Implementing Principal Component Analysis from Scratch (July 2023)

Kaustuv Karki[1], Undergraduate, IOE, Nikhil Pradhan[2], Undergraduate, IOE

[1]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal
[2]Institute of Engineering Thapathali Campus, Tribhuvan University, Kathmandu, Nepal

Corresponding author: Kaustuv Karki (e-mail: karkikaustuv@gmail.com),
Nikhil Pradhan (e-mail: nikhilpradhan20b@gmail.com)

ABSTRACT Principal component analysis (PCA) is a widely used technique for dimensionality reduction and data visualization. It aims to find orthogonal vectors that capture the most variance in the data, and project the data onto a lower dimension to counter the problem created by high number of dimensions in a dataset. This report implements PCA from scratch using Python and NumPy and applies it to three datasets: a randomly generated dummy Dataset, Iris Dataset and Diabetes Prediction Dataset. After the data from the dataset were subjected to PCA and plots were generated to effectively visualize the transformed data. The dataset was subjected to our PCA implementation and the PCA from scikit-learn library and the results were compared. This report demonstrates the effectiveness of custom implementation of PCA and PCA from scikit-learn library for effective dimension reduction.

INDEX TERMS Covariance matrix, Dimensionality Reduction, Eigen Values, Eigen Vectors, Principal Component Analysis

I. INTRODUCTION

As the number of dimensions grows many problems arise in the dataset. The data becomes increasingly sparse meaning the datapoints are spread out very thinly making it hard to find any kind of pattern or relationship in the data and very high computational power is required for the high dimensional dataset. Also, there is a problem of visualizing the data in high dimensions. This problem is known as the curse of dimensionality. In the real world the collected data may have many features. Among these features there may be features whose values are irrelevant to the context of the dataset or are heavily correlated to other features present in the dataset. These features do not provide any valuable information or insights to the context of the dataset but only increase the complexity of the problem. So, it becomes a very important task to remove such irrelevant features and keep only the features which are relevant to the subject. And one of the methods to achieve this task is known as dimensionality reduction.

Principal component analysis (PCA) is a widely used technique for dimensionality reduction and data visualization. It aims to reduce the dimension of the dataset while preserving as much information as possible. It aims to find a set of orthogonal vectors, called principal components, that capture the most variance in the data. The orthogonal vectors are in such order that the first principal component captures the most variance while each subsequent principal component captures the next highest variance in the dataset. The principal components are orthogonal to each other signifying that they are not correlated with each other.

II. METHODOLOGY

A. BRIRF THEORY

The main concept of PCA[1] is removing any redundant dimensions and keeping the dimensions with the highest variance. PCA selects the principal components that capture all the major variations across the dataset encompassing most of the information present in the dataset

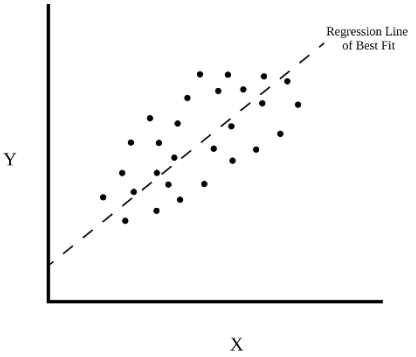


Figure 1 : PCA FIGURE 1

Suppose we have a dataset with features X and Y then we can fit a regression line onto the dataset. Drawing an orthogonal line to the best fit line we get,

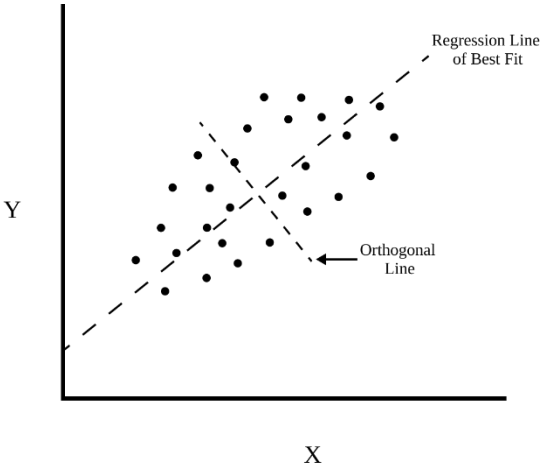


Figure 2 : PCA Figure 2

Since data varies mostly along the best fit line. Now, we can project the points onto our new axis and get our new x-axis and y-axis. We can keep drawing lines perpendicular to both lines to get more axis.

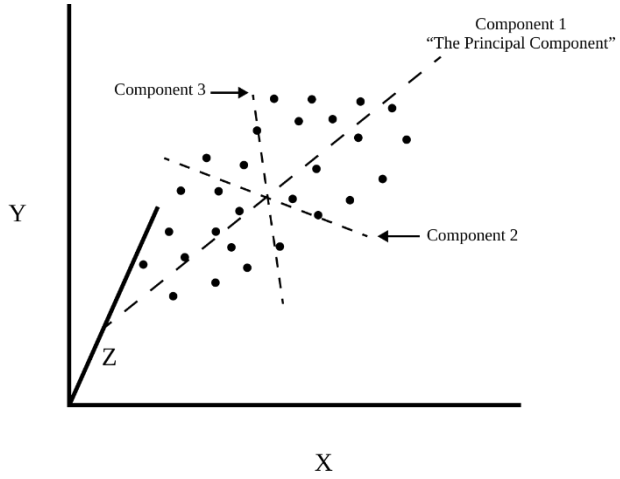


Figure 3 : PCA Figure 3

These new axes are Principal Components. PC1 is generally used to denote the component that captures the most variation, PC2 the next and so on.

B. WORKING PRINCIPLE

Initially PCA was performed on a random (20 X 2) matrix was generated from a standard normal distribution. Thus, generated matrix had no correlation so PCA could not be applied. Random (2 X 2) matrix was then generated from a uniform distribution. This matrix was then multiplied with the original matrix to generate a new (20 X 2) matrix. Newly generated data was then plotted which resulted in a correlated data point. Standardization was not required as the data from standard normal distribution had 0 mean and multiplying with the (2 X 2) matrix is scaling. Covariance matrix was calculated this, and its value was checked. The required criteria of diagonal element high and off diagonal element nearly 0 was not achieved. Eigenvalues and eigenvectors were extracted from this covariance matrix. Initially both principal components were taken, and a change of basis vector was performed. The variance along Principal Component 1 was very high compared to Principal Component 2 so PC2 was ignored and PC1 was taken. PCA was again using the scikit-learn library and then compared with PCA from scratch. Thus, PCA of random data was completed.

PCA for iris dataset[2] firstly involved creating a new data frame and loading the iris data set into the data frame. Features were separated from the class and a matrix of features was created. Calculation of mean and standard deviation showed that the dataset needed to be standardized. Standardization was performed using StandardScaler from scikit-learn. Thus, the dataset had mean 0 and standard deviation of 1. Covariance matrix of the standardized matrix was calculated which did not satisfy our required properties. New basis vectors for the dataset were calculated by eigen decomposition of covariance matrix. Eigenvectors were used to perform changes of basis and projected features were obtained. PCA was again done using decomposition from scikit-learn and compared with PCA from scratch. Thus, PCA of random data was completed.

Diabetes prediction dataset was chosen as our next candidate for PCA. Dataset consisted of 8 features and class. Categorial classes were converted into numerical form. Some of the features were NaN (Not a Number) which was filled with mean value using SimpleImputer. Features was separated from the class and matrix was created. Calculation of mean and standard deviation showed that the dataset needed to be standardized. Standardization was performed. Thus, the dataset had mean 0 and standard deviation of 1. Covariance matrix of the standardized matrix was calculated which did not satisfy our required properties. New basis vectors for the dataset were calculated by eigen decomposition of covariance matrix. Eigenvectors were used to perform change of basis and projected features was obtained. PCA was again done using scikit-learn library and compared with PCA from scratch. Newly obtained datapoints were the plotted to analyze the result. Thus, PCA of random data was completed.

C. SYSTEM BLOCK DIAGRAM

1) CREATE A MATRIX OF FEATURE VECTORS

Initially features are selected from which the target is interpreted. These features allow the model to make predictions on new data to assign classes.

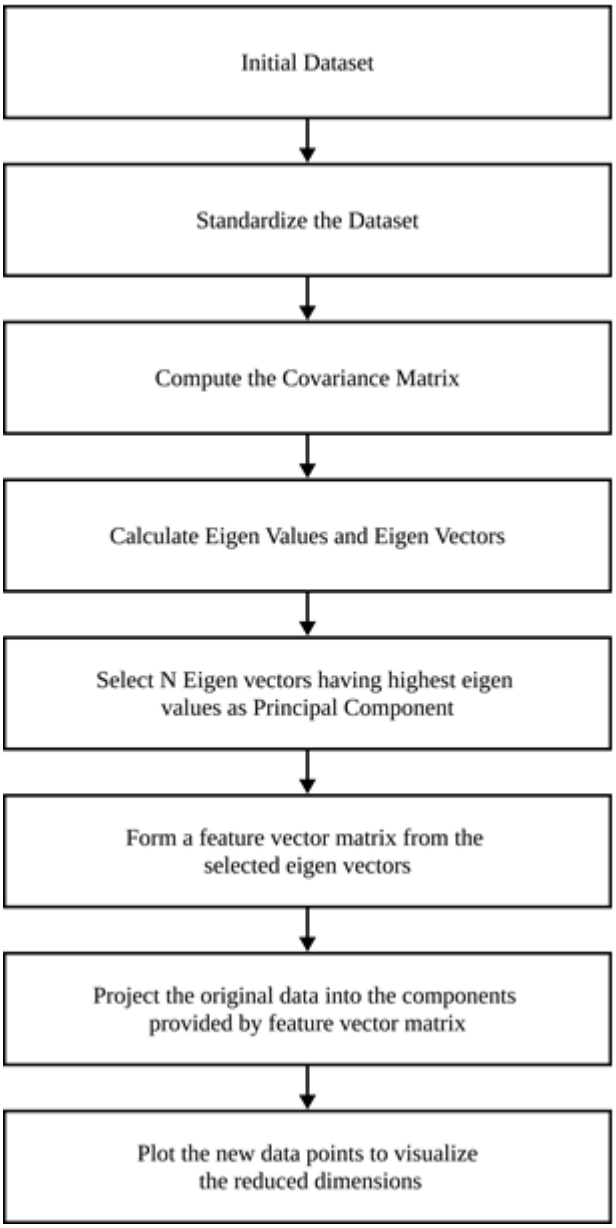


Figure 4 : System Block Diagram

2) STANDARDIZE THE DATA

Before starting the PCA process, the feature vectors must be standardized so that they have zero mean and standard deviation of 1. This ensures that all the features have the same scale so that PCA gives equal importance to each feature preventing features with larger scale to disproportionately influence the result. Units are also removed during standardization making interpretation of principal components more meaningful.

3) COVARIANCE MATRIX COMPUTATION

After standardization of features, covariance matrix of the standardized features is calculated. Covariance matrix shows the linear relationship between the features. For PCA decomposition, the diagonal elements of the covariance matrix i.e., Variance of a feature should be maximum, and the non-diagonal elements of the covariance matrix must be minimum. Covariance matrix allows PCA to find the direction of maximum variance.

4) CALCULATION OF EIGEN VALUES AND EIGEN VECTORS

The covariance matrix is used of eigen decomposition to get the eigenvalues and eigenvectors. Eigenvalues represent the amount of variance captured by each principal component and eigenvectors represent the direction of each principal component. Each eigenvector is new basis vectors that are used to project the original data into new coordinates.

5) SELECTION OF PRINCIPAL COMPONENT

Eigen values are used to calculate the importance of each principal component. If n number of features is required, then eigen values are arranged in ascending order and eigenvectors with top n eigenvalue is selected. This ensures that maximum amount of variance is captured thus retaining essential information.

6) FORMATION OF FEATURE MATRIX

Selected eigenvectors are used to form a feature new feature matrix. The feature matrix acts as a new basis for the data, and the new features are orthogonal to each other.

7) PROJECT ORIGINAL DATA INTO THE PRINCIPAL COMPONENTS

Change of basis is performed on the original dataset to project them onto new coordinates. This results in new dataset with reduced number of features.

8) VISUALIZATION

New data is then plotted to visualize the changes study the result.

D. MAJOR MATHEMATICAL FORMULAS

1) VARIANCE

Variance is used to measure how scattered the data are from a mean value. It is calculated by.

$$Var(x) = \frac{\sum_{i=0}^{n}(x_i - \bar{x})}{n-1} \quad (1)$$

Where:

x_i = Individual data points

$\overline{x}$ = Mean of all the value

n = Number of data points in the given data.

2) COVARIANCE

Covariance is the measure of variance between variables. It describes how variable change to together. Covariance is calculated as

$$Cov(x,y) = \frac{\sum_{i=0}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1} \quad (2)$$

Where:

x_i = Individual data points of feature x

$\overline{x}$ = Mean of all the value of feature x

y_i = Individual data points of feature y

$\overline{y}$ = Mean of all the value of feature y

n = Number of data points in the given data.

Covariance gives an idea about the trend of the data. Positive covariance indicates that with the increase of x, y also increases whereas negative covariance indicates that with increase in x, y decreases. Zero covariance indicates that there is no relationship between x and y.

3) COVARIANCE MATRIX

Covariance Matrix is a square matrix with diagonal elements representing variance and non-diagonal elements representing covariance. Generally, Covariance matrix is represented as

$$\begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(x,y) & Var(y) \end{bmatrix} \quad (3)$$

Where:
$Var(x)$ = Variance of x
$Var(y)$= Variance of y
$Cov(x,y)$ = Covariance between x and y

Covariance Matrix is calculated as

$$S_x = \frac{1}{n-1} XX^T \quad (4)$$

Where:
S_x= Square symmetric (n*n) matrix
X = n-dimensional vector of feature x

4) CHANGE OF BASIS

Basis of a vector space is the set of linearly independent vectors that span all the vector space. For e.g., Consider a 2-dimensional space with unit vector $\hat{\imath}$ along x axis and $\hat{\jmath}$ along y axis. These unit vectors can then be scaled so that it can span all the possible points in the 2-dimensional space. In PCA, change on basis plays a vital role to as we map the original data into new one by changing the basis vector. Changing the basis does not change the data only its representation is changed. Changing the basis only projects the data vectors on the basis vectors.
Change of basis is achieved by:

$$Y = PX \quad (5)$$

Where:
Y = Data points obtained after linear transformation
P = Basis vectors used for linear transformation
X = Original data points

5) EIGEN VALUES AND EIGEN VECTORS

Eigen v
Let A be an $n \times n$ matrix, then
An eigen vector is a non-zero vector v such that

$$Av = \lambda v \quad (6)$$

For some scalar λ.
An eigen value is a scalar λ such that

$$Av = \lambda v \quad (6)$$

Has some non-trivial solution.
If $Av = \lambda v$ and $v \neq 0$, then λ is eigenvalue for v, and v is the eigen vector for λ.
All eigen vectors of a symmetric matrix are perpendicular to each other.

C. INSTRUMENTATION

In the implementation of Principal Component Analysis (PCA) using Python, the following libraries and functions were utilized. Numpy, a powerful library for numerical computing, was employed for vector and matrix operations, providing efficient computation capabilities. Pandas, a popular data manipulation library, was used for storing and handling data in the form of dataframes. Scikit-learn, a comprehensive machine learning library, contributed the StandardScaler class for scaling the data, ensuring that all features have similar ranges. The np.linalg.eig function from Numpy was utilized to calculate the eigenvalues and eigenvectors, crucial components of PCA. Lastly, the PCA class from scikit-learn was employed to compare the results against the hand-coded implementation, facilitating a convenient and validated approach to PCA analysis. Also, for the visualization of the results using various 2D and 3D plots matplotlib and plotly were used.

III. RESULTS

A. RANDOM DATASET

The plot (fig 5) shows the datapoint obtained from standard normal distribution. Since the data is extracted randomly from

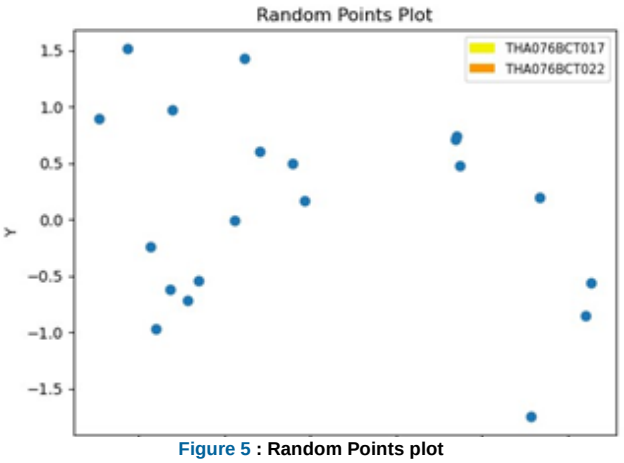


Figure 5 : Random Points plot

standard normal distribution, datapoints show no correlation thus PCA cannot be applied. The values generated form a

standard normal distribution generates a mean of zero and standard deviation close to 1

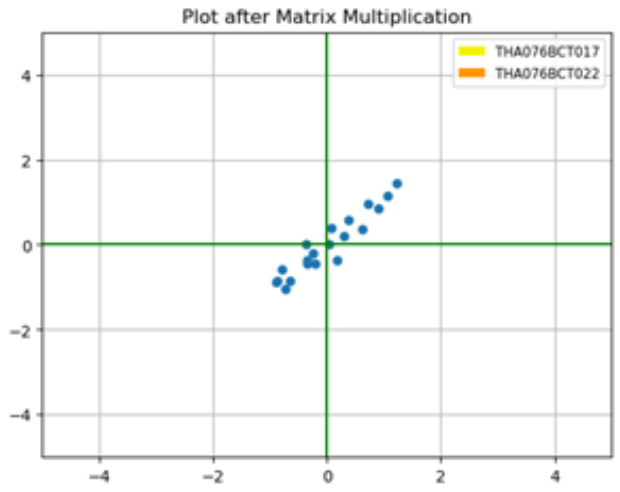


Figure 6 : Plot after Matrix Multiplication

Multiplication with (2 X 2) matrix from uniform distribution resulted in the above plot (fig 6). It is seen that the datapoints are correlated. This is because multiplication with the (2 X 2) matrix introduces a linear transformation upon the data. Because the transformation involves a linear combination of the original variables leading to a change in their relationship, as a result the transformed data points become correlated.

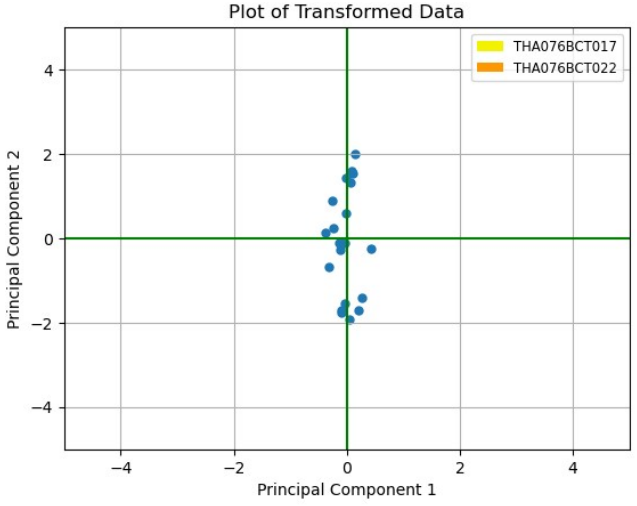


Figure 7 : Plot of Transformed Data

The plot (fig 7) was obtained by performing a change of basis vector and data point was projected upon new coordinated system defined by the principal components generated by applying PCA. The Principal Component 1 is as x-axis and Principal Component 2 is at y-axis.

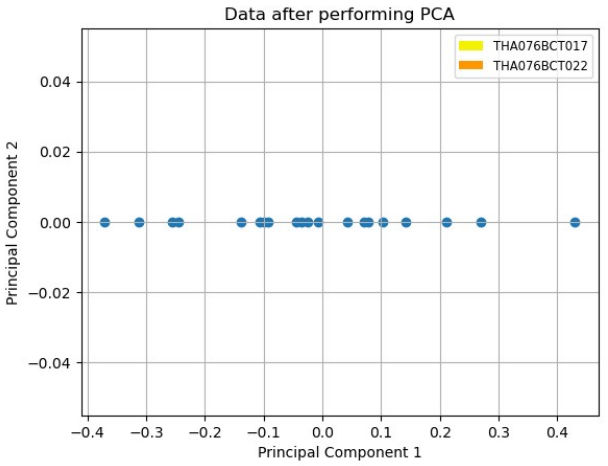


Figure 8 : Data after performing PCA

The plot (fig 8) demonstrated the reduction of the original two-dimensional data into a single dimension using PCA. In this plot the data are varied only in a single dimension x-axis using principal component 1 and the y-axis value which is principal component 2 remains zero.

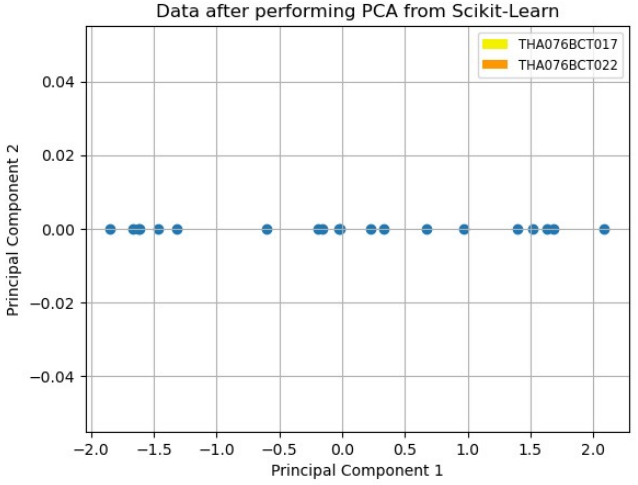


Figure 9 : Data after performing PCA from Scikit-Learn

The plot (fig 9) was obtained by using the PCA provided by scikit-learn library to reduce the dimension from two-dimensions to a single dimension.

B. IRIS DATASET

Like the random generated dataset PCA was also applied to the Iris Dataset. The results obtained from applying the PCA on the Iris Dataset are discussed below.

TABLE I
EXPLAINED VARIANCE FOR IRIS DATASET

Principal Component	Explained Variance	Approx. (in %)
PC1	0.7296	72.96
PC2	0.2285	22.85
PC3	0.03669	3.66
PC4	0.00517	0.51

Table I gives the values of variance captured by every principal component after applying PCA.

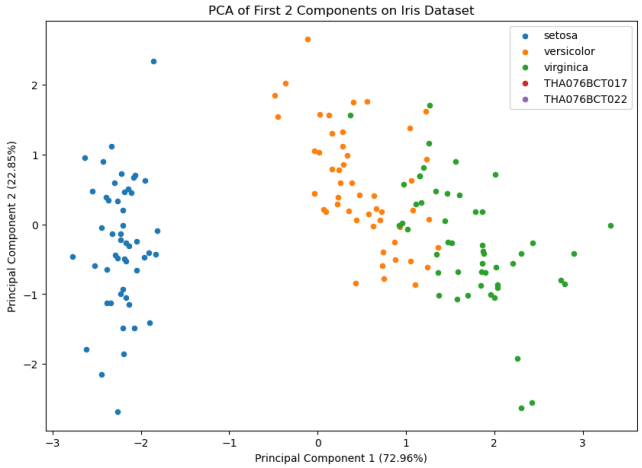


Figure 10 : PCA of first 2 Component on Iris Dataset

The 2D plot (fig 10) new data points of Iris Dataset after undergoing PCA. The goal of PCA is to retain as much information about the dataset in reduced dimensions and the plot shows the distribution of points along the two axes namely principal component 1 and principal component 2. This 2-D plot depicts PC1 which covers 72.96% of the variation and PC2 which covers 22.85% of the variation.

This plot provides us with insights and intuitions about the pattern of the data and the clusters. The data points that are closer to each other tend to be related so we can deduce a pattern from the plot. In this plot we can see that the principal component 1 has higher variance and it is seen in the plot as the points are largely spread out in this axis making it a more important axis. The scatter plot also reveals how clearly the classes present in the Iris dataset are separated into clusters. With only two dimensions of the initial four-dimensional dataset, the class setosa can be seen separated from the other two flower classes but there still lies some confusion between versicolor and virginica.

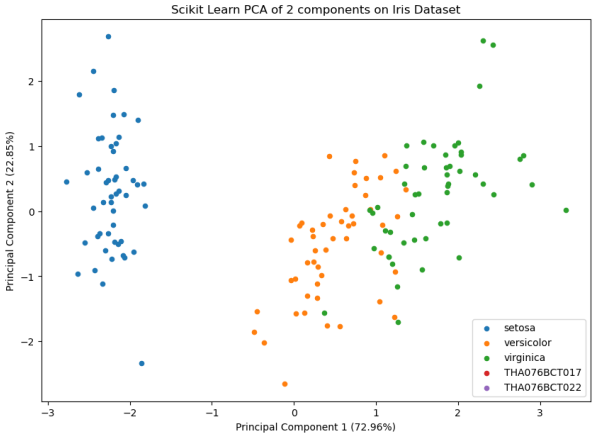


Figure 11 Scikit-Learn PCA of 2 component on iris Dataset

The 2D plot (fig 11) shows the top two PCA accomplished using the decomposition function from scikit-learn. We have observed that the eigenvalues and eigenvectors result in the same value using both methods i.e., applying PCA from scratch and PCA from scikit-learn. The only difference observed was the direction of eigenvectors which was introduced due to a sign difference. This sign difference does not cause any issue for PCA. But for the plot it caused the plot to flip along the y-axis. This occurred due to some difference in mathematical implementation.

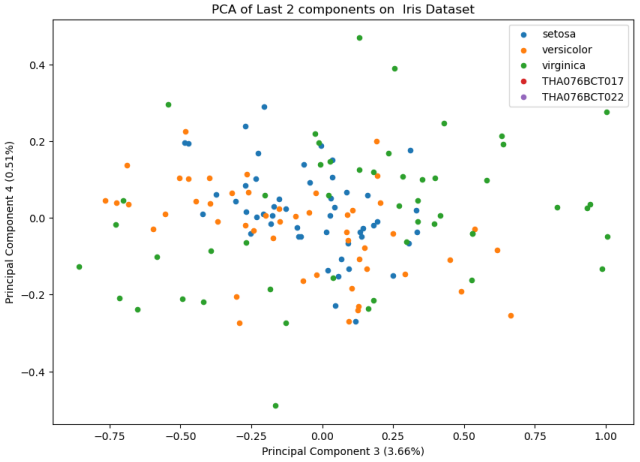


Figure 12 : PCA of Last component on its Dataset

The 2D plot (fig 12) shows the PC3 which is represented by the x-axis which captures 3.66% of the variation and PC4 which is represented by the y-axis and captures 0.51% of the variation. Separating the flowers was not possible from the available data points as most of the essential information about the original data was lost because the principal components could not retain most of the information from the dataset.

Figure 13 : PCA of First 3 component on Iris Dataset

The 3D plot consists of PC1, PC2 and PC3. This combination of Principal Component captures the largest amount of variation in the dataset while being easily visualizable. Flowers can be seen to be easily separatable as crucial

information of the original dataset was retained. The 3D plot provides more information about the dataset compared to the 2D plots for the Iris dataset.

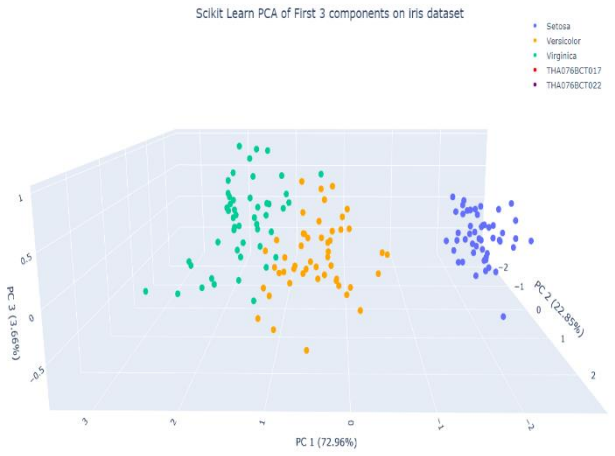


Figure 14 : Scikit-Learn PCA of First 3 Component on Iris Dataset

The 3D (fig 14) plot represents the dataset in a 3D environment by making the PC1 which capture 72.96% of the variance, PC2 which captures 22.85% of the variance and PC3 which captures 3.66% of the variance as its axis which is derived from decomposition using PCA from the scikit-learn library.

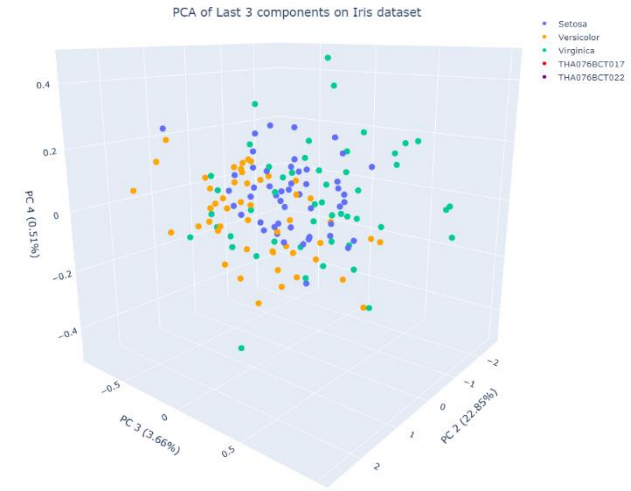


Figure 15 : PCA of Last 3 Component on Iris Dataset

Figure 15 shows the 3D plot of PC with least amount of variation as PC2 which captures 22.85% of the variance, PC3 which captures 3.66% of the variance, PC4 which captures 0.51% of the variance were taken as the axes. As crucial information is lost, separation of the flowers cannot be done, and important analysis and information cannot be deduced from this plot.

C. DIABETES DATASET

Finally, PCA was applied to the Diabetes Detection Dataset. The results obtained from applying the PCA on the Diabetes Detection Dataset are discussed below.

TABLE II
EXPLAINED VARIANCE FOR DIABETES PREDICTION DATASET

Principal Component	Explained Variance	Approx. (in %)
PC1	0.2150	21.5
PC2	0.1380	13.8
PC3	0.1322	13.22
PC4	0.1196	11.96
PC5	0.1099	10.99
PC6	0.1056	10.56
PC7	0.1041	10.41
PC8	0.0752	7.52

Table II gives the values of variance captured by every principal component after applying PCA on the diabetes detection dataset.

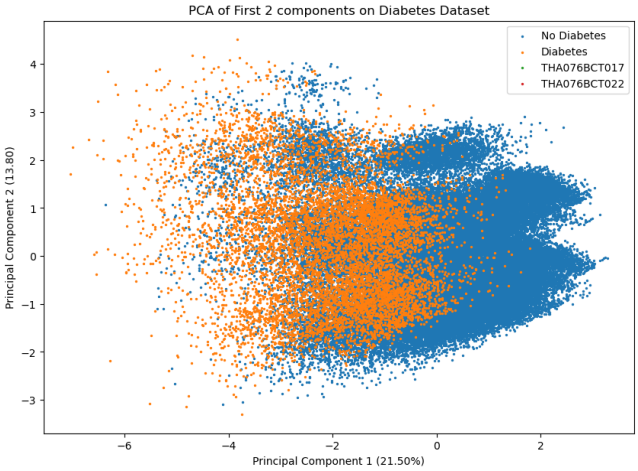


Figure 16 : PCA of First 2 Component on Diabetes Dataset

The 2D plot (fig 16) shows the top two principal components for diabetes prediction. Since the original dataset had 8 dimensions, two principal components are not enough to show the difference so not many insights can be gained from this figure. The 2D plot depicts PC1 which captures 21.5% of the variance and PC2 which captures 13.8% of the variance which is not enough variance captured to give any sort of meaningful information. As the principal components cannot retain major information about the dataset any sort of deduction is not possible from this 2D plot. In the plot the points are color coded for Diabetes and No Diabetes. As there is no clear separation between the classes PCA while reducing the dataset into 2 dimensions has not done a proper job of retaining the information.

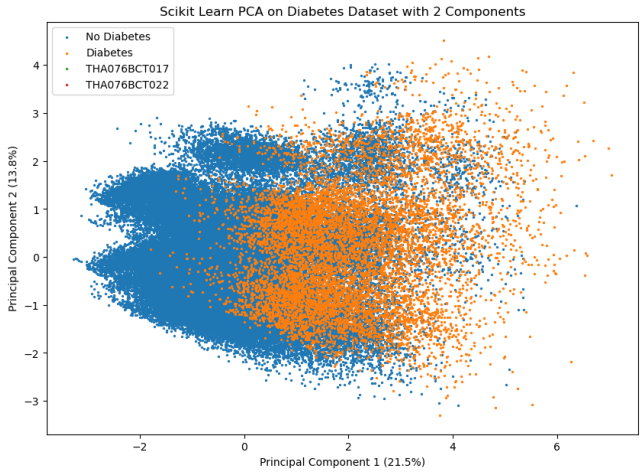


Figure 18: Scikit Learn PCA on Diabetes Dataset with 2 Component

The 2D plot (fig 18) shows top two PC from decomposition from scikit learn. Because of the mathematical implementation of PCA in scikit-learn library and custom one we can see that the plots are flipped on the x-axis.

Figure 17 : PCA on First 3 Component on Diabetes Dataset

The 3D (fig 17) plot shows the plot of the datapoints from the dataset using the PCA that captures the highest variance namely PC1, PC2 and PC3. The PC1 captures 21.5% of the variance, PC2 captures 13.8% of the variance and PC3 captures 13.22% of the variance as the three principal components do a better job than two components more information about the dataset is captured using higher dimension 3D rather than lower dimensions 2D. But even using three dimensions all the major information about the datasets is not captured properly.

Figure 19 : PCA of last 3 component on Diabetes Dataset

The 3D plot shows the points plotted by taking PC with the least variance PC6 with variance 10.56, PC7 with variance 10.41% and PC8 with variance 7.52% as the three axes. Because these PC do not capture retain much information no proper deduction can be made from the 3D plot. No proper separation and clustering of classes is observed in the plot.

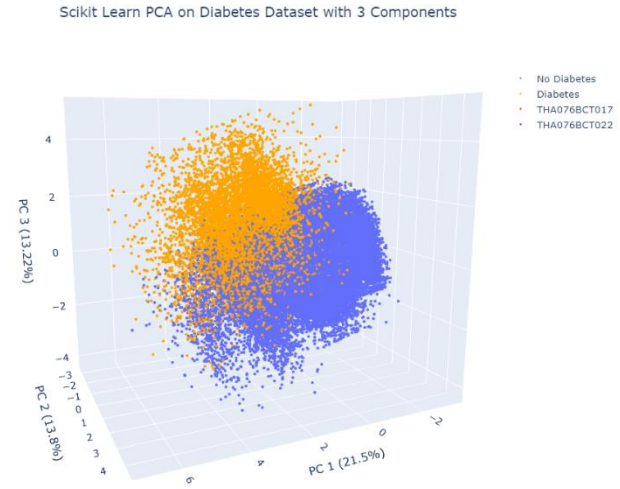


Figure 20 : Scikit-Learn PCA on Diabetes Dataset with 3 Components

The 3D plot (fig 20) shows top three PC from decomposition from scikit learn. Because of the mathematical implementation of PCA in scikit-learn library and custom one, the plots are flipped.

IV. DISCUSSION AND ANALYSIS

This report includes the process of implementing PCA from scratch. The findings of this report have a strong impact on giving us in-depth intuition and insights on the workings of the algorithms used for PCA. The PCA from this report can be used for dimensionality reduction which is an important method for visualization and analysis of the data. During the analysis of plots for PCA for a randomly generated dummy dataset we notice that the at first the random generated from

standard normal distribution had no correlation but after multiplying it with unform distribution matrix correlation was observed. This is because multiplying the matrix causes the linear combination of original variables. Similarly, PCA was applied on the Iris and Diabetes dataset and for Iris dataset adequate information was found to be found while reducing to two dimensions and the classes were totally separated after three dimensions. For the Diabetes Detection dataset, we found that even three dimensions could not retain all the necessary information about the dataset making the classes overlap in some cases.

The PCA from scikit-learn utilizes efficient linear algebra libraries resulting in excellent performance even for very large datasets. The PCA provided by the scikit learn library is also faster and efficient due to various optimization and parallelization techniques and can be implemented in a few lines of code with additional flexibility. Whereas the custom implementation of PCA requires extensive knowledge of the internal workings of the algorithm and the process is slower due to the lack of proper optimization.

V. CONCLUSION

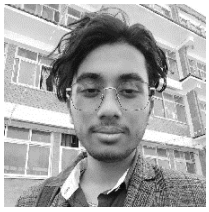
In conclusion we implemented Principal Component Analysis (PCA) from scratch on three different datasets: Randomly generated Dummy Dataset, Iris Dataset and Diabetes Detection Dataset and gained valuable intuition and insights on the concept of dimensionality reduction using PCA. The comparison between our custom PCA and the PCA provided by the scikit -learn library showed that the mathematics involved in the implementation of PCA in scikit-learn library is slightly different from our own implementation as change in the sign of the eigen vectors were found to be present in the eigen vectors from the scikit-learn PCA causing the plots of the graph to reflect around some axis.

Therefore, our project implements PCA from scratch giving us the theoretical and mathematical intuition about a widely used process for dimensionality reduction and benefits as well as the limitations of PCA in representing the information present in the data. Also implementing PCA from scratch gave us valuable insights and intuition about the inner workings of the algorithm. By applying PCA on various datasets, we were able to reduce the dimensions of complex datasets while retaining the most important information. This dimensionality reduction enabled us to visualize the data in a lower-dimensional space, making it easier to interpret and understand. PCA facilitated the extraction of relevant information from the datasets, enabling us to make well-informed decisions by utilizing the transformed data.

REFERENCES

[1] J. Shlens, “A Tutorial on Principal Component Analysis,” Apr. 2014, [Online]. Available: http://arxiv.org/abs/1404.1100

[2] “The Iris Dataset — scikit-learn 1.3.0 documentation.” https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html (accessed Jul. 03, 2023).



KAUSTUV KARKI is a devoted individual currently pursuing a graduate degree in Computer Engineering Program from Institute of Engineering Thapathali Campus under Tribhuvan University He is keen and highly motivated towards the field of Artificial Intelligence and Machine Learning. He actively seeks out various sources to learn more about these fields, including books, research papers, online courses, and tutorials. He believes in continuous learning and keeping up with the latest advancements in AI and ML. Overall, his dedication, curiosity, and proactive approach make him a promising individual in the field of Artificial Intelligence and Machine Learning.

NIKHIL PRADHAN.
Nikhil Pradhan is a dedicated and ambitious individual currently pursuing graduate studies in computer engineering from Institute of Engineering Thapathali Campus. With a strong passion for artificial intelligence (AI), he is actively engaged in expanding his knowledge and expertise in this rapidly evolving field. His academic pursuits provide him with a solid foundation in computer engineering and a deep understanding of programming languages and frameworks commonly used in AI applications. With his enthusiasm, academic background, and commitment to learning, He is well-positioned to make significant contributions to the AI industry.

APPENDIX

A. RANDOM GENERATED DUMMY DATASET

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Generates random data from standard normal
distribution
X = np.random.randn(20,2)
X.mean()
X.std()

#Plot of random points
plt.scatter(X[:,0], X[:,1], linewidths=1)
plt.title("Random Points Plot")
one = mpatches.Patch(facecolor='#f3f300',
label='THA076BCT017', linewidth = 0)
two = mpatches.Patch(facecolor='#ff9700', label =
'THA076BCT022', linewidth = 0)
plt.xlabel("X")
plt.ylabel("Y")

# Generated Data from uniform distribution
x1 = np.random.rand(2,2)

# Multiplying Standard Normal Distribution Matrix
and Uniform Distribution Matrix
Y = np.matmul(X, x1)

# Plot of the points after matrix multiplication
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.grid()
plt.title("Plot after Matrix Multiplication")
plt.axhline(y=0, color="g")
plt.axvline(x=0, color="g")
plt.xlabel("X")
plt.ylabel("Y")
plt.scatter(Y[:,0], Y[:,1], linewidths=0.01)
one = mpatches.Patch(facecolor='#f3f300',
label='THA076BCT017', linewidth = 0)
two = mpatches.Patch(facecolor='#ff9700', label =
'THA076BCT022', linewidth = 0)

# Covariance of the matrix
cov matrix = np.cov(Y.T)

# Variance of the correlated data
np.matmul(Y.T, Y) / (Y.shape[0] -1)

# Variance of random data
np.matmul(X.T, X) / (X.shape[0] -1)

from numpy.linalg import eig
eigen values, eigen vectors = eig(cov matrix)

# proportion of variance
total = np.sum(eigen values)
# gives the proportion of variance explained by a
particular eigen value/ principal component in the
dataset
def explained_variance(eigenvalues):
    sum = eigenvalues.sum()
    return (eigenvalues/sum) * 100
value = explained variance(eigen values)

transformed = np.matmul(eigen vectors.T, Y.T).T
np.cov(transformed.T)

#Plot of the transformed Data
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.grid()
plt.title("Plot of Transformed Data")
plt.axhline(y=0, color="g")
plt.axvline(x=0, color="g")
plt.scatter(transformed[:,0], transformed[:,1],
linewidths=0.01)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
one = mpatches.Patch(facecolor='#f3f300',
label='THA076BCT017', linewidth = 0)
two = mpatches.Patch(facecolor='#ff9700', label =
'THA076BCT022', linewidth = 0)

oned_transformed = np.matmul(eigen_vectors.T[0].T,
Y.T).T
oned transformed.shape
zeros = np.zeros(oned_transformed.shape[0],)

#Plot of the transformed Data after applying PCA
plt.grid()
plt.title("Data after performing PCA")
plt.scatter(oned transformed, zeros)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
one = mpatches.Patch(facecolor='#f3f300',
label='THA076BCT017', linewidth = 0)
two = mpatches.Patch(facecolor='#ff9700', label =
'THA076BCT022', linewidth = 0)

# Using PCA from scikit-learn library
from sklearn.decomposition import PCA
pca = PCA(n_components=1, random_state=0)
pca.fit(Y)

new_Y = pca.transform(Y)
new_Y.shape

# Plotting the points after using PCA from scikit-
learn
plt.grid()
plt.title("Data after performing PCA from Scikit-
Learn")
plt.scatter(new_Y, zeros)
plt.legend(["17, 22"])
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
one = mpatches.Patch(facecolor='#f3f300',
label='THA076BCT017', linewidth = 0)
two = mpatches.Patch(facecolor='#ff9700', label =
'THA076BCT022', linewidth = 0)
```

B. IRIS DATASET

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


# Loading the iris dataset from sklearn datsets
iris = datasets.load_iris()

R = np.array(iris.data)
print(R.shape)
iris.feature_names
# iris.data
# Making a dataframe with the data from the iris
dataset set and giving the comlumn names the
features.
```

```
df_iris = pd.DataFrame(iris.data,
columns=iris.feature names)
df_iris['target'] = pd.Series(iris.target)
df_iris.head()

# Taking the first columns and leaving the target
value
x = df_iris.iloc[:,0:4].values
print("The mean and the standard deviations are")
np.mean(x), np.std(x)

# Standardize the inital data and calculate the
mean and standard deviation
def standardize(x):
    x_std = StandardScaler().fit_transform(x)
    return x_std, x_std.mean(), x_std.std()

#Calculate the covariance matrix
def covariance_matrix(x_std):
    return np.cov(x_std.T)

#Calculate the eigen values and vectors
def eigVecVal(sx):
    eigVal, eigVec = np.linalg.eig(sx)
    return eigVal, eigVec

# Sorting the eigen values
def sort_eigen(eigenvalues, eigenvectors):
    sorted indices = np.argsort(eigenvalues)[::-1]
# Sort indices in descending order
    sorted_eigenvalues =
eigenvalues[sorted_indices] #Sorts the eigenValues
    sorted eigenvectors = eigenvectors[:,
sorted_indices] # Sorts the column according to
the indices
    return sorted_eigenvalues, sorted_eigenvectors

# gives the proportion of variance explained by a
particular eigen value/ pricipal component in the
dataset
def explained_variance(eigenvalues):
    sum = eigenvalues.sum()
    return (eigenvalues/sum) * 100

# For the start parameter give the index of the
first eigenvalue and for the end give the index +
1
def transformed_data(standardized_data, eigVec,
start, end):
    transformed x = np.matmul(eigVec.T[start:end],
standardized_data.T).T
    return transformed_x
import time

x_std, x_std_mean, x_std_standard_deviation =
standardize(x)


# The covariance matrix is
sx = covariance_matrix(x_std)

# Calculating the eigen values and eigen vectors
eigenValues, eigenVectors = eigVecVal(sx)

sorted_eigenValues, sorted_eigenVectors =
sort_eigen(eigenValues, eigenVectors)

variance_eigen =
explained variance(sorted eigenValues)

# Taking two eigen vectors with highest eigen
values and variance
transformed_x1 = transformed_data(x_std,
sorted eigenVectors, 0, 2)

members_roll = ["THA076BCT017", "THA076BCT022"]
targetNames = iris.target names
targetNames = list(targetNames) + members roll
plt.figure(figsize=(10, 7))
y = iris.target
for i, targetName in zip([0,1,2,3,4],
targetNames):
    plt.scatter(transformed x1[y==i, 0],
transformed x1[y==i, 1],label=targetName, s=20)
plt.xlabel("Principal Component 1 (72.96%)")
plt.ylabel("Principal Component 2 (22.85%)")
plt.legend()
plt.title('PCA of First 2 Components on Iris
Dataset')
plt.show()


# Taking two eigen vectors with least eigen values
and variance
transformed x2 = transformed data(x std,
sorted eigenVectors, 2, 4)
targetNames = iris.target_names
targetNames = list(targetNames) + members roll
plt.figure(figsize=(10, 7))
y = iris.target
for i, targetName in zip([0,1,2,3,4],
targetNames):
    plt.scatter(transformed_x2[y==i, 0],
transformed_x2[y==i, 1],label=targetName, s=20)
plt.xlabel("Principal Component 3 (3.66%)")
plt.ylabel("Principal Component 4 (0.51%)")
plt.legend()
plt.title('PCA of Last 2 components on  Iris
Dataset')
plt.show()


# Taking the top three eigen Vectors
transformed_x3d1 = transformed_data(x_std,
sorted eigenVectors, 0, 3)
import plotly.graph_objects as go
import plotly.offline as pyo
trace1 = go.Scatter3d(
    x = transformed x3d1[y==0][:, 0],
    y = transformed_x3d1[y==0][:, 1],
    z= transformed x3d1[y==0][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color="light green"
    ),
    name="Setosa"

)

trace2 =go.Scatter3d(
    x = transformed x3d1[y==1][:, 0],
    y = transformed x3d1[y==1][:, 1],
    z= transformed_x3d1[y==1][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="orange"
    ),
    name ="Versicolor"
)

trace3 = go.Scatter3d(
```

```
    x = transformed_x3d1[y==2][:, 0],
    y = transformed_x3d1[y==2][:, 1],
    z= transformed_x3d1[y==2][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="light blue"
    ),
    name ="Virginica"
)

trace4 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace5 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)


fig = go.Figure(data=[trace1, trace2, trace3,
trace4, trace5])

fig.update_layout(
    title="PCA of First 3 components on Iris
dataset",
        margin=dict(
        l=10,
        r=10,
        b=10,
        t=10,
        pad=4
    ),
    scene=dict(
    xaxis_title='PC 1 (72.96%)',
    yaxis_title='PC 2 (22.85%)',
    zaxis_title='PC 3 (3.66%)'
    )
)

fig.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig.update_layout(title_y=0.85, title_x=0.5)
pyo.init_notebook_mode(connected=False)
pyo.plot(fig, auto_open=True)

# Taking the last three eigen Vectors
transformed_x3d2 = transformed_data(x_std,
sorted_eigenVectors, 1, 4)
transformed_x3d2.shape
trace1 = go.Scatter3d(
    x = transformed_x3d2[y==0][:, 0],
    y = transformed_x3d2[y==0][:, 1],
    z= transformed_x3d2[y==0][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color="light green"
    ),
    name="Setosa"

)

trace2 =go.Scatter3d(
    x = transformed_x3d2[y==1][:, 0],
    y = transformed_x3d2[y==1][:, 1],
    z= transformed_x3d2[y==1][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="orange"
    ),
    name ="Versicolor"
)

trace3 = go.Scatter3d(
    x = transformed_x3d2[y==2][:, 0],
    y = transformed_x3d2[y==2][:, 1],
    z= transformed_x3d2[y==2][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="light blue"
    ),
    name ="Virginica"
)

trace4 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace5 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)

fig = go.Figure(data=[trace1, trace2, trace3,
trace4, trace5])

fig.update_layout(
    title="PCA of Last 3 components on Iris
dataset",
        margin=dict(
        l=10,
        r=10,
        b=10,
        t=10,
```

```
        pad=4
    ),
    scene=dict(
    xaxis_title='PC 2 (22.85%)',
    yaxis_title='PC 3 (3.66%)',
    zaxis_title='PC 4 (0.51%)'
    )
)

fig.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig.update_layout(title_y=0.85, title_x=0.5)
pyo.init_notebook_mode(connected=False)
pyo.plot(fig, auto_open=True)


# PCA from the library
from sklearn.decomposition import PCA
pca_from_lib_2d = PCA(n_components=2,
random_state=20)
pca_from_lib_2d.fit(x_std)
new_x2d = pca_from_lib_2d.transform(x_std)
variance_proportions =
pca_from_lib_2d.explained_variance_ratio_
variance_proportions
pcaeigenvalues =
pca_from_lib_2d.explained_variance_
pcaeigenvectors = pca_from_lib_2d.components_
targetNames = iris.target_names
targetNames = list(targetNames) + members_roll
plt.figure(figsize=(10, 7))
y = iris.target
for i, targetName in zip([0,1,2,3,4],
targetNames):
    plt.scatter(new_x2d[y==i, 0], new_x2d[y==i,
1],label=targetName, s=20)
plt.xlabel("Principal Component 1 (72.96%)")
plt.ylabel("Principal Component 2 (22.85%)")
plt.legend()
plt.title('Scikit Learn PCA of 2 components on
Iris Dataset')
plt.show()
pca_from_lib3d = PCA(n_components=4,
random_state=20)
pca_from_lib3d.fit(x_std)
new_x3d = pca_from_lib3d.transform(x_std)
variance_proportions_3 =
pca_from_lib3d.explained_variance_ratio_
pca3eigenvalues =
pca_from_lib3d.explained_variance_
pca3eigenvectors = pca_from_lib3d.components_
trace1_1 = go.Scatter3d(
    x = transformed_x3d1[y==0][:, 0],
    y = transformed_x3d1[y==0][:, 1],
    z= transformed_x3d1[y==0][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color="light green"
    ),
    name="Setosa"

)

trace2_1 =go.Scatter3d(
    x = transformed_x3d1[y==1][:, 0],
    y = transformed_x3d1[y==1][:, 1],
    z= transformed_x3d1[y==1][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="orange"
    ),
    name ="Versicolor"
)

trace3_1 = go.Scatter3d(
    x = transformed_x3d1[y==2][:, 0],
    y = transformed_x3d1[y==2][:, 1],
    z= transformed_x3d1[y==2][:, 2],
    mode = "markers",
    marker = dict(
        size=5,
        symbol="circle",
        color ="light blue"
    ),
    name ="Virginica"
)

trace4_1 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace5_1 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=5,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)


fig1 = go.Figure(data=[trace1_1, trace2_1,
trace3_1, trace4_1, trace5_1])

fig1.update_layout(
    title="Scikit Learn PCA of First 3 components
on iris dataset",
        margin=dict(
        l=10,
        r=10,
        b=10,
        t=10,
        pad=4
    ),
    scene=dict(
    xaxis_title='PC 1 (72.96%)',
    yaxis_title='PC 2 (22.85%)',
    zaxis_title='PC 3 (3.66%)'
    )
)

fig1.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig1.update_layout(title_y=0.85, title_x=0.5)
pyo.init_notebook_mode(connected=False)
pyo.plot(fig1, auto_open=True)
```

C. DIABETES DETECTION DATASET

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

# Loading the diabetes dataset
dataset =
pd.read csv("diabetes prediction dataset.csv")
dataset.shape
X = dataset.iloc[:,8].values

print(X.mean())
print(X.std())

# Changing the values of smoking into 0 , 1 and 2
smoking histor dict =
{'never':0,'former':1,'current':2}

dataset['smoking_history'] =
dataset.smoking history.map(smoking histor dict)
dataset.head()

# Making the Female 0 and Male 1
gender_dict = {'Female': 0, 'Male': 1}
dataset['gender'] =
dataset.gender.map(gender_dict)
dataset.isnull().sum()

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')

# Filling the NaN values with mean
dataset = imputer.fit_transform(dataset)

target = dataset[:,8]
target = pd.DataFrame(target,columns=['diabetes'])
X = dataset[:,0:8]
X.shape

# Feature names to columns
features = pd.DataFrame(X,columns=['gender',
'age', 'hypertension', 'heart disease',
'smoking_history', 'bmi', 'HbA1c_level',
'blood glucose level'])
features.head(50)

dataset = pd.DataFrame(dataset,columns=['gender',
'age', 'hypertension', 'heart disease',
'smoking history', 'bmi', 'HbA1c level',
'blood_glucose_level','diabetes'])


print("The mean and the standard deviations are")
X.mean(), X.std()

# Standardize the inital data and calculate the
mean and standard deviation
def standardize(x):
    x_std = StandardScaler().fit_transform(x)
    return x_std, x_std.mean(), x_std.std()

#Calculate the covariance matrix
def covariance_matrix(x_std):
    return np.cov(x_std.T)

#Calculate the eigen values and vectors
def eigVecVal(sx):
    eigVal, eigVec = np.linalg.eig(sx)
    return eigVal, eigVec

# Sorts the eigen vectors and eigen values
def sort_eigen(eigenvalues, eigenvectors):
    sorted_indices = np.argsort(eigenvalues)[::-1]
# Sort indices in descending order
    sorted_eigenvalues =
eigenvalues[sorted_indices] #Sorts the eigenValues
    sorted eigenvectors = eigenvectors[:,
sorted indices] # Sorts the column according to
the indices
    return sorted_eigenvalues, sorted_eigenvectors

# gives the proportion of variance explained by a
particular eigen value/ pricipal component in the
dataset
def explained_variance(eigenvalues):
    sum = eigenvalues.sum()
    return (eigenvalues/sum) * 100


# For the start parameter give the index of the
first eigenvalue and for the end give the index +
1
def transformed_data(standardized_data, eigVec,
start, end):
    transformed x = np.matmul(eigVec.T[start:end],
standardized data.T).T
    return transformed_x

X_std, X_std_mean, X_std_standard_deviation =
standardize(X)
print("The new mean and standard deviation after
standardization are:")
X_std_mean, X_std_standard_deviation

# The covariance matrix is
Sx = covariance_matrix(X_std)

# Calculating the eigen values and eigen vectors
eigenValues, eigenVectors = eigVecVal(Sx)
sorted eigenValues, sorted eigenVectors =
sort_eigen(eigenValues, eigenVectors)


# Gives the proportion of eigen vectors
variance eigen =
explained_variance(sorted_eigenValues)

# Plotting the PCA of First 2 components on
Diabetes Dataset
members_roll = ["THA076BCT017", "THA076BCT022"]
# Taking the top 2 eigen values
transformed x = transformed data(X std,
sorted_eigenVectors, 0, 2)
targetNames = ["No Diabetes", "Diabetes"]
targetNames = list(targetNames) + members_roll
plt.figure(figsize=(10, 7))
for i, targetName in zip([0,1,2,3,4],
targetNames):
    plt.scatter(transformed x[dataset['diabetes']
== i][:, 0], transformed_x[dataset['diabetes'] ==
i][:, 1],label=targetName, s=2)
plt.xlabel("Principal Component 1 (21.50%)")
plt.ylabel("Principal Component 2 (13.80)")
plt.legend()
plt.title('PCA of First 2 components on Diabetes
Dataset')
plt.show()


transformed_low = transformed_data(X_std,
sorted eigenVectors, 6, 8)

# Plotting the PCA of last 2 components of the
Diabetes Datset
targetNames = ["No Diabetes", "Diabetes"]
```

```
targetNames = list(targetNames) + members_roll
plt.figure(figsize=(10, 7))
for i, targetName in zip([0,1,2,3,4],
targetNames):

plt.scatter(transformed_low[dataset['diabetes'] ==
i][:, 0], transformed_low[dataset['diabetes'] ==
i][:, 1],label=targetName, s=2)
plt.xlabel("Principal Component 7 (10.41%)")
plt.ylabel("Principal Component 8 (7.52%)")
plt.legend()
plt.title('PCA of Last 2 components on Diabetes
Dataset')
plt.show()


# Taking three eigen vectors with highest eigen
values and variance
transformed_x1 = transformed_data(X_std,
sorted_eigenVectors, 0, 3)

import plotly.graph_objects as go
import plotly.offline as pyo

# Plotting the PCA of 3 highest components on
Diabetes Dataset
trace1 = go.Scatter3d(
    x = transformed_x1[dataset['diabetes'] ==
0][:, 0],
    y = transformed_x1[dataset['diabetes'] ==
0][:, 1],
    z= transformed_x1[dataset['diabetes'] == 0][:,
2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color="light green"
    ),
    name="No Diabetes"

)

trace2 =go.Scatter3d(
    x = transformed_x1[dataset['diabetes'] ==
1][:, 0],
    y = transformed_x1[dataset['diabetes'] ==
1][:, 1],
    z= transformed_x1[dataset['diabetes'] == 1][:,
2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color ="orange"
    ),
    name ="Diabetes"
)

trace3 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace4 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)


fig = go.Figure(data=[trace1, trace2, trace3,
trace4])

fig.update_layout(
    title = "PCA of First 3 components on Diabetes
Dataset",
    scene=dict(
    xaxis_title='PC 1 (21.5%)',
    yaxis_title='PC 2 (13.8%)',
    zaxis_title='PC 3 (13.22%)'
))

fig.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig.update_layout(title_y=0.85, title_x=0.5)

# Opens the plot in new tab as a HTML
pyo.init_notebook_mode(connected=False)
pyo.plot(fig, auto_open=True)


# Plotting the PCA of lowest three eigen vectors
transformed_x2 = transformed_data(X_std,
sorted_eigenVectors, 5, 8)

trace1 = go.Scatter3d(
    x = transformed_x2[dataset['diabetes'] ==
0][:, 0],
    y = transformed_x2[dataset['diabetes'] ==
0][:, 1],
    z= transformed_x2[dataset['diabetes'] == 0][:,
2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color="light green"
    ),
    name="No Diabetes"

)

trace2 =go.Scatter3d(
    x = transformed_x2[dataset['diabetes'] ==
1][:, 0],
    y = transformed_x2[dataset['diabetes'] ==
1][:, 1],
    z= transformed_x2[dataset['diabetes'] == 1][:,
2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color ="orange"
    ),
    name ="Diabetes"
)

trace3 = go.Scatter3d(
```

```
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace4 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)


fig2 = go.Figure(data=[trace1, trace2, trace3,
trace4])

fig2.update_layout(
    title = "PCA of last 3 components on Diabetes
Dataset",
    scene=dict(
    xaxis_title='PC 6 (10.56%)',
    yaxis_title='PC 7 (10.41%)',
    zaxis_title='PC 8 (7.52%)'
))

fig2.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig2.update_layout(title_y=0.85, title_x=0.5)

pyo.init_notebook_mode(connected=False)
pyo.plot(fig2, auto_open=True)


# Calculating PCA from scikit-learn library
from sklearn.decomposition import PCA
pca_from_lib_2d = PCA(n_components=3,
random_state=20)
pca_from_lib_2d.fit(X_std)
new_x2d = pca_from_lib_2d.transform(X_std)

variance_proportions =
pca_from_lib_2d.explained_variance_ratio_
pcaeigenvalues =
pca_from_lib_2d.explained_variance
pcaeigenvectors = pca_from_lib_2d.components_

# Plotting Scikit Learn PCA on Diabetes Dataset
with 3 Components
targetNames = ["No Diabetes", "Diabetes"]
targetNames = list(targetNames) + members_roll
plt.figure(figsize=(10, 7))
for i, targetName in zip([0,1,2,3,4],
targetNames):
    plt.scatter(new_x2d[dataset['diabetes'] ==
i][:, 0], new_x2d[dataset['diabetes'] == i][:,
1],label=targetName, s=2)
plt.xlabel("Principal Component 1 (21.5%)")
plt.ylabel("Principal Component 2 (13.8%)")
plt.legend()
plt.title('Scikit Learn PCA on Diabetes Dataset
with 2 Components')
plt.show()
pca_from_lib3d = PCA(n_components=3,
random_state=20)
pca_from_lib3d.fit(X_std)
new_x3d = pca_from_lib3d.transform(X_std)
variance_proportions_3 =
pca_from_lib3d.explained_variance_ratio_
variance_proportions_3
trace1_3 = go.Scatter3d(
    x = new_x3d[dataset['diabetes'] == 0][:, 0],
    y = new_x3d[dataset['diabetes'] == 0][:, 1],
    z= new_x3d[dataset['diabetes'] == 0][:, 2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color="light green"
    ),
    name="No Diabetes"

)

trace2_3 =go.Scatter3d(
    x = new_x3d[dataset['diabetes'] == 1][:, 0],
    y = new_x3d[dataset['diabetes'] == 1][:, 1],
    z= new_x3d[dataset['diabetes'] == 1][:, 2],
    mode = "markers",
    marker = dict(
        size=2,
        symbol="circle",
        color ="orange"
    ),
    name ="Diabetes"
)

trace3_3 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="red"
    ),
    name="THA076BCT017"
)

trace4_3 = go.Scatter3d(
    x=[None],
    y=[None],
    z=[None],
    mode="markers",
    marker=dict(
        size=2,
        symbol="circle",
        color="purple"
    ),
    name="THA076BCT022"
)


fig3 = go.Figure(data=[trace1_3, trace2_3,
trace3_3, trace4_3])

fig3.update_layout(
    title = "Scikit Learn PCA on Diabetes Dataset
with 3 Components",
    scene=dict(
    xaxis_title='PC 1 (21.5%)',
    yaxis_title='PC 2 (13.8%)',
```

```
    zaxis_title='PC 3 (13.22%)'
))

fig3.update_layout(legend=dict(yanchor="top",
y=0.85, xanchor="left", x=0.7))
fig3.update_layout(title_y=0.85, title_x=0.5)

pyo.init_notebook_mode(connected=False)
pyo.plot(fig3, auto_open=True)
```