]) plt.show()

# 4. 3D visualization to show PC1, PC2, and PC3 together from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

plt.title('AFTER PCA: 3D Visualization of First Three Principal Components', fontsize=14)

for cluster\_id, cluster\_name in enumerate(target\_names): ax.scatter(

pca\_df[pca\_df['target'] == cluster\_id]['PC1'], pca\_df[pca\_df['target'] == cluster\_id]['PC2'], pca\_df[pca\_df['target'] == cluster\_id]['PC3'], label=cluster\_name,

alpha=0.7, s=50

)

ax.set\_xlabel(f'PC1 ({explained\_variance[0]:.2%})') ax.set\_ylabel(f'PC2 ({explained\_variance[1]:.2%})') ax.set\_zlabel(f'PC3 ({explained\_variance[2]:.2%})') plt.legend()

plt.tight\_layout() plt.show()

# 5. Examine PCA components/loadings plt.figure(figsize=(12, 8))

loadings = pca.components\_.T

loading\_df = pd.DataFrame(loadings, columns=[f'PC{i+1}' for i in range(loadings.shape[1])], index=feature\_names)

sns.heatmap(loading\_df, annot=True, cmap='coolwarm') plt.title('Feature Contributions to Principal Components (Loadings)') plt.tight\_layout()

plt.show()

# 6. Comparison of original feature space and PCA space (2D only for simplicity) plt.figure(figsize=(12, 5))

plt.suptitle('BEFORE vs AFTER PCA Comparison', fontsize=16, y=1.05)

# Before PCA: One representative plot (Feature\_1 vs Feature\_2) plt.subplot(1, 2, 1)

for cluster\_id, cluster\_name in enumerate(target\_names): plt.scatter(

df[df['target'] == cluster\_id]['Feature\_1'], df[df['target'] == cluster\_id]['Feature\_2'], label=cluster\_name,

alpha=0.7

)

plt.xlabel('Feature\_1') plt.ylabel('Feature\_2')

plt.title('BEFORE: Original Feature Space\n(Feature\_1 vs Feature\_2)') plt.legend()

# After PCA: PC1 vs PC2 (most variance explained) plt.subplot(1, 2, 2)

for cluster\_id, cluster\_name in enumerate(target\_names): plt.scatter(

pca\_df[pca\_df['target'] == cluster\_id]['PC1'], pca\_df[pca\_df['target'] == cluster\_id]['PC2'], label=cluster\_name,

alpha=0.7

)

plt.xlabel(f'PC1 ({explained\_variance[0]:.2%} variance)') plt.ylabel(f'PC2 ({explained\_variance[1]:.2%} variance)') plt.title('AFTER: Principal Component Space\n(PC1 vs PC2)') plt.legend()

plt.tight\_layout(rect=[0, 0, 1, 0.95]) plt.show()

# 7. Demonstrate dimensionality reduction by keeping only top components # Determine how many components we need to retain 95% variance n\_components\_95 = np.argmax(cumulative\_variance >= 0.95) + 1

print(f"\nNumber of components needed to retain 95% variance: {n\_components\_95} out of

{X.shape[1]}")

# Create a reduced PCA model

pca\_reduced = PCA(n\_components=n\_components\_95) X\_pca\_reduced = pca\_reduced.fit\_transform(X\_scaled)

# Reconstruct the data from reduced components

X\_reconstructed = pca\_reduced.inverse\_transform(X\_pca\_reduced) X\_reconstructed = scaler.inverse\_transform(X\_reconstructed) # Unscale the data

# Calculate reconstruction error

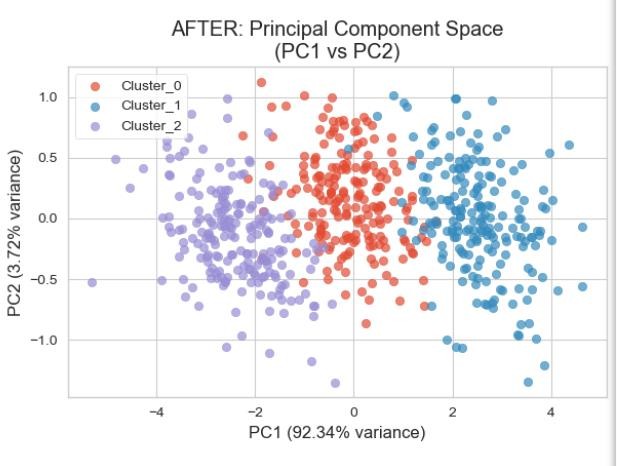
mse = np.mean((X - X\_reconstructed) \*\* 2) print(f"Mean Squared Reconstruction Error: {mse:.4f}")

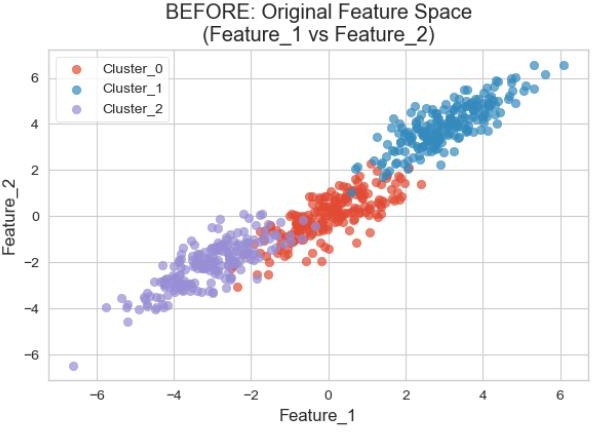
# 8. Create a scree plot to visualize explained variance plt.figure(figsize=(10, 6))

plt.plot(range(1, len(explained\_variance) + 1), explained\_variance, 'o-', linewidth=2, color='blue') plt.title('Scree Plot: Explained Variance by Principal Component')

plt.xlabel('Principal Component') plt.ylabel('Explained Variance Ratio') plt.xticks(range(1, len(explained\_variance) + 1)) plt.grid(True)

plt.tight\_layout() plt.show()





**CONCLUSION: -**

In summary, PCA helps in distilling complex data into its most informative elements, making it simpler and more efficient to analyze.

It identifies the directions (called principal components) where the data varies the most.

It projects the data onto these directions, reducing the number of dimensions while retaining as much information as possible.

The new set of uncorrelated variables (principal components) is easier to work with and can be used for tasks like regression, classification, or visualization.