# PCA: Principal Component Analysis

Linear dimensionality reduction technique

## 💡 Core Concepts

🎯 **Goal:** Find directions of maximum variance

📐 **Principal Components:** Orthogonal eigenvectors of covariance matrix

📊 **Eigenvalues:** Variance explained by each component

## 🔄 Implementation Steps

1. Center data (subtract mean)

↓

2. Compute covariance matrix

↓

3. Eigen decomposition

↓

## 📈 Component Selection

Use scree plot to choose k components

**Example: Select k explaining 95% variance**

## ⚠️ Limitations

- ⚠ Linear transformation only
- ⚠ Sensitive to feature scaling
- ⚠ May lose interpretability

## 🎯 Applications

💾 Data compression

🔇 Noise reduction

📊 Visualization

4 Project onto top k components

# 📐 PCA Step-by-Step Numerical Example

2D to 1D Dimension Reduction

## 🔢 Sample Dataset

| Sample | $X_1$ | $X_2$ |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 5 |
| 3 | 4 | 3 |
| 4 | 5 | 6 |
| 5 | 6 | 7 |
| 6 | 7 | 8 |

### Step 1: Calculate Mean Vector (μ)

$$\mu_1 = (2+3+4+5+6+7)/6 = 4.5$$
$$\mu_2 = (1+5+3+6+7+8)/6 = 5.0$$

**Mean Vector:** $\mu = [4.5, 5.0]$

### Step 2: Center the Data (Subtract Mean)

```
X_centered = X - μ [-2.5, -4.0] [-1.5, 0.0] [-0.5, -2.0] [ 0.5, 1.0] [ 1.5, 2.0] [ 2.5, 3.0]
```

### Step 3: Compute Covariance Matrix

$$Cov(X) = (1/(n-1)) \times X\_centered^T \times X\_centered$$

```
Covariance Matrix: [ 3.5 3.75 ] [ 3.75 10.0 ]
```

💡 **Interpretation:** The diagonal elements (3.5, 10.0) represent variances of $X_1$ and $X_2$. Off-diagonal elements (3.75) show positive correlation between variables.

## 🔍 Eigendecomposition & Component Selection

### ⚙️ Computing Eigenvectors and Eigenvalues

### Step 4: Solve Characteristic Equation

$$det(Cov(X) - \lambda I) = 0$$

```
| 3.5-λ 3.75 | | 3.75 10.0-λ | = 0 (3.5-λ)(10.0-λ) - (3.75)² = 0 λ² - 13.5λ + 20.94 = 0
```

**Eigenvalues:**
$\lambda_1$ = 11.93 (88.4% variance explained)

$\lambda_2 = 1.57$ (11.6% variance explained)

### Step 5: Calculate Eigenvectors

For $\lambda_1 = 11.93$: Eigenvector $v_1 = [0.478, 0.878]$ (normalized) For $\lambda_2 = 1.57$: Eigenvector $v_2 = [-0.878, 0.478]$ (normalized)

🎯 **Principal Component 1 (PC1):** Direction of maximum variance, pointing towards [0.478, 0.878]

### Step 6: Variance Explained

| Component | Eigenvalue | % Variance | Cumulative % |
|---|---|---|---|
| PC1 | 11.93 | 88.4% | 88.4% |
| PC2 | 1.57 | 11.6% | 100.0% |

### Step 7: Project Data onto PC1

$$Z = X\_centered \times v_1$$

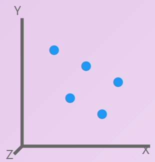
Transformed Data (1D): $z_1 = [-2.5, -4.0] \cdot [0.478, 0.878] = -4.71$ $z_2 = [-1.5, 0.0] \cdot [0.478, 0.878] = -0.72$ $z_3 = [-0.5, -2.0] \cdot [0.478, 0.878] = -1.99$ $z_4 = [0.5, 1.0] \cdot [0.478, 0.878] = 1.12$ $z_5 = [1.5, 2.0] \cdot [0.478, 0.878] = 2.47$ $z_6 = [2.5, 3.0] \cdot [0.478, 0.878] = 3.83$

# 📊 PCA Visualization: 3D to 2D Example

Understanding dimensionality reduction visually

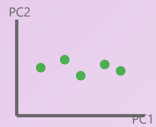
## 🎨 3D Data Visualization

### Original 3D Data Space




$X_1$ $X_2$ $X_3$

### Reduced 2D Space (PC1 & PC2)




PC1 PC2

📉 **Dimension Reduction:** From 3D ($X_1$, $X_2$, $X_3$) → 2D (PC1, PC2) while preserving 95%+ of variance

## 🔍 Visual Interpretation Guide

🎯 **PC1 Direction:** Points along the direction of maximum spread (variance) in the data. This is where the data varies the most.

📐 **PC2 Direction:** Orthogonal to PC1, captures the second largest variance. Always perpendicular to all previous components.

🔄 **Data Projection:** Each data point is projected onto the new PC axes. Distance from origin indicates the component score.

📊 **Information Loss:** The reduction from 3D to 2D means discarding PC3. The eigenvalue of PC3 tells us how much information is lost.

## 💻 Python Implementation Example

Using NumPy and scikit-learn

### 🐍 Method 1: NumPy (From Scratch)

```
# Import libraries
import numpy as np

# Sample data
X = np.array([[2, 1], [3, 5], [4, 3],
              [5, 6], [6, 7], [7, 8]])
```

```
# Step 1: Center the data
X_mean = np.mean(X, axis=0)
X_centered = X - X_mean

# Step 2: Compute covariance matrix
cov_matrix = np.cov(X_centered.T)

# Step 3: Eigendecomposition
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 4: Sort by eigenvalues
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Step 5: Project data onto PC1
PC1 = eigenvectors[:, 0]
X_pca = X_centered @ PC1

print("Eigenvalues:", eigenvalues)
print("PC1:", PC1)
print("Transformed data:", X_pca)
```

## ⚡ Method 2: scikit-learn (Production Ready)

```
# Import PCA from scikit-learn
from sklearn.decomposition import PCA
import numpy as np

# Sample data
```

```python
X = np.array([[2, 1], [3, 5], [4, 3],
              [5, 6], [6, 7], [7, 8]])

# Create PCA object (reduce to 1 component)
pca = PCA(n_components=1)

# Fit and transform
X_pca = pca.fit_transform(X)

# Access results
print("Explained variance ratio:",
      pca.explained_variance_ratio_)
print("Principal components:",
      pca.components_)
print("Transformed data:", X_pca)

# For 3D to 2D reduction
pca_3d = PCA(n_components=2)
X_3d_to_2d = pca_3d.fit_transform(X_3d_data)
```

### 📈 Before PCA

✓ Original dimensions: 3D or more
✓ All features present
✓ Full information retained
✗ Computational cost high
✗ Difficult to visualize
✗ Possible multicollinearity

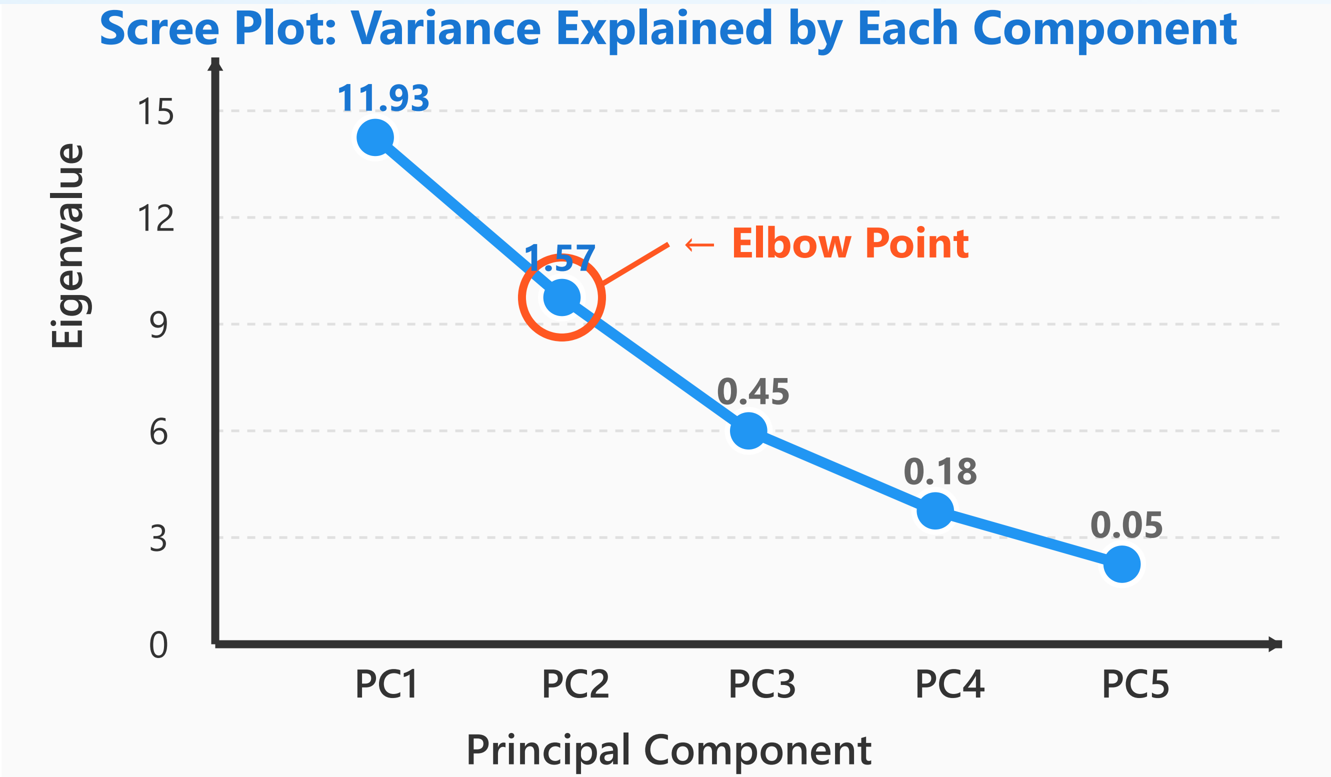
### 📉 After PCA

✓ Reduced dimensions: 2D or 1D
✓ Uncorrelated components
✓ 95%+ variance retained
✓ Faster computation
✓ Easy visualization
✗ Interpretability reduced

# 📈 Component Selection & Scree Plot

How many components should we keep?

## 📊 Scree Plot Analysis




🎯 **Elbow Method:** Choose the number of components at the "elbow" point where the eigenvalue curve flattens out. In this example, keeping 2 components is optimal.

### Decision Criteria for Component Selection

| Method | Rule | Example |
| --- | --- | --- |
| **Variance Threshold** | Keep components explaining 95% variance | If PC1+PC2 = 96%, keep 2 components |
| **Kaiser Rule** | Keep eigenvalues > 1 | If $\lambda_1$=11.93, $\lambda_2$=1.57, keep both |
| **Elbow Method** | Visual inspection of scree plot | Choose point where curve bends |
| **Fixed Dimension** | Reduce to 2D or 3D for visualization | Always keep exactly 2 or 3 PCs |

## 🎓 Practical Guidelines

### ✅ When to Use PCA:

- High-dimensional data (p > 50 features)
- Features are highly correlated
- Need for data visualization
- Computational efficiency required
- Noise reduction in signal processing

### ❌ When NOT to Use PCA:

- Need to maintain feature interpretability
- Features have different scales (standardize first!)
- Non-linear relationships in data (use Kernel PCA)
- Small sample size (n < p)
- Sparse data (many zeros)

## 🌍 Real-World PCA Applications

From theory to practice

### 🔬 Case Study 1: Image Compression

**Problem:** A 100×100 grayscale image has 10,000 pixels (dimensions)

#### PCA Solution

**Step 1:** Treat each image as a 10,000-dimensional vector

**Step 2:** Apply PCA to find principal components

**Step 3:** Keep top 100 components (99% variance)

**Step 4:** Reconstruct image using only 100 components

**Result:** 100× compression ratio with minimal quality loss!
Storage: 10,000 → 100 coefficients per image

### 📊 Case Study 2: Stock Market Analysis

**Problem:** Analyzing 500 stocks (S&P 500) with daily returns

### PCA Insights

```
# Stock returns data: 252 days × 500 stocks
from sklearn.decomposition import PCA
import pandas as pd

# Apply PCA
pca = PCA(n_components=10)
pc_scores = pca.fit_transform(stock_returns)

# Analyze variance explained
variance_explained = pca.explained_variance_ratio_
cumulative_variance = variance_explained.cumsum()

print("PC1 explains:", variance_explained[0])
# Output: PC1 explains: 0.42 (42% of market movement!)
```

**Interpretation:** PC1 often represents "the market" - overall market movement. PC2-PC3 capture sector-specific effects.

## 🧬 Case Study 3: Gene Expression Analysis

**Problem:** 20,000 genes measured across 100 patients

### Bioinformatics Application

| Original | After PCA | Benefit |
|---|---|---|
| 20,000 dimensions | 50 dimensions | 400× reduction |
| Impossible to visualize | 2D/3D plot | Pattern discovery |
| High noise | Filtered signal | Better classification |

💡 PCA revealed hidden patient subgroups that weren't visible in original data!

📖 Reference

**Interactive 3D Visualization Tutorial:**
LearnPCA - Visualizing PCA in 3D