Darshan UNIVERSITY
योग: कर्मसु कौशलम्

Data Mining

Lab - 6

```
# Dimensionality Reduction using NumPy
```

Name: Smit Maru

Enrollment No: 23010101161

🔍 What is Data Reduction?

Data reduction refers to the process of reducing the amount of data that needs to be processed and stored, while preserving the essential patterns in the data.

Why do we reduce data?

- To reduce computational cost.
- To remove noise and redundant features.
- To improve model performance and training time.
- To visualize high-dimensional data in 2D or 3D.

Common data reduction techniques include:

- Principal Component Analysis (PCA)
- Feature selection
- Sampling

📉 What is Principal Component Analysis (PCA)?

PCA is a **dimensionality reduction technique** that transforms a dataset into a new coordinate system. It identifies the **directions (principal components)** where the variance

of the data is maximized.

Key Concepts:

- **Principal Components**: New features (linear combinations of original features) capturing most variance.
- **Eigenvectors & Eigenvalues**: Used to compute these principal directions.
- **Covariance Matrix**: Measures how features vary with each other.

PCA helps in **visualizing high-dimensional data**, **noise reduction**, and **speeding up algorithms**.

🧠 NumPy Functions Summary for PCA

Function	Purpose
`np.mean(X, axis=0)`	Compute mean of each column (feature-wise mean).
`X - np.mean(X, axis=0)`	Centering the data (zero mean).
`np.cov(X, rowvar=False)`	Compute covariance matrix for features.
`np.linalg.eigh(cov_mat)`	Get eigenvalues and eigenvectors (for symmetric matrices).
`np.argsort(values)[::-1]`	Sort values in descending order.
`np.dot(X, eigenvectors)`	Project original data onto new axes.

Step 1: Load the Iris Dataset

```
In [1]:  import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt

         df = pd.read_csv('iris.csv')
```

Step 2: Standardize the data (zero mean)

```
In [2]:  X = df.iloc[:, 0:4].values
         X_meaned = X - np.mean(X, axis=0)
```

Step 3: Compute the Covariance Matrix

```
In [3]:  cov_mat = np.cov(X_meaned, rowvar=False)
         print(cov_mat)
```

```
[[ 0.68569351 -0.042434     1.27431544  0.51627069]
 [-0.042434     0.18997942 -0.32965638 -0.12163937]
 [ 1.27431544 -0.32965638  3.11627785  1.2956094 ]
 [ 0.51627069 -0.12163937  1.2956094    0.58100626]]
```

Step 4: Compute eigenvalues and eigenvectors

```
In [4]:  eigen_vals, eigen_vecs = np.linalg.eigh(cov_mat)
         print(eigen_vals)
         print(eigen_vecs)
```

```
[0.02383509 0.0782095  0.24267075 4.22824171]
[[ 0.31548719  0.58202985  0.65658877 -0.36138659]
 [-0.3197231  -0.59791083  0.73016143  0.08452251]
 [-0.47983899 -0.07623608 -0.17337266 -0.85667061]
 [ 0.75365743 -0.54583143 -0.07548102 -0.3582892 ]]
```

Step 5: Compute eigenvalues and eigenvectors

```
In [5]:  sorted_indices = np.argsort(eigen_vals)[::-1]
         eigen_vals = eigen_vals[sorted_indices]
         eigen_vecs = eigen_vecs[:, sorted_indices]
         print(eigen_vals)
         print(eigen_vecs)
```

```
[4.22824171 0.24267075 0.0782095   0.02383509]
[[-0.36138659  0.65658877  0.58202985  0.31548719]
 [ 0.08452251  0.73016143 -0.59791083 -0.3197231 ]
 [-0.85667061 -0.17337266 -0.07623608 -0.47983899]
 [-0.3582892  -0.07548102 -0.54583143  0.75365743]]
```

Step 6: Select the top k eigenvectors (top 2)

```
In [6]:  top_2_eigen_vecs = eigen_vecs[:, :2]
         top_2_eigen_vecs
```

```
Out[6]:  array([[-0.36138659,  0.65658877],
                [ 0.08452251,  0.73016143],
                [-0.85667061, -0.17337266],
                [-0.3582892 , -0.07548102]])
```

Step 7: Project the data onto the top k eigenvectors

In [7]:
```python
X_reduced = np.dot(X_meaned, top_2_eigen_vecs)
print("Reduced data shape:", X_reduced.shape)
```

Reduced data shape: (150, 2)

Step 8: Plot the PCA-Reduced Data

In [8]:
```python
species = df['species'].values
unique_species = np.unique(species)
colors = {'setosa': 'red', 'versicolor': 'green', 'virginica': 'blue'}

for sp in unique_species:
    idx = np.where(species == sp)
    plt.scatter(X_reduced[idx, 0], X_reduced[idx, 1], label=sp, color=colors[sp])

plt.title("PCA - Iris Dataset")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.grid(True)
plt.show()
```
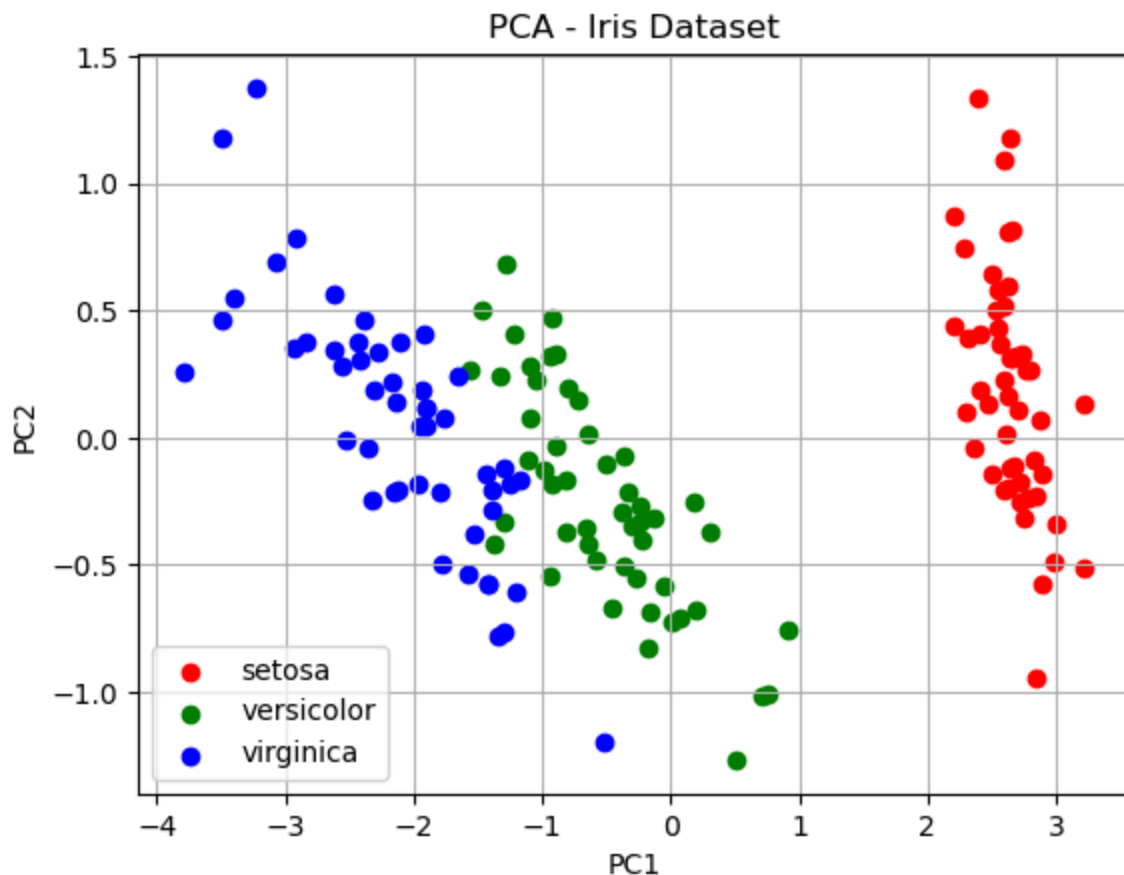

Extra - Bining Method

5,10,11,13,15,35,50,55,72,92,204,215.

Partition them into three bins by each of the following methods: (a) equal-frequency (equal-depth) partitioning (b) equal-width partitioning

In [9]:
```python
data = pd.Series([5, 10, 11, 13, 15, 35, 50, 55, 72, 92, 204, 215])
equal_freq = pd.qcut(data, q=3)
print("Equal-Frequency Bins:")
print(equal_freq)
```

```
Equal-Frequency Bins:
0        (4.999, 14.333]
1        (4.999, 14.333]
2        (4.999, 14.333]
3        (4.999, 14.333]
4       (14.333, 60.667]
5       (14.333, 60.667]
6       (14.333, 60.667]
7       (14.333, 60.667]
8        (60.667, 215.0]
9        (60.667, 215.0]
10       (60.667, 215.0]
11       (60.667, 215.0]
dtype: category
Categories (3, interval[float64, right]): [(4.999, 14.333] < (14.333, 60.667] < (60.
667, 215.0]]
```

In [10]:
```python
equal_width = pd.cut(data, bins=3)
print("Equal-Width Bins:")
print(equal_width)
```

```
Equal-Width Bins:
0         (4.79, 75.0]
1         (4.79, 75.0]
2         (4.79, 75.0]
3         (4.79, 75.0]
4         (4.79, 75.0]
5         (4.79, 75.0]
6         (4.79, 75.0]
7         (4.79, 75.0]
8         (4.79, 75.0]
9        (75.0, 145.0]
10      (145.0, 215.0]
11      (145.0, 215.0]
dtype: category
Categories (3, interval[float64, right]): [(4.79, 75.0] < (75.0, 145.0] < (145.0, 21
5.0]]
```