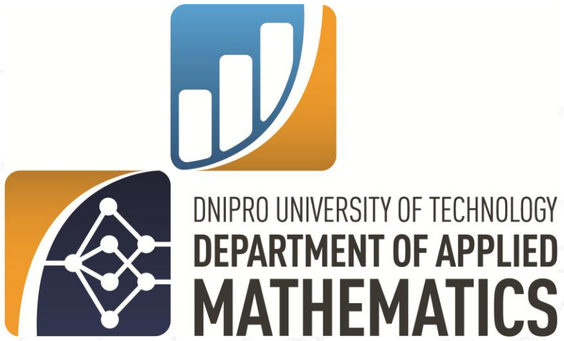


Principal Component Analysis

Course "Machine Learning: From Mathematical Foundations to Implementation in PYTHON"

Lecture by prof. Dmytro Babets

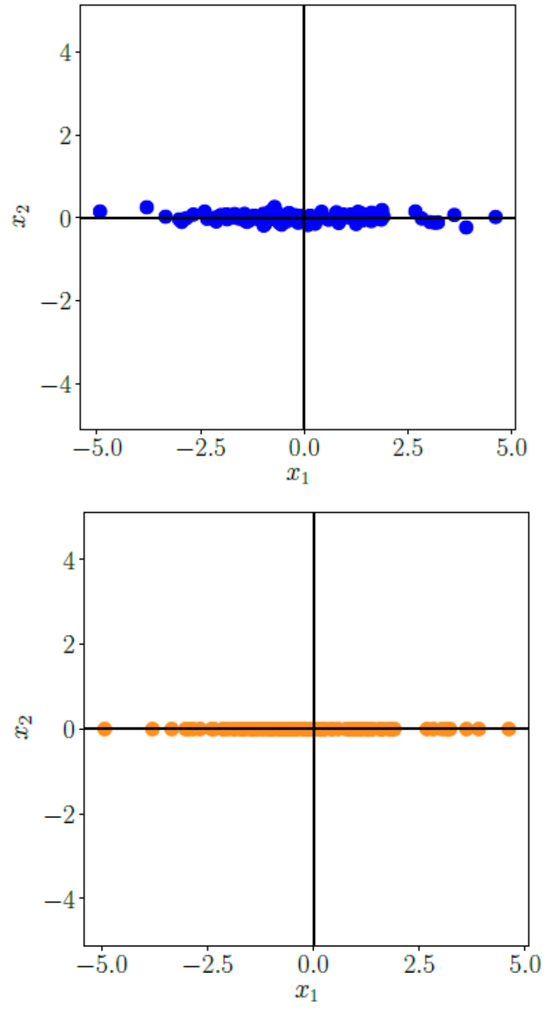
1. Why Dimensionality Reduction?

In modern machine learning, we often deal with high-dimensional data:

- A grayscale image of size 100×100 has 10,000 features (pixels).
- A document represented by word frequencies (bag-of-words) may have tens of thousands of features.
- Genomic datasets can have millions of measurements per sample.

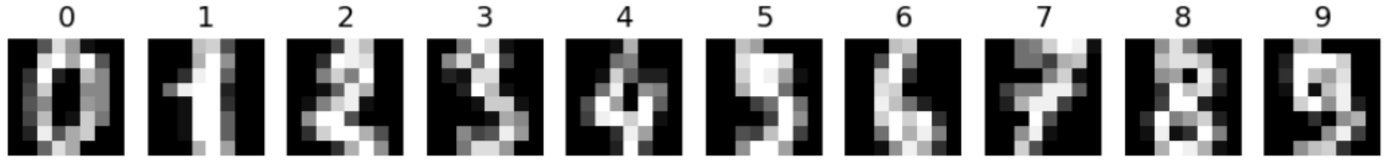
Challenges:

1. Curse of Dimensionality
   - As dimensionality increases, the volume of the feature space grows exponentially.
   - Data points become sparse, making it difficult for models to generalize.
   - Distance-based methods (like k-NN) lose effectiveness since all points appear equally distant.
2. Computational Cost
   - Training models on thousands of features requires significant memory and processing time.
   - Algorithms such as linear regression, clustering, or SVM become slow or unstable.
3. Visualization
   - Humans cannot easily interpret data in more than 3 dimensions.
   - PCA allows projecting high-dimensional data into 2D or 3D, making it possible to visualize clusters, separability, and patterns.



The original dataset does not vary much along the x_2 direction. This data can be represented using the x_1-coordinate alone with nearly no loss

Example: dataset of handwritten digits (MNIST)



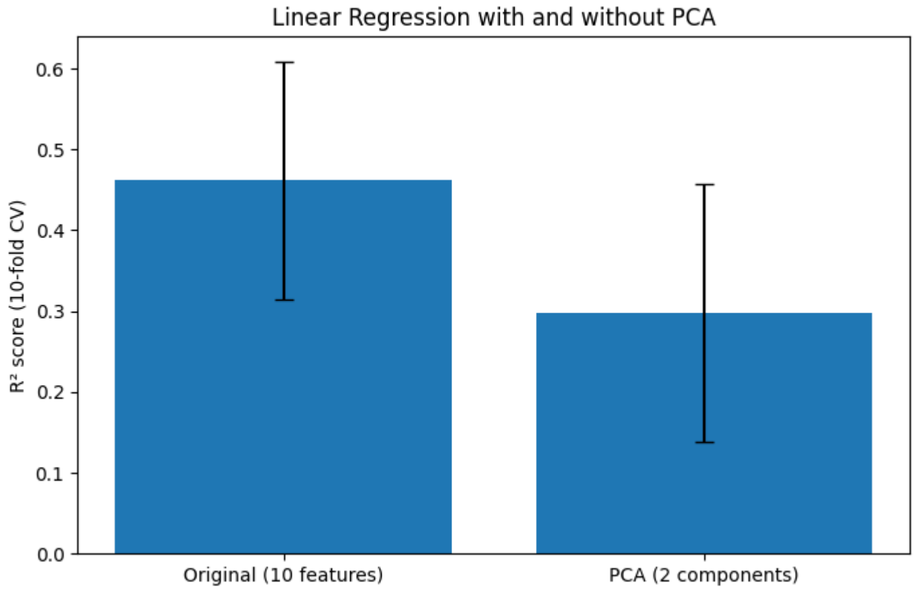
Each digit image has 784 features (28×28 pixels).

- Training a classifier directly on 784 dimensions is expensive.
- PCA can reduce it to, say, 50 dimensions while preserving ~95% of the variance.
- This results in faster training, less noise, and still accurate recognition.

Example: using the Diabetes dataset from `sklearn`

- Diabetes dataset has many correlated features – linear regression may become unstable due to multicollinearity.
- Using PCA, we can reduce the data to a few uncorrelated principal components, making the regression more stable and sometimes improving generalization.

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641	135.0
5	-0.092695	-0.044642	-0.040696	-0.019442	-0.068991	-0.079288	0.041277	-0.076395	-0.041176	-0.096346	97.0
6	-0.045472	0.050680	-0.047163	-0.015999	-0.040096	-0.024800	0.000779	-0.039493	-0.062917	-0.038357	138.0
7	0.063504	0.050680	-0.001895	0.066629	0.090620	0.108914	0.022869	0.017703	-0.035816	0.003064	63.0
8	0.041708	0.050680	0.061696	-0.040099	-0.013953	0.006202	-0.028674	-0.002592	-0.014960	0.011349	110.0
9	-0.070900	-0.044642	0.039062	-0.033213	-0.012577	-0.034508	-0.024993	-0.002592	0.067737	-0.013504	310.0



Problem Setting

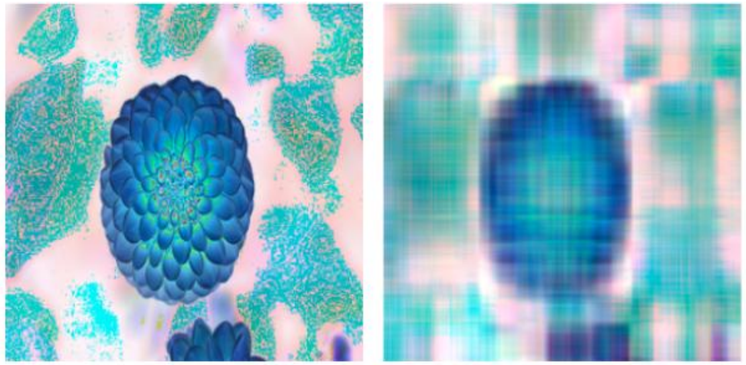
We have a dataset of N samples: $X = \{x_1, \ldots, x_N\}, \quad x_n \in \mathbb{R}^D$

where each vector has high dimensionality (D features).

- For simplicity, assume the data is centered (mean = 0).
- The covariance matrix describes how features vary together: $S = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^\top \quad \in \mathbb{R}^{D \times D}.$

Goal of PCA:

- Project data into a lower-dimensional space ($M<D$)
- Preserve as much of the original variability as possible
- Ensure projected points are “close” to the originals (minimal reconstruction error).



Intuition: Imagine compressing a high-resolution image into fewer pixels, while still keeping it recognizable

Compression Idea

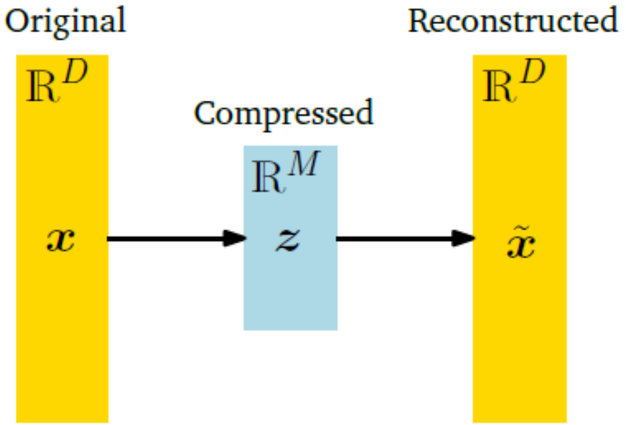
We search for a set of projection directions: $B = [b_1, \ldots, b_M] \in \mathbb{R}^{D \times M}$

where the columns b_i are orthonormal vectors (mutually perpendicular, unit length).

Encoding step (compression) $z_n = B^\top x_n \in \mathbb{R}^M$

Each original point x_n is represented by an M-dimensional code z_n.

Decoding step (reconstruction): $\tilde{x}_n = Bz_n = BB^\top x_n$



here $\tilde{x}_n$ is the approximation of the original point in the lower-dimensional subspace spanned by b_1,…,b_M.

Objective:

Find the matrix B such that:

- The variance of the projected data is maximized,

OR

- The reconstruction error $\|x_n - \tilde{x}_n\|^2$ is minimized.

Intuition:

- z_n = compressed "code" (like storing fewer coordinates).
- B = dictionary/basis that defines the new subspace.
- **PCA** guarantees that this compression loses as little information as possible for the chosen dimension M

Linear Algebra Foundations of PCA

Principal Component Analysis (PCA) is fundamentally a linear algebra method. To understand it deeply, we need to recall several key concepts.

Vectors, Inner Product, and Orthogonality

• A **vector** represents a data point in a D-dimensional space:

$$x = [x_1, x_2, \dots x_D]^\top \in \mathbb{R}^D,$$

• The **inner product** (dot product) measures similarity between two vectors:

$$\langle x, y \rangle = x^\top y = \sum_{i=1}^{D} x_i \, y_i$$

• The **norm (length)** of a vector:

$$\| x \| = \sqrt{x^\top x}$$

• Two vectors are **orthogonal** if:

$$\langle x, y \rangle = 0$$

• Orthogonality means they point in **independent directions** — no shared information.

In PCA, we will find a new orthogonal coordinate system (basis) where data variance is maximized.

Variance and Covariance of Data

Variance of a variable x measures how much data spreads along one axis:

$$\mathrm{Var}(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Covariance between two variables x and y measures how much two variables vary together:
If positive → both increase/decrease together.
If negative → one increases, the other decreases.
If zero → they are linearly uncorrelated.

$$\mathrm{Cov}(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

Covariance Matrix (The covariance matrix captures how features vary individually and jointly)

For multivariate data $X \in \mathbb{R}^{N\times D}$ (rows = samples, columns = features):

$$S = \frac{1}{N}X^{\top}X$$

Each element s_{ij} of S represents the covariance between features i and j:

$$S = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) & \dots \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Var}(x_2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

we can use **either of two conventions**:

$S = \frac{1}{n}X_c^T X_c$ *(ML convention)* - Used in PCA, SVD, and many ML libraries. Here we treat Xc as the whole dataset, not a sample

$S = \frac{1}{n-1}X_c^T X_c$ *(statistical convention)* - Used in statistics when the dataset is considered a sample from a larger population. Dividing by n−1 gives an unbiased estimator

Eigenvalues and Eigenvectors

If we perform **eigen-decomposition** of the covariance matrix:

$$Sb_i = \lambda_i b_i$$

then:

- b_i – **eigenvectors** (principal directions / axes of maximal variance)
- λ_i – **eigenvalues** (amount of variance explained by each axis)

The eigenvectors form an **orthonormal basis**, meaning: $b_i^\top b_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$

We can then sort eigenvalues: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$

The explained variance ratio (for component i) is:

$$\text{Explained Variance Ratio}_i = \frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j}$$

This gives the fraction (or percentage) of total variance explained by the i-th principal component

Select the first M eigenvectors to form:

$$B = [b_1, b_2, \ldots, b_M]$$

This matrix B defines the **projection** into a lower-dimensional subspace:

$$Z = B^\top X$$

Key Insight

1. PCA = finding orthogonal directions (basis vectors) that capture maximum variance in data.
2. These directions are **eigenvectors** of the **covariance matrix**.
3. The corresponding **eigenvalues** tell us how important each direction is (Each eigenvalue tells how much variance that component captures)
4. Total variance in the data = sum of all eigenvalues

Decoding (Reconstruction to original space)

The **decoded (reconstructed)** approximation of the original data is:

$$\boxed{\tilde{X} = ZB^{\top} = X_c BB^{\top}.}$$

Reconstruction error (per sample)

The reconstruction **error vector** for the i-th observation is:

$$e_i = x_{c,i} - \tilde{x}_i = x_{c,i} - BB^{\top} x_{c,i}.$$

The **squared reconstruction error** (per sample) is:

$$\boxed{E_i = \| e_i \|^2 = \| x_{c,i} - BB^{\top} x_{c,i} \|^2.}$$

This measures how far the reconstructed point lies from the original point in the feature space.

Total reconstruction error (Frobenius norm form)

Over all N samples, the **total squared reconstruction error** is:

$$\boxed{E_{total} = \| X_c - \tilde{X} \|_F^2 = \| X_c - X_c BB^{\top} \|_F^2.}$$

Key Insight

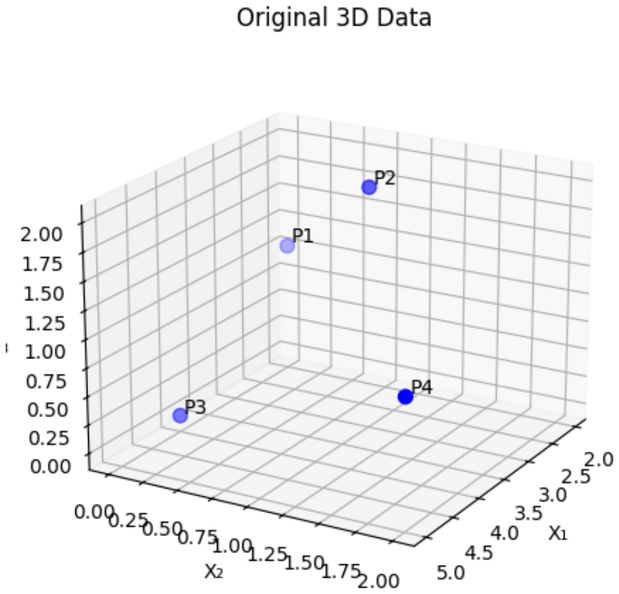
Decoding (or Reconstruction) in PCA is the process of **projecting data back** from the low-dimensional principal component space into the original feature space.

Each reconstructed point lies in the subspace spanned by the top M eigenvectors — it's the closest possible linear approximation (in the least-squares sense) of the original point using only those M directions of maximal variance.

Intuitive Summary

- PCA "compresses" data by keeping only the directions that matter most.
- Decoding "unpacks" that compressed representation — not perfectly, but with **minimal possible error** among all rank-M linear reconstructions.
- The reconstruction shows **how much information was preserved** by the chosen components.

PCA Example (2D projection and reconstruction)



Data

We have $N = 4$ points in $\mathbb{R}^3$ (rows are samples): $X = \begin{bmatrix} 2 & 0 & 1 \\ 3 & 1 & 2 \\ 4 & 0 & 0 \\ 5 & 2 & 1 \end{bmatrix}$

1) Mean vector and centering. Mean (feature-wise): $\bar{x} = \frac{1}{4}\sum_{n=1}^{4} x_n = [3.5,\ 0.75,\ 1.0].$

Center the data: $x_n^{(c)} = x_n - \bar{x}$. The centered data matrix X_c (each row a centered sample):

$$X_c = \begin{bmatrix} -1.5 & -0.75 & 0.0 \\ -0.5 & 0.25 & 1.0 \\ 0.5 & -0.75 & -1.0 \\ 1.5 & 1.25 & 0.0 \end{bmatrix}.$$

Outer-products and covariance matrix

Compute outer product $x_n^{(c)} x_n^{(c)\top}$ for each sample, then sum and divide by N.
Outer-products for each centered sample:

Sample 1 $x^{(c)} = [-, 1.5, -0.75, 0]$:

$$x^{(c)} x^{(c)\top} = \begin{bmatrix} 2.25 & 1.125 & -0.0 \\ 1.125 & 0.5625 & -0.0 \\ -0.0 & -0.0 & 0 \end{bmatrix}$$

Sample 2 $x^{(c)} = [-, 0.5, 0.25, 1.0]$:

$$\begin{bmatrix} 0.25 & -0.125 & -0.5 \\ -0.125 & 0.0625 & 0.25 \\ -0.5 & 0.25 & 1.0 \end{bmatrix}$$

Sample 3 $x^{(c)} = [0.5, -, -0.75, -1.0]$:

$$\begin{bmatrix} 0.25 & -0.375 & -0.5 \\ -0.375 & 0.5625 & 0.75 \\ -0.5 & 0.75 & 1.0 \end{bmatrix}$$

Sample 4 $x^{(c)} = [1.5, -, 1.25, 0]$:

$$\begin{bmatrix} 2.25 & 1.875 & 0 \\ 1.875 & 1.5625 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Sum of outer-products:

$$S_{sum} = \begin{bmatrix} 5.0 & 2.5 & -1.0 \\ 2.5 & 2.75 & 1.0 \\ -1.0 & 1.0 & 2.0 \end{bmatrix}.$$

Covariance matrix (using $1/N$ normalization):

$$S = \frac{1}{4} S_{sum} = \begin{bmatrix} 1.25 & 0.625 & -0.25 \\ 0.625 & 0.6875 & 0.25 \\ -0.25 & 0.25 & 0.5 \end{bmatrix}.$$

Eigen-decomposition of S

Solve

$$Sb_i = \lambda_i b_i$$

Eigenvalues (**sorted descending**) and corresponding eigenvectors (columns):

$$\lambda_1 \approx 1.6592492719, \qquad \lambda_2 = 0.75, \qquad \lambda_3 \approx 0.0282507281.$$

Eigenvectors (columns correspond to $\lambda_1, \lambda_2, \lambda_3$)

$$V = \begin{bmatrix} -0.847037 & -0.267261 & -0.459456 \\ -0.527035 & 0.534522 & 0.660697 \\ 0.069011 & 0.801784 & -0.593616 \end{bmatrix}.$$

Select top M components and build B.

Choose $M = 2$. The projection matrix B (**columns are top-2 eigenvectors**):

$$B = [b_1 \; b_2] = \begin{bmatrix} -0.847037 & -0.267261 \\ -0.527035 & 0.534522 \\ 0.069011 & 0.801784 \end{bmatrix} \in \mathbb{R}^{3\times 2}.$$

Key Insight

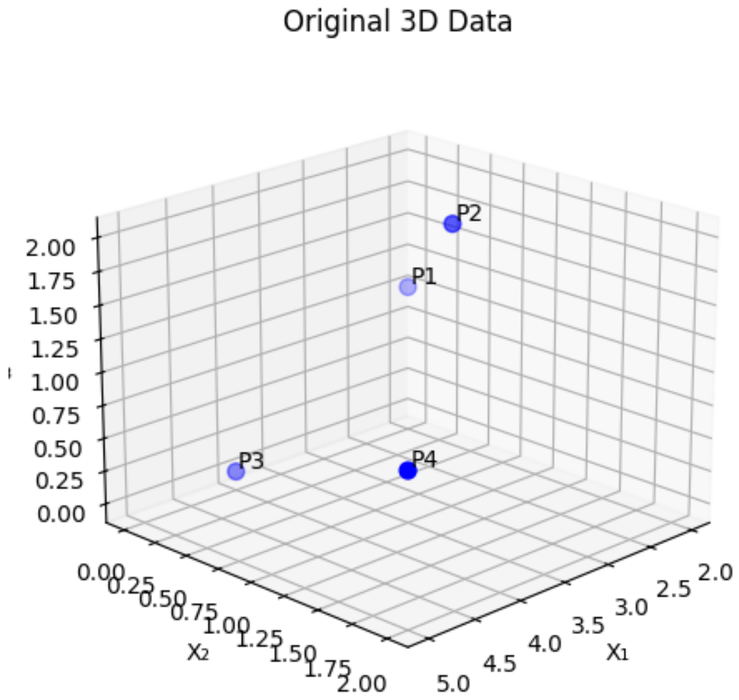
1. PCA = finding orthogonal directions (basis vectors) that capture maximum variance in data.
2. These directions are **eigenvectors** of the **covariance matrix**.
3. The corresponding **eigenvalues** tell us how important each direction is (Each eigenvalue tells how much variance that component captures)
4. Total variance in the data = sum of all eigenvalues

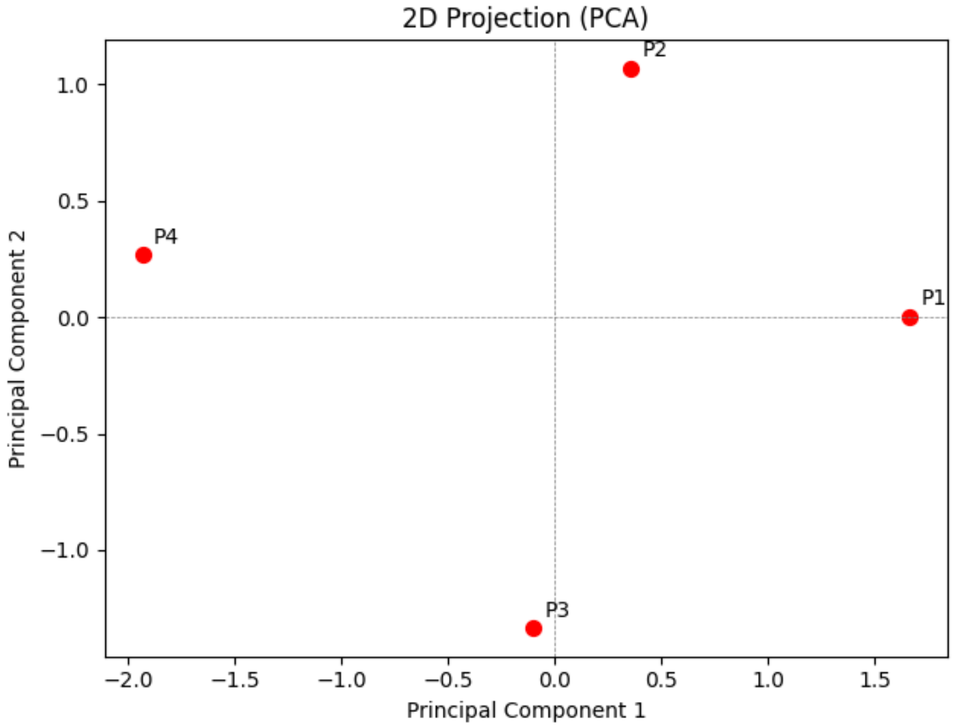
Encoding (projecting)

Encoding (projecting) — compute $Z = X_c B$

Projected coordinates Z (each row is the 2D code z_n)

```
Z =  [[ 1.66583177  0.        ]
      [ 0.36077064  1.06904497]
      [-0.09725331 -1.33630621]
      [-1.9293491   0.26726124]]
```





Decoding (reconstruction)

Decoding (reconstruction) — compute

$$\tilde{X} = ZB^{\top} =$$

```
[[ 2.08897856 -0.12795109  1.11496025]
 [ 2.90869957  1.13128994  1.8820399 ]
 [ 3.93952002  0.08697015 -0.07814009]
 [ 5.06280184  1.90969101  1.08113994]]
```

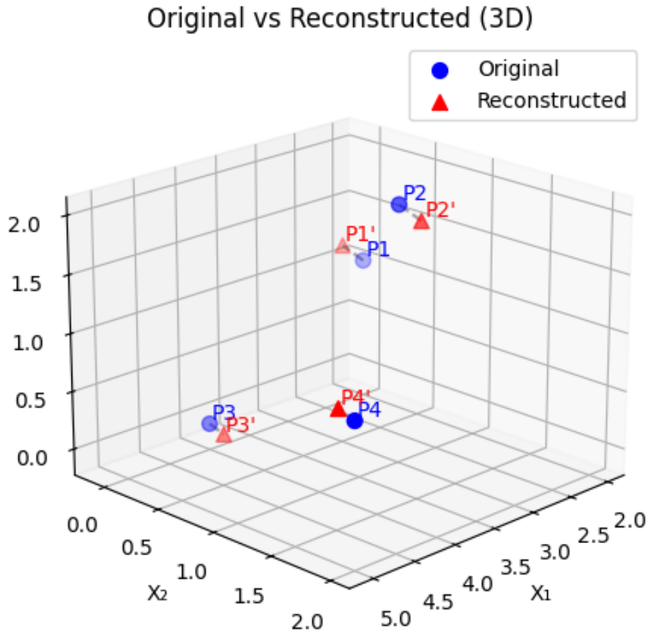
When we compress data to a lower-dimensional space and then decode it back, some information is inevitably lost.

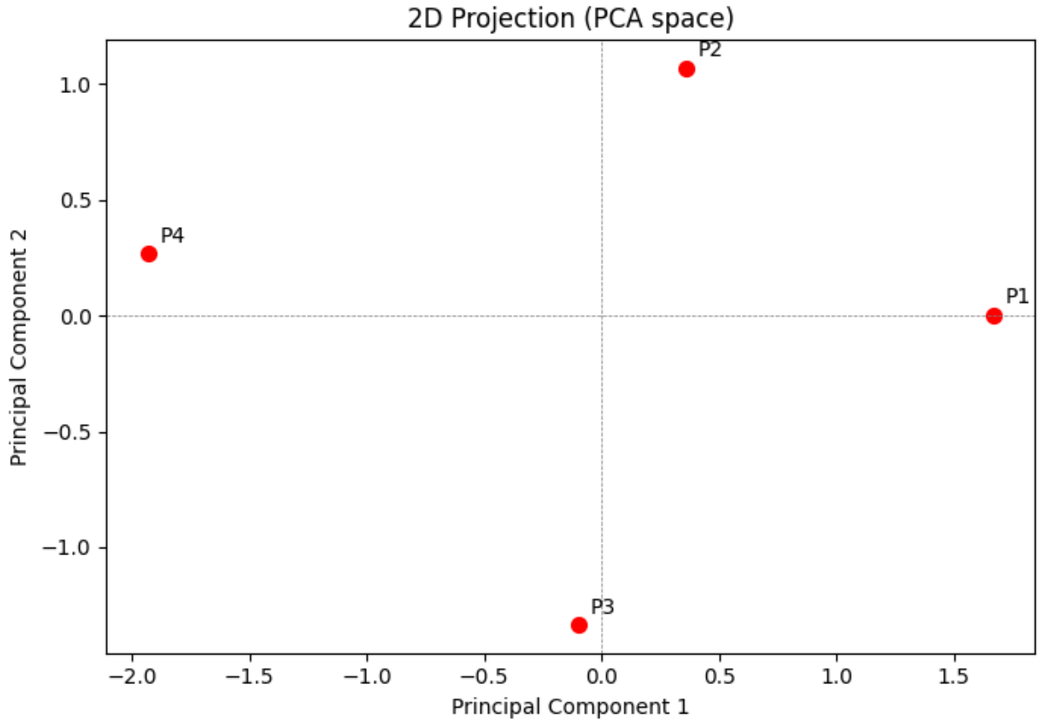
This **loss** is quantified by the **reconstruction error** — how far the reconstructed data $\tilde{X}$ is from the original centered data X_c.

The **Mean Squared Error (MSE)** version:

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N} \| x_n - \tilde{x}_n \|^2 = 0.028251$$

where $\|\cdot\|_F$ denotes the **Frobenius norm** (sum of squared elements).





Python code (PCA)

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # 3D plotting

# === 1. Original data (4 points in 3D) ===
X = np.array([
    [2.0, 0.0, 1.0],
    [3.0, 1.0, 2.0],
    [4.0, 0.0, 0.0],
    [5.0, 2.0, 1.0]])
print("1. Initial data: \n", X)

# === 2. Center the data ===
mean = X.mean(axis=0)
Xc = X - mean
print("\n2. Centered data: \n", Xc)

# === 3. Covariance matrix and eigen decomposition ===
S = (Xc.T @ Xc) / X.shape[0]
print("\n3. Covariance matrix: \n", S)

eigvals, eigvecs = np.linalg.eigh(S)
idx = np.argsort(eigvals)[::-1]  # sort by decreasing eigenvalues
eigvals = eigvals[idx]
eigvecs = eigvecs[:, idx]

print('\nEigenvalues:', eigvals)
print('Eigenvectors (columns are PCs):\n', eigvecs)
```

```
1. Initial data:
 [[2. 0. 1.]
 [3. 1. 2.]
 [4. 0. 0.]
 [5. 2. 1.]]

2. Centered data:
 [[-1.5  -0.75  0.  ]
 [-0.5   0.25  1.  ]
 [ 0.5  -0.75 -1.  ]
 [ 1.5   1.25  0.  ]]

3. Covariance matrix:
 [[ 1.25    0.625  -0.25  ]
 [ 0.625   0.6875  0.25  ]
 [-0.25    0.25    0.5   ]]

Eigenvalues: [1.65924927 0.75       0.02825073]
Eigenvectors (columns are PCs):
 [[-0.84703718 -0.26726124 -0.4594556 ]
 [-0.52703467  0.53452248  0.66069673]
 [ 0.06901072  0.80178373 -0.59361636]]
```

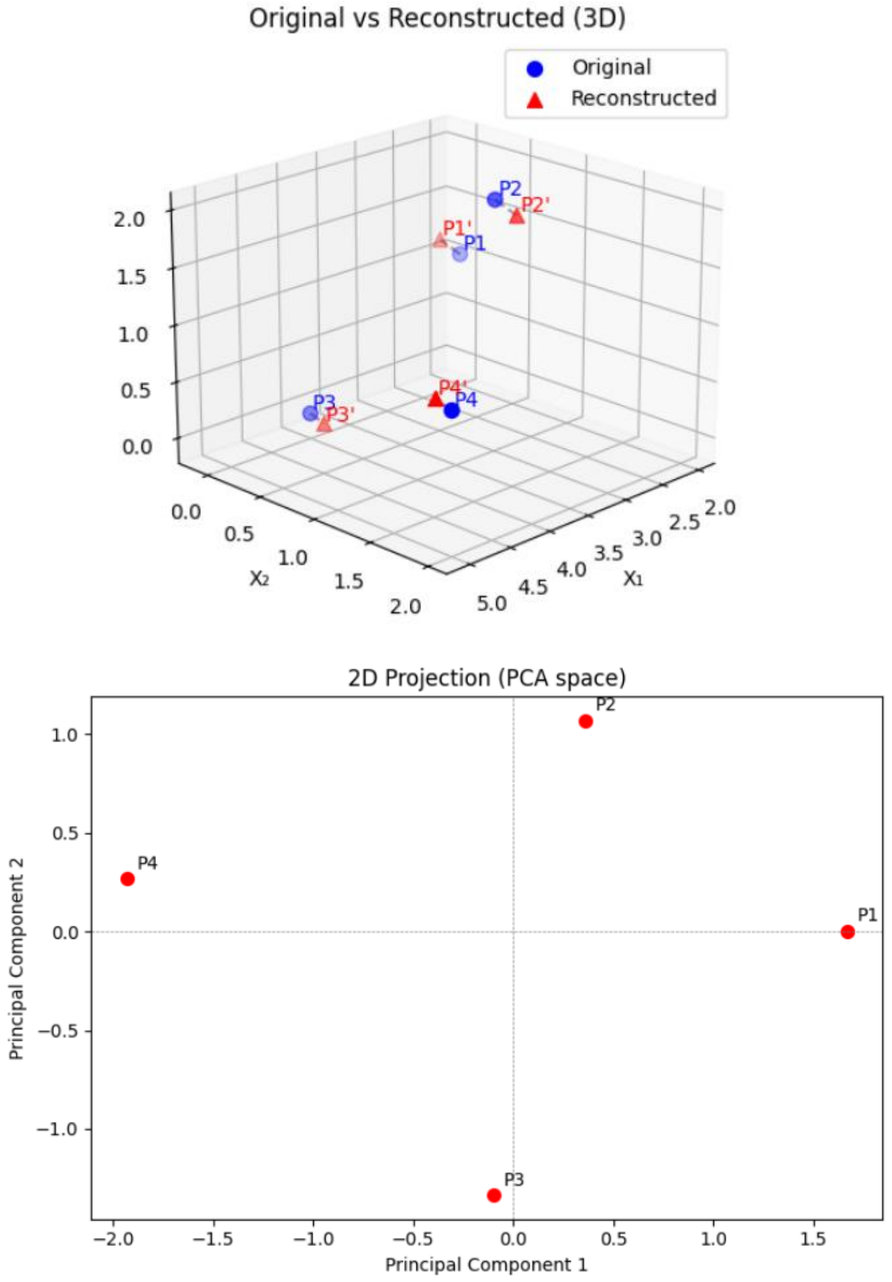
Python code (PCA)

```python
# === 4. Explained variance ===
# Step 1: Total variance in the data = sum of all eigenvalues
total_variance = np.sum(eigvals)
# Step 2: Explained variance ratio for each principal component
# Each eigenvalue tells how much variance that component captures
explained_variance_ratio = eigvals / total_variance
# Step 3: Cumulative sum (how much total variance is captured up to each PC)
cumulative_variance = np.cumsum(explained_variance_ratio)
# Step 4: Print results clearly
print("\n4. Explained variance per component:")
for i in range(len(eigvals)):
    print(f" PC{i+1}:")
    print(f"   Eigenvalue (λ): {eigvals[i]:.4f}")
    print(f"   Explained variance: {explained_variance_ratio[i]*100:.2f}%")
    print(f"   Cumulative variance: {cumulative_variance[i]*100:.2f}%")

# === 5. Project data onto first 2 principal components ===
B = eigvecs[:, :2]  # projection matrix
Z = Xc @ B          # low-dimensional representation
print('\n5. 2D PCA projection (Z):\n', Z)

# === 6. Reconstruct data back to 3D ===
X_reconstructed = Z @ B.T + mean
print("\n6. Reconstructed 3D data:\n", X_reconstructed)

# === 7. Reconstruction loss (Mean Squared Error) ===
reconstruction_loss = np.mean(np.sum((X - X_reconstructed) ** 2, axis=1))
print(f"\n7. Mean Squared Reconstruction Loss: {reconstruction_loss:.6f}")
```

```
4. Explained variance per component:
 PC1:
   Eigenvalue (λ): 1.6592
   Explained variance: 68.07%
   Cumulative variance: 68.07%
 PC2:
   Eigenvalue (λ): 0.7500
   Explained variance: 30.77%
   Cumulative variance: 98.84%
 PC3:
   Eigenvalue (λ): 0.0283
   Explained variance: 1.16%
   Cumulative variance: 100.00%

5. 2D PCA projection (Z):
 [[ 1.66583177  0.        ]
 [ 0.36077064  1.06904497]
 [-0.09725331 -1.33630621]
 [-1.9293491   0.26726124]]

6. Reconstructed 3D data:
 [[ 2.08897856 -0.12795109  1.11496025]
 [ 2.90869957  1.13128994  1.8820399 ]
 [ 3.93952002  0.08697015 -0.07814009]
 [ 5.06280184  1.90969101  1.08113994]]

7. Mean Squared Reconstruction Loss: 0.028251
```

Python code (PCA) - Visualization

```python
# === 8. Visualization ===
fig = plt.figure(figsize=(13, 5))

# --- 3D plot: original and reconstructed points ---
ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.scatter(X[:, 0], X[:, 1], X[:, 2], color='blue', s=50, label="Original")
ax1.scatter(X_reconstructed[:, 0], X_reconstructed[:, 1], X_reconstructed[:, 2],
            color='red', s=50, label="Reconstructed", marker='^')
for i, (x, y, z) in enumerate(X):
    ax1.text(x + 0.07, y + 0.07, z + 0.07, f"P{i+1}", color='blue')
for i, (x, y, z) in enumerate(X_reconstructed):
    ax1.text(x + 0.07, y + 0.07, z + 0.07, f"P{i+1}'", color='red')
# arrows between original and reconstructed
for i in range(len(X)):
    ax1.plot([X[i, 0], X_reconstructed[i, 0]],
             [X[i, 1], X_reconstructed[i, 1]],
             [X[i, 2], X_reconstructed[i, 2]], color='gray', linestyle='--', alpha=0.7)
ax1.set_title("Original vs Reconstructed (3D)")
ax1.set_xlabel("X₁")
ax1.set_ylabel("X₂")
ax1.set_zlabel("X₃")
ax1.legend()
ax1.view_init(elev=20, azim=45)

# --- 2D plot: PCA projection ---
ax2 = fig.add_subplot(1, 2, 2)
ax2.scatter(Z[:, 0], Z[:, 1], color='red', s=50)
for i, (x, y) in enumerate(Z):
    ax2.text(x + 0.05, y + 0.05, f"P{i+1}")
ax2.set_title("2D Projection (PCA space)")
ax2.set_xlabel("Principal Component 1")
ax2.set_ylabel("Principal Component 2")
ax2.axhline(0, color='gray', linestyle='--', linewidth=0.5)
ax2.axvline(0, color='gray', linestyle='--', linewidth=0.5)

plt.tight_layout()
plt.show()
```



PCA in Scikit-learn (`sklearn.decomposition.PCA`)

Category	Name / Parameter / Method	Description
Class	sklearn.decomposition.PCA	Performs Principal Component Analysis (PCA) for dimensionality reduction, feature extraction, and data compression.
Initialization Parameters	n_components	Number of principal components to keep. Can be: • integer → exact number of components • float ∈ (0,1) → fraction of explained variance • 'mle' → choose automatically using Minka's MLE • None → keep all components
	copy	If False, data is overwritten during fitting (saves memory). Default = True.
	whiten	If True, scales each component to unit variance — useful for some ML algorithms but removes original variance scale.
	tol	Tolerance for singular value decomposition when using 'arpack'.
	iterated_power	Number of power iterations when using 'randomized' solver (improves accuracy).
	random_state	Ensures reproducibility when using randomized solver.
Attributes (after fitting)	components_	Principal axes in feature space (each row is a principal component vector).
	explained_variance_	Eigenvalues — amount of variance explained by each component.
	explained_variance_ratio_	Proportion of total variance explained by each component.
	mean_	Per-feature empirical mean used for centering.
	n_components_	Actual number of components used.
	noise_variance_	Estimated noise variance (for MLE mode).

PCA in Scikit-learn (`sklearn.decomposition.PCA`)

Category	Name / Parameter / Method	Description
Core Methods	.fit(X)	Learns components and explained variance from data (X).
	.transform(X)	Projects data (X) into the lower-dimensional PCA space.
	.fit_transform(X)	Combines fitting and transforming in one step.
	.inverse_transform(Z)	Reconstructs data from the lower-dimensional representation (Z).
	.get_covariance()	Returns the estimated covariance matrix in the original space.
	.get_precision()	Returns the estimated precision matrix (inverse covariance).
	.score(X)	Returns the log-likelihood of the data under the model.
Related Utilities	sklearn.decomposition.IncrementalPCA	Handles large datasets in mini-batches (useful for big data).
	sklearn.decomposition.KernelPCA	Nonlinear extension of PCA using kernel functions.
	sklearn.decomposition.SparsePCA	Variant that produces sparse (interpretable) components.

PCA in `scikit-learn` automatically handles **centering** and provides **easy access** to explained variance and reconstruction — making it ideal for both conceptual learning and applied ML pipelines.

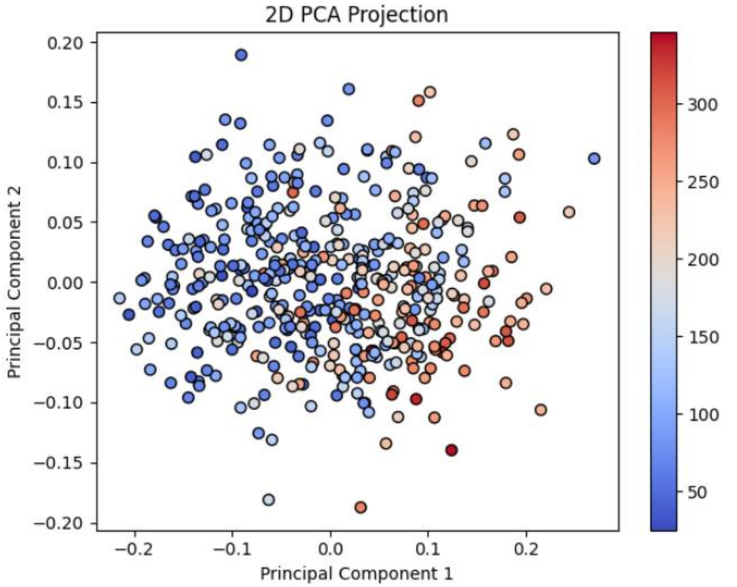
Python code (PCA - `scikit-learn`)

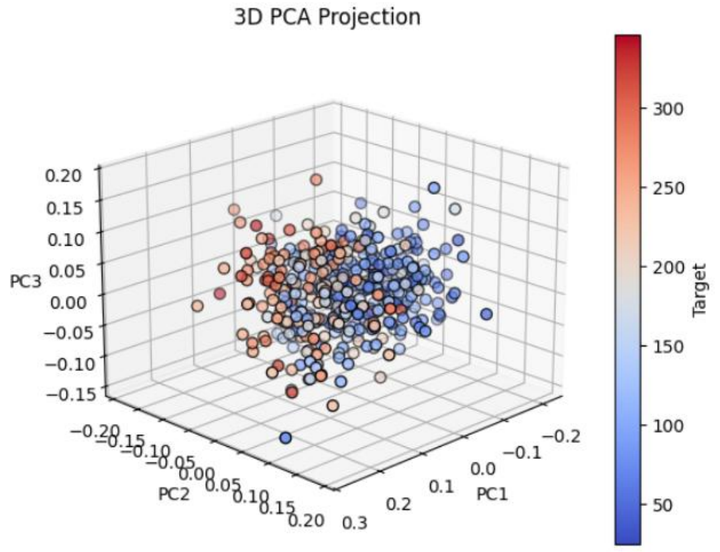
```python
# === PCA Example using the Diabetes dataset ===
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D  # for 3D visualization

# === 1. Load dataset ===
X, y = load_diabetes(return_X_y=True)
print("Dataset shape:", X.shape)  # (442 samples, 10 features)
print(f"Original dimensionality: {X.shape[1]}")

# === 2. Apply PCA 2D ===
pca2 = PCA(n_components=2)
X_pca2 = pca2.fit_transform(X)
X_reconstructed_2 = pca2.inverse_transform(X_pca2) #Reconstruction (Decoding step)

# === 3. Show metrics 2D ===
print("\nExplained variance per component:", pca2.explained_variance_)
print("Explained variance ratio:", pca2.explained_variance_ratio_)
total_explained_variance_2 = np.sum(pca2.explained_variance_ratio_)
print(f"Total explained variance (2 components): {total_explained_variance_2:.4f}")
mse_2d = mean_squared_error(X, X_reconstructed_2)
print("Reconstruction MSE (2D):", round(mse_2d, 6))

# === 4. Apply PCA 3D ===
pca3 = PCA(n_components=3)
X_pca3 = pca3.fit_transform(X)
X_reconstructed_3 = pca3.inverse_transform(X_pca3) #Reconstruction (Decoding step)

# === 5. Show metrics 3D ===
print("\nExplained variance per component:", pca3.explained_variance_)
print("Explained variance ratio:", pca3.explained_variance_ratio_)
total_explained_variance_3 = np.sum(pca3.explained_variance_ratio_)
print(f"Total explained variance (3 components): {total_explained_variance_3:.4f}")
mse_3d = mean_squared_error(X, X_reconstructed_3)
print("Reconstruction MSE (3D):", round(mse_3d, 6))
```

```
Dataset shape: (442, 10)
Original dimensionality: 10

Explained variance per component: [0.00912519 0.00338394]
Explained variance ratio: [0.40242108 0.14923197]
Total explained variance (2 components): 0.5517
Reconstruction MSE (2D): 0.001014

Explained variance per component: [0.00912519 0.00338394 0.00273462]
Explained variance ratio: [0.40242108 0.14923197 0.12059663]
Total explained variance (3 components): 0.6722
Reconstruction MSE (3D): 0.000742
```

Python code (PCA - `scikit-learn`)

```python
# === 6. 2D Visualization ===
plt.figure(figsize=(12, 5))

# --- 2D projection ---
plt.subplot(1, 2, 1)
plt.scatter(X_pca2[:, 0], X_pca2[:, 1], c=y, cmap='coolwarm', s=40, edgecolor='k')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA Projection')
plt.colorbar(label='Target (disease progression)')

# === 7. 3D Visualization ===
ax = plt.subplot(1, 2, 2, projection='3d')
p = ax.scatter(X_pca3[:, 0], X_pca3[:, 1], X_pca3[:, 2],
               c=y, cmap='coolwarm', s=40, edgecolor='k')
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')
ax.set_title('3D PCA Projection')
plt.colorbar(p, ax=ax, label='Target')
ax.view_init(elev=20, azim=45)
plt.tight_layout()
plt.show()
```





НТУ "ДНІПРОВСЬКА ПОЛІТЕХНІКА"
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ

Let’s proceed to the practical exercises

link:
https://colab.research.google.com/drive/18UU3EWyRnLZqeoH34G9hdPjFbJFQaRH_?usp=sharing

REFERENCES

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in R* (2nd ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1
2. Cohen, M. X. (2022). *Practical linear algebra for data science: From Core Concepts to Applications Using Python*. “O’Reilly Media, Inc.”
3. Cohen, M. X. (2021). *Linear algebra: theory, intuition, code*.