**Step 1: Import Required Libraries**

We will use Python with common libraries such as **Pandas**, **NumPy**, and **Scikit-learn**. First, ensure you have installed these libraries:

bash

Copy

pip install pandas numpy scikit-learn matplotlib

Now, let's import them into the script:

python

Copy

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

**Step 2: Load the Sample Diabetic Dataset**

For demonstration purposes, we can use a sample dataset of diabetic patients. A well-known dataset for this purpose is the **Pima Indians Diabetes Database**, which can be downloaded from various sources (like Kaggle).

Let's assume we have a CSV file (diabetes.csv) with features like Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, and Outcome.

python

Copy

# Load the dataset (make sure to adjust the path if needed)

df = pd.read\_csv("diabetes.csv")

# Display first few rows to get an idea of the data

print(df.head())

This might produce a dataframe like:

| **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

**Step 3: Preprocess the Data**

Before applying PCA, we need to standardize the data. PCA is sensitive to the scale of the data, so we will use the **StandardScaler** from Scikit-learn to scale the data to have zero mean and unit variance.

python

Copy

# Selecting the features (excluding the 'Outcome' column as it is the target variable)

X = df.drop(columns=['Outcome'])

# Standardize the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Check the standardized data

print(X\_scaled[:5]) # print the first 5 rows

**Step 4: Apply PCA**

Now, let’s apply PCA to reduce the number of features. We’ll start by trying to reduce it to 2 principal components for visualization.

python

Copy

# Apply PCA

pca = PCA(n\_components=2) # Reduce to 2 dimensions

X\_pca = pca.fit\_transform(X\_scaled)

# Check the resulting transformed data

print(X\_pca[:5]) # print the first 5 rows of the PCA-transformed data

**Step 5: Variance Explained by Principal Components**

PCA also allows us to check how much of the original variance is captured by each principal component.

python

Copy

# Explained variance ratio (how much variance each component explains)

explained\_variance = pca.explained\_variance\_ratio\_

print(f"Explained variance ratio: {explained\_variance}")

This will output something like:

python

Copy

Explained variance ratio: [0.3426, 0.2398]

This means that the first principal component explains 34.26% of the variance, and the second principal component explains 23.98% of the variance.

**Step 6: Plot the Data**

Let’s visualize the transformed data in 2D space using a scatter plot. We will plot the first and second principal components:

python

Copy

# Create a DataFrame with the PCA results

df\_pca = pd.DataFrame(X\_pca, columns=['PC1', 'PC2'])

# Plot the data

plt.figure(figsize=(8, 6))

plt.scatter(df\_pca['PC1'], df\_pca['PC2'], c=df['Outcome'], cmap='coolwarm', edgecolors='k')

plt.title('PCA of Diabetic Patients Data')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(label='Outcome')

plt.show()

This scatter plot will show the data in 2D, with points colored according to the **Outcome** (whether the patient is diabetic or not). This visualization helps us see if there's a clear distinction between the two groups.

**Step 7: Interpret the Results**

* From the scatter plot, if the points are well separated, it indicates that PCA has successfully reduced the dimensionality while preserving important patterns in the data.
* You can also interpret the loadings of the original features on the principal components by examining the components\_ attribute of the PCA model:

python

Copy

# View the components (i.e., the contribution of each original feature to the principal components)

print("PCA components (loadings):\n", pca.components\_)

This will give us the contributions of the original features (such as Glucose, BMI, Age, etc.) to each principal component.

**Step 8: Explaining the Resulting Data**

The reduced data (X\_pca) contains the first two principal components. These components can now be used for further analysis or machine learning models like classification (to predict diabetes status), clustering, etc.

For example, if you want to perform classification using the PCA-transformed data:

python

Copy

# Train a classifier (e.g., Logistic Regression) on the PCA-transformed data

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Prepare data for classification

y = df['Outcome']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_pca, y, test\_size=0.3, random\_state=42)

# Train the model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy of Logistic Regression on PCA data: {accuracy:.2f}")

**Summary**

* **PCA** was applied to reduce the dimensionality of the dataset (e.g., 8 features down to 2 components).
* The **variance explained** by each component was evaluated.
* We visualized the results using a scatter plot to see how well the data can be separated into classes (diabetic vs. non-diabetic).
* We used the transformed PCA data to build a classifier (Logistic Regression) and tested its accuracy.

This is a simple example of how PCA can be used for dimensionality reduction in a diabetes dataset.