**PRINCIPAL COMPONENT ANALYSIS (PCA)**

**CSE 303: Machine Learning**

Submitted by

Name: Ridhi Guntur

Roll No: AP22110011467

Section: CSE M

Lab Date: 15/10/2024

Submission Date: 21/10/2024

**A picture containing text

Description automatically generated**

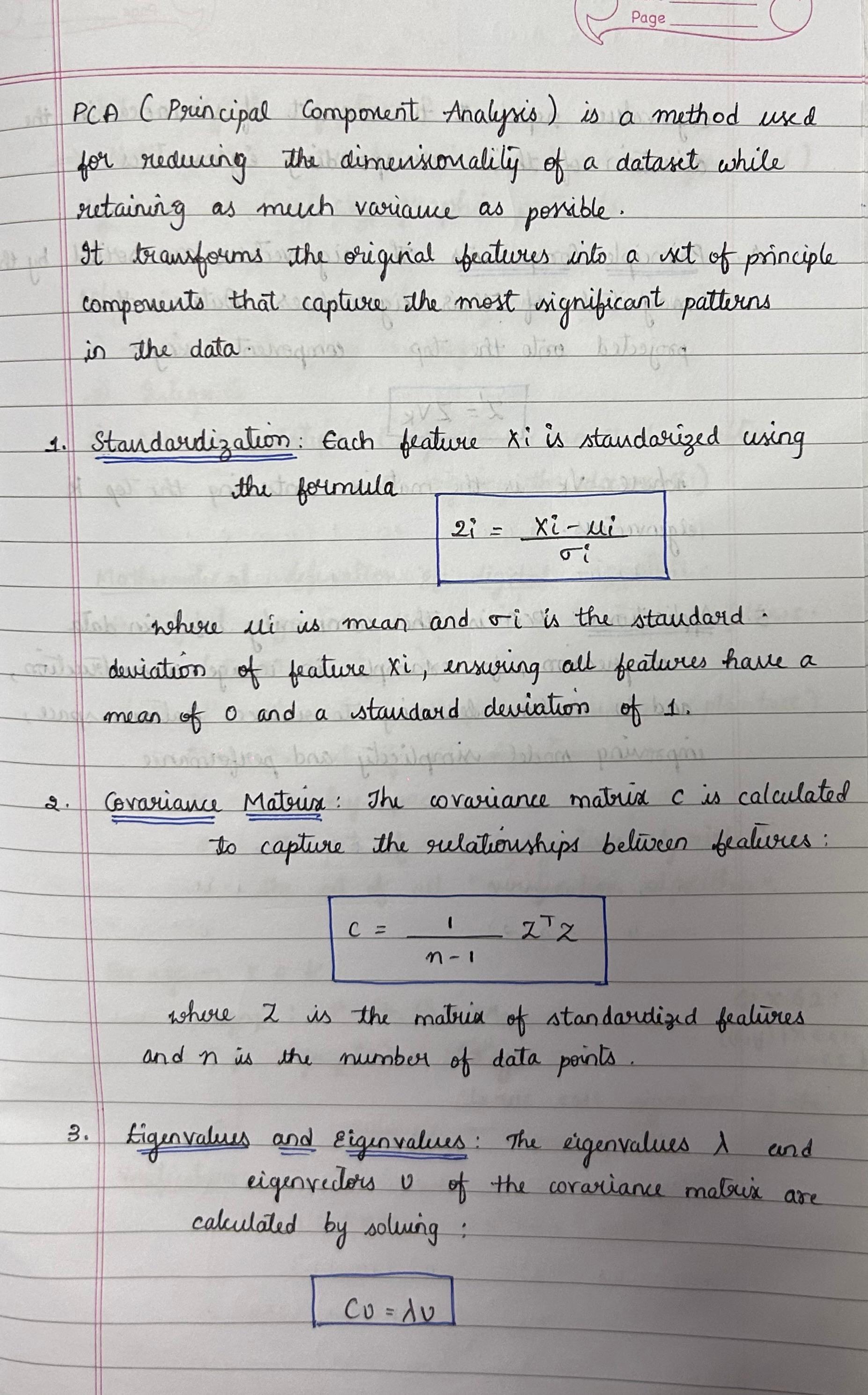
**Department Computer Science and Engineering**

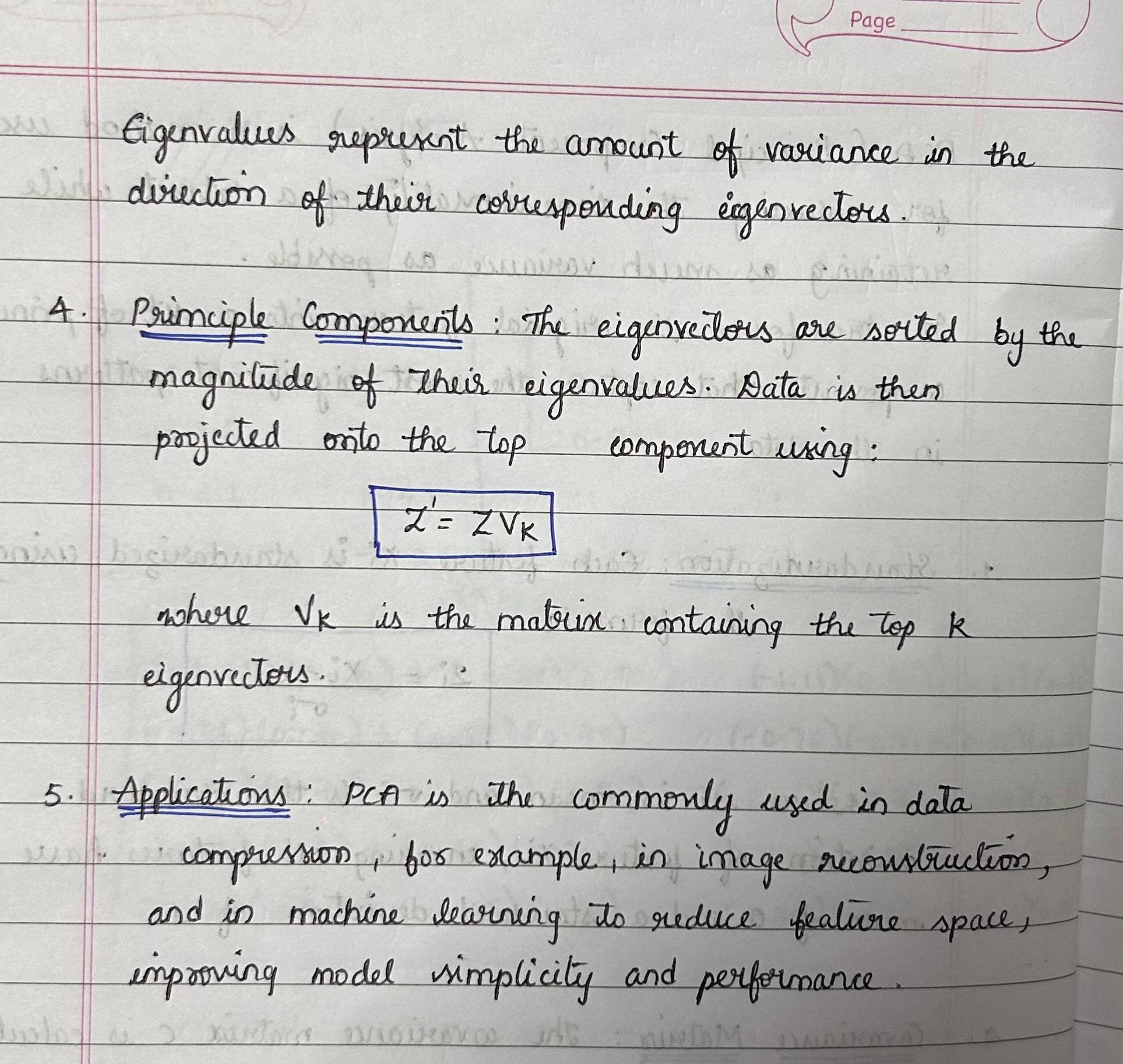
**School of Engineering and Sciences**

**SRM University–AP**

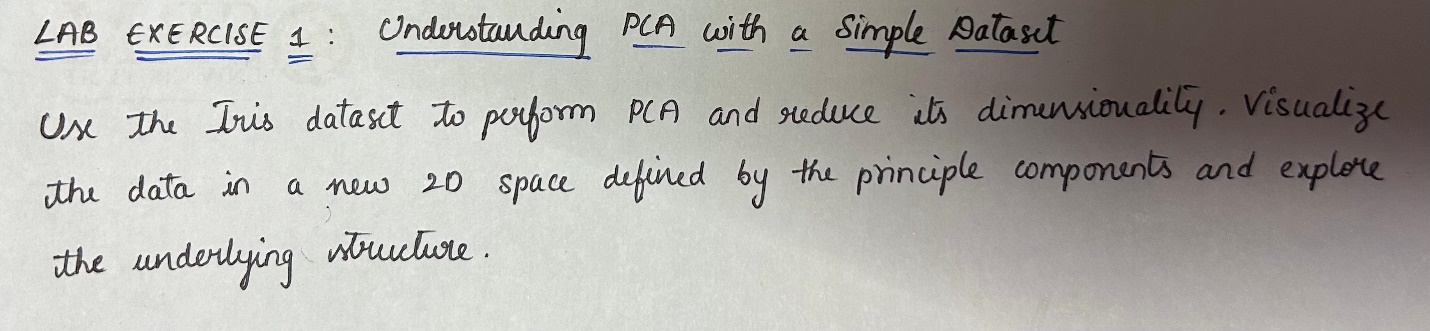
**Amaravati, Andhra Pradesh – 522 240, India**

1. **Algorithm Description**

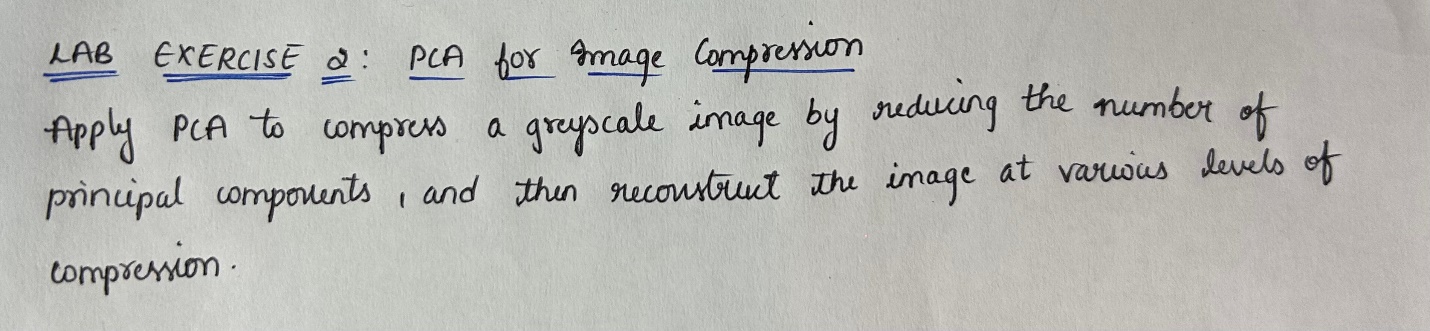




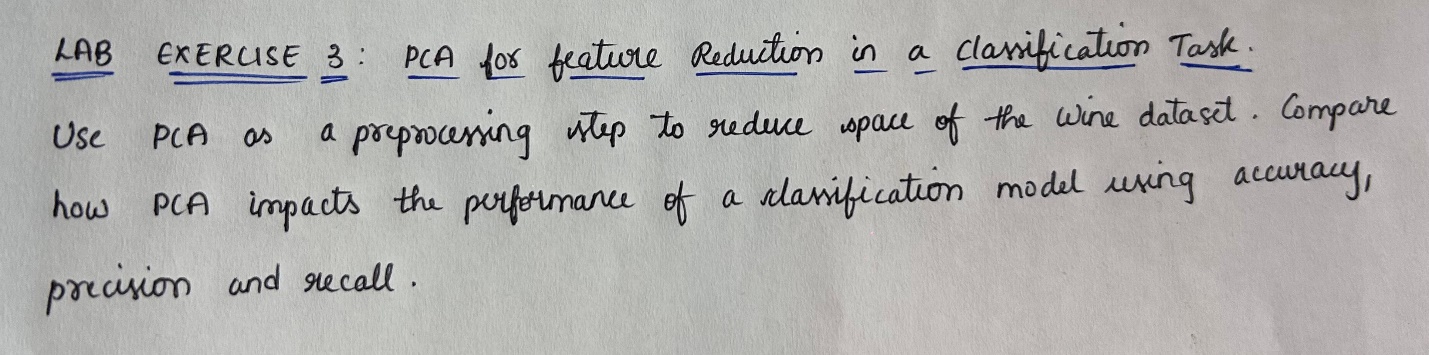
1. **Solution**



|  |
| --- |
| import pandas as pd  import seaborn as sns  # Load the Iris dataset from seaborn  data = sns.load\_dataset('iris')  print(data.head())  # Create boxplots to see the distribution of features by species  import matplotlib.pyplot as plt  fig, axes = plt.subplots(2, 2, figsize=(12, 8))  for ax, feature in zip(axes.flatten(), data.columns[:-1]):  sns.boxplot(x='species', y=feature, data=data, ax=ax)  ax.set\_title(f'{feature} by Species')  plt.tight\_layout()  plt.show()  from sklearn.preprocessing import StandardScaler  X = data[['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width']]  scaler = StandardScaler()  X\_standardized = scaler.fit\_transform(X)  X\_standardized\_df = pd.DataFrame(X\_standardized, columns=X.columns)  print(X\_standardized\_df.head())  import numpy as np  cov\_matrix = np.cov(X\_standardized.T)  print(cov\_matrix)  eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)  print("Eigenvalues:")  print(eigenvalues)  print("\nEigenvectors:")  print(eigenvectors)  sorted\_indices = np.argsort(eigenvalues)[::-1]  sorted\_eigenvalues = eigenvalues[sorted\_indices]  sorted\_eigenvectors = eigenvectors[:, sorted\_indices]  print("Sorted Eigenvalues:")  print(sorted\_eigenvalues)  print("\nSorted Eigenvectors:")  print(sorted\_eigenvectors)  top\_2\_eigenvectors = sorted\_eigenvectors[:, :2]  X\_pca = np.dot(X\_standardized, top\_2\_eigenvectors)  X\_pca\_df = pd.DataFrame(X\_pca, columns=['PC1', 'PC2'])  print(X\_pca\_df.head())  import matplotlib.pyplot as plt  X\_pca\_df['species'] = data['species']  plt.figure(figsize=(8, 6))  sns.scatterplot(x='PC1', y='PC2', hue='species', data=X\_pca\_df, palette='Set1', s=100)  plt.title("Iris Dataset Projected onto the First Two Principal Components", fontsize=14)  plt.xlabel("Principal Component 1", fontsize=12)  plt.ylabel("Principal Component 2", fontsize=12)  plt.show() |



|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from skimage import color, io  image = io.imread(r"C:\Users\guntu\Downloads\image-1.jpeg", as\_gray=True) # Provide the correct path to your image  image\_resized = np.array(image)  plt.figure(figsize=(6, 6))  plt.imshow(image\_resized, cmap='gray')  plt.title('Original Image')  plt.axis('off')  plt.show()  print(f"Original image shape: {image.shape}")  X = image\_resized.reshape(-1, image\_resized.shape[1])  print("Shape of reshaped image: ", X.shape)  from sklearn.decomposition import PCA  def apply\_pca\_and\_reconstruct(X, num\_components):  pca = PCA(n\_components=num\_components)  X\_pca = pca.fit\_transform(X) # Applying PCA  X\_reconstructed = pca.inverse\_transform(X\_pca) # Reconstructing the image from the PCA components  return X\_reconstructed  X\_reconstructed\_50 = apply\_pca\_and\_reconstruct(X, 50)  image\_reconstructed\_50 = X\_reconstructed\_50.reshape(image\_resized.shape)  components\_list = [5, 20, 50, 100]  reconstructed\_images = []  for n in components\_list:  X\_reconstructed = apply\_pca\_and\_reconstruct(X, n)  reconstructed\_images.append(X\_reconstructed.reshape(image\_resized.shape))  plt.figure(figsize=(10, 10))  plt.subplot(3, 2, 1)  plt.imshow(image\_resized, cmap='gray')  plt.title('Original Image')  plt.axis('off')  for i, n in enumerate(components\_list):  plt.subplot(3, 2, i+2)  plt.imshow(reconstructed\_images[i], cmap='gray')  plt.title(f'Reconstructed with {n} components')  plt.axis('off')  plt.tight\_layout()  plt.show() |



|  |
| --- |
| # Import necessary libraries  import numpy as np  import matplotlib.pyplot as plt  from sklearn.datasets import load\_wine  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.linear\_model import LogisticRegression  from sklearn.svm import SVC  from sklearn.metrics import accuracy\_score, precision\_score, recall\_score  from sklearn.decomposition import PCA  wine = load\_wine()  X = wine.data # Features  y = wine.target # Labels  print("Feature shape:", X.shape)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  log\_reg = LogisticRegression(max\_iter=10000)  log\_reg.fit(X\_train, y\_train)  y\_pred = log\_reg.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  precision = precision\_score(y\_test, y\_pred, average='macro')  recall = recall\_score(y\_test, y\_pred, average='macro')  print("Baseline Model (No PCA) - Logistic Regression")  print(f"Accuracy: {accuracy:.4f}")  print(f"Precision: {precision:.4f}")  print(f"Recall: {recall:.4f}")  components\_list = [2, 5, 10]  X\_train\_pca\_list = [] # Reduced training data for each PCA run  pca\_models = []  for n in components\_list:  pca = PCA(n\_components=n)  X\_train\_pca = pca.fit\_transform(X\_train) # Fit PCA and transform the training data  X\_train\_pca\_list.append(X\_train\_pca) # Store the reduced training data  pca\_models.append(pca) # Store the PCA model for later use on test data  print(f"Training data transformed with {n} components. Shape: {X\_train\_pca.shape}")  X\_test\_pca\_list = []  for i, pca in enumerate(pca\_models):  X\_test\_pca = pca.transform(X\_test) # Transform the test data using the same PCA model  X\_test\_pca\_list.append(X\_test\_pca) # Store the reduced test data  print(f"Test data transformed with {components\_list[i]} components. Shape: {X\_test\_pca.shape}")  results = []  for i, n in enumerate(components\_list):  log\_reg\_pca = LogisticRegression(max\_iter=10000)  log\_reg\_pca.fit(X\_train\_pca\_list[i], y\_train)  # Evaluate performance  y\_pred\_pca = log\_reg\_pca.predict(X\_test\_pca\_list[i])  accuracy\_pca = accuracy\_score(y\_test, y\_pred\_pca)  precision\_pca = precision\_score(y\_test, y\_pred\_pca, average='macro')  recall\_pca = recall\_score(y\_test, y\_pred\_pca, average='macro')    results.append({  'components': n,  'accuracy': accuracy\_pca,  'precision': precision\_pca,  'recall': recall\_pca  })  print(f"\nModel with {n} PCA components:")  print(f"Accuracy: {accuracy\_pca:.4f}")  print(f"Precision: {precision\_pca:.4f}")  print(f"Recall: {recall\_pca:.4f}")  components = [result['components'] for result in results]  accuracies = [result['accuracy'] for result in results]  plt.figure(figsize=(8, 6))  plt.plot(components, accuracies, marker='o', linestyle='-', color='b')  plt.title('Accuracy vs Number of PCA Components')  plt.xlabel('Number of PCA Components')  plt.ylabel('Accuracy')  plt.grid(True)  plt.show() |

1. **Code Repository:**

GitHub Link: