IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Comparative Analysis of PCA for Dimensionality Reduction in Diverse Datasets

PRABIN BOHARA[1], PRABIN SHARMA POUDEL[2], AND SANTOSH PANDEY.[3]

[1]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: prabinbohara10@gmail.com)
[2]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: prabinsharmapoudel@gmail.com)
[3]Institute of Engineering, Thapathali Campus, Kathmandu, Nepal (e-mail: suntoss.pandey@gmail.com)

* All Authors contributed equally to this work.

ABSTRACT High-dimensional datasets pose significant challenges in data analysis and machine learning tasks. With the increasing volume and complexity of data in various domains, deriving insights from such data is becoming challenging, mentally and computationally. Principal Component Analysis (PCA) enables feature extraction, data compression, and easier visualization, facilitating faster training and improving model performance. This report presents a comparative analysis of PCA applied to diverse datasets, including Random Numbers, Iris, and Covertype, with the aim of addressing these challenges. In our study, we applied PCA by standardizing the data, computing the covariance matrix, performing eigenvalue-eigenvector decomposition, and reducing the dimensionality. Our work showcases the versatility of PCA in addressing real-world challenges associated with high-dimensional data analysis and machine learning. Insights gained from this study can guide future applications of PCA for enhancing data understanding, modeling, and decision-making processes.

INDEX TERMS Dimensionality reduction, Eigendecomposition, Principal Component Analysis

I. INTRODUCTION

THE digital revolution has created an era of unprecedented data accumulation, and this immense data footprint presents a formidable challenge for data analysis. Principal Component Analysis (PCA) is a technique that addresses the complexities of high-dimensional data by transforming it into a lower-dimensional representation. By projecting the data onto principal components, PCA aims to capture the essential trends and patterns inherent in the dataset while minimizing the distance between the original data and its transformed representation. This reduction in complexity not only enables more efficient data analysis but also preserves a substantial amount of meaningful information and the underlying structure of the dataset. As a result, PCA has emerged as a powerful tool for navigating and extracting knowledge from the vast realms of data that the digital age has bestowed upon us.

In this report, we explore the application of PCA to three distinct datasets: Random Numbers following a Gaussian distribution, Iris Flower, and the Covertype identification. The primary objective is to investigate the effectiveness of PCA in addressing the challenges associated with high-dimensional data analysis and machine learning. By applying a standardized approach that includes data standardization, covariance matrix computation, eigenvalue-eigenvector decomposition, and dimensionality reduction, we aim to assess the impact of PCA on data compression, visualization, and model performance. In addition to our customized implementation of Principal Component Analysis (PCA), we also leverage the power of existing PCA libraries for comparative analysis. By employing PCA directly through the library, we aim to evaluate the consistency and performance of our results and explore the benefits of using established PCA implementations in terms of computational efficiency and ease of use.

II. METHODOLOGY

A. THEORY

The variance of data over multiple dimensions is reduced to few prominent dimensions with the application of PCA. This technique figure outs the high quality new components that captures the maximum notion of overall dataset. The number of newly generated components (k) is dependent upon the developer's configuration whose cumulative proportion of

variance is above the required threshold. If an instance of principal component is obtained, the new co-ordinate of dataset is obtained by projecting the original dataset to unit vector of a principal component.

Considering the original points of dataset is spreading over 2D plane (XY-plane) represented by 'x'. Let, the new dimension given by unit vector '**u**' captures the maximum variance of the original data. So, the new co-ordinates of points 'x' in Figure 1 can be found out by projecting it to the assumed principal component '**u**'.

So, the variance in original dataset with respect to X-axis and Y-axis is given by the formulas:

$$\text{Variance1} = \text{var}(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{1}$$

$$\text{var}(y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{2}$$

However, the relative variance between two or more dimensions gives the relation of variation amongst each other. The co-variance(cov) of dataset in X dimension with respect to Y dimension is given as:

$$\text{cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{3}$$

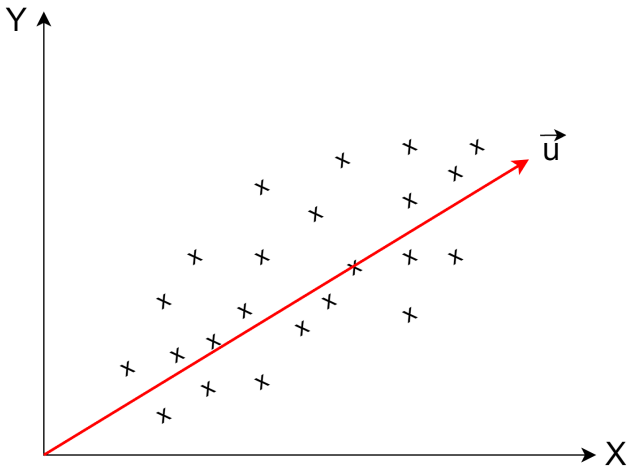


FIGURE 1: Distribution of original Dataset

Thus, the covariance matrix captures all the spread in own dimension and relation between others in remaining dimensions. So, from the relation above,

$$\text{var}(x) = \text{cov}(x, x) \tag{4}$$
$$\text{cov}(x, y) = \text{cov}(y, x) \tag{5}$$

$$\text{Covariance Matrix} = \begin{bmatrix} \text{var}(x) & \text{cov}(x, y) \\ \text{cov}(y, x) & \text{var}(y) \end{bmatrix}$$

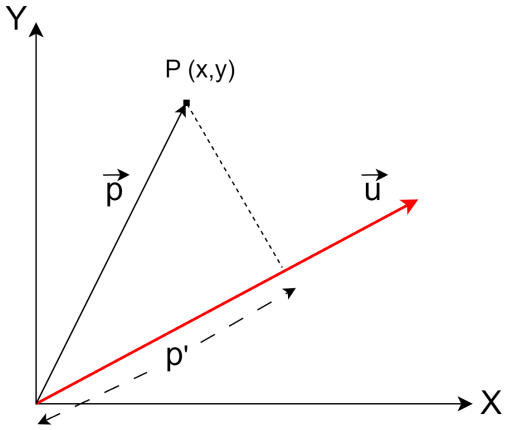


FIGURE 2: Projection of point(P) on assumed principal component

Taking P(x,y) as an arbitrary point in Figure 2, the projection of p over the principal component (u) is calculated as:

$$p' = \frac{u \cdot p}{\|u\|} = u \cdot p = u^T p \tag{6}$$

Now, the variance of data points with respect to new principal component (u) is given as:

$$\text{Variance2} = \frac{1}{n}\sum_{i}(u^T x_i - u^T x_m)^2 \tag{7}$$

Thus, the ultimate goal of PCA is to find out the unit vector (u) that maximizes the Variance2 which is an optimization problem. The principal unit vector (u) can be formulated by Rayleigh Quotient which happens to be an Eigenvector with the maximum Eigenvalue of the covariance matrix of the original dataset. This is because linear transformation by covariance matrix to the greatest Eigenvector results in the component that captures the maximum variance of original data. Thus, the Eigendecomposition by the application of transformation makes no linear changes rather than scaling by a factor called Eigenvalue (λ).

B. SYSTEM BLOCK DIAGRAM

The system architecture in Figure 3 shows various blocks in sequence. Diverse datasets were collected and data visualization techniques like scatter plots, histograms, and heatmaps were employed to gain insights into the data's distribution and relationships. Under preprocessing standardization or normalization was done to bring the data to a common scale, ensuring that different features contribute equally to the analysis. Afterward, covariance matrix computation was done to measure the linear relationships between pairs of features. Eigenvalues and eigenvectors of the covariance matrix were computed, so that the top k eigenvectors could be selected to find out the most significant directions of variation in the data. Both the selected eigenvectors and the training

data were arranged row-wise before employing the matrix multiplication between them to obtain the projected final value which could be visualized.

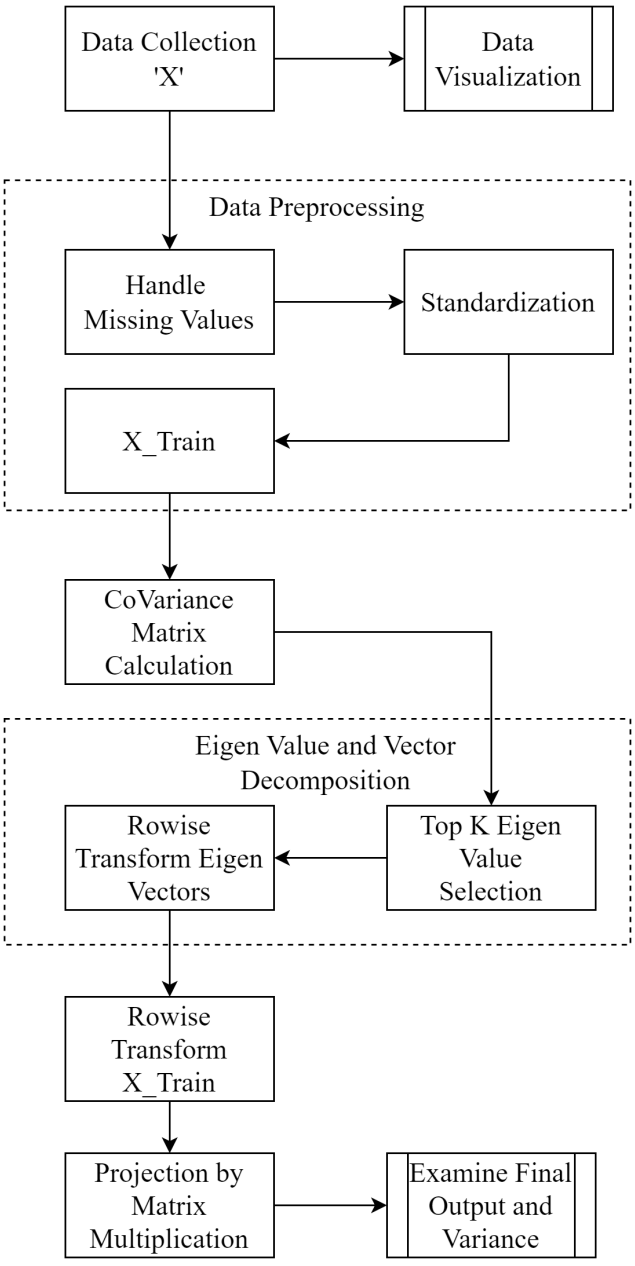


FIGURE 3: System Architecture of PCA

C. INSTRUMENTATION

The implementation of Principal Component Analysis (PCA) in our study involved the use of several libraries and tools to facilitate data visualization and mathematical operations.

1) Pandas

A powerful Python library used for data manipulation, Pandas provides data structures and functions which help us efficiently work with structured data, such as tabular data, and time series. The exact significance of Pandas is that it's commonly used for data preprocessing, cleaning, exploration, and analysis. It offers efficient data structures, like DataFrame that help us in easy indexing and manipulation of data. We used Pandas to load and read data from CSV files, such as the cancer dataset. It was used to inspect the data, extract features, and preprocess the dataset by handling missing values, encoding categorical variables, and scaling numerical data. For instance, we used label encoding for the target feature in the Cancer dataset, and the use of Pandas made the manipulation process convenient. Additionally, Pandas was used to select specific columns, filter rows based on conditions, and perform descriptive statistics on the data.

2) Matplotlib

Matplotlib is a data visualization tool, that helps us gain insights in visual form. It helps us gain insights, communicate findings, and present results. It offers flexible options for creating line plots, scatter plots, bar plots, histograms, heat maps, and more. We used Matplotlib to create scatter plots, box plots, histograms, and heat maps to visualize different aspects of the data and analyze patterns and relationships. Matplotlib's functions, such as scatter(), boxplot(), hist(), and imshow(), were utilized for these visualizations

3) NumPy

NumPy or Numerical Python provides a multidimensional array object and a collection of functions for performing mathematical operations on arrays efficiently. The general usefulness of NumPy can't be overstated as it offers high-performance array operations, such as mathematical, logical, shape manipulation, and linear algebra operations. We used NumPy to perform various array operations, such as element-wise operations, matrix multiplication, transposing matrices, and calculating eigenvalues and eigenvectors. NumPy's functions, like np.dot(), np.transpose(), and np.linalg.eig(), were used for these tasks. For the application of PCA in particular, np.linalg.eig() was a lot helpful since the eigen vectors primarily defined the direction of the components while eigenvalues gave the relative magnitude of the components. This mathematical insight is what PCA directly relies on, thus the use of NumPy made the implementation much easier for us.

By utilizing these libraries, we were able to seamlessly integrate data visualization and mathematical operations into our PCA implementation. This instrumentation allowed us to effectively analyse the datasets, visualize the results, and gain valuable insights from the dimensionality reduction process.

D. WORKING PRINCIPLE

The working principle of Principal Component Analysis (PCA) involves a pipeline of steps from the input data to the output representation. Here, we provide a detailed description of the PCA pipeline:

Initially, random data was sampled from the Gaussian distribution. Thus, the dataset obtained did not contain any categorical values. To visualize this set of values, a scatter plot is shown in Figure 4. Similarly, for the IRIS, three classes of Iris flowers were represented in the target variable and the dataset consisted of four features: sepal length, sepal width, petal length, and petal width. The initial data can be visualized using a pair plot, as shown in Figure 5, and using a scatterplot in Figure 6 . Likewise, the Cancer dataset consists of various features related to cancer tumors, such as radius, texture, perimeter, area, smoothness, and more. It can be visualized using Figure 7.

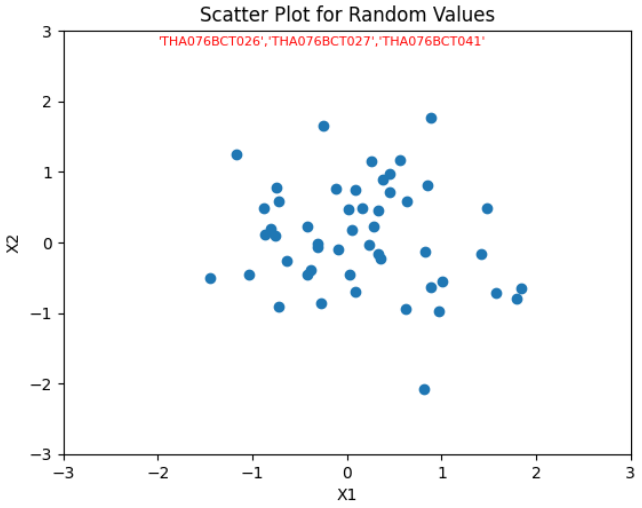


FIGURE 4: Scatter Plot for Random Values

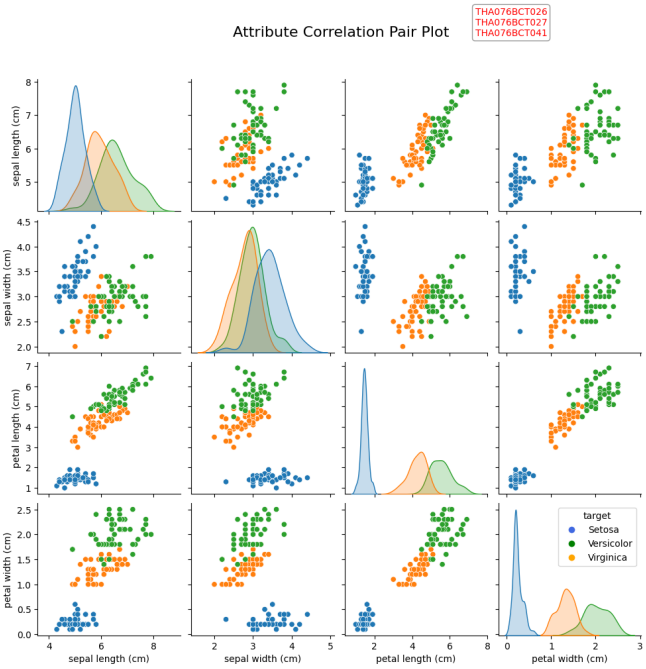


FIGURE 5: Pair Plot for Iris Dataset

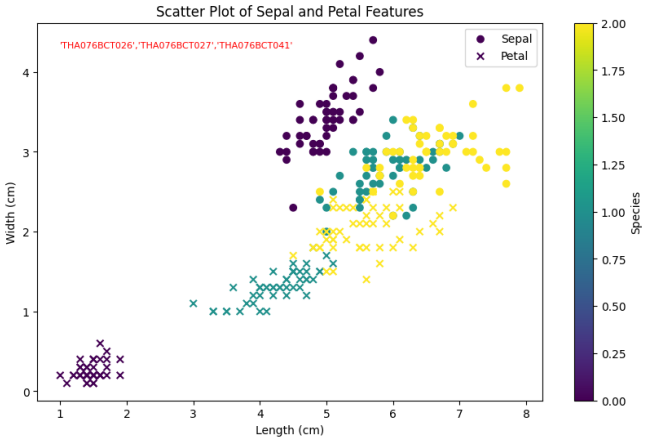


FIGURE 6: Sepal and Petal Features before Preprocessing

1) Data Pre-processing

The input dataset is pre-processed to ensure its suitability for PCA. This typically involves steps such as handling missing values, scaling the data, and removing any outliers. In our implementation, we applied data pre-processing techniques specific to each dataset, such as standardization or normalization. Encoding the categorical target variable was necessary to convert it into a numerical form. A technique like label encoding was used to represent the two classes (malignant and benign) in a numeric format. The processed dataset can be visualized using the given figures.

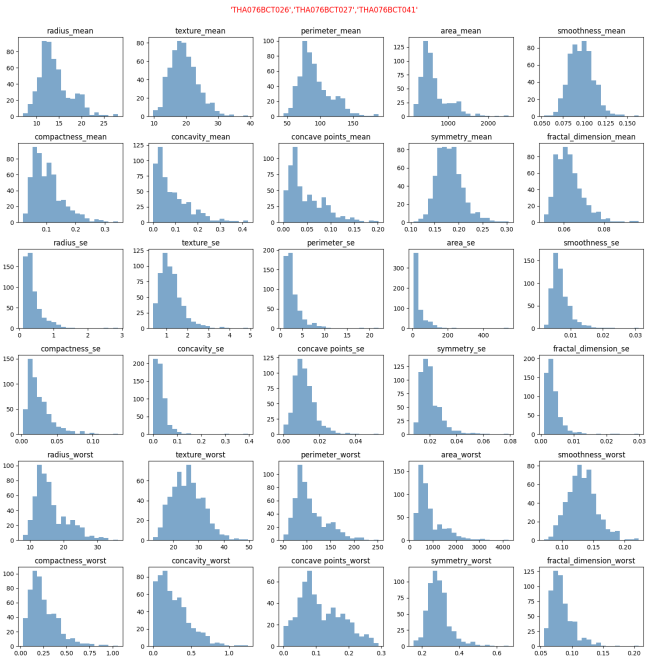


FIGURE 7: Histogram for each feature of Cancer Dataset

2) Covariance Matrix Computation

The covariance matrix is computed based on the pre-processed dataset. The covariance matrix captures the relationships between the variables and provides information about the variance and covariance of the data. This step involves calculating pairwise covariances between the features in the dataset. The covariance matrix can be used to identify highly correlated features.

3) Eigenvalue-Eigenvector Decomposition

The covariance matrix is then decomposed into its eigenvalues and eigenvectors. This step is performed using numerical methods such as the eigenvalue decomposition provided by libraries like NumPy. The eigenvalues represent the variance explained by each eigenvector, while the eigenvectors define the principal components.

4) Selection of Principal Components

The principal components are selected based on the eigenvalues. The eigenvectors with higher eigenvalues capture more variance in the data and are chosen as the principal components. The number of principal components to retain can be determined by analysing the cumulative proportion of variance explained

5) Dimensionality Reduction

The dimensionality of the data is reduced by projecting it onto the selected principal components. The original data is transformed into a lower-dimensional space by multiplying it with the matrix of selected eigenvectors.

6) Output Representation

The transformed data, represented in terms of the principal components, serves as the output of the PCA process. This lower-dimensional representation retains the most important information from the original data while reducing the complexity and dimensionality.

By following this pipeline, PCA enables the extraction of meaningful features, data compression, and easier visualization. The reduced-dimensional representation obtained through PCA can be used for various tasks such as data exploration, clustering, classification, or regression, facilitating more efficient and accurate data analysis and model training.

III. RESULTS

A. RANDOM NUMBERS DATASET

The dataset comprises 20 random samples drawn from a Gaussian distribution, which are visually scattered throughout Figure 4. Before extracting information from these samples, it is necessary to perform PCA analysis. Since the samples follow a Gaussian distribution, they are transformed into a normal distribution by multiplying them with a 2D matrix. This transformation aligns the random samples in a single direction, as depicted in Figure 8.

After computing the eigenvalues and eigenvectors, the principal component (PC1) is determined and visualized in

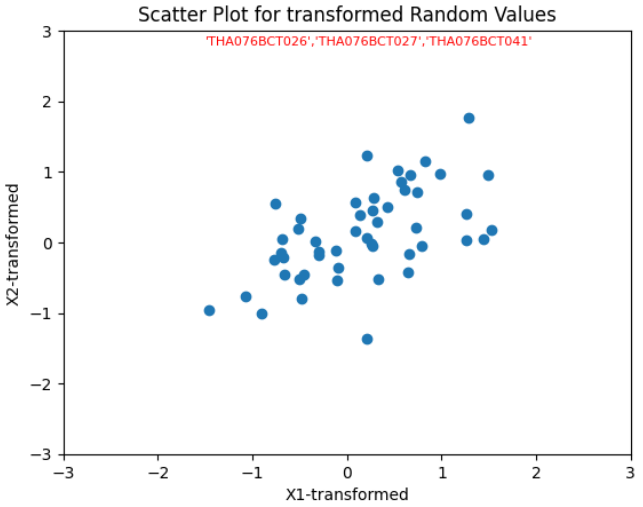


FIGURE 8: Scatter Plot for Transformed Random Values

Figure 9. By selecting only one principal component, all the data points become aligned in a singular direction.

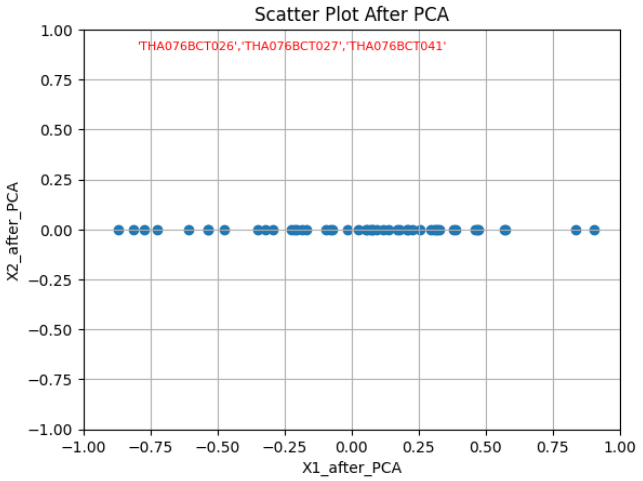


FIGURE 9: Scatter Plot After PCA on Random Values

B. IRIS DATASET

The Iris dataset imported from scikit-learn's library is widely used in machine learning and statistics. It includes measurements of four features from three species of Iris flowers: Setosa, Versicolor, and Virginica. The goal is to classify the flowers based on these features. The dataset has 150 samples, with 50 for each species. It is popular for beginners to learn classification algorithms due to its manageable size and well-structured nature.

Principal Component Analysis (PCA) was applied to extract the most informative components, and scatter plots visualized their directions and magnitudes. Covariance matrices were also examined for different PCA combinations. PCA was further compared using scikit-learn, revealing that both

explicitly specifying two principal components and retaining 95% of the information required the same number of components. Two models were trained for the PCA from the library, 1 model was trained for the PCA from scratch and 1 model was trained without applying PCA so that we could compare the performance of the model. It was observed that the respective model accuracies were 0.93, 0.93, 0.96, and 0.93.

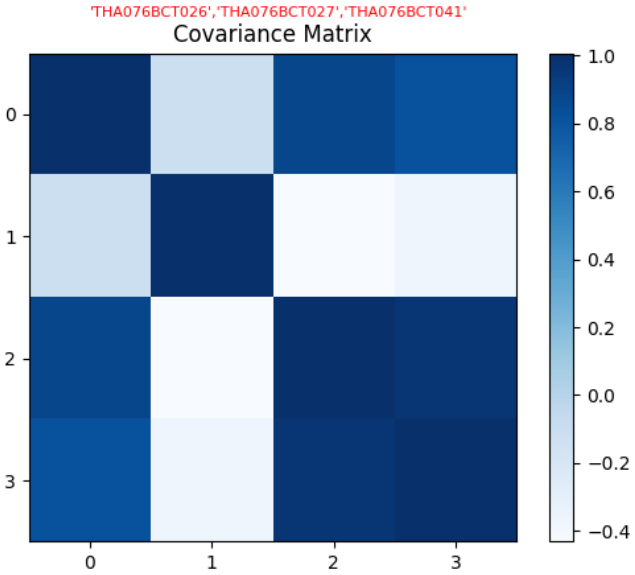


FIGURE 10: Covariance Matrix of Iris Dataset

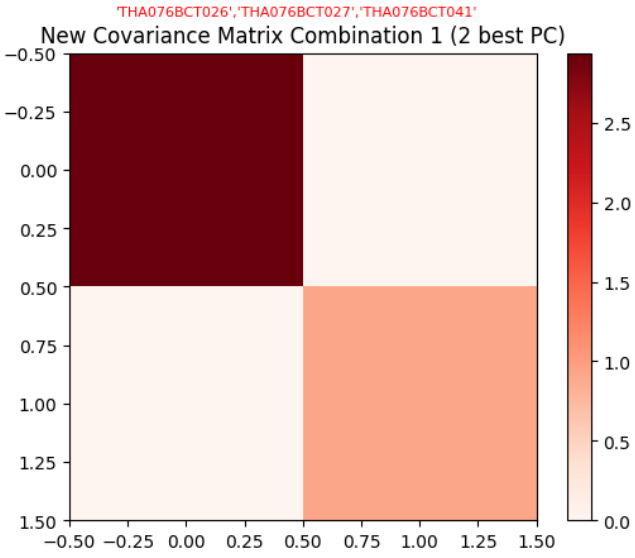


FIGURE 11: Covariance Matrix for two high variance components for Iris Dataset

C. CANCER DATA

The Cancer data was downloaded from Kaggle public dataset that contains data of 570 cancer cells with 30 features to determine whether the cancer cells data are benign or malignant.

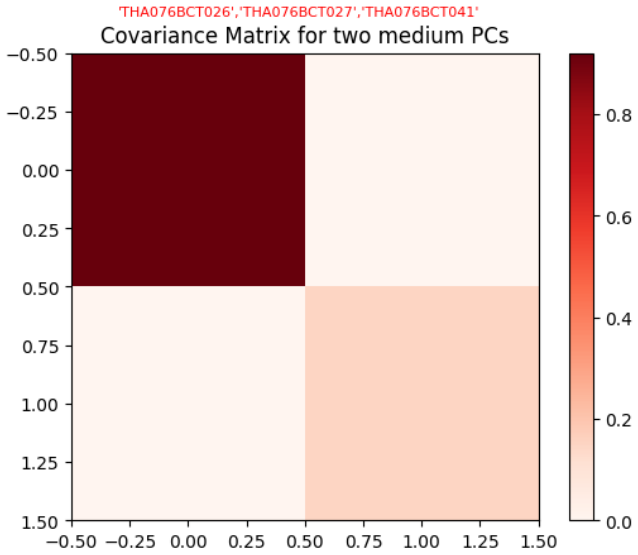


FIGURE 12: Covariance Matrix for two medium variance components for Iris Dataset

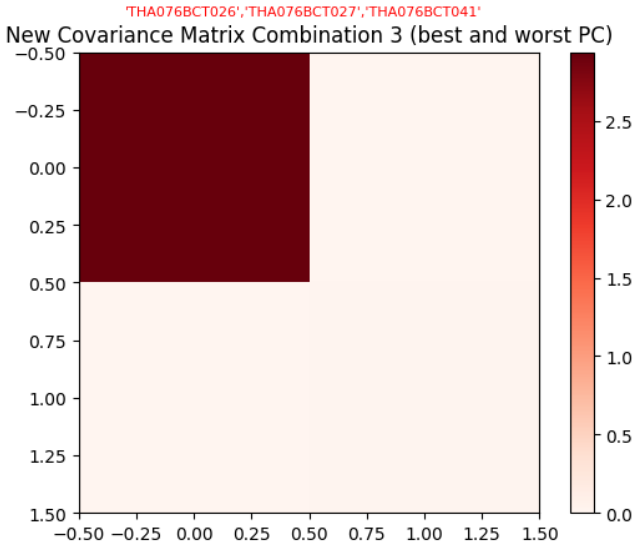


FIGURE 13: Covariance Matrix for one high and one low variance components for Iris Dataset

The Principal Component Analysis (PCA) was done on the dataset and Scree Plot was plotted and the plot highlights the valuable insights into the explained variance by each principal component. By examining the scree plot, we can identify the principal components that contribute the most to the overall variance. Therefore, by analyzing the scree plot, we determined the optimal number of principal components to retain for further analysis. This decision is crucial as it balances the trade-off between capturing sufficient variance and minimizing the dimensionality of the dataset.

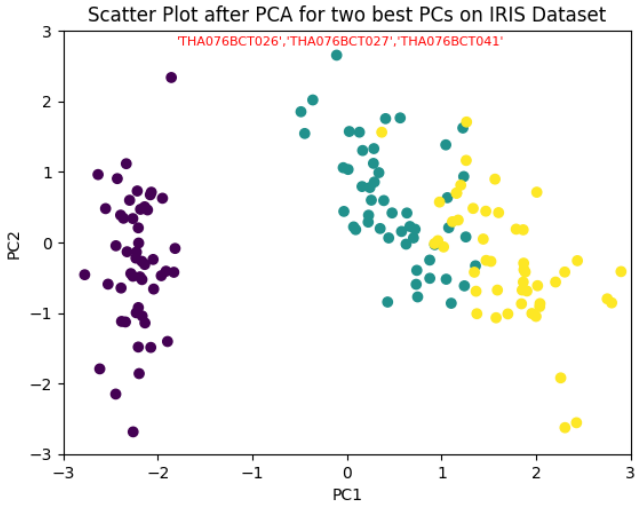


FIGURE 14: Scatter Plot after PCA for two high variance components for Iris Dataset

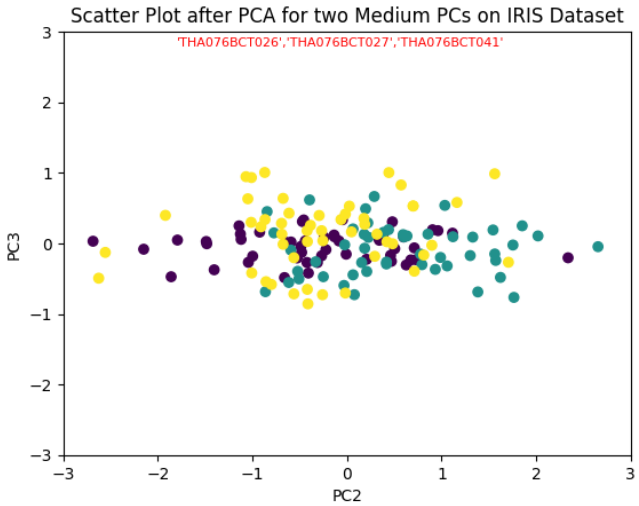


FIGURE 15: Scatter Plot after PCA for two medium variance components for Iris Dataset

The covariance in the processed data of cancer dataset is shown in the Figure 20 and the scree plot can be seen in the Figure 21.

Likewise, PCA was applied from the library to visualize the difference. Again, two instances were created for a similar motivation, that is one by explicitly specifying two components and another by specifying a 0.95 retention value. This time it was observed that to retain 95% of the information, 10 principal components were required.

IV. DISCUSSION AND ANALYSIS

A. RANDOM NUMBERS DATASET

The data was standardized by multiplying it with a 2x2 random uniform matrix, where elements were sampled from a uniform distribution. This transformation resulted in aligning

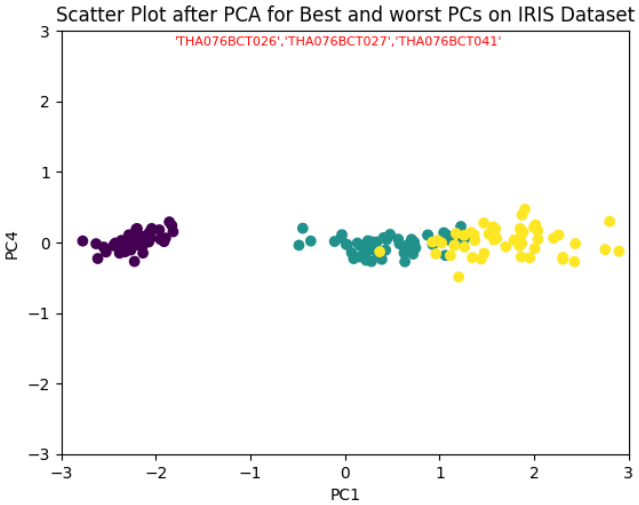


FIGURE 16: Scatter Plot after PCA for one high and one low variance components for Iris Dataset

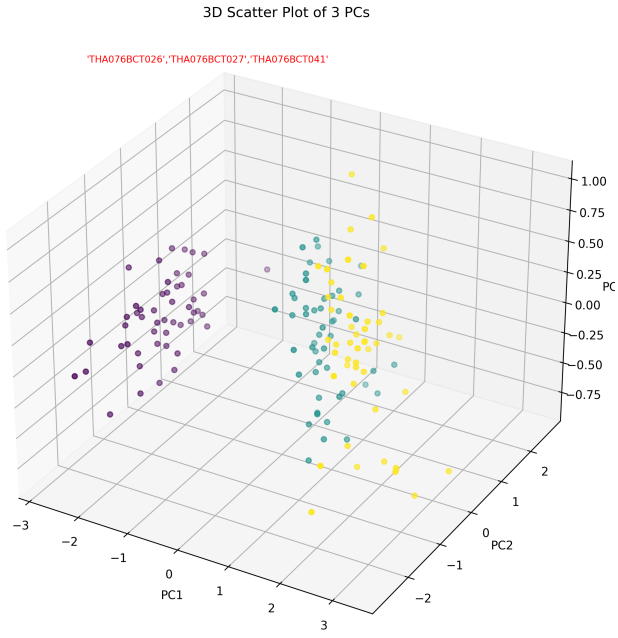


FIGURE 17: 3D Scatter Plot of 3 PCs for Iris Dataset

the random samples along the principal axes of a normal distribution. The purpose of this alignment was to simplify the subsequent PCA analysis by orienting the data in a specific direction. Applying PCA to the dataset of 20 random samples from a Gaussian distribution with 2 features has allowed us to transform the data and visualize it in a lower-dimensional space. The principal components obtained through PCA capture the major sources of variance in the data, providing a concise representation for further analysis.

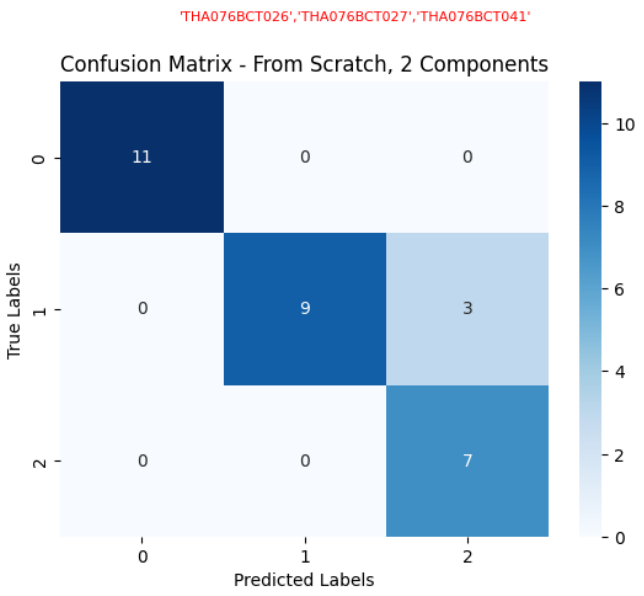


FIGURE 18: Confusion Matrix for the model after applying PCA from Scratch for Iris

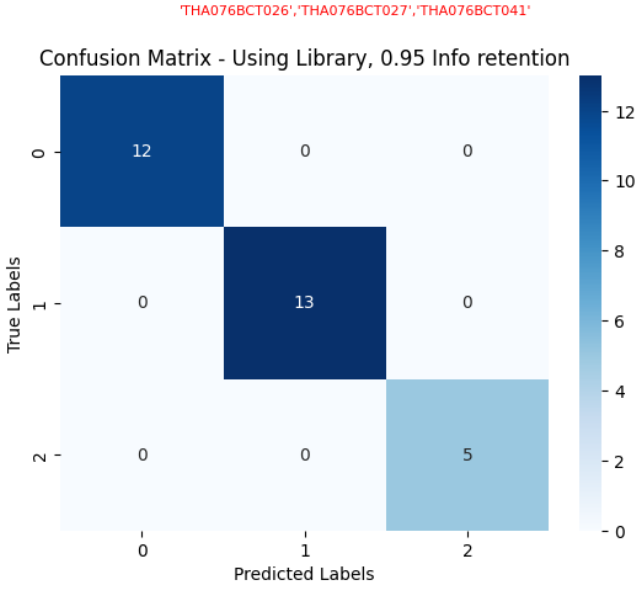


FIGURE 19: Confusion Matrix for the model after applying PCA using Library for Iris

B. IRIS DATASET

During the Principal Component Analysis on the IRIS dataset, it was observed that using principal components beyond the first and second ones in the PCA analysis of the Iris dataset resulted in a mixed-up or differently clustered pattern. This is due to limitations of the third and fourth components (described as medium PCs) in capturing significant variations. The first and second components were more effective in representing the dataset's structure and differentiating between Iris species. Lower-order components show

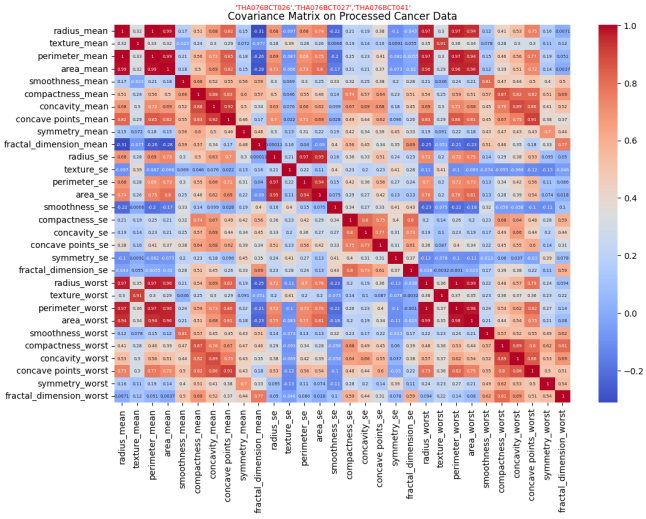


FIGURE 20: Covariance Matrix on Preprocessed Cancer Data

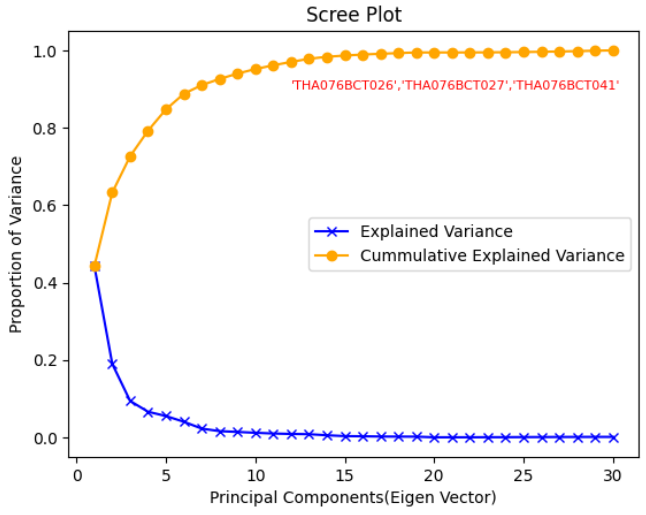


FIGURE 21: Scree Plot of Variance for Cancer Data

clear separations, while higher-order ones may be influenced by noise or irrelevant features. Careful selection of principal components is crucial for extracting meaningful information from the dataset.

C. CANCER DATA

In the PCA analysis of the Cancer dataset, the Scree plot was utilized to examine the variance explained by each principal component. Higher eigenvalues in the plot indicate a larger amount of captured variance. By analysing the Scree plot, we can identify the principal components that contribute the most to the overall variance. Usually, there is a significant drop in eigenvalues after a certain number of components, indicating that the additional components provide diminishing returns in terms of explained variance. Hence it indicated that the first few principal components tend to

be the most informative. Analyzing the Scree plot assists in determining the optimal number of components to retain, striking a balance between capturing sufficient variance and reducing dimensionality. This process is vital for accurately distinguishing between the two types of cancer cells based on their features.

V. CONCLUSION

In conclusion, this report presented a comparative analysis of Principal Component Analysis (PCA) applied to diverse datasets, including Random Numbers, Iris, and Cancer datasets. The main objectives were to explore the capabilities of PCA in addressing real-world challenges associated with high-dimensional data analysis and machine learning. Through our analysis, we successfully applied PCA to each dataset, following a standardized pipeline that involved data pre-processing, covariance matrix computation, eigenvalue-eigenvector decomposition, selection of principal components, and dimensionality reduction. By leveraging the power of PCA, we were able to reduce the dimensionality of the datasets while preserving a significant amount of meaningful information and the underlying structure. The results demonstrated the effectiveness of PCA in achieving the objectives of the lab. Additionally, PCA provided a valuable means of data compression, allowing us to represent the data in a lower-dimensional space without significant loss of information.

Furthermore, PCA greatly enhanced the visualization of the datasets by projecting them onto the principal components. We observed that PCA enabled us to extract the most important features from the datasets, leading to improved data understanding and facilitating more efficient model training, and supporting better decision-making processes. Overall, the objectives of the lab were successfully met through the application of PCA to the datasets. The comparative analysis highlighted the versatility and usefulness of PCA in handling high-dimensional data, deriving meaningful insights, and enhancing data analysis and machine learning tasks. The findings from this study can guide future applications of PCA in various domains and contribute to the advancement of data analysis techniques.

REFERENCES

[1] Abdi, H., & Williams, L. J. (2010). Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4), 433-459. doi:10.1002/wics.101.

[2] I. T. Jolliffe, "Principal component analysis and factor analysis," Principal Component Analysis, pp. 115-128, 1986.

[3] K. Pearson, "On lines and planes of closest fit to systems of points in space," Philosophical Magazine, vol. 2, no. 11, pp. 559-572, 1901.

[4] H. Abdi and L. J. Williams, "Principal component analysis," Wiley Interdisciplinary Reviews: Computational Statistics, vol. 2, no. 4, pp. 433-459, 2010.

[5] M. Ringnér, "What is principal component analysis?," Nature Biotechnology, vol. 26, no. 3, pp. 303-304, 2008.

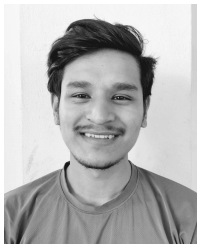
[6] scikit-learn developers, "sklearn.decomposition.PCA," scikit-learn documentation, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[7] J. Brownlee, "How to Use Principal Component Analysis for Feature Selection," Machine Learning Mastery, [Online]. Available: https://machinelearningmastery.com/principal-components-analysis-for-feature-selection-in-python/.

[8] D. Pattnaik, "Principal Component Analysis (PCA) - Step by Step Tutorial," Towards Data Science, [Online]. Available: https://towardsdatascience.com/principal-component-analysis-pca-step-by-step-tutorial-81b6387c55c1.

[9] A. Dey, "PCA — Machine Learning Mystery," Machine Learning Mystery, [Online]. Available: https://machinelearningmastery.com/pca-machine-learning-mystery/.

[10] A. Géron, "Principal Component Analysis (PCA) Explained," Medium, [Online]. Available: https://towardsdatascience.com/principal-component-analysis-pca-explained-and-implemented-eeab7cb73b72.



PRABIN BOHARA is an enthusiastic individual currently pursuing a Bachelor's degree in Computer Engineering, set to graduate in 2024. He has a genuine passion for coding, web technologies, and machine learning. Prabin's primary focus revolves around AI research and exploring the various applications of data science in general. He actively contributes to open-source projects and continuously expands his knowledge through academic endeavors. Prabin actively participates in webinars, both nationally and internationally, as well as competing in diverse technology competitions. He also shares his expertise by conducting workshops on various technology topics. With a strong drive for excellence, Prabin aims to make valuable contributions in the field of AI while keeping up with the latest advancements.

PRABIN SHARMA POUDEL , an undergraduate senior at IOE's Thapathali Campus, has forever been in pursuit of knowledge and wisdom. His involvement extends beyond technical confines, with involvement in non-technical organizations like AYON. With a vision to become a scholarly figure, he aims to bridge multiple disciplines of knowledge. His current field of interest encompasses AI in conflict resolution, Computer vision, and AI in creative industries Beyond academics, Prabin finds joy in playing chess, composing music, and journaling his thoughts through poetry.

SANTOSH PANDEY is a fourth-year Bachelor's student at IOE, Thapathali Campus, with a passion for coding and technology. Currently working as an intern at NAAMII organization, he is driven by the goal of applying his skills to earn a living through coding. With a keen interest in full-stack development, graphic designing, mobile app development, and computer vision, Santosh is constantly exploring new technologies and honing his skills in these areas. In his free time, he actively participates in coding competitions and enjoys participating in marathons. Santosh's dedication, enthusiasm, and continuous learning make him a promising professional in the field of technology and coding.

CODE

A. RANDOM NUMBERS DATASET

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import random

# Generates and visualizes a scatter plot of two sets of 50 random data points each. [randn=from normal distribution.]
# x1 and x2 represent the 2 axes
x1 = np.random.randn(50)
x2 = np.random.randn(50)
print(x1)
print(x2)
plt.scatter(x1,x2)
plt.xlabel("X1")
plt.ylabel("X2")
plt.text(-2, 2.8, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.title('Scatter Plot for Random Values')
plt.ylim(-3,3)
plt.xlim(-3,3)
plt.savefig('Scatter Plot for Random Values', bbox_inches='tight')

# Mean and standard deviation
mean1 = np.mean(x1)
mean2 = np.mean(x2)
sd1 = np.std(x1)
sd2 = np.std(x2)
print(mean1,sd1)
print(mean2,sd2)

# 2x2 random matrix with elements sampled from a uniform distribution (range [0, 1]).
random_matrix = np.random.rand(2,2)
print(random_matrix)

# Converts the lists x1 and x2 into NumPy arrays, then creates a data matrix by column stacking the arrays.
x1_array = np.array(x1)
x2_array = np.array(x2)
data_matrix = np.column_stack((x1_array, x2_array))
print(data_matrix.shape)
print(data_matrix)


# **STEP 1: Reshape the data by multiplication with a random matrix .** To visualize the linear transformation, click on the link : https://www.geogebra.org/m/YCZa8TAH?fbclid=IwAR3LK2VOQooxb-qzH-LLhlyzVEhi8Qsb8Of_V1zeiqWb3FcjYibVPbL0NGo

transformed_data_by_random_matrix = data_matrix @ random_matrix
print(transformed_data_by_random_matrix.shape)

# first column [:,0] as x-axis value. similarly 2nd column as y-axis
plt.scatter(
    transformed_data_by_random_matrix[:,0],
    transformed_data_by_random_matrix[:,1])
plt.xlabel("X1-transformed")
plt.ylabel("X2-transformed")
plt.ylim(-3,3)
plt.xlim(-3,3)
plt.text(-1.5, 2.8, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.title('Scatter Plot for transformed Random Values')
plt.savefig('transformed Random Values', bbox_inches='tight')


# **STEP 2: Calculate covariance matrix** The covariance matrix provides insights into the relationships between different dimensions of the transformed_data_by_random_matrix.
# See if elements in main diagonal are max and other diagonal are min. If not, we have to proceed further .
covariance_matrix = np.cov(transformed_data_by_random_matrix.T)
covariance_matrix
#same as above
covariance_matrix = np.cov(np.transpose(transformed_data_by_random_matrix))
print(covariance_matrix)


# **STEP 3: Eigen values and vector calculation**
# The eigenvalues represent the variance explained by each principal component (magnitude of new feature space)
# The eigenvectors define the directions of the principal components. ( direction of new feature space)
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
print(eigen_values)
print(eigen_vectors)

# Just To check if the eigenvalues really represent the variance, lets find out the proportion of variance for each components
proportion1 = eigen_values[0] / np.sum(eigen_values)
print(proportion1*100)
proportion2 = eigen_values[1] / np.sum(eigen_values)
print(proportion2*100)


# **STEP 4: Transform the data with the selected eigen vector.**
#first write the eigenvectors row wise ( do the transpose, from column wise to row wise)
```

```
arranged_eigen_vector = np.transpose(
    eigen_vectors)
print(arranged_eigen_vector.shape)
print(arranged_eigen_vector)

#also transpose the (50,2) data matrix to
     (2,50)
transposed_data_matrix =
    transformed_data_by_random_matrix.T
print(transposed_data_matrix.shape)

#Project data into first principal
    component (by matrix multiplication)
new_data = np.dot(arranged_eigen_vector,
    transposed_data_matrix)
print(new_data.shape)
#print(new_data)

#transpose back to (50,2)
trans_new_data=new_data.T
print(trans_new_data.shape)

# again see if the elements in main
    diagonal are max
covariance_of_new_data = np.cov(new_data)
print(covariance_of_new_data)

#lets select the best eigen vectors as
    specified in the step 4
best_eigen_vector = np.transpose(
    eigen_vectors[:,0])
print(best_eigen_vector.shape)
print(best_eigen_vector)
new_new_data = np.dot(best_eigen_vector,
    transposed_data_matrix)
print(new_new_data.shape)
print(new_new_data)


# **STEP 6: plot the new data**
plt.scatter(new_data[0,:], new_data[1,:]
    )
plt.ylim(-3,3)
zeros = np.zeros(50)
plt.scatter(new_data[1,:],zeros)
plt.xlabel("X1_after_PCA")
plt.ylabel("X2_after_PCA")
plt.title('Scatter Plot After PCA')
plt.text(-0.8, 0.9, "'THA076BCT026','
    THA076BCT027','THA076BCT041'",
    fontsize=8,color='red')
plt.ylim(-1,1)
plt.xlim(-1,1)
plt.grid()
plt.savefig('scatter plot after PCA',
    bbox_inches='tight')

print(new_data.shape)
print(new_new_data)
```

B. IRIS DATASET

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from numpy import random
import sklearn


# **STEP 1: Load the iris flower dataset
    **

from sklearn.datasets import load_iris
iris = load_iris()

X = iris.data    # X is 2d array, where
    row=samples, column=features
y = iris.target  # y is 1d array,
    representing class label for all
    samples
print(X.shape)
print(y.shape)
print(iris.feature_names)
print("Target names:", iris.target_names)
print(X)

# **STEP 2: Create a dataframe and see
    the data statistics**

# assign meaningful columns(feature_names
    ) to feature values for 150 samples(
    iris.data) in the dataframe.
df = pd.DataFrame(data=iris.data,columns=
    iris.feature_names)
# add a new target column.
df['target']=iris.target
df.head(2)   # view 1st 2 data, both are
    setosa(numbering starts from 0)

#view 49th to 52nd dataset to see 2
    classes(setosa and vesicolor)
subset = df.iloc[48:52]
print(subset)

# statistics computed for each column
summary_stats = df.describe()
print(summary_stats)

# data visualization, box plot for each
    feature
df.plot(kind='box', figsize=(10, 6))
plt.title('Box Plot of Iris Dataset')
plt.text(1.5, 8, "'THA076BCT026','
    THA076BCT027','THA076BCT041'",
    fontsize=8,color='red')

plt.savefig('Box Plot of Iris Dataset',
    bbox_inches='tight')
plt.show()

import seaborn as sns
# Set the size of the figure
plt.figure(figsize=(12, 7))
df['target'] = df['target'].replace({0: '
    Setosa', 1: 'Versicolor', 2: '
    Virginica'})
# Create a pair plot with hue based on
    the 'target' column in the dataframe
g=sns.pairplot(df, hue='target',plot_kws
    ={'legend': False})
g._legend.set_title('')
# # Set the title of the plot
plt.suptitle('Attribute Correlation Pair
    Plot', fontsize=16, y=0.98)
```

```
# Set the style and color of the text box
textbox_props = dict(boxstyle='round', facecolor='white', edgecolor='gray', alpha=0.8)

# Add a text box with the highlighted feature names
plt.text(0.1, 8, "THA076BCT026\nTHA076BCT027\nTHA076BCT041", fontsize=10, color='red', ha='center', bbox=textbox_props)
legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='Setosa', markerfacecolor='royalblue', markersize=8),
                   plt.Line2D([0], [0], marker='o', color='w', label='Versicolor', markerfacecolor='g', markersize=8),
                   plt.Line2D([0], [0], marker='o', color='w', label='Virginica', markerfacecolor='orange', markersize=8)]

# Add the legend to the plot
plt.legend(handles=legend_elements, title='target')

# Adjust the spacing between subplots
plt.subplots_adjust(top=0.9)
# Set custom tick labels for the legend
# Save the plot to a file
plt.savefig('./plots/pair_plot.png', bbox_inches='tight')

# Display the plot
plt.show()

# Visualization of data points based on the target class (purple=setosa,green=Versicolour,yellow=Virginica )
df['target']=iris.target
plt.figure(figsize=(10, 6))
plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'], c=df['target'], label='Sepal')
plt.scatter(df['petal length (cm)'], df['petal width (cm)'], c=df['target'], marker='x', label='Petal')
plt.xlabel('Length (cm)')
plt.ylabel('Width (cm)')
plt.title('Scatter Plot of Sepal and Petal Features')
plt.colorbar(label='Species')
plt.text(1, 4.3, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.legend()
plt.savefig('./plots/Sepal and Petal Features', bbox_inches='tight')
plt.show()
```

```
# **STEP 3: Data preprocessing**

# Handling missing values:
# df.isna() returns a boolean, where True means missing values(NaN) and .sum() sums the values along each column
missing_values = df.isna().sum()
print("Columns with missing values:")
print(missing_values[missing_values > 0])

# Standardizing to eliminate the chance of PCA being influenced by magnitude of some features' values.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = df.iloc[:,1:].values
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)

# IMPORTANT STEP!
# Create new input and output after preprocessing X and y.
X_train = df_scaled.drop('target', axis=1).values
y_train = df['target']
print("shape of new input X",X_train.shape)
print("shape of new output Y",y_train.shape)
print(X_train)
print(y_train)

df_plot=df_scaled.copy()
df_plot['target'] = df['target'].replace({0: 'Setosa', 1: 'Versicolor', 2: 'Virginica'})
plt.figure(figsize=(12, 8))
# Create a pair plot with hue based on the 'target' column in the dataframe
g=sns.pairplot(df_plot, hue='target', plot_kws={'legend': False})
g._legend.set_title('')

plt.suptitle('Attribute Correlation Pair Plot of Transformed Data', fontsize=16, y=0.98)

# Set the style and color of the text box
textbox_props = dict(boxstyle='round', facecolor='white', edgecolor='gray', alpha=0.8)

# Add a text box with the highlighted feature names
plt.text(1.5, 6, "THA076BCT026\nTHA076BCT027\nTHA076BCT041", fontsize=10, color='red', ha='center', bbox=textbox_props)

# Adjust the spacing between subplots
plt.subplots_adjust(top=0.9)

legend_elements = [plt.Line2D([0], [0], marker='o', color='w', label='Setosa', markerfacecolor='royalblue',
```

```
    markersize=8),
                   plt.Line2D([0], [0],
                       marker='o', color=
                       'w', label='
                       Versicolor',
                       markerfacecolor='g
                       ', markersize=8),
                   plt.Line2D([0], [0],
                       marker='o', color=
                       'w', label='
                       Virginica',
                       markerfacecolor='
                       orange',
                       markersize=8)]

# Add the legend to the plot
plt.legend(handles=legend_elements, title
    ='target')
# Save the plot to a file
plt.savefig('./plots/
    pair_plot_transformed.png',
    bbox_inches='tight')

# Display the plot
plt.show()

# lets see the new plot after
    standardization
plt.figure(figsize=(10, 6))
plt.scatter(df_scaled['sepal length (cm)'
    ], df_scaled['sepal width (cm)'], c=
    df_scaled['target'], label='Sepal')
plt.scatter(df_scaled['petal length (cm)'
    ], df_scaled['petal width (cm)'], c=
    df_scaled['target'], marker='x',
    label='Petal')
plt.xlabel('Length (cm)')
plt.ylabel('Width (cm)')
plt.title('Scatter Plot of Sepal and
    Petal Features')
plt.colorbar(label='Species')
plt.ylim(-3,4)
plt.xlim(-3,3)
plt.text(-1, 3.5, "'THA076BCT026','
    THA076BCT027','THA076BCT041'",
    fontsize=8,color='red')
plt.legend()
plt.savefig('./plots/processed Sepal and
    Petal Feature', bbox_inches='tight')
plt.show()

# **STEP 4: Covariance matrix calculation
    .**

# X_train is already an array so no need
    to do data_matrix = np.array(X)
data_matrix = X_train
data_matrix.shape

# covariance matrix will be 4*4 since x
    =(150,4)
covariance_matrix = np.cov(X_train,
    rowvar=False)              # row of
    X_train != variable(so column of
    X_train means variable, and rows
    means observation)
print("Covariance matrix shape:",
    covariance_matrix.shape)
print(covariance_matrix)

# visualize covariance matrix
plt.imshow(covariance_matrix, cmap='Blues
    ', interpolation='nearest')
plt.colorbar()
plt.title('Covariance Matrix')
plt.text(0, -0.8, "'THA076BCT026','
    THA076BCT027','THA076BCT041'",
    fontsize=8,color='red')
plt.xticks(np.arange(4), np.arange(4))
    #tick labels on axes changed from 0
    to 4
plt.yticks(np.arange(4), np.arange(4))
plt.savefig('Iris Covariance Matrix',
    bbox_inches='tight')
plt.show()

# **STEP 5: Eigen values and eigen vector
     calculation**

eigenvalues, eigenvectors = np.linalg.eig
    (covariance_matrix)
#descending sort
sorted_indices = np.argsort(eigenvalues)
    [::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:,
    sorted_indices]
print("eigen values are",eigenvalues)
print("eigen vectors are",eigenvectors)
print(eigenvalues.shape)
print(eigenvectors.shape)

# **STEP 6: See the variance explained by
     each eigen values.**

# Proportion of variance for each eigen
    values
pov_list=[]
for i in range (0,4):
  pov=eigenvalues[i]/sum(eigenvalues)
  pov_list.append(pov)
  print("Proportion of variance for",
      eigenvalues[i],"eigen value is",
      pov)

# Plot the scree plot
plt.plot(range(1, len(pov_list) + 1),
    pov_list, marker='o',color='blue')
varlegend= "'THA076BCT026','THA076BCT027
    ','THA076BCT041'"
plt.text(2.5,0.7,varlegend,color='red')
      #plt.text(x_position, y_position,
    varlegend)
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance')
plt.title('Scree Plot')
plt.show()


# **STEP 7: Select the eigen vectors as
    needed to compute the final output**

print("Shape of x train is",X_train.shape
    )
```

```
transposed_X_train = np.transpose(X_train)
transposed_X_train.shape
eigenvectors_transposed=np.transpose(eigenvectors)

final_data = {}        # Dictionary to store the final projected data

# Define the combinations of selected components
selected_components = [[0], [0, 1], [0,2], [0, 3], [1], [1,2], [1,3], [0, 1, 2, 3]]
count=len(selected_components)

# Iterate over the selected components
for i in range (0,count):
  selected_eigenvectors = eigenvectors_transposed[selected_components[i]]
                  #select the combination of components from above list
  final_data[i] = np.transpose(selected_eigenvectors @ transposed_X_train)  #projection
  print(final_data[i].shape)

eigenvectors_transposed[selected_components[2]]

# **STEP 8: Computing and visualizing covariance for the new data.**

new_covariance_matrices_dictionary = {}
# Compute covariance matrix for each projected data
for i in range(0, count):
    print(" * Covariance matrix for combination no", i)
    new_covariance_matrix = np.cov(final_data[i], rowvar=False)
    new_covariance_matrices_dictionary[i] = new_covariance_matrix
    # Print the shape of the covariance matrix
    print("  --->Covariance matrix shape: ", new_covariance_matrix.shape)

#Lets access the first combination(combination 0), which is (eigen[0])                        (best component)
new_covariance_matrices_dictionary[0]

#Lets access combination 1, which is eigen[0] and eigen[1]                (2 best components)
new_covariance_matrices_dictionary[1]

#Lets access combination 3, which is eigen[0] and eigen[3]                (best and worst component)
new_covariance_matrices_dictionary[3]

#Lets access combination 5, which is eigen[1] and eigen[2]        (2 medium medium best components)
new_covariance_matrices_dictionary[5]

#Lets access combination 7, which is eigen[0],eigen[1],eigen[2] and eigen[3]      (all components combined)
new_covariance_matrices_dictionary[7]

# Heatmap plot, darker shade indicate higher values. (combination starts from 0 to 7)
plt.imshow(new_covariance_matrices_dictionary[1], cmap='Reds', interpolation='nearest')  #for smoother interpolation of color on pixel values, instead of 'nearest', we can use 'bilinear','bicubic','spline16', 'spline36', 'hanning', 'hamming', 'hermite',etc..
plt.colorbar()
plt.title("New Covariance Matrix Combination 1 (2 best PC)")
plt.text(-0.3, -0.66, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('Covariance Matrix for 2 best PC', bbox_inches='tight')
plt.show()

plt.imshow(new_covariance_matrices_dictionary[3], cmap='Reds', interpolation='nearest')
plt.colorbar()
plt.title("New Covariance Matrix Combination 3 (best and worst PC)")
plt.text(-0.3, -0.66, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('Covariance Matrix for best and worst PC', bbox_inches='tight')
plt.show()

plt.imshow(new_covariance_matrices_dictionary[5], cmap='Reds', interpolation='nearest')
plt.colorbar()
plt.title("Covariance Matrix for two medium PCs")
plt.text(-0.3, -0.66, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('Covariance Matrix for two medium PCs', bbox_inches='tight')
plt.show()

plt.imshow(new_covariance_matrices_dictionary[7], cmap='Reds', interpolation='nearest')
plt.colorbar()
plt.title("New Covariance Matrix Combination 7 (All Components)")
plt.show()
```

```
# **STEP 9: Obtain final data and visualize**

# Two best PCs
final_data[1].shape
#final_data[1]

plt.ylim(-3,3)
plt.xlim(-3,3)
# plt.title("2 best PCs")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title("Scatter Plot after PCA for two best PCs on IRIS Dataset")
plt.text(-1.8, 2.8, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.scatter(final_data[1][:,0],final_data[1][:,1],c=y)
plt.savefig('Scatter Plot after PCA for two best PCs', bbox_inches='tight')

#Best and worst PCs
final_data[3].shape
#final_data[3]

plt.ylim(-3,3)
plt.xlim(-3,3)
plt.xlabel('PC1')
plt.ylabel('PC4')
plt.title("Scatter Plot after PCA for Best and worst PCs on IRIS Dataset")
plt.text(-1.8, 2.8, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.scatter(final_data[3][:,0],final_data[3][:,1],c=y)
plt.savefig('Scatter Plot after PCA for Best and worst PCs', bbox_inches='tight')

#Best and mediumly worse PCs
final_data[5].shape
#final_data[5]

plt.ylim(-3,3)
plt.xlim(-3,3)
plt.title("Scatter Plot after PCA for two Medium PCs on IRIS Dataset")
plt.text(-1.8, 2.8, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.xlabel('PC2')
plt.ylabel('PC3')
plt.scatter(final_data[5][:,0],final_data[5][:,1],c=y)
plt.savefig('Scatter Plot after PCA for two Medium PCs', bbox_inches='tight')

#All components
final_data[7].shape
#final_data[7]

plt.ylim(-3,3)
plt.xlim(-3,3)
plt.title("All PCs")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(final_data[7][:,0],final_data[7][:,1],c=y)

# Create a 3D scatter plot
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(final_data[7][:,0],final_data[7][:,1],final_data[7][:,2],c=y, marker='o')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title('3D Scatter Plot of 3 PCs')
ax.text(-6, 2.8,1, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.tight_layout()
plt.savefig('3D Scatter Plot of 3 PCs', bbox_inches='tight',dpi=300)
plt.show()


# **STEP 10: PCA with library.**

from sklearn.decomposition import PCA
# Create instances of PCA with the desired number of components. (pca1 = 2components. pca2=so as to retain 95% info)
pca1 = PCA(n_components=2)
pca2 = PCA(0.95)

# Fit the PCA model to the data and transform the data
reduced_data1 = pca1.fit_transform(X)
reduced_data2 = pca2.fit_transform(X)

# Print the shape of the reduced data
print("Shape of reduced data:", reduced_data1.shape)
print("reduced through first", reduced_data1)

print("Shape of reduced data:", reduced_data2.shape)
print("reduced through second", reduced_data2)


# **STEP 11: Lets train 4 models and visualize the results.** 2 models are implemented after applying PCA using library. The third model is implemented after applying PCA using Math(using all the earlier above steps). The fourth model is trained without PCA.

# Create 4 sets of test-train for 4 models.
from sklearn.model_selection import train_test_split
```

```
x_train1,x_test1,y_train1,y_test1 = train_test_split(reduced_data1,y, test_size=0.2)   # using library, 2 components
x_train2,x_test2,y_train2,y_test2 = train_test_split(reduced_data2,y, test_size=0.2)   # using library, 0.95 info
x_train3,x_test3,y_train3,y_test3 = train_test_split(final_data[1],y, test_size=0.2)   # from scratch, 2 components
x_train4,x_test4,y_train4,y_test4 = train_test_split(X_train,y,test_size=0.2)          # no PCA

#training of the model
# model 1: using library, 2 components
from sklearn.svm import SVC
svm1 = SVC()
svm1.fit(x_train1, y_train1)

#library, 0.95 info
svm2=SVC()
svm2.fit(x_train2, y_train2)

#from scratch, 2 components
svm3=SVC()
svm3.fit(x_train3, y_train3)

#No PCA
svm4=SVC()
svm4.fit(x_train4, y_train4)

# Accuracy check for 4 models
from sklearn.metrics import accuracy_score

y_pred1 = svm1.predict(x_test1)
accuracy1 = accuracy_score(y_test1, y_pred1)
print("Accuracy1:", accuracy1)

y_pred2 = svm2.predict(x_test2)
accuracy2 = accuracy_score(y_test2, y_pred2)
print("Accuracy2:", accuracy2)

y_pred3 = svm3.predict(x_test3)
accuracy3 = accuracy_score(y_test3, y_pred3)
print("Accuracy3:", accuracy3)

y_pred4 = svm4.predict(x_test4)
accuracy4 = accuracy_score(y_test4, y_pred4)
print("Accuracy4:", accuracy4)

#prediciton from 4 models
predictions1 = svm1.predict(x_test1)
predictions2 = svm2.predict(x_test2)
predictions3 = svm3.predict(x_test3)
predictions4 = svm4.predict(x_test4)
predictions1.shape

# Lets define a function to plot confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
def plot_confusion_matrix(y_true, y_pred, model_name,text):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.xlabel("Predicted Labels")
    plt.ylabel("True Labels")
    plt.text(0.2, -0.5, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
    plt.savefig(f'./plots/{model_name}.png', bbox_inches='tight',dpi=300)
    plt.show()

# Plot confusion matrix for model 1
plot_confusion_matrix(y_test1, predictions1, "UsingLibrary, 2 Components")

# for model 2
plot_confusion_matrix(y_test2, predictions2, "Using Library, 0.95 Info retention")

#for model 3
plot_confusion_matrix(y_test3, predictions3, "From Scratch, 2 Components")

#for model4
plot_confusion_matrix(y_test4, predictions4, "No PCA")
```

C. CANCER DATASET

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns


# **STEP 1 : Load the dataset into a dataframe and analyze the data**
#specify path of csv file relative to "/content/drive" directory.
file_path = './dataset/Cancer_Data.csv'
df = pd.read_csv(file_path)
df.head(10)
df.shape

# Print the column names
print(df.columns)

# statistics computed for each column in the df
summary_stats = df.describe()
print(summary_stats)

# see the unique values in diagnosis column
```

```
unique_diagnosis = df['diagnosis'].unique()
print(unique_diagnosis)

# Looks like diagnosis is our target column. Lets assign X and y accordingly
# Separate the data and target variables
X = df.drop(columns=['id', 'diagnosis', 'Unnamed: 32'])
y = df['diagnosis']

print("Data:")
print(X.shape)
print(X.head())
print()
print("Target:")
print(y.shape)
print(y.head())
X.columns

# Create a heatmap of the attribute correlations
# High correlation (+ve or -ve) by bright colors, low correlation by dull colors.
plt.figure(figsize=(12, 8))
sns.heatmap(X.corr(), cmap='coolwarm', annot=True, annot_kws={'size': 5})
plt.title('Attribute Correlation Heatmap')
plt.text(8, -1.2, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('./plots/Corelation Matrix', bbox_inches='tight')
plt.show()


# **STEP 2: Preprocessing**
# Perform label encoding on the 'diagnosis' target variable
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
print(y_encoded)

# change the dataframe's diagnosis column
df['diagnosis'] = y_encoded
df.head()

# Standardizing everything to have a common ground so that magnitude of some feature's value doesn't affect the PCA.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
# Scaling of the whole dataframe.
df_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)

# Define new X and y
X_train = df_scaled
y_train = df['diagnosis']
print("shape of new input X",X_train.shape)
print("shape of new output Y",y_train.shape)
df_scaled.head()

# Assign different colors to the data points based on the 'diagnosis' column
# Used to visually distinguish 2 different categories in that column
colors = ['blue' if t == 0 else 'red' for t in df['diagnosis'].map({0: 'B', 1: 'M'})]

X_train.cov()


# **STEP 3:CoVariance Matrix Calculation**
# covariance matrix will be 30*30 since X=(569,30). [each 30 features has covariance with each 30. so 30*30 matrix]
covariance_matrix = np.cov(X_train, rowvar=False)
print("Covariance matrix shape:", covariance_matrix.shape)
attribute_labels = X.columns
covariance_matrix = pd.DataFrame(covariance_matrix, columns=attribute_labels, index=attribute_labels)
print(covariance_matrix.head())

# Visualize covariance matrix
plt.figure(figsize=(12, 8))
sns.heatmap(covariance_matrix, cmap='coolwarm', annot=True, annot_kws={'size': 5})
plt.title('Covariance Matrix on Processed Cancer Data')
plt.text(8, -1.2, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('./plots/Covariance Matrix Processed Cancer Data', bbox_inches='tight')
plt.show()


#### **STEP 4: Eigen vectors/values calculation**
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
print("shape of eigenvalue:",eigenvalues.shape)
print("shape of eigen vectors:", eigenvectors.shape)
print("eigen values are",eigenvalues)
print("eigen vectors are",eigenvectors)


# **STEP 5: See the variance explained by each eigen values.**
# Element wise array division to obtain a
```

```
    new array
# proportion_of_variance array provides insights into relative importance of each PC in capturing the overall variance in the dataset
proportion_of_variance = eigenvalues/sum(eigenvalues)
proportion_of_variance

# Plot a scree plot to visualize how much variance is captured by which PC
cumulative_variance = np.cumsum(proportion_of_variance)

# range(1 to 30) on x axis, cummulative pov on y axis
plt.plot(range(1, len(proportion_of_variance) + 1), proportion_of_variance, marker='x', color='blue')
plt.plot(range(1, len(proportion_of_variance) + 1), cumulative_variance, marker='o', color='orange')
plt.xlabel('Principal Components(Eigen Vector)')
plt.ylabel('Proportion of Variance')
plt.title('Scree Plot')
legend_class = ["Explained Variance", "Cummulative Explained Variance"]
plt.legend(labels = legend_class)
plt.text(12, 0.9, "'THA076BCT026','THA076BCT027','THA076BCT041'", fontsize=8,color='red')
plt.savefig('./plots/Scree Plot', bbox_inches='tight')
plt.show()


# **STEP 6: Select the eigen vectors as needed to compute the final output** Formula: New data (Y) = Row feature vector * Row data
# Transpose data into row wise
transposed_X_train = np.transpose(X_train)
print("shape of transposed x train is", transposed_X_train.shape)
# Transpose eigen vectors into row wise for row feature vector
eigenvectors_transposed=np.transpose(eigenvectors)

# Dictionary to store the final Y(because we want to have multiple Y, each obtained by selecting our choice of combination of PC).
Y = {}
selected_components = [[0], [0, 1], [0,3], [2,3], [0,29]]
                            # best PC, 2 best PCs, best+medium, medium+medium, best+worst
count=len(selected_components)

# Iterate over the selected components
for i in range (0,count):
  selected_eigenvectors = eigenvectors_transposed[selected_components[i]]       # select the combination of components from above list
  Y[i] = np.transpose(selected_eigenvectors @ transposed_X_train)               # projection
  print(Y[i].shape)

# checking the nature of Y
print(type(Y))
print(Y.keys())
Y[1].shape
print(type(Y[1]))


# **STEP 7: Computing and visualizing covariance for the new data.**
# Dictionary to store all covariance matrices(obtained by selecting our choice of combination of PC).
new_covariance_matrices_dictionary = {}
for i in range(0, count):
  new_covariance_matrix = np.cov(Y[i], rowvar=False)
  new_covariance_matrices_dictionary[i] = new_covariance_matrix                 # Combination:
  print("*Covariance matrix shape for combination",i,"is", new_covariance_matrix.shape)
  # 0=best PC, 1=2 best PCs, 2= best+medium PCs
  print("-->Covariance matrix:", new_covariance_matrices_dictionary[i])         # 3= medium+medium PCs, 4= best+worst PCs

# Heatmap to visualize covariance matrix combination.
plt.imshow(new_covariance_matrices_dictionary[1], cmap='Reds', interpolation='nearest')  #instead of 'nearest', we can use 'bilinear','bicubic','spline16', 'spline36', 'hanning', 'hamming', 'hermite',etc..
plt.colorbar()
plt.title("New Covariance Matrix Combination 1 (2 best PCs)")
plt.show()

#lets try bilinear,pixel coloring is smoother(linear interpolation between 4 nearest data points)
plt.imshow(new_covariance_matrices_dictionary[2], cmap='Reds', interpolation='bilinear')
plt.colorbar()
plt.title("New Covariance Matrix Combination 2 (Best and medium PCs)")
plt.show()
```

```

#lets try bicubic,pixel coloring is again smoother(linear interpolation based on 16 nearest data points)
plt.imshow(new_covariance_matrices_dictionary[3], cmap='Reds', interpolation='bicubic')
plt.colorbar()
plt.title("New Covariance Matrix Combination 3 (Two medium PCs)")
plt.show()

#lets try spline16,pixel coloring is even smoother(16 degree of polynomial used for interpolation, for flexibility)
plt.imshow(new_covariance_matrices_dictionary[4], cmap='Reds', interpolation='spline16')
plt.colorbar()
plt.title("New Covariance Matrix Combination 4 (Best and worst PCs)")
plt.show()


# **STEP 8:Visualize the final output**
#Combination 0 = Y[0] = Best PC
#plt.ylim(-10,10)
#plt.xlim(-10,10)
varlegend= "Roll: 26,27,41"
plt.text(5,0.04,varlegend,color='red')
plt.title("1 best PC")
plt.xlabel('PC1')
plt.scatter(Y[0], np.zeros_like(Y[0]), c=y_train)   # colors = based on y_train

# Combination 1 = Y[1] = 2 best PCs
#plt.ylim(-3,3)
#plt.xlim(-3,3)
varlegend= "Roll: 26,27,41"
plt.text(5,4,varlegend,color='red')
plt.title("2 best PCs")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(Y[2].iloc[:, 0], Y[2].iloc[:, 1], c=y_train)    # iloc used to access data

# Combination 3 = Y[3] = 2 medium PCs
#plt.ylim(-3,3)
#plt.xlim(-3,3)
varlegend= "Roll: 26,27,41"
plt.text(2,4.5,varlegend,color='red')
plt.title("2 medium PCs")
plt.xlabel('PC3')
plt.ylabel('PC4')
plt.scatter(Y[3].iloc[:,0],Y[3].iloc[:,1],c=y_train)

# Combination 4 = Y[4] = Best and worst PCs
plt.ylim(-3,3)
#plt.xlim(-3,3)
varlegend= "Roll: 26,27,41"
plt.text(5,2,varlegend,color='red')
plt.title("Best and worst PCs")
plt.xlabel('PC1')
plt.ylabel('PC30')
plt.scatter(Y[4].iloc[:,0],Y[4].iloc[:,1],c=y_train)


# **STEP 9: PCA using Library**
from sklearn.decomposition import PCA
# Create instances of PCA with the desired number of components. (pca1 = 2components. pca2=so as to retain 95% info)
pca1 = PCA(n_components=2)
pca2 = PCA(0.95)

# Fit the PCA model to the data and transform the data
reduced_data1 = pca1.fit_transform(X_train)
reduced_data2 = pca2.fit_transform(X_train)

# Print the shape of the reduced data
print("Shape of reduced data:", reduced_data1.shape)
print("reduced through first", reduced_data1)
print("Shape of reduced data:", reduced_data2.shape)
print("reduced through second", reduced_data2)


# Here it can be seen that to retain 95% info, 10 principle components are required
# Visualize the output for the first PCA done by library
#plt.ylim(-3,3)
#plt.xlim(-3,3)
varlegend= "Roll: 26,27,41"
plt.text(10,10,varlegend,color='red')
plt.title("Two best PCs-Using Library")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(reduced_data1.T[0], reduced_data1.T[1], c= y_train) # After applying transform method of PCA,Each row was represented as a sample. So need to transpose.

# For the 2nd PCA done by library(to retain 95% info), 10 components were required! We will try to plot 3d scatter plot to visualize.
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
varlegend= "Roll: 26,27,41"
ax.text(-5,12.5,10,varlegend,color='red')


    # 3d coordinates required to specify
```

```
    the legend position
ax.scatter(reduced_data2.T[0],
    reduced_data2.T[1],reduced_data2.T
    [2],c=y_train)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title('3D Scatter Plot of 3 PCs')
plt.show()

# For the next step, we can train a model
    and compare the performance to see
    the impact of PCA we did so far.
```