Date of submission June 05, 2024

Lab 1

Principal Component Analysis

ATUL SHREEWASTAV (THA077BCT013)[1], OM PRAKASH SHARMA (THA077BCT030)[2]
[1,2]Department of Electronics and Computer Engineering, Thapathali Campus, Institute of Engineering, Kathmandu 44600 NP)
[1]e-mail: atul.077bct013@tcioe.edu.np
[2]e-mail: om.077bct030@tcioe.edu.np

For the fulfillment of Lab Requirement, *CT72502 Data Mining*

ABSTRACT This study explores the application of Principal Component Analysis (PCA) on two distinct datasets: a randomly generated dataset using NumPy and the Computer Hardware dataset from the UCI Machine Learning Repository. The primary objective was to understand the effectiveness of PCA in reducing dimensionality and extracting meaningful patterns from both synthetic and real-world data. The analysis involved standardizing the datasets, performing PCA, and visualizing the results to interpret the principal components. The findings demonstrate the capability of PCA to highlight key features and underlying structures within the data, providing valuable insights for both synthetic and real-world scenarios.

INDEX TERMS Computer Hardware Dataset, Data Analysis, Data Transformation, Dimensionality Reduction, Feature Extraction, Machine Learning, Principal Component Analysis (PCA), Random Data Generation, UCI Machine Learning Repository

I. INTRODUCTION

As datasets grow in complexity and size, the challenges of managing and analyzing high-dimensional data become increasingly pronounced. The phenomenon known as the "curse of dimensionality" highlights these issues, where the volume of data required for statistically significant results rises exponentially with the number of features or dimensions. This leads to problems such as overfitting, increased computational burden, and reduced model accuracy, particularly in machine learning applications. The exponential growth in possible feature combinations makes it computationally expensive to perform tasks like clustering or classification effectively.

To mitigate the curse of dimensionality, various feature engineering techniques, including feature selection and feature extraction, are employed. Dimensionality reduction, a type of feature extraction, aims to decrease the number of input features while retaining as much of the original information as possible. Among the most popular dimensionality reduction techniques is Principal Component Analysis (PCA).

Introduced by mathematician Karl Pearson in 1901 [1], Principal Component Analysis (PCA) transforms data from a higher-dimensional space to a lower-dimensional one while maximizing the variance retained in the lower-dimensional representation. PCA achieves this through an orthogonal transformation that converts a set of correlated variables into a set of uncorrelated variables, called principal components. PCA is extensively used in exploratory data analysis and machine learning for building predictive models. As an unsupervised learning technique, it examines the interrelationships among variables without prior knowledge of target variables, making it a versatile tool in various analytical contexts. The primary goal of PCA is to reduce the dimensionality of a dataset while preserving its most significant patterns and relationships.

PCA identifies a set of orthogonal axes, termed principal components, which capture the maximum variance in the data. These components are linear combinations of the original variables and are ordered by the amount of variance they capture, from the highest to the lowest. The first principal component captures the most variance in the dataset, with each subsequent component capturing the maximum variance that is orthogonal to all previous components.

PCA has a wide range of applications, including data visualization, feature selection, and data compression. In data visualization, PCA can reduce high-dimensional data to two or three dimensions, making it easier to visualize and interpret. For feature selection, PCA helps identify the most important variables in a dataset, aiding in the selection of relevant features for model building. In data compression, PCA reduces the size of a dataset while retaining its essential information, facilitating efficient data storage and processing.

In essence, PCA simplifies complex datasets, making them more manageable and understandable. By focusing on the variance of features, PCA ensures that the most informative aspects of the data are retained, enabling more effective anal-

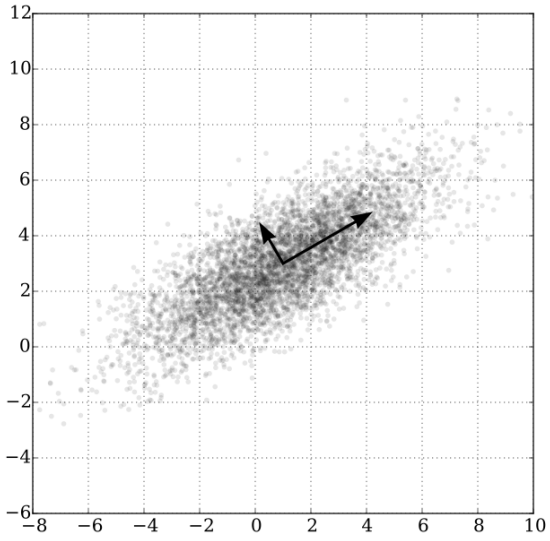

FIGURE 1. PCA of a multivariate Gaussian distribution centered at (1,3) with a standard deviation of 3 in roughly the (0.866, 0.5) direction and of 1 in the orthogonal direction. The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean. [5]

ysis and modeling.

II. RELATED WORKS

In the realm of dimensionality reduction and feature extraction algorithms, Principal Component Analysis (PCA) has been the subject of extensive study. Jolliffe's book, Principal Component Analysis, offers a thorough exploration of PCA, covering its mathematical foundations, practical implementations, and a wide range of applications [2]. Shlens' tutorial paper, "A Tutorial on Principal Component Analysis," further delves into the complexities of PCA, explaining the algorithm, the interpretation of principal components, and its effectiveness in dimensionality reduction [3]. Additionally, the paper by Turk and Pentland, "Eigenfaces for Recognition," examines the use of PCA in face recognition tasks, demonstrating its efficiency in extracting discriminative facial features [4]. Collectively, these works enhance the understanding and assessment of PCA as a robust feature extraction technique across various fields.

III. METHODOLOGY

A. DATASET DESCRIPTION

The dataset contains Relative CPU Performance Data, described in terms of its cycle time, memory size, and other relevant parameters. The dataset aims to provide comprehensive performance metrics for various CPU models from different vendors. The performance metrics were estimated using a linear regression method, as descibed by Feldmesser,Jacob in his paper Computer Hardware [6]. The data was sourced from the UCI Machine Learning Repository (UCIMLRepo), ensuring its credibility and accessibility for research purposes.

The variables used in the database are as follows:

- **VendorName**: List of vendors (e.g., adviser, amdahl, etc.).
- **ModelName**: Unique model symbol for each CPU.
- **MYCT**: Machine Cycle Time (nanoseconds).
- **MMIN**: Minimum Main Memory (kilobytes).
- **MMAX**: Maximum Main Memory (kilobytes).
- **CACH**: Cache Memory (kilobytes).
- **CHMAX**: Maximum Channels (units).
- **CHMIN**: Minimum Channels (units).
- **PRP**: Published Relative Performance.
- **ERP**: Estimated Relative Performance.

The dataset offers detailed insights into the relative performance values of CPUs, allowing for comparison and analysis based on various hardware specifications. The inclusion of both published and estimated relative performance metrics provides a robust framework for evaluating CPU performance across different models and vendors. For more detailed information on the methodology and calculations of the performance metrics, please refer to the original article provided by the authors.

B. PROPOSED METHODOLOGY

Principal Component Analysis (PCA) is a powerful dimensionality reduction technique that transforms a high-dimensional dataset into a lower-dimensional form while preserving as much variance as possible. The process of PCA can be broken down into several key steps.

The first step in PCA is standardization. This involves scaling the dataset such that each feature has a mean of 0 and a standard deviation of 1. This is done to ensure that each feature contributes equally to the analysis and prevents features with larger scales from dominating the results. The standardization formula is $Z = \frac{X-\mu}{\sigma}$, where μ represents the mean of the independent features $\{\mu_1, \mu_2, \cdots, \mu_m\}$ and σ represents the standard deviation of the independent features $\{\sigma_1, \sigma_2, \cdots, \sigma_m\}$.

The second step involves computing the covariance matrix, which measures the strength of the joint variability between two or more variables. This matrix helps to understand how variables in the dataset change together. The covariance between two variables x_1 and x_2 can be calculated using the formula $\text{cov}(x_1, x_2) = \frac{\sum_{i=1}^{n}(x_{1i}-\bar{x}_1)(x_{2i}-\bar{x}_2)}{n-1}$. Covariance values can be positive (indicating that as x_1 increases, x_2 also increases), negative (indicating that as x_1 increases, x_2 decreases), or zero (indicating no direct relationship).

The third step is to compute the eigenvalues and eigenvectors of the covariance matrix, which are essential for identifying the principal components. Given a square $n \times n$ matrix A and a non-zero vector X, the relationship $AX = \lambda X$ holds for some scalar value λ, where λ is the eigenvalue and X is the eigenvector of matrix A. This relationship can be rewritten as $(A - \lambda I)X = 0$, where I is the identity matrix

of the same shape as A. The equation $|A - \lambda I| = 0$ is used to find the eigenvalues. Once the eigenvalues are determined, the corresponding eigenvectors can be found using the equation $AX = \lambda X$.

PCA employs a linear transformation to reduce dimensionality by preserving the maximum variance in the data with the fewest number of dimensions. This involves projecting the data onto the new axes defined by the principal components, which are derived from the eigenvectors of the covariance matrix. The principal components are ordered by the amount of variance they capture, with the first principal component capturing the most variance, and each subsequent component capturing progressively less variance. By focusing on these principal components, PCA simplifies complex datasets, making them easier to analyze and interpret while retaining their most informative aspects.

C. MATHEMATICAL FORMULAE

To perform Principal Component Analysis (PCA) on this dataset, we need to follow several mathematical steps involving linear algebra.

1. Standardize the data

$$\mathbf{z_{ij}} = \frac{\mathbf{x_{ij}} - \mu_\mathbf{j}}{\sigma_\mathbf{j}} \tag{1}$$

2. Calculate the covariance matrix

$$\mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} \mathbf{z}_i \mathbf{z}_i^T \tag{2}$$

3. Compute the eigenvalues and eigenvectors

$$\mathbf{Cv} = \lambda \mathbf{v} \tag{3}$$

4. Sort the eigenvalues and eigenvectors

Sort eigenvalues $(\lambda_1, \lambda_2, \ldots, \lambda_p)$ in descending order, and sort eigenvectors correspondingly.

5. Compute the principal components

$$\mathbf{P} = \mathbf{ZV} \tag{4}$$

6. Calculate Proportion of Variance

The proportion of variance explained by each principal component is given by:

$$\mathbf{ProportionofVariance} = \frac{\lambda_\mathbf{i}}{\sum_{\mathbf{j=1}}^{\mathbf{p}} \lambda_\mathbf{j}} \tag{5}$$

where λ_i is the eigenvalue of the i-th principal component, and p is the total number of principal components.

D. INSTRUMENTATION DETAILS

In this study, Principal Component Analysis was conducted using Python, leveraging libraries such as NumPy, Pandas, Seaborn, and Matplotlib for data manipulation, visualization, and analysis.

1) Software and Libraries Used

- **Python**:Chosen for its versatility and extensive ecosystem of libraries suitable for data analysis and machine learning tasks.
- **UciMlRepo**:Used to load the dataset as a dataframe in the program directly.
- **NumPy**:UUtilized for numerical operations and efficient manipulation of large datasets. NumPy provided the necessary functions to perform all calculations related to PCA, including matrix operations, eigenvalue decomposition, and data standardization.
- **Pandas**:Utilized for data manipulation and analysis, offering powerful data structures like DataFrames that simplify the process of cleaning and preparing data.
- **Seaborn**:Employed for statistical data visualization, making it easier to create informative and attractive visual representations of the data and PCA results.
- **Matplotlib**:Used alongside Seaborn to customize and enhance plots, providing fine-grained control over the visual elements of the graphs.

2) Data Preparation

- **Dataset Acquisition:** The Computer Hardware dataset was downloaded from the UCI Machine Learning Repository to ensure a reliable and standardized dataset for analysis.
- **Data Loading:** The dataset was loaded into a Pandas DataFrame to facilitate efficient data manipulation and analysis.

3) Data Preprocessing

- **Handling Missing Values:** The dataset was checked for missing values, and appropriate imputation methods were applied if necessary to maintain data integrity
- **Feature Selection:** Relevant features were selected for PCA, excluding non-numeric and irrelevant columns to focus on the most significant variables.

4) Principal Component Analysis

- **Standardization:**The selected features were standardized to have a mean of 0 and a standard deviation of 1, which is essential for PCA to ensure that each feature contributes equally to the analysis.
- **PCA Implementation:**PCA was performed using NumPy. The steps involved were:
  - Computing the covariance matrix of the standardized data.
  - Performing eigenvalue decomposition on the covariance matrix to obtain eigenvalues and eigenvectors.
  - Sorting eigenvalues and their corresponding eigenvectors to identify the principal components.
  - Transforming the original data into the new principal component space using the selected eigenvectors.

- **Explained Variance:**The explained variance ratio of the principal components was computed to understand the variance captured by each component, aiding in the interpretation of the PCA results.

5) Data Visualization

- **Pairplot:** A pairplot of the selected features was created using Seaborn to visualize the relationships and interactions between different variables. The pairplot provided insights into the data distribution, potential correlations, and patterns among the features.
- **Scatter Plot:** A scatter plot of the first two principal components was created using Seaborn and Matplotlib to visualize the data distribution and identify potential patterns or clusters.

6) Development Environment

- **Jupyter Notebook**: The Jupyter Notebook environment was used for developing and running the Python scripts, enabling an interactive and iterative approach to data analysis.

E. SYSTEM BLOCK DIAGRAM

The block diagram as shown in 2 for Principal Component Analysis (PCA) outlines the main steps in the PCA process in a sequential manner. It begins with the input dataset, which consists of m samples and n features. The first step involves centering the data by subtracting the mean of each feature from the dataset, resulting in a centered data matrix. This step ensures that each feature has a mean of zero. Following this, the centered data undergoes standardization, which scales the features so that they have unit variance, ensuring that all features contribute equally to the analysis, regardless of their original scales.

The standardized data is then used to compute the covariance matrix, which measures the pairwise covariances between features and has dimensions $n \times n$. Eigenvalues and eigenvectors are calculated from this covariance matrix. The eigenvalues represent the proportion of variance explained by each principal component, while the eigenvectors represent the directions of these components.

The next step involves sorting the eigenvectors based on their corresponding eigenvalues in descending order. The top p eigenvectors are selected to form the principal components, resulting in a matrix of size $n \times p$, where p is the number of principal components chosen. The centered data matrix (of size $m \times n$) is then projected onto these principal components by taking the dot product, resulting in a new matrix, often referred to as the score matrix or PC scores, of size $m \times p$. The final output is the matrix of principal component scores, where each sample is represented by its projections onto the principal components. This process transforms the original dataset into a new space defined by the principal components, effectively reducing the dimensionality while retaining most of the variance in the data.

IV. EXPERIMENTAL RESULTS

A. PCA ON RANDOM DATASET

1) Data Standardization

The original features of the dataset were standardized to have a mean of 0 and a standard deviation of 1. This step was crucial to ensure that each feature contributed equally to the PCA, avoiding any bias due to differences in the scale of the features.

2) Explained Variance

The PCA computation revealed the following explained variance ratios for the first two principal components:

Principal Component 1 (PC1): (99.1%) Principal Component 2 (PC2): (0.8%) PC1 Component itself is sufficient to describe the data, which consist of more than 99% of the contribution in the final data.

3) Principal Component Visualization

Almost all of the data was shown to be aligned along the PC1, so PC2 can be ignored.

B. PCA ON UCIML DATASET

This section presents the findings from the Principal Component Analysis (PCA) performed on the Computer Hardware dataset. The results are discussed in terms of data standardization, explained variance, principal component visualization, and pairplot analysis.

1) Data Standardization

The original features of the dataset were standardized to have a mean of 0 and a standard deviation of 1. This step was crucial to ensure that each feature contributed equally to the PCA, avoiding any bias due to differences in the scale of the features.

2) Explained Variance

The PCA computation revealed the following explained variance ratios for the first three principal components:

Principal Component 1 (PC1): (55.9%) Principal Component 2 (PC2): (66.8%) Principal Component 3 (PC3): (82.1%) Together, these three components explained approximately 83% of the total variance in the dataset. This indicates that a substantial portion of the variability in the original data could be captured by these two principal components.

3) Principal Component Visualization

The scatter plot of the first two principal components (PC1 and PC2) provided a visual representation of the dataset in the reduced dimensional space. The plot, color-coded by the vendor name, highlighted potential patterns and clusters within the data. The separation of data points suggested that certain vendors might have distinctive hardware characteristics that were captured by the principal components.

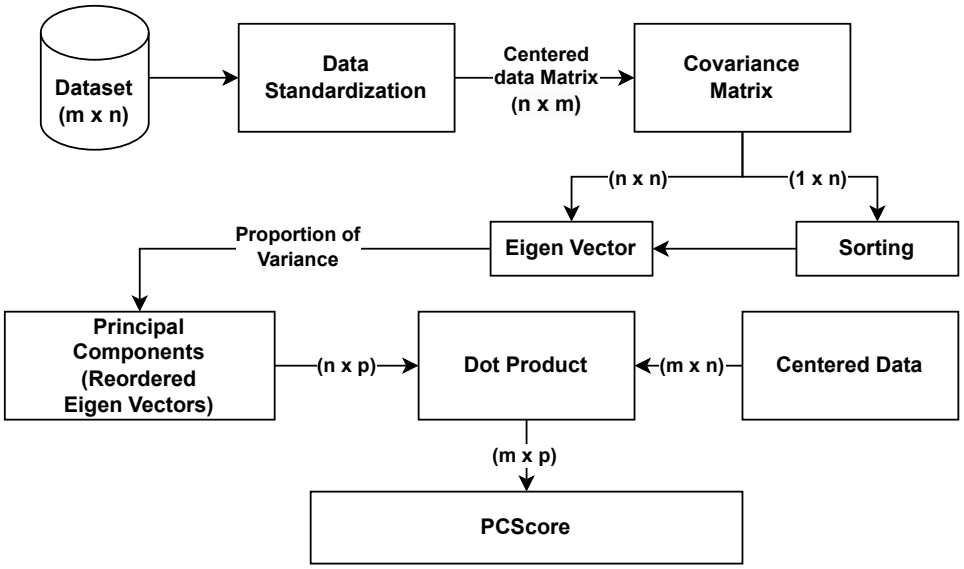

FIGURE 2. Block Diagram for Principal Component Analysis

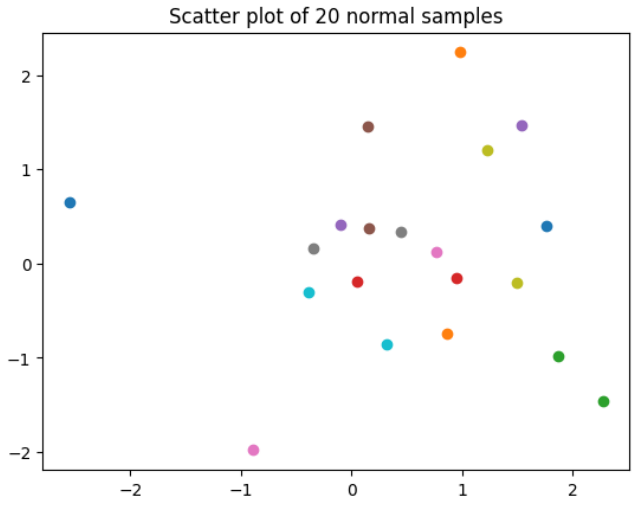

FIGURE 3. Scatter Plot of 20 Normal Samples

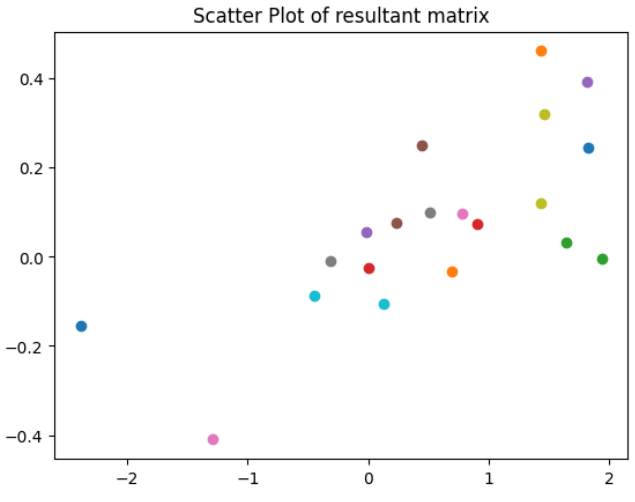

FIGURE 4. Scatter Plot of the resultant marix after PCA

4) Pairplot Analysis

To further explore the relationships between the original features, a pairplot was created. This visual representation displayed scatter plots for each pair of features, along with the distribution of each feature along the diagonal.

The key observations from the pair plot were found to be:

- **Correlation Patterns:** Certain features showed strong linear relationships, indicating high correlation. For example, MMIN (minimum main memory) and MMAX (maximum main memory) exhibited a positive correlation, as expected.
- **Cluster Identification:** The pairplot revealed potential clusters within the data, corresponding to different hardware configurations and performance metrics.
- **Outlier Detection:** Some features displayed outliers, which were further analyzed to understand their impact on the overall PCA results.

5) Analysis of Principal Components

The principal components were analyzed to understand their contributions from the original features. The first principal component (PC1) was found to be heavily influenced by features related to memory (MMIN and MMAX) and cache size (CACH). The second principal component (PC2) was significantly impacted by features related to machine cycle

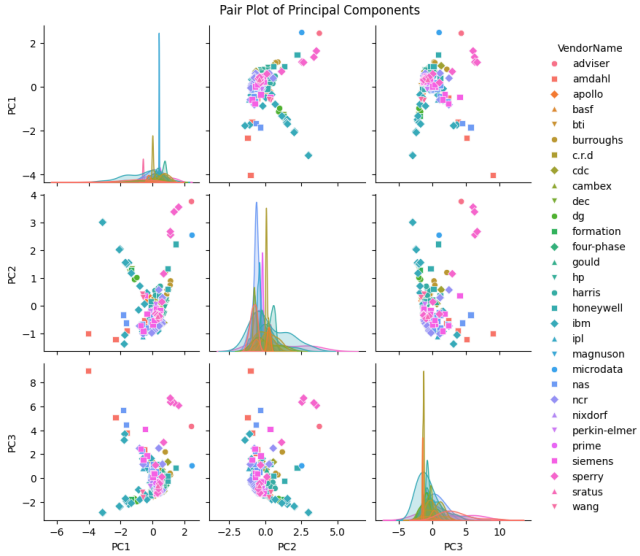

FIGURE 5. Pair Plot of Principal Components

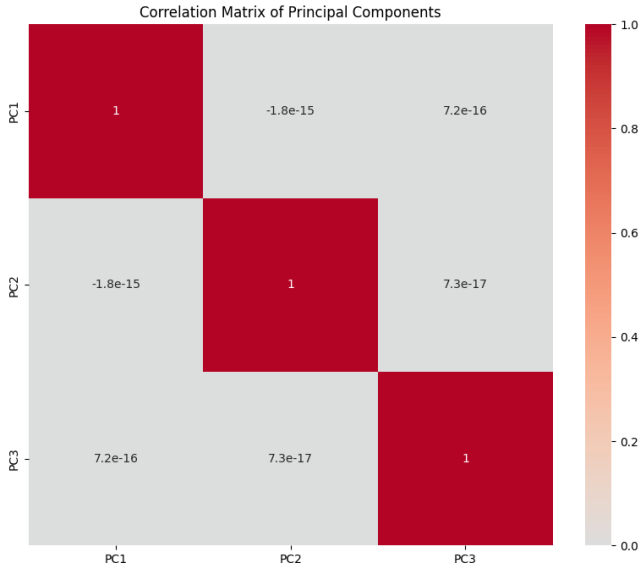

FIGURE 6. Confusion Matrix of Principal Components

time (MYCT) and channel minimum (CHMIN).

V. DISCUSSION AND ANALYSIS

A. RANDOM DATASET

The application of PCA on the randomly generated dataset serves as a baseline for understanding the technique's behavior in the absence of inherent structure. The results indicated that:

- **Uniform Distribution:** The scatter plot of the principal components was uniformly distributed, which is expected given the random nature of the data. This suggests that PCA can effectively reduce dimensionality but may not reveal meaningful patterns in data without intrinsic

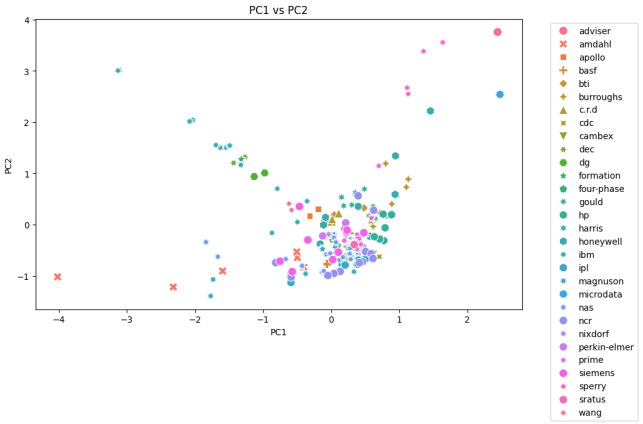

FIGURE 7. PC1 vs PC2

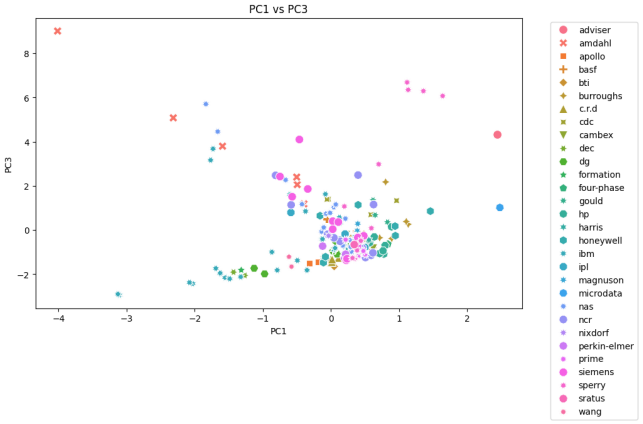

FIGURE 8. PC1 vs PC3

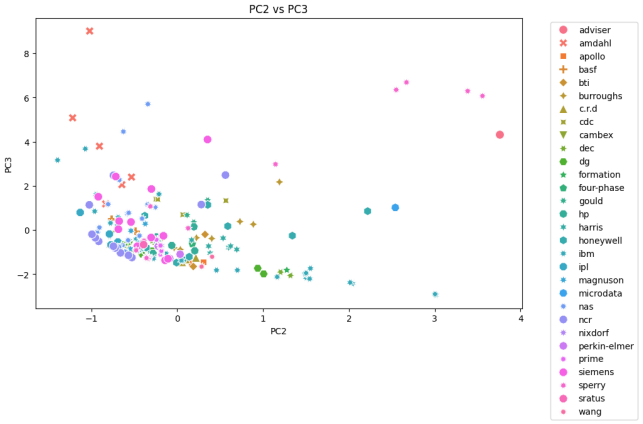

FIGURE 9. PC2 vs PC3

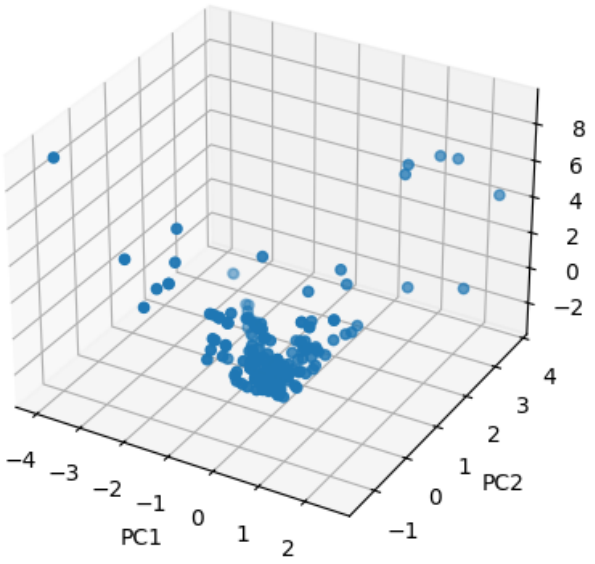

FIGURE 10. 3D Plot for PC1, PC2 and PC3

structure.

- **Variance Capture:** The first two principal components captured a significant portion of the variance (om ji%), demonstrating PCA's ability to condense information even in random data.

B. COMPUTER HARDWARE DATASET

The PCA on the Computer Hardware dataset provided more insightful results, reflecting its real-world nature:

- **Variance Explanation:** The first two principal components explained a substantial portion of the variance (om ji%), indicating that these components effectively capture the key variations in the dataset.
- **Cluster Formation:** The scatter plot revealed distinct clusters, suggesting that certain features, such as memory size and cache, are primary drivers of variance and help differentiate between different hardware configurations.
- **Vendor Differentiation:** The color-coded plot by vendor name showed some separation among vendors, implying that PCA can help identify underlying patterns related to specific manufacturers.

C. COMPARATIVE ANALYSIS

The comparison between the random and real-world datasets highlights several points:

- **Pattern Detection:** While the random dataset did not reveal any significant patterns due to its lack of inherent structure, the Computer Hardware dataset showed clear clusters and separations. This underscores PCA's strength in identifying meaningful structures in real-world data.
- **Dimensionality Reduction:** In both datasets, PCA effectively reduced dimensionality while preserving a significant portion of the variance, demonstrating its utility in simplifying complex datasets.

VI. CONCLUSION

The PCA successfully reduced the dimensionality of both a randomly generated dataset and the Computer Hardware dataset from UCI Machine Learning Repository while retaining a significant portion of the variance. The visualization of the principal components and the pairplot analysis provided valuable insights into the relationships and patterns within the data. These findings can aid in further analysis and understanding of the hardware characteristics and their impact on performance metrics.

The experimental results demonstrate the effectiveness of PCA in summarizing and visualizing complex datasets, making it a powerful tool for exploratory data analysis and feature reduction.

REFERENCES

[1] Ma, Yuan Zhe. (2014). A Tutorial on Principal Component Analysis. 10.13140/2.1.1593.1684. G. O. Young, Synthetic structure of industrial
[2] I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. Springer, 2002.
[3] J. Shlens, "A Tutorial on Principal Component Analysis," 2014. [Online]. Available: https://arxiv.org/abs/1404.1100
[4] M. Turk and A. Pentland, "Eigenfaces for recognition," Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991.
[5] By Nicoguaro - Own work, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=46871195.
[6] Feldmesser,Jacob. (1987). Computer Hardware. UCI Machine Learning Repository. https://doi.org/10.24432/C5830D.

ATUL SHREEWASTAV is currently pursuing his undergraduate degree in Computer Engineering at IOE, Thapathali Campus. His research interests encompass various areas, including Machine Learning Techniques, Natural Language Processing, and Computer Vision. Additionally, he is intrigued by automation, attention mechanism in transformers, agile development methodologies, advancement in neural net architectures, linux.(THA077BCT013)

OM PRAKASH SHARMA is currently pursuing his undergraduate degree in Computer Engineering at IOE, Thapathali Campus. His research interests encompass various areas, including computational genomics, cloud computing, and system designing. Additionally, he is intrigued by artificial intelligence, cybersecurity, data analytics, machine learning, Internet of Things (IoT), quantum computing, software development methodologies, network security, computer vision, and natural language processing.(THA077BCT030)

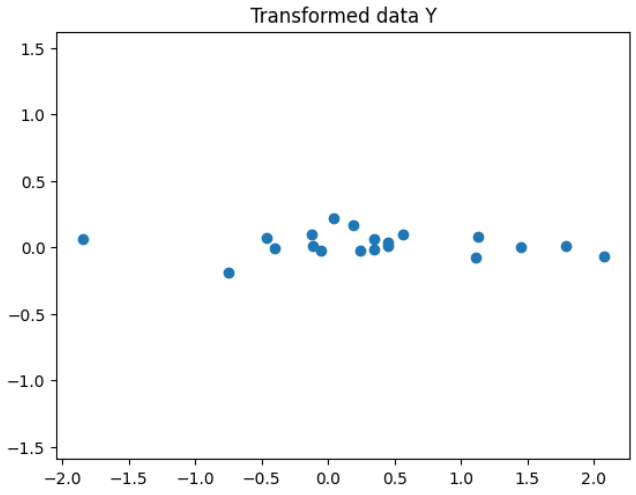

FIGURE 11. Transformed data Y

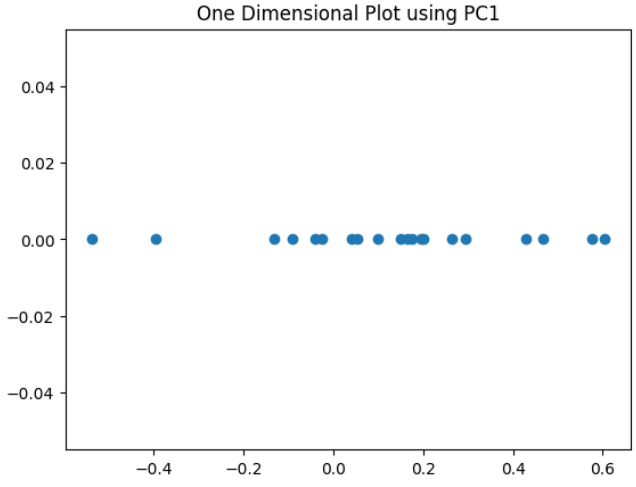

FIGURE 12. One Dimensional Plot using PC1

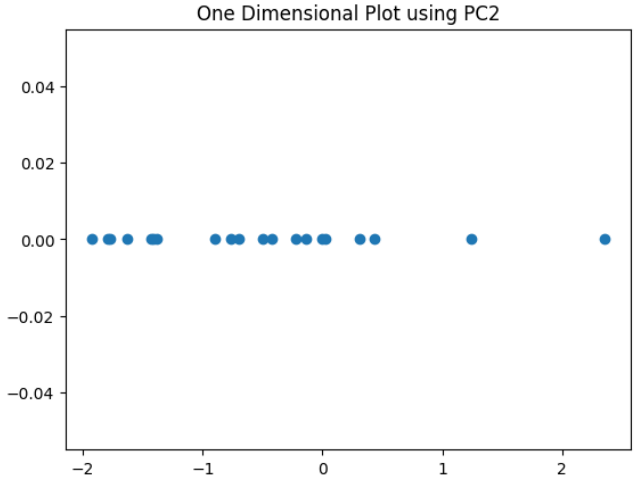

FIGURE 13. One Dimensional Plot using PC2

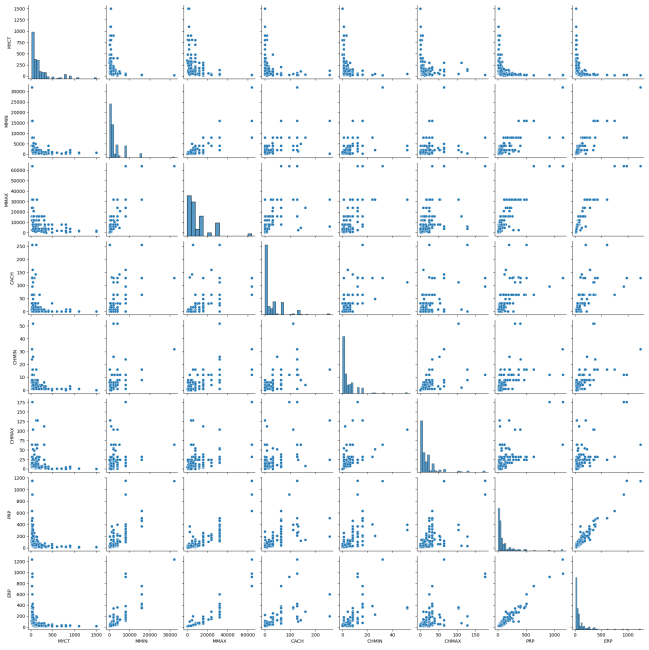

FIGURE 14. Pair Plot of the Original Hardware Data

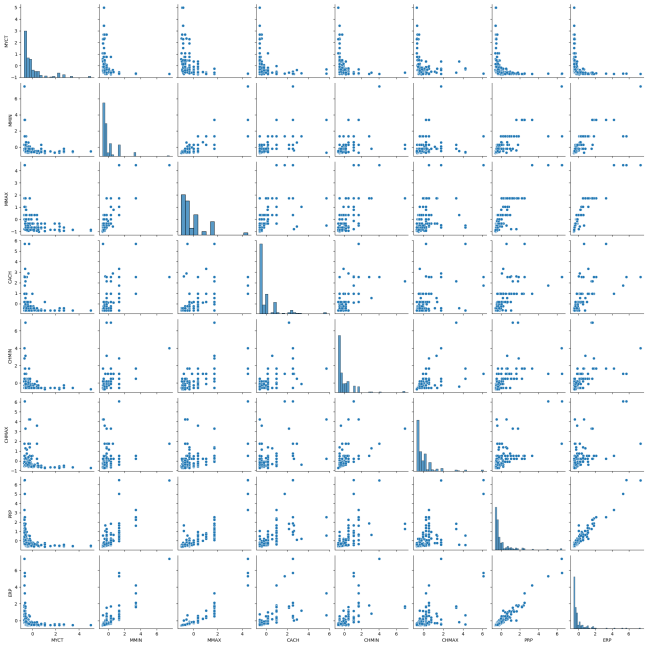

FIGURE 15. Pair Plot of the Standardized Hardware Data

APPENDIX A
RESULTS OF PCA ON RANDOM DATASET
APPENDIX B
RESULTS OF PCA ON HARDWARE DATASET

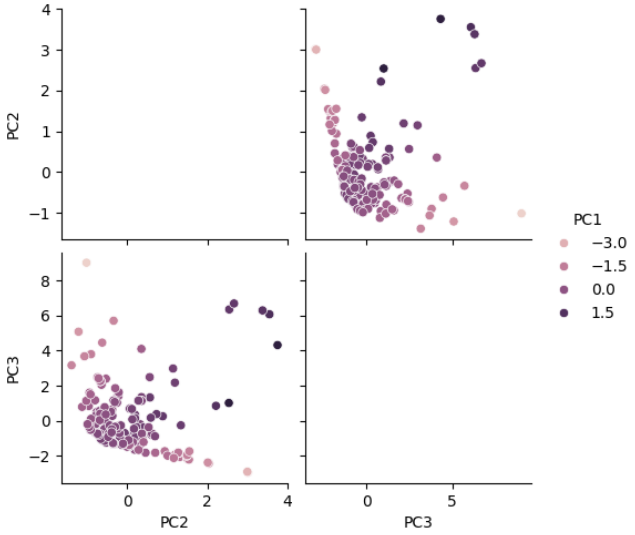

FIGURE 16. Pair Plot of PC2 and PC3 as marker for PC1

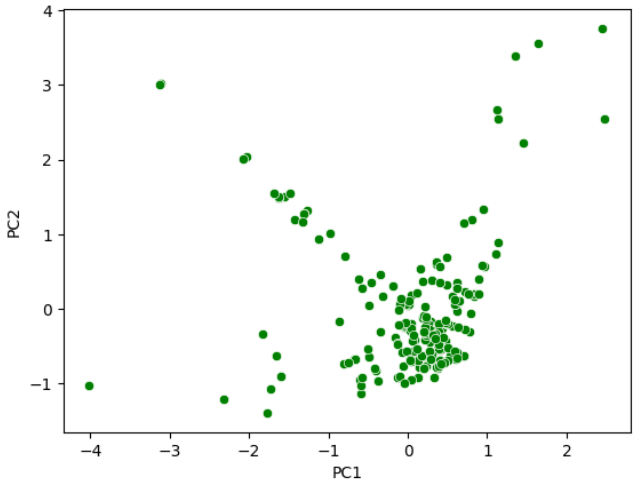

FIGURE 17. Plot of PC1 and PC2

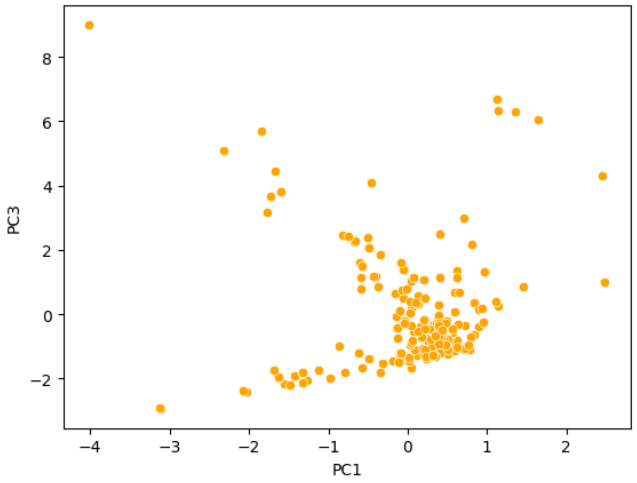

FIGURE 18. Plot of PC1 and PC3

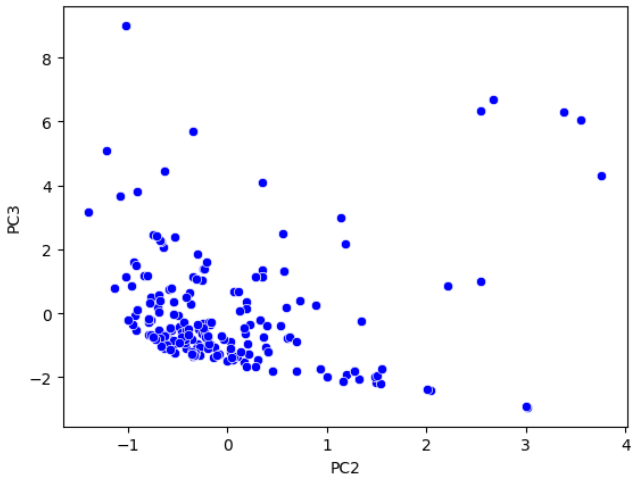

FIGURE 19. Plot of PC2 and PC3

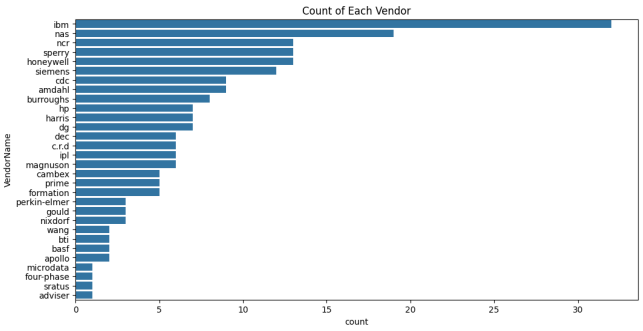

FIGURE 20. Count of Each Vendor in the dataset

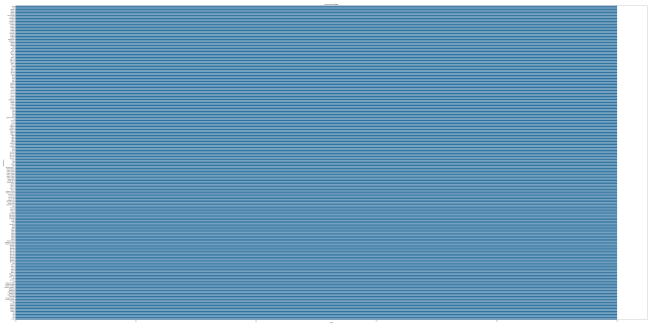

FIGURE 21. Count of Each Model

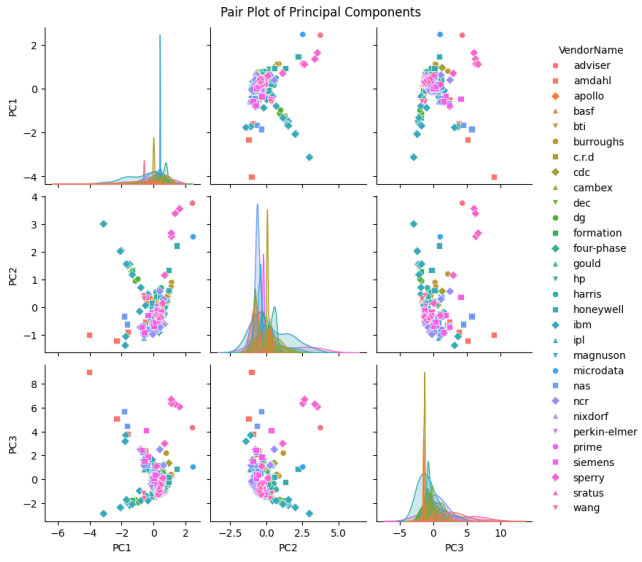

FIGURE 22. Pair Plor of Principal Components

Appendix C: Code for PCA in Random Dataset

Goal: Perform PCA from scratch on a random dataset

This notebook contains the lab solutions of PCA lab for Data Mining prepared by Dinesh Baniya Kshatri.

Author: Atul Shreewastav and Om Prakash Sharma

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Task-1: Initialize Psuedo-Random Number Generator with random seed set to 0

[ ]:
```
np.random.seed(0)
```

Task-2: Sample from a Gaussian distribution an array of 20 by 2 samples

[ ]:
```
samples = np.random.normal( size=(20,2))
print(samples)
```

```
[[ 1.76405235  0.40015721]
 [ 0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788]
 [ 0.95008842 -0.15135721]
 [-0.10321885  0.4105985 ]
 [ 0.14404357  1.45427351]
 [ 0.76103773  0.12167502]
 [ 0.44386323  0.33367433]
 [ 1.49407907 -0.20515826]
 [ 0.3130677  -0.85409574]
 [-2.55298982  0.6536186 ]
 [ 0.8644362  -0.74216502]
 [ 2.26975462 -1.45436567]
 [ 0.04575852 -0.18718385]
 [ 1.53277921  1.46935877]
 [ 0.15494743  0.37816252]
 [-0.88778575 -1.98079647]
 [-0.34791215  0.15634897]
 [ 1.23029068  1.20237985]
 [-0.38732682 -0.30230275]]
```

Task-3: Plot the sample values

[ ]:
```
for i in range(20):
        plt.title("Scatter plot of 20 normal samples")
        plt.scatter(samples[i][0],samples[i][1])
```

Task-4: Create a 2x2 matrix and fill it with random values from a uniform distribution having values between 0 and 1

[ ]:
```
mat = np.random.random(size=(2,2))
mat
```

```
array([[0.98837384, 0.10204481],
       [0.20887676, 0.16130952]])
```

Task-5: Multiply the matrices from Tasks (2) and (4)

[ ]:
```
mult = np.matmul(samples, mat)
print(mult)
```

```
[[ 1.82712673  0.24456155]
 [ 1.43542952  0.46135253]
 [ 1.64171483  0.03293038]
 [ 0.90742753  0.07253623]
 [-0.01625433  0.0557005 ]
 [ 0.44613283  0.24928706]
 [ 0.77760486  0.09728729]
 [ 0.50839962  0.09911878]
 [ 1.43385588  0.11936904]
 [ 0.13102718 -0.10582684]
 [-2.38678261 -0.15508446]
 [ 0.6993651  -0.03150705]
 [ 1.9395829  -0.00298634]
 [ 0.00612817 -0.02552512]
 [ 1.82187377  0.39343372]
 [ 0.23213534  0.07681279]
 [-1.29120655 -0.41011525]
 [-0.3112096  -0.01028205]
 [ 1.46713632  0.31950009]
 [-0.44596771 -0.088289  ]]
```

Task-6: Plot the resulting sample values

[ ]:
```
for i in range(20):
        plt.title("Scatter Plot of resultant matrix")
        plt.scatter(mult[i][0],mult[i][1])
```

Task-7: Compute the variance among the data axes

[ ]:
```
xvar=np.var(mult[0])
print(xvar)

yvar = np.var(mult[1])
print(yvar)
```

```
0.0002642888402092033
0.015330782954394414
```

Task-8: Compute the covariance matrix

[ ]:
```
covx = np.dot(mult.T,mult)
print(covx)
```

```
[[9.73929705 8.64990129]
 [8.64990129 7.96920795]]
```

Task-9: Compute the eigenvalues and eigenvectors of the covariance matrix

[ ]:
```
eigenvalue, eigenvector = np.linalg.eig(covx)
print(eigenvalue)
print(eigenvector)
```

```
[17.54931409  0.15919091]
[[ 0.74222201 -0.67015407]
 [ 0.67015407  0.74222201]]
```

Task-10: Keeping both eigenvectos perform a change of basis

[ ]:
```
eigenvec_T = np.transpose(eigenvector)
Y = np.matmul(eigenvec_T,mult.T)
Y = np.transpose(Y)
```

Task-11: Compute the proportion of variance due to both eigenvectors

[ ]:
```
prop_1= eigenvalue[0]/(eigenvalue[0]+eigenvalue[1])
prop_2= eigenvalue[1]/(eigenvalue[0]+eigenvalue[1])
print(prop_1)
print(prop_2)
```

```
0.991010482856861
0.008989517143138961
```

Task-12: Plot the data values in the new basis

[ ]:
```
plt.title("Transformed data Y")
plt.scatter(Y[:,0],Y[:,1])
plt.axis('equal')
print(Y)
```

```
[[ 1.12994028  0.08067709]
 [ 2.07474114 -0.06946645]
 [ 0.19158331  0.16467453]
 [ 0.34546458  0.06420909]
 [ 0.24513374 -0.02927704]
 [ 1.11175501 -0.07384636]
 [ 0.45092799  0.0376885 ]
 [ 0.45139681  0.00711713]
 [ 0.56683296  0.09911389]
 [-0.46290637  0.06666065]
 [-0.7512339  -0.18650664]
 [-0.119159    0.09262046]
```

```
 [ 0.04163678  0.21522422]
 [-0.11237145  0.01327087]
 [ 1.78619469  0.00663635]
 [ 0.34523987 -0.01232139]
 [-1.84475209  0.06007712]
 [-0.05412874 -0.02922293]
 [ 1.45018537  0.0040257 ]
 [-0.4018824  -0.00558088]]
```

Task-13: Calculate the new covariance matrix

[ ]:
```
covy = np.dot(Y.T,Y)
print(covy)
```

```
[[ 20.34973421 -13.86287965]
 [-13.86287965   9.99328881]]
```

Task-14: Repeat Tasks (10) and (12) using only one eigenvector

[ ]:
```
principal_eigenvector = eigenvector[:,np.argmax(eigenvalue)]
print(principal_eigenvector)
pc_y = np.dot(principal_eigenvector,Y.T)
plt.title("One Dimensional Plot using PC1")
plt.scatter(pc_y, np.zeros(20))
plt.show()
```

```
[0.74222201 0.67015407]
```

[ ]:
```
secondary_eigenvector = eigenvector[:,np.argmin(eigenvalue)]
print(secondary_eigenvector)
pc_y = np.dot(secondary_eigenvector,Y.T)
plt.title("One Dimensional Plot using PC2")
plt.scatter(pc_y, np.zeros(20))
plt.show()
```

```
[-0.67015407  0.74222201]
```

Appendix D: Code for Principal Component Analysis on Computer Hardware Dataset form ucimlrepo

Dataset Description This dataset contains Relative CPU Performance Data, described in terms of its cycle time, memory size, etc.

The estimated relative performance values were estimated by the authors using a linear regression method. See their article https://doi.org/10.24432/C5830D. (pp 308-313) for more details on how the relative performance values were set.

Variable Name	Description	Units
VendorName	List of Vendors(adviser,amdahl,etc.)	-
ModelName	Unique Model Symbol	-
MYCT	Machine Cycle Time	nanoseconds
MMIN	Minimum Main Memory	kilobytes
MMAX	Maximum Main Memory	kilobytes
CACH	Cache Memory	kilobytes
CHMAX	Maximum Channels	units
CHMIN	Minimum Channels	units
PRP	Published Relative Performance	-
ERP	Estimated Relative Performance	-

[1]:
```
from ucimlrepo import fetch_ucirepo
```

[2]:
```
# fetch dataset
computer_hardware = fetch_ucirepo(id=29)

# data (as pandas dataframes)
df = computer_hardware.data.features
```

[3]:
```
df.head()
```

[3]:

	VendorName	ModelName	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
0	adviser	32/60	125	256	6000	256	16	128	198	199
1	amdahl	470v/7	29	8000	32000	32	8	32	269	253
2	amdahl	470v/7a	29	8000	32000	32	8	32	220	253
3	amdahl	470v/7b	29	8000	32000	32	8	32	172	253
4	amdahl	470v/7c	29	8000	16000	32	8	16	132	132

[4]:
```
df.columns
```

[4]:
```
Index(['VendorName', 'ModelName', 'MYCT', 'MMIN', 'MMAX', 'CACH', 'CHMIN',
       'CHMAX', 'PRP', 'ERP'],
      dtype='object')
```

[5]:
```
df.describe()
```

[5]:
```
              MYCT          MMIN          MMAX        CACH       CHMIN  \
count   209.000000    209.000000    209.000000  209.000000  209.000000
mean    203.822967   2867.980861  11796.153110   25.205742    4.698565
std     260.262926   3878.742758  11726.564377   40.628722    6.816274
min      17.000000     64.000000     64.000000    0.000000    0.000000
25%      50.000000    768.000000   4000.000000    0.000000    1.000000
50%     110.000000   2000.000000   8000.000000    8.000000    2.000000
75%     225.000000   4000.000000  16000.000000   32.000000    6.000000
max    1500.000000  32000.000000  64000.000000  256.000000   52.000000

             CHMAX          PRP          ERP
count   209.000000   209.000000   209.000000
mean     18.267943   105.622010    99.330144
std      25.997318   160.830733   154.757102
min       0.000000     6.000000    15.000000
25%       5.000000    27.000000    28.000000
50%       8.000000    50.000000    45.000000
75%      24.000000   113.000000   101.000000
max     176.000000  1150.000000  1238.000000
```

[6]:
```
# check for missing values in each feature
df.isnull().sum()
```

[6]:
```
VendorName    0
ModelName     0
MYCT          0
MMIN          0
MMAX          0
CACH          0
CHMIN         0
CHMAX         0
PRP           0
ERP           0
dtype: int64
```

[131]:
```
# check the datatype of each feature
df.dtypes
df.loc[df['ERP'] == max(df['ERP'])]
# df.loc[df['ERP'] == min(df['ERP'])]
```

[131]:

	VendorName	ModelName	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
9	amdahl	580-5880	23	32000	64000	128	32	64	1144	1238

[132]:
```
df.loc[df['ERP'] == min(df['ERP'])]
```

[132]:

	VendorName	ModelName	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP	ERP
14	bti	5000	350	64	64	0	1	4	10	15
99	ibm	370/125-2	480	96	512	0	1	1	6	15

[8]:
```
# get only numerical features
num_features = df.drop(['VendorName','ModelName'],axis=1)
num_features
```

[9]:
```
num_features.corr()
```

[9]:
```
            MYCT      MMIN      MMAX      CACH     CHMIN     CHMAX       PRP  \
MYCT    1.000000 -0.335642 -0.378561 -0.321000 -0.301090 -0.250502 -0.307099
MMIN   -0.335642  1.000000  0.758157  0.534729  0.517189  0.266907  0.794931
MMAX   -0.378561  0.758157  1.000000  0.537990  0.560513  0.527246  0.863004
CACH   -0.321000  0.534729  0.537990  1.000000  0.582245  0.487846  0.662641
CHMIN  -0.301090  0.517189  0.560513  0.582245  1.000000  0.548281  0.608903
CHMAX  -0.250502  0.266907  0.527246  0.487846  0.548281  1.000000  0.605209
PRP    -0.307099  0.794931  0.863004  0.662641  0.608903  0.605209  1.000000
ERP    -0.288396  0.819292  0.901202  0.648620  0.610580  0.592156  0.966472

            ERP
MYCT  -0.288396
MMIN   0.819292
MMAX   0.901202
CACH   0.648620
CHMIN  0.610580
CHMAX  0.592156
PRP    0.966472
ERP    1.000000
```

[10]:
```
import seaborn as sns

sns.pairplot(data=num_features)
```

[10]:
```
<seaborn.axisgrid.PairGrid at 0x7f2830b3a1b0>
```

0.0.1 Standardize the Data

[11]:
```
import numpy as np
```

[12]:
```
mean = np.mean(num_features, axis=0)
mean
```

[12]:
```
MYCT       203.822967
MMIN      2867.980861
MMAX     11796.153110
CACH        25.205742
CHMIN        4.698565
CHMAX       18.267943
PRP        105.622010
ERP         99.330144
```

```
dtype: float64
```

[13]:
```
std = np.std(num_features, axis=0)
std
```

[13]:
```
MYCT       259.639541
MMIN      3869.452343
MMAX     11698.476757
CACH        40.531407
CHMIN        6.799947
CHMAX       25.935049
PRP        160.445509
ERP        154.386426
dtype: float64
```

[14]:
```
standardized_data = (num_features - mean)/std
standardized_data
```

[14]:
```
          MYCT      MMIN      MMAX      CACH     CHMIN     CHMAX       PRP  \
0    -0.303586 -0.675026 -0.495462  5.694208  1.661989  4.231033  0.575759
1    -0.673330  1.326291  1.727049  0.167629  0.485509  0.529479  1.018277
2    -0.673330  1.326291  1.727049  0.167629  0.485509  0.529479  0.712877
3    -0.673330  1.326291  1.727049  0.167629  0.485509  0.529479  0.413710
4    -0.673330  1.326291  0.359350  0.167629  0.485509 -0.087447  0.164405
..         ...       ...       ...       ...       ...       ...       ...
204  -0.307438 -0.482751 -0.324500 -0.621882 -0.543911 -0.395910 -0.396533
205  -0.407576 -0.482751 -0.324500  0.167629 -0.396851 -0.395910 -0.371603
206  -0.303586 -0.224316 -0.324500 -0.621882 -0.396851 -0.164563 -0.334207
207   1.063694 -0.608867 -0.324500  0.167629 -0.690971 -0.704373 -0.240717
208   1.063694 -0.482751 -0.666425 -0.621882 -0.690971 -0.704373 -0.377836

          ERP
0    0.645587
1    0.995359
2    0.995359
3    0.995359
4    0.211611
..        ...
204 -0.403728
205 -0.319524
206 -0.377819
207 -0.338956
208 -0.481455

[209 rows x 8 columns]
```

[15]:
```
standardized_data.std()
```

[15]:
```
MYCT     1.002401
MMIN     1.002401
MMAX     1.002401
CACH     1.002401
CHMIN    1.002401
CHMAX    1.002401
PRP      1.002401
ERP      1.002401
dtype: float64
```

[16]:
```
sns.pairplot(data=standardized_data)
```

[16]:
```
<seaborn.axisgrid.PairGrid at 0x7f281e078ef0>
```

0.0.2 Insights From Pairplot

We can observe linear relationship between PRP and ERP so those two features will be optimum for performing PCA

[40]:
```
data = standardized_data
data = data.drop(["PRP", "ERP"], axis=1) # ERP is the class attribute
data
```

[41]:
```
#sns.scatterplot(x='PRP', y='ERP', data=data)
```

[42]:
```
cov_matrix = np.cov(data, rowvar=False)
# for i in range(cov_matrix.shape[0]):
#     for j in range(cov_matrix.shape[1]):
#         print(f"{cov_matrix[i,j]:.2f}", end=" ")
#     print()
```

[43]:
```
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
print(f"Eigen Values:\n{eigenvalues}")
print(f"\nEigen Vectors:\n{eigenvectors}")
eigenvectors.shape
```

```
Eigen Values:
[0.17475717 0.40636886 0.49871024 0.74277939 0.83334797 3.37288253]

Eigen Vectors:
[[-0.02708287  0.04523943  0.02695622 -0.66860232  0.68218424 -0.28998081]
 [ 0.63140861 -0.0083375   0.08818826 -0.5476845  -0.33298195  0.42736541]
 [-0.67176548  0.1352712   0.47695161 -0.26428174 -0.11408717  0.46913674]
 [-0.1208669   0.51868367 -0.71365487 -0.01993454  0.15163789  0.42856007]
 [-0.09879599 -0.81213824 -0.25459238  0.03019271  0.27463756  0.43533454]
 [ 0.35348461  0.22579707  0.4357669   0.42644906  0.55884878  0.37416675]]
```

[43]:
```
(6, 6)
```

[44]:
```
np.dot(eigenvectors[0],eigenvectors[1])
```

[44]:
```
1.6653345369377348e-16
```

[45]:
```
descending_eigenvalues = eigenvalues[::-1]
```

[46]:
```
descending_eigenvalues
```

[46]:
```
array([3.37288253, 0.83334797, 0.74277939, 0.49871024, 0.40636886,
       0.17475717])
```

[47]:
```
sum_eigenvalues = np.sum(descending_eigenvalues)
```

[48]:
```
for i in range(1,len(descending_eigenvalues)+1):
    sum_of_i_eigen = np.sum(descending_eigenvalues[:i])
    print(f"Proportion of {i} = {sum_of_i_eigen/sum_eigenvalues}")
```

```
Proportion of 1 = 0.5594573887493591
Proportion of 2 = 0.6976841653838327
Proportion of 3 = 0.8208884022552221
Proportion of 4 = 0.9036090801353474
Proportion of 5 = 0.9710131649120174
Proportion of 6 = 1.0
```

The first three principal components corresponds to more than 70% of the proportions. So, the other principal components can be removed.

[49]:
```
feature_vector = eigenvectors[:,-3:]
feature_vector = feature_vector.transpose()
data = data.transpose()
data
```

[50]:
```
final_data = np.matmul(feature_vector, data)
final_data
```

[51]:
```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
final_data = final_data.transpose()
```

[52]:
```
fig = plt.figure(figsize=(10,10))
# plot final_data in a 3d plot
```

```
<Figure size 1000x1000 with 0 Axes>
```

[53]:
```
ax = plt.axes(projection="3d")
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
ax.set_zlabel("PC3")
ax.scatter(final_data[0], final_data[1], final_data[2])
```

[53]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f280f16aba0>
```

[54]:
```
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.scatter(final_data[0],final_data[1],c=final_data[2],cmap='rainbow')
```

[54]:
```
<matplotlib.collections.PathCollection at 0x7f280f188fb0>
```

[57]:
```
final_data.rename(columns={0:'PC1',1:'PC2',2:'PC3'}, inplace=True)
final_data
```

[57]:

	PC1	PC2	PC3
0	2.444610	3.758606	4.314055
1	-0.495514	-0.643343	2.053597
2	-0.495514	-0.643343	2.053597
3	-0.495514	-0.643343	2.053597
4	-0.397143	-0.832075	1.181126
..	...	...	...
204	0.382847	-0.476893	-0.920828
205	0.438502	-0.385099	-0.489416
206	0.341830	-0.390644	-0.660916
207	-0.616545	0.407413	-1.213411
208	-0.579514	0.284708	-1.658276

[209 rows x 3 columns]

[62]:
```
sns.pairplot(final_data)
```

[68]:
```
sns.scatterplot(x='PC1', y='PC2', data=final_data, c='green')
```

[71]:
```
sns.scatterplot(x='PC1', y='PC3', data=final_data, c='orange')
```

[70]:
```
sns.scatterplot(x='PC2', y='PC3', data=final_data, c='blue')
```

[103]:
```
import pandas as pd
processed_data = pd.concat([df[['VendorName','ModelName']], final_data], axis =␣
↪1).T.drop_duplicates().T
top_vendors = processed_data['VendorName'][:20].unique()
```

```
top_vendors
```

[103]:
```
array(['adviser', 'amdahl', 'apollo', 'basf', 'bti', 'burroughs'],
      dtype=object)
```

[88]:
```
processed_data['VendorName']
```

[88]:
```
0       adviser
1        amdahl
2        amdahl
3        amdahl
4        amdahl
         ...
204      sperry
205      sperry
206      sratus
207        wang
208        wang
Name: VendorName, Length: 209, dtype: object
```

[108]:
```
processed_data.head(20)
```

[108]:

	VendorName	ModelName	PC1	PC2	PC3
0	adviser	32/60	2.44461	3.758606	4.314055
1	amdahl	470v/7	-0.495514	-0.643343	2.053597
2	amdahl	470v/7a	-0.495514	-0.643343	2.053597
3	amdahl	470v/7b	-0.495514	-0.643343	2.053597
4	amdahl	470v/7c	-0.397143	-0.832075	1.181126
5	amdahl	470v/b	-0.503527	-0.531506	2.395301
6	amdahl	580-5840	-1.592605	-0.904715	3.794381
7	amdahl	580-5850	-1.592605	-0.904715	3.794381
8	amdahl	580-5860	-2.315521	-1.216789	5.077657
9	amdahl	580-5880	-4.01443	-1.018466	9.00749
10	apollo	dn320	-0.313587	0.16775	-1.516155
11	apollo	dn420	-0.192006	0.306025	-1.449999
12	basf	7/65	0.374145	-0.487922	-0.051623
13	basf	7/68	-0.063914	-0.764323	0.501256
14	bti	5000	0.046864	0.188654	-1.652576
15	bti	8000	0.483433	0.325084	-0.200477
16	burroughs	b1955	0.599524	0.037492	-0.884401
17	burroughs	b2900	0.892037	0.40376	-0.385882
18	burroughs	b2925	0.618771	-0.034522	-0.811165
19	burroughs	b4955	0.802557	1.192026	2.173784

[109]:
```
plt.figure(figsize=(12, 6))
sns.countplot(y='VendorName', data=processed_data,␣
↪order=processed_data['VendorName'].value_counts().index)
```

```
plt.title('Count of Each Vendor')
plt.show()
```

[112]:
```
plt.figure(figsize=(100, 50))
sns.countplot(y='ModelName', data=processed_data,␣
 ↪order=processed_data['ModelName'].value_counts().index)
plt.title('Count of Each Model')
plt.show()
```

[114]:
```
sns.pairplot(processed_data, vars=['PC1', 'PC2', 'PC3'], hue='VendorName',␣
 ↪markers=['o', 's', 'D', '^', 'v'])
plt.suptitle('Pair Plot of Principal Components', y=1.02)
plt.show()
```

[116]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(processed_data[['PC1', 'PC2', 'PC3']].corr(), annot=True,␣
 ↪cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Principal Components')
plt.show()
```

[118]:
```
# Scatter plot for PC1 vs PC2
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC1', y='PC2', hue='VendorName', style='VendorName', s=100,␣
 ↪data=processed_data)
plt.title('PC1 vs PC2')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

[120]:
```
# Scatter plot for PC1 vs PC3
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC1', y='PC3', hue='VendorName', style='VendorName', s=100,␣
 ↪data=processed_data)
plt.title('PC1 vs PC3')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

[121]:
```
# Scatter plot for PC2 vs PC3
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC2', y='PC3', hue='VendorName', style='VendorName', s=100,␣
 ↪data=processed_data)
plt.title('PC2 vs PC3')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```