National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Ma'am Neelma Naz **Date:** December 18, 2025

Semester: 7th **Group:** 02

CS471 Machine Learning

Lab 13: Principal Component Analysis

		PLO4	PLO5	PLO5	PLO8	PLO9
		CLO4	CLO5	CLO5	CLO6	CLO7
Student Name	**Reg. No**	**Viva / Quiz / Demo** 5 Marks	**Analysis of Data in Report** 5 Marks	**Modern Tool Usage** 5 Marks	**Ethics** 5 Marks	**Individual and Teamwork** 5 Marks
Hanzla Sajjad	403214					
Irfa Farooq	412564					

Introduction

This laboratory exercise will focus on the concept of dimensionality reduction of the features of a dataset. Principal Component Analysis (PCA) is a technique used to decrease the n features of a dataset into k features while retaining the majority of the information of the dataset. The decrease in the number of features allows faster training of machine learning implementations. Despite belonging to the techniques of unsupervised learning, dimensionality reduction is used in datasets pertaining to supervised learning programs.

Objectives

The following are the main objectives of this lab:

- Perform normalization of the dataset
- Acquire the covariance matrix using the features of the dataset
- Implement PCA in python using Eigen decomposition
- Plot the projections along the principal components
- Use PCA on dataset before implementing supervised learning

Lab Conduct

- Respect faculty and peers through speech and actions
- The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
- In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code/screenshot/plot between the #START and #END parts of these commented lines. Do NOT remove the commented lines.
- Use the tab key to provide the indentation in python.
- When you provide the code in the report, keep the font size at 12

Theory

Principal Component Analysis (or PCA) is a method that is used on a dataset to find a new set of dimensions that are linearly independent and ranked according to the variance of their data. PCA is implemented by normalizing the dataset features and using the result to get a covariance matrix which highlights the correlations of the features with each other. The Eigen vectors of the covariance matrix give a new representation of the features. By choosing a smaller number of vectors, it is possible to represent majority of the information of the dataset across a smaller set of features. Thus, PCA is a tool that is useful for compressing the dataset and hence saving costs on computation. PCA is also useful to help visualize the dataset when dealing with a very large number of features.

A brief summary of the relevant keywords and functions in python is provided below:

print()	output text on console
input()	get input from user on console
range()	create a sequence of numbers
len()	gives the number of characters in a string
if	contains code that executes depending on a logical condition
else	connects with **if** and **elif**, executes when conditions are not met
elif	equivalent to **else if**
while	loops code as long as a condition is true
for	loops code through a sequence of items in an iterable object
break	exit loop immediately
continue	jump to the next iteration of the loop
def	used to define a function
pd.read_csv	import csv file as a dataframe
df.to_csv	export dataframe as a csv file

For this lab, you will need to use different functions from NumPy. The following functions can be helpful:

```
np.dot
np.divide
np.mean
np.std
np.linalg.eig
np.linalg.svd
np.argsort
```

Lab Task 1 – Feature Normalization _____

Write a python program that loads a dataset of at least 8 features (X) and normalizes the features using the given equation:

$$x_j^{(i)} \leftarrow \frac{x_j{}^{(i)} - \mu_j}{s_j}$$

In the above equation μ_j is the mean of the j-th feature and s_j is the standard deviation of the j-th feature. The training example number is denoted by the superscript i. Provide the code and all relevant screenshots.

Code

```
# Important Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```python
# Task 1: Feature Normalization
# Using dataset for 8 features
from sklearn.datasets import load_diabetes
data = load_diabetes()

X = data.data        # Features
y = data.target     # Labels

print("Original Feature Shape:", X.shape)

# Computing mean and standard deviation
mu = np.mean(X, axis=0)
sigma = np.std(X, axis=0)

# Feature normalization function
def normalize_features(X, mu, sigma):
    return (X - mu) / sigma

# Normalized features
X_norm = normalize_features(X, mu, sigma)

print("First 5 normalized samples:\n", X_norm[:5])
```

Console Output

```
Original Feature Shape: (442, 10)
First 5 normalized samples:
 [[ 0.80050009  1.06548848  1.29708846  0.45984057 -0.92974581 -0.73206462
  -0.91245053 -0.05449919  0.41853093 -0.37098854]
 [-0.03956713 -0.93853666 -1.08218016 -0.55350458 -0.17762425 -0.40288615
   1.56441355 -0.83030083 -1.43658851 -1.93847913]
 [ 1.79330681  1.06548848  0.93453324 -0.1192138  -0.95867356 -0.71889748
  -0.68024452 -0.05449919  0.06015558 -0.54515416]
 [-1.87244107 -0.93853666 -0.24377122 -0.77064997  0.25629203  0.52539714
  -0.75764652  0.72130245  0.47698252 -0.19682291]
 [ 0.11317236 -0.93853666 -0.76494435  0.45984057  0.08272552  0.32789006
   0.17117751 -0.05449919 -0.67250161 -0.98056821]]
```

Lab Task 2 – The Covariance Matrix _____

In this task, you will use the normalized features to construct the covariance matrix that you will need to perform PCA in the next task. The covariance matrix can be achieved using two equivalent approaches:

$$X_{cov} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \qquad \text{Loop Method}$$

$$X_{cov} = \frac{1}{m} X^T . X \qquad \text{Matrix Method}$$

In the previous equations, $x^{(i)}$ is an n x 1 vector (n is the number of features), X is an m x n matrix (m is the number of training examples). In both equations, the normalized features are being used. Write a python program to get the covariance matrix using both methods and show that they are equivalent. Provide the code and all relevant screenshots.

Code

```
# Task 2: Covariance Matrix
m = X_norm.shape[0]

# Method 1: Loop Method
cov_loop = np.zeros((X_norm.shape[1], X_norm.shape[1]))

for i in range(m):
    x_i = X_norm[i].reshape(-1, 1)
    cov_loop += np.dot(x_i, x_i.T)

cov_loop = cov_loop / m

# Method 2: Matrix Method
cov_matrix = np.dot(X_norm.T, X_norm) / m
```

```
# Display comparison
print("Covariance Matrix (Loop Method):\n", cov_loop)
print("\nCovariance Matrix (Matrix Method):\n", cov_matrix)

# Check equivalence
print("\nAre both covariance matrices equal?",
      np.allclose(cov_loop, cov_matrix))
```

Console Output

```
Covariance Matrix (Loop Method):
 [[ 1.          0.1737371   0.18508467  0.33542759  0.26006082  0.21924314
  -0.07518097  0.2038409   0.27077424  0.30173101]
 [ 0.1737371   1.          0.0881614   0.24101049  0.03527682  0.14263726
  -0.37908963  0.33211509  0.14991614  0.20813322]
 [ 0.18508467  0.0881614   1.          0.3954109   0.24977742  0.26116991
  -0.36681098  0.4138066   0.44615654  0.38867999]
 [ 0.33542759  0.24101049  0.3954109   1.          0.24246402  0.18554846
  -0.17876163  0.25765005  0.39348011  0.39043002]
 [ 0.26006082  0.03527682  0.24977742  0.24246402  1.          0.89666296
   0.05151936  0.54220728  0.51550292  0.32571675]
 [ 0.21924314  0.14263726  0.26116991  0.18554846  0.89666296  1.
  -0.19645512  0.65981689  0.31835667  0.29060038]
 [-0.07518097 -0.37908963 -0.36681098 -0.17876163  0.05151936 -0.19645512
   1.         -0.73849273 -0.39857729 -0.2736973 ]
 [ 0.2038409   0.33211509  0.4138066   0.25765005  0.54220728  0.65981689
  -0.73849273  1.          0.61785897  0.41721211]
 [ 0.27077424  0.14991614  0.44615654  0.39348011  0.51550292  0.31835667
  -0.39857729  0.61785897  1.          0.46466885]
 [ 0.30173101  0.20813322  0.38867999  0.39043002  0.32571675  0.29060038
  -0.2736973   0.41721211  0.46466885  1.        ]]
```

```
Covariance Matrix (Matrix Method):
 [[ 1.          0.1737371   0.18508467  0.33542759  0.26006082  0.21924314
  -0.07518097  0.2038409   0.27077424  0.30173101]
 [ 0.1737371   1.          0.0881614   0.24101049  0.03527682  0.14263726
  -0.37908963  0.33211509  0.14991614  0.20813322]
 [ 0.18508467  0.0881614   1.          0.3954109   0.24977742  0.26116991
  -0.36681098  0.4138066   0.44615654  0.38867999]
 [ 0.33542759  0.24101049  0.3954109   1.          0.24246402  0.18554846
  -0.17876163  0.25765005  0.39348011  0.39043002]
 [ 0.26006082  0.03527682  0.24977742  0.24246402  1.          0.89666296
   0.05151936  0.54220728  0.51550292  0.32571675]
 [ 0.21924314  0.14263726  0.26116991  0.18554846  0.89666296  1.
  -0.19645512  0.65981689  0.31835667  0.29060038]
 [-0.07518097 -0.37908963 -0.36681098 -0.17876163  0.05151936 -0.19645512
   1.         -0.73849273 -0.39857729 -0.2736973 ]
 [ 0.2038409   0.33211509  0.4138066   0.25765005  0.54220728  0.65981689
  -0.73849273  1.          0.61785897  0.41721211]
 [ 0.27077424  0.14991614  0.44615654  0.39348011  0.51550292  0.31835667
  -0.39857729  0.61785897  1.          0.46466885]
 [ 0.30173101  0.20813322  0.38867999  0.39043002  0.32571675  0.29060038
  -0.2736973   0.41721211  0.46466885  1.        ]]

Are both covariance matrices equal? True
```

Lab Task 3 – Eigen Decomposition

In this task, you will write a python program to get the Eigen vectors of the covariance matrix X_{cov}. You may directly use the Eigen vector function (you may have to sort them in terms of decreasing Eigen value) or you can use Singular Value Decomposition function to get the U matrix (which contains left singular Eigen vectors of $X_{cov}.X_{cov}^T$). The U matrix contains the Eigen vectors. Next, reduce the U matrix to k columns (specify the value of k of your choice such that k < n) by discarding the columns from the right side of the matrix. Finally, use this reduced matrix to get the new features:

$$x_{new}^{i} = U_{reduced}{}^{T} . x^{(i)}$$

Provide the code and all relevant screenshots.

Code

```
# Task 3: Eigen Decomposition
eigen_values, eigen_vectors = np.linalg.eig(cov_matrix)

# Sorting eigenvalues in descending order
sorted_indices = np.argsort(eigen_values)[::-1]
eigen_values_sorted = eigen_values[sorted_indices]
eigen_vectors_sorted = eigen_vectors[:, sorted_indices]

print("Top 5 Eigenvalues:\n", eigen_values_sorted[:5])

# Selecting k principal components
k = 2
U_reduced = eigen_vectors_sorted[:, :k]

# Projecting data onto new feature space
Z = np.dot(X_norm, U_reduced)

print("Reduced Feature Shape:", Z.shape)
print("First 5 reduced samples:\n", Z[:5])
```

Console Output

```
Top 5 Eigenvalues:
 [4.02421075 1.49231968 1.20596626 0.9554764  0.66218139]
Reduced Feature Shape: (442, 2)
First 5 reduced samples:
 [[-0.58719913 -1.9468322 ]
 [ 2.83162538  1.37208173]
 [-0.27212855 -1.63490124]
 [-0.04928114  0.38227803]
 [ 0.75642136  0.81196025]]
```

Lab Task 4 – Projection Plotting _____

To verify the new features obtained in the PCA are correct, you will do 3-D to 2-D projection of a planar plot. You have been provided with a file that allows you to make a 3-D plot of a planar shape (n = 3). Use your PCA algorithm to reduce the dimensions to 2 (k = 2). Plot both the original (3-D) and the new (2-D) features. Check if you can obtain the projection of the planar shape. Provide the code and all relevant screenshots.

Code

```
# Lab Task 4 (PCA Projection) on CSV File
# Loading the CSV File
data_csv = pd.read_csv('lab13_planar_shape2.csv')
print("First 5 rows of the CSV data:")
print(data_csv.head())

# Extract features
X_csv = data_csv[['x', 'y', 'z']].values
print("\nShape of dataset:", X_csv.shape)

# Normalising Features
mu_csv = np.mean(X_csv, axis=0)
sigma_csv = np.std(X_csv, axis=0)
X_csv_norm = normalize_features(X_csv, mu_csv, sigma_csv)
print("\nFirst 5 normalized samples:\n", X_csv_norm[:5])

# Covariance Matrix
m_csv = X_csv_norm.shape[0]

# Loop Method
cov_loop_csv = np.zeros((X_csv_norm.shape[1], X_csv_norm.shape[1]))
for i in range(m_csv):
    x_i = X_csv_norm[i].reshape(-1, 1)
    cov_loop_csv += np.dot(x_i, x_i.T)
cov_loop_csv = cov_loop_csv / m_csv
```

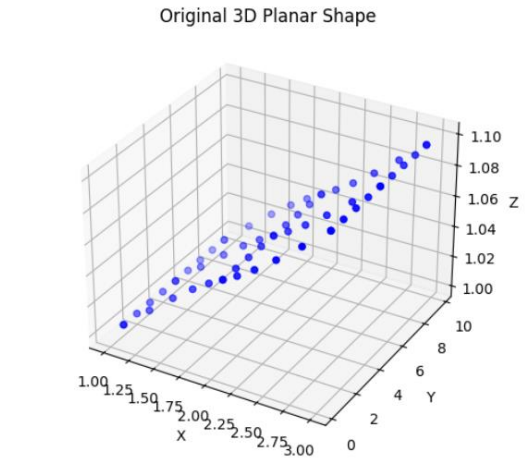
```
# Matrix Method
cov_matrix_csv = np.dot(X_csv_norm.T, X_csv_norm) / m_csv

print("\nCovariance Matrix (Loop Method):\n", cov_loop_csv)
print("\nCovariance Matrix (Matrix Method):\n", cov_matrix_csv)
print("\nAre both covariance matrices equal?", np.allclose(cov_loop_csv,
cov_matrix_csv))

# Eigen Decomposition
eigen_values_csv, eigen_vectors_csv = np.linalg.eig(cov_matrix_csv)
sorted_indices_csv = np.argsort(eigen_values_csv)[::-1]
eigen_vectors_sorted_csv = eigen_vectors_csv[:, sorted_indices_csv]

# Select top 2 principal components
k = 2
U_reduced_csv = eigen_vectors_sorted_csv[:, :k]

# Project data onto new 2D features
Z_csv = np.dot(X_csv_norm, U_reduced_csv)
print("\nReduced Feature Shape:", Z_csv.shape)
print("First 5 projected 2D samples:\n", Z_csv[:5])

# Plot Original 3D and 2D PCA Projection
fig = plt.figure(figsize=(12,5))

# 3D Original Data
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X_csv[:,0], X_csv[:,1], X_csv[:,2], c='blue', marker='o')
ax1.set_title("Original 3D Planar Shape")
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_zlabel("Z")

# 2D PCA Projection
ax2 = fig.add_subplot(122)
ax2.scatter(Z_csv[:,0], Z_csv[:,1], c='red', marker='o')
ax2.set_title("2D PCA Projection")
ax2.set_xlabel("Principal Component 1")
ax2.set_ylabel("Principal Component 2")
```

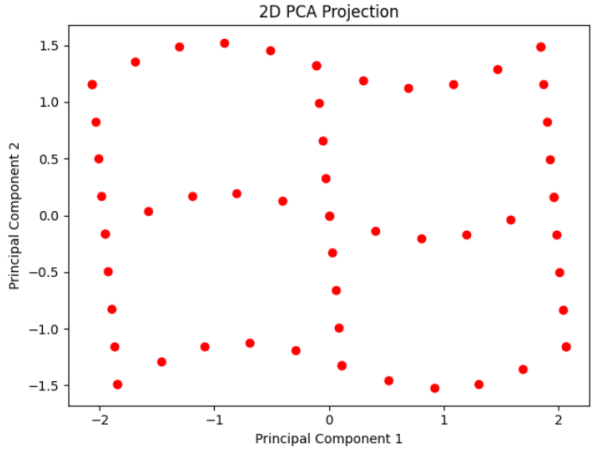
```
plt.tight_layout()
plt.show()
```

Console Output

```
First 5 rows of the CSV data:
     x    y    z
0  1.0  1.0  1.0
1  1.0  2.0  1.0
2  1.0  3.0  1.0
3  1.0  4.0  1.0
4  1.0  5.0  1.0

Shape of dataset: (60, 3)

First 5 normalized samples:
 [[-1.38675049 -1.33003696 -1.38675049]
 [-1.38675049 -0.99752772 -1.38675049]
 [-1.38675049 -0.66501848 -1.38675049]
 [-1.38675049 -0.33250924 -1.38675049]
 [-1.38675049  0.         -1.38675049]]
```

```
Covariance Matrix (Loop Method):
 [[ 1.         -0.05994396  1.        ]
 [-0.05994396  1.         -0.05994396]
 [ 1.         -0.05994396  1.        ]]

Covariance Matrix (Matrix Method):
 [[ 1.         -0.05994396  1.        ]
 [-0.05994396  1.         -0.05994396]
 [ 1.         -0.05994396  1.        ]]

Are both covariance matrices equal? True

Reduced Feature Shape: (60, 2)
First 5 projected 2D samples:
 [[-1.84269196 -1.48984513]
 [-1.87058161 -1.15850759]
 [-1.89847126 -0.82717006]
 [-1.92636091 -0.49583252]
 [-1.95425056 -0.16449499]]
```





Lab Task 5 – PCA for Supervised Learning _____

Download a dataset containing at least 8 features and 1 label column. Perform linear regression on the dataset (you may use Sci-kit Learn for this portion of the task). Next, use your PCA algorithm to reduce the number of features from n to k. Perform linear regression on the reduced dataset and make plots of training and validation losses. Make a comparison of the result between the original and reduced features.

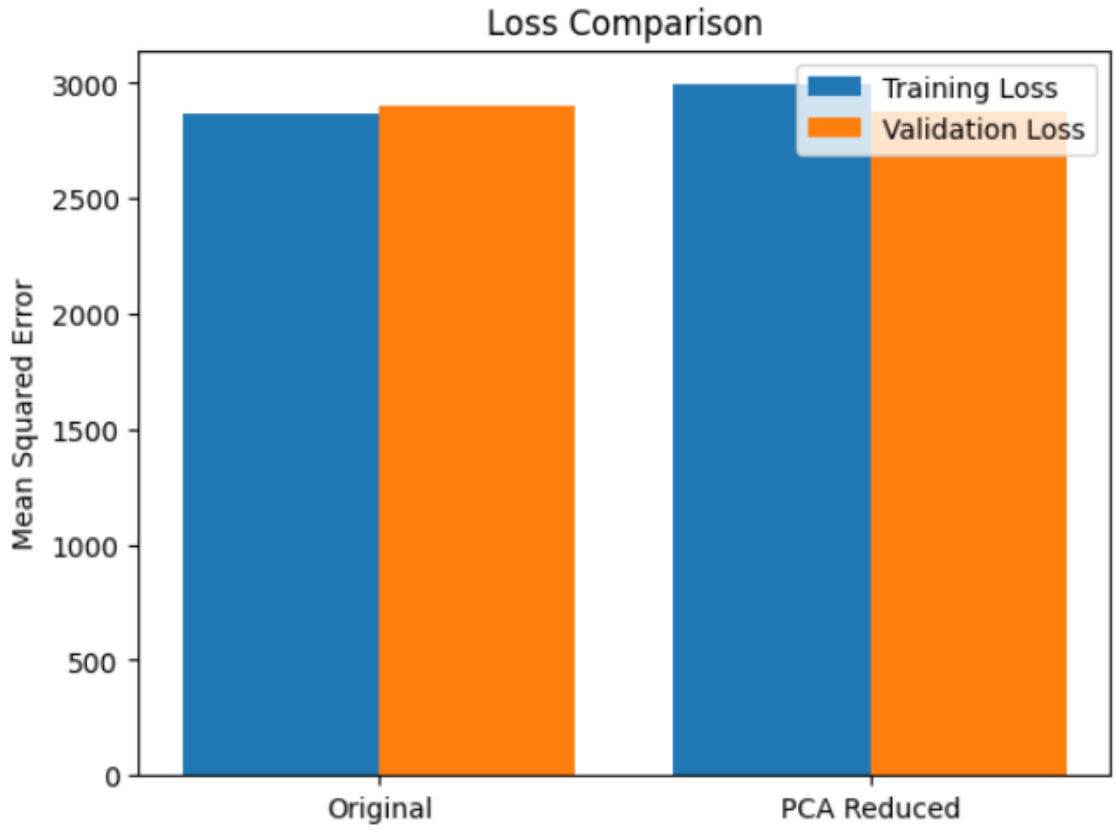
Code

```
# Task 5: PCA for Supervised Learning
# Restore y to the diabetes target as it was overwritten in previous
tasks.
y = data.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_norm, y, test_size=0.2, random_state=42)

# Linear Regression (Original Features)
lr_original = LinearRegression()
lr_original.fit(X_train, y_train)

y_train_pred = lr_original.predict(X_train)
y_test_pred = lr_original.predict(X_test)

train_loss_orig = mean_squared_error(y_train, y_train_pred)
test_loss_orig = mean_squared_error(y_test, y_test_pred)

print("Original Features Training Loss:", train_loss_orig)
print("Original Features Validation Loss:", test_loss_orig)

# PCA Reduction
k = 5
U_k = eigen_vectors_sorted[:, :k]
```

```
X_train_pca = np.dot(X_train, U_k)
X_test_pca = np.dot(X_test, U_k)

# Linear Regression (Reduced Features)
lr_pca = LinearRegression()
lr_pca.fit(X_train_pca, y_train)

y_train_pred_pca = lr_pca.predict(X_train_pca)
y_test_pred_pca = lr_pca.predict(X_test_pca)

train_loss_pca = mean_squared_error(y_train, y_train_pred_pca)
test_loss_pca = mean_squared_error(y_test, y_test_pred_pca)

print("\nPCA Features Training Loss:", train_loss_pca)
print("PCA Features Validation Loss:", test_loss_pca)

# Loss Comparison Plot
labels = ['Original', 'PCA Reduced']
train_losses = [train_loss_orig, train_loss_pca]
test_losses = [test_loss_orig, test_loss_pca]

x_axis = np.arange(len(labels))

plt.bar(x_axis - 0.2, train_losses, width=0.4, label='Training Loss')
plt.bar(x_axis + 0.2, test_losses, width=0.4, label='Validation Loss')

plt.xticks(x_axis, labels)
plt.ylabel("Mean Squared Error")
plt.title("Loss Comparison")
plt.legend()
plt.show()
```

Console Output

```
Original Features Training Loss: 2868.5497028355776
Original Features Validation Loss: 2900.193628493481

PCA Features Training Loss: 2992.7764973753974
PCA Features Validation Loss: 2879.592419510312
```



Code Explanation

In this task, the already loaded Diabetes dataset (10 numerical features and 1 label) was used to analyse the effect of Principal Component Analysis (PCA) on supervised learning performance using Linear Regression.

First, the normalized feature matrix was split into training and validation sets. A Linear Regression model was trained on the original feature space, and the Mean Squared Error (MSE) was calculated for both training and validation data. This established a baseline performance using all original features.

Next, PCA was applied using the eigenvectors obtained in previous tasks. The dimensionality of the dataset was reduced from n = 10 features to k principal components. The training and validation data were projected onto this reduced feature space, and Linear Regression was again trained and evaluated using MSE.