Principal Component Analysis

Seeing Through the Noise

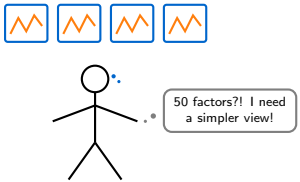
Methods & Algorithms

MSc Data Science – Spring 2026

A Portfolio Manager Tracks 500 Stocks with 50 Risk Factors — How?

The Dimensionality Problem

- 500 stocks $\times$ 50 features $=$ a 500-row, 50-column matrix
- Too many dimensions to visualize, too many correlations to track
- Many factors move together – can we find a smaller set of independent drivers?



The Question

Can we compress 50 correlated risk factors into 3–5 independent components without losing the signal?

PCA was invented by Pearson (1901) and independently by Hotelling (1933) – one of the oldest and most used tools in data science

Learning Objectives

After this lecture, you will be able to:

1. **Derive** PCA as variance maximization via eigendecomposition of the covariance matrix
2. **Analyze** scree plots and reconstruction error to select the number of components
3. **Evaluate** PCA's linear assumption using the Swiss roll and identify when non-linear methods are needed
4. **Apply** PCA to yield curve decomposition and portfolio risk reduction

Prerequisites

Linear algebra (matrix multiplication, eigenvectors), basic statistics (variance, covariance), Python with scikit-learn.

All learning objectives target Bloom's level 4+ (analyze, evaluate, apply, derive)

What Is PCA in Plain English?

The Camera Rotation Analogy

- Imagine a 3D cloud of data points viewed from a bad angle – all points overlap
- **PCA rotates the camera** to the angle that shows the widest spread
- The first axis (PC1) captures the most variance
- The second axis (PC2) is perpendicular and captures the next most

Key Insight

PCA does not discard data – it *reorders* it by importance. You choose how many axes to keep.

Key Terms

Principal Component: a new axis (linear combination of original features)

Explained Variance: how much spread each PC captures

Loadings: weights showing each original feature's contribution to a PC

Scree Plot: bar chart of explained variance per PC

PCA is a rotation followed by a projection – the rotation is data-driven, choosing axes that maximize variance

Why Reduce Dimensions?

Three Reasons to Compress Your Data

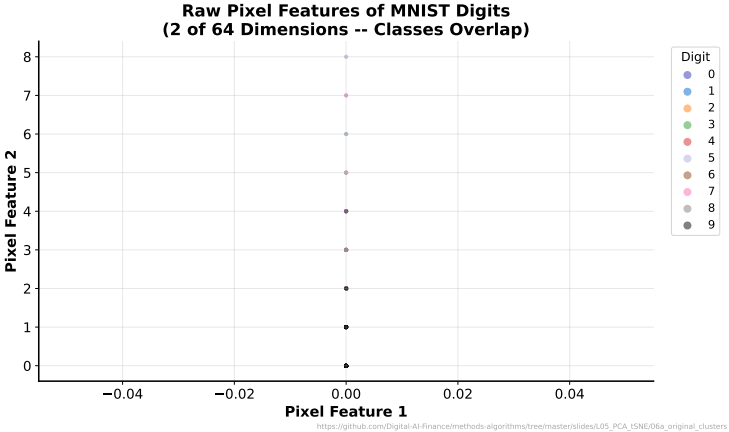
Reason	Details
Visualization	Project 50D data onto 2D to discover clusters, outliers, and trends that are invisible in tables
Noise Reduction	Low-variance directions are mostly noise. Discarding them improves signal-to-noise ratio.
Speed	Fewer features = faster training, lower memory, less overfitting. PCA from 1000 to 50 features can cut training time 20×.

When NOT to Reduce

If every feature is informative and uncorrelated, PCA will not help. PCA compresses correlated features – if there is no redundancy, there is nothing to compress.

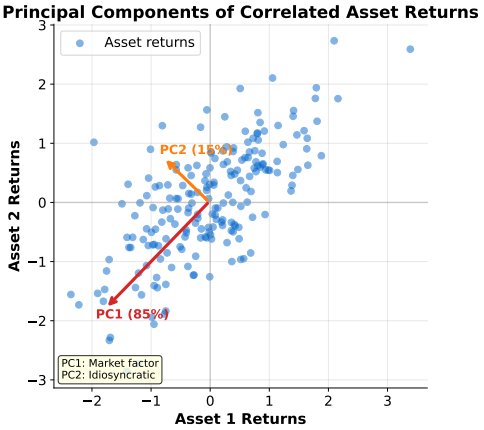
The curse of dimensionality: in high-D, all points become equidistant and nearest-neighbor methods fail

What Does the Data Look Like Before Reduction?



High-dimensional data contains clusters that may be linearly separable – PCA can reveal them by projecting onto the top PCs

How Does PCA Find the Best Axes?



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/02_principal_components

Geometric Intuition

- **PC1**: direction of maximum variance in the data cloud
- **PC2**: perpendicular to PC1, captures the next most variance
- Each subsequent PC is orthogonal to all previous ones
- The arrows show *loadings* – how each original feature contributes

Key Property

PCs are uncorrelated by construction. This is why PCA is so useful – it converts correlated features into independent components.

PCA finds the eigenvectors of the covariance matrix – each eigenvector defines a principal component direction

The Variance Maximization Objective

PCA Optimization Problem

$$\max_{\mathbf{w}} \ \mathbf{w}^T \mathbf{S} \mathbf{w} \quad \text{subject to} \quad \|\mathbf{w}\| = 1$$

where $\mathbf{S} = \frac{1}{n-1} \mathbf{X}_c^T \mathbf{X}_c$ is the sample covariance matrix of centered data $\mathbf{X}_c$.

Solution via Lagrange Multipliers

- Set up Lagrangian: $\mathcal{L} = \mathbf{w}^T \mathbf{S} \mathbf{w} - \lambda(\mathbf{w}^T \mathbf{w} - 1)$
- Take derivative, set to zero: $\mathbf{S}\mathbf{w} = \lambda \mathbf{w}$
- This is an eigenvalue equation – $\mathbf{w}$ is an eigenvector of $\mathbf{S}$, λ is the eigenvalue
- The eigenvector with the largest eigenvalue = PC1 (direction of maximum variance)

The eigenvalue λ_k equals the variance explained by PCk – larger eigenvalue means more important component

Worked Example: 2D to 1D by Hand

Data: 5 points in 2D: $(1,2)$, $(3,4)$, $(5,6)$, $(2,3)$, $(4,5)$

Step 1: Center – Mean $= (3,4)$. Centered: $(-2,-2)$, $(0,0)$, $(2,2)$, $(-1,-1)$, $(1,1)$

Step 2: Covariance matrix

$$\mathbf{S} = \frac{1}{4}\begin{pmatrix} 10 & 10 \\ 10 & 10 \end{pmatrix} = \begin{pmatrix} 2.5 & 2.5 \\ 2.5 & 2.5 \end{pmatrix}$$

Step 3: Eigenvalues – $\det(\mathbf{S} - \lambda \mathbf{I}) = 0 \;\Rightarrow\; \lambda_1 = 5,\ \lambda_2 = 0$

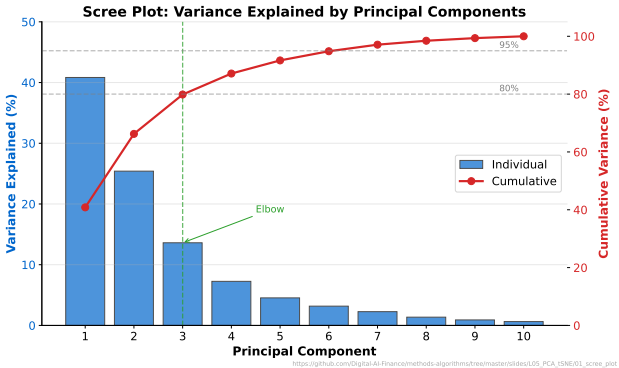
Step 4: Eigenvectors – $\mathbf{w}_1 = \frac{1}{\sqrt{2}}(1,1)^T$ (PC1), $\mathbf{w}_2 = \frac{1}{\sqrt{2}}(1,-1)^T$ (PC2)

Step 5: Project – $z_i = \mathbf{w}_1^T \mathbf{x}_i^c$: scores $= (-2\sqrt{2},\ 0,\ 2\sqrt{2},\ -\sqrt{2},\ \sqrt{2})$

Result: PC1 explains 100% of variance $(\lambda_1/(\lambda_1+\lambda_2) = 5/5 = 1.0)$. All data lies on the line $y = x$.

This example is degenerate (all points on a line) – real data has $\lambda_2 > 0$ but small, meaning PC2 captures residual noise

How Much Variance Does Each Component Explain?



Reading the Scree Plot

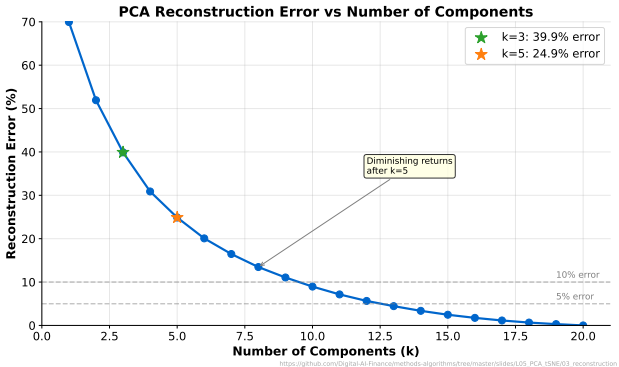
- Each bar shows the explained variance ratio of one PC
- The cumulative line shows total variance retained
- **90% threshold**: keep enough PCs to cross this line
- The "elbow" is where additional PCs add diminishing returns

Example

If 3 PCs explain 92% of variance in 50 features, you have compressed 50D to 3D with only 8% information loss.

The scree plot is named after geological scree (rubble at the base of a cliff) – stop where the cliff flattens

Reconstruction: Going Back to Original Space



The Reconstruction Formula

Project to k PCs and reconstruct:

$$\hat{\mathbf{X}} = \mathbf{Z}\mathbf{W}_k^T + \boldsymbol{\mu}$$

- $\mathbf{Z}$: scores in PC space ($n \times k$)
- $\mathbf{W}_k$: top k eigenvectors ($p \times k$)
- $\boldsymbol{\mu}$: original mean vector
- **Reconstruction error** = variance of discarded PCs

Insight

More components $\Rightarrow$ better reconstruction but less compression. The scree plot guides the tradeoff.

Reconstruction error equals $\sum_{j=k+1}^{p} \lambda_j$ – the sum of eigenvalues of the discarded components

SVD and PCA: Two Paths, One Answer

Singular Value Decomposition

Any centered data matrix can be decomposed as:

$$\mathbf{X}_c = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

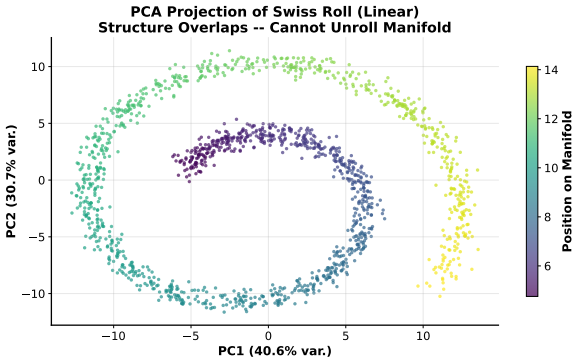
- $\mathbf{V}$ (right singular vectors) = eigenvectors of $\mathbf{X}_c^T\mathbf{X}_c$ = PCA loadings
- $\mathbf{\Sigma}$ (singular values) relate to eigenvalues: $\lambda_j = \sigma_j^2/(n-1)$
- $\mathbf{U}\mathbf{\Sigma}$ = PCA scores (projections onto principal components)

Why sklearn Uses SVD

SVD is numerically more stable than computing $\mathbf{S} = \mathbf{X}^T\mathbf{X}/(n-1)$ explicitly. It avoids squaring condition numbers and works for $p > n$ (more features than samples).

In practice, always use sklearn's PCA which calls randomized SVD – never compute the covariance matrix by hand for large datasets

When Does PCA Fail?



https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L05_PCA_tSNE/05a_pca_swiss_roll

The Linear Assumption

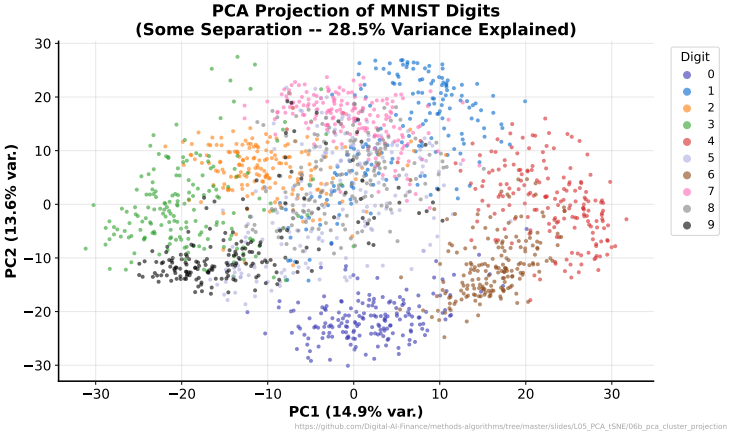
- PCA finds the best *linear* projection
- If structure is non-linear (curved manifold), PCA mixes up points
- The Swiss roll: nearby points on the surface become far apart after PCA projection
- Colors interleave instead of staying separated

Bridge to t-SNE

When data lies on a curved surface, you need non-linear methods: t-SNE, UMAP, or kernel PCA.

PCA projects onto a hyperplane – if the data manifold is curved, the projection flattens and destroys neighborhood structure

PCA Projection Preserves Linear Clusters



When clusters are linearly separable, PCA projection works well – the top 2 PCs separate the groups cleanly

Yield Curve PCA: The Canonical Finance Example

The Three Factors of Interest Rates

- Apply PCA to daily yield curve data (1Y, 2Y, 5Y, 10Y, 30Y maturities)
- **PC1 (∼85%)**: Level – all rates move together (parallel shift)
- **PC2 (∼10%)**: Slope – short rates move opposite to long rates
- **PC3 (∼3%)**: Curvature – the belly of the curve bows

Impact

3 components explain 98% of yield curve variation. Fixed income risk management is built on this decomposition.

Portfolio Hedging with PCA:

Instead of tracking 10 maturities, hedge 3 factor exposures:

$$\Delta P \approx D_1 \cdot \Delta\text{PC1} + D_2 \cdot \Delta\text{PC2} + D_3 \cdot \Delta\text{PC3}$$

where D_k = factor duration

Litterman & Scheinkman (1991) showed 3 PCA factors explain >98% of US Treasury yield curve movements

Portfolio Risk Decomposition

From 50 Stocks to 5 Principal Portfolios

1. Compute daily returns matrix: $\mathbf{R}$ ($T \times 50$, where T = trading days)
2. Standardize each column (zero mean, unit variance)
3. Run PCA: obtain eigenvectors $\mathbf{w}_1, \ldots, \mathbf{w}_{50}$
4. **PC1** = market factor (all stocks load positively) – explains $\sim$40% of variance

Interpretation

Each PC defines a "principal portfolio" – a linear combination of stocks. PC1 is the market, PC2 is often sector rotation (tech vs. energy), PC3 may capture momentum vs. value.

PCA on returns is the basis for statistical factor models – an alternative to Fama-French style predefined factors

Choosing the Number of Components

Four Methods

- **Scree plot elbow**: visual inspection for the "cliff edge"
- **Kaiser criterion**: keep PCs with eigenvalue > 1 (when using correlation matrix)
- **Cumulative variance**: keep enough PCs for 90–95% of total variance
- **Cross-validation**: choose k that optimizes a downstream task (classification, regression)

Method	Pro	Con
Scree elbow	Simple	Subjective
Kaiser	Automatic	Arbitrary
Cum. var.	Interpretable	Ignores task
CV	Task-optimal	Expensive

Practical Advice

Start with the 90% cumulative variance rule. If you have a downstream model, use cross-validation to fine-tune.

In sklearn: `PCA(n_components=0.95)` **automatically selects enough PCs to explain 95% of variance**

PCA as Preprocessing for ML

sklearn Pipeline: Scale → PCA → Classify

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('pca',    PCA(n_components=0.95)),
    ('clf',    LogisticRegression())
])

pipe.fit(X_train, y_train)
print(f"Accuracy: {pipe.score(X_test, y_test):.3f}")
print(f"Components kept: {pipe['pca'].n_components_}")
```

Why This Works

PCA removes noise (low-variance directions) and reduces multicollinearity. The classifier sees cleaner, lower-dimensional input.

Always scale before PCA – features with larger units dominate the covariance matrix and distort the principal components

Interpreting Principal Components

The Loadings Matrix

- Each PC is a weighted sum of original features
- The weight of feature j on PC k is called the **loading** w_{jk}
- Large positive loading: feature moves with the PC
- Large negative loading: feature moves against the PC

Example: Stock Returns

Stock	PC1	PC2
Apple	$+0.18$	$+0.35$
JPMorgan	$+0.20$	-0.28
Exxon	$+0.15$	-0.31
Google	$+0.19$	$+0.33$

PC1: all positive $\Rightarrow$ **market factor**
PC2: tech positive, banks/energy negative $\Rightarrow$ **sector rotation**

Loadings are the eigenvector components – they tell you what each PC "means" in terms of original features

When to Use PCA — and When Not To

Pros

- Interpretable components (loadings tell you what each PC means)
- Fast: $O(np^2)$ or less with randomized SVD
- Preserves global variance structure
- Invertible: can reconstruct original data

Cons

- Linear only – fails on curved manifolds (Swiss roll)
- Sensitive to feature scaling (must standardize first)
- Loses local neighborhood structure
- Maximizes variance, which may not equal "importance"

Rule of Thumb

Use PCA when you need preprocessing, interpretability, or compression. Switch to t-SNE/UMAP when you need to *visualize* non-linear structure.

PCA is the default first step in dimensionality reduction – try it before anything more complex

PCA vs Factor Analysis vs Autoencoders

Property	PCA	Factor Analysis	Autoencoder
Linearity	Linear	Linear	Non-linear
Interpretability	High (loadings)	High (factors)	Low (black box)
Component selection	Scree/CV	Likelihood ratio	Architecture choice
Computational cost	Low	Medium	High (GPU)
Non-linear capability	No	No	Yes
Noise model	None	Explicit	Implicit
Reconstruction	Exact (with all PCs)	Approximate	Approximate

Practical Choice

Start with PCA (fast, interpretable). If you need a latent variable model with noise, use Factor Analysis. If you need non-linear compression, use an autoencoder.

Factor Analysis assumes data = factors + noise; PCA makes no noise assumption – it just maximizes explained variance

Hands-on: Decompose Portfolio Risk

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np

# Simulate 250 days x 50 stocks
np.random.seed(42)
returns = np.random.randn(250, 50) @ np.random.randn(50, 50) * 0.01

# Scale and fit PCA
scaler = StandardScaler()
X = scaler.fit_transform(returns)
pca = PCA().fit(X)

# Scree: cumulative explained variance
cumvar = np.cumsum(pca.explained_variance_ratio_)
k_90 = np.searchsorted(cumvar, 0.90) + 1
print(f"Components for 90% variance: {k_90}")
```

Full notebook: `notebooks/L05_pca.ipynb`

Try this yourself: load real stock returns from Yahoo Finance and see how many PCs you need for 90% explained variance

Key Takeaways

Concepts

- PCA = eigenvectors of the covariance matrix
- Eigenvalue = variance explained by that PC
- Reconstruction error = discarded eigenvalue sum
- Scree plot guides component selection

Practice

- Always standardize features before PCA
- Use the scree plot 90% rule as a starting point
- Interpret loadings to understand what each PC means
- Test with the Swiss roll: if PCA fails, try t-SNE or UMAP

One-Sentence Summary

PCA rotates your data to find the axes of maximum variance, letting you compress, denoise, and visualize high-dimensional data.

PCA is a tool you will use in nearly every data science project – master it.

What's Next: t-SNE for Visualization

PCA Finds Global Axes. t-SNE Finds Local Neighborhoods.

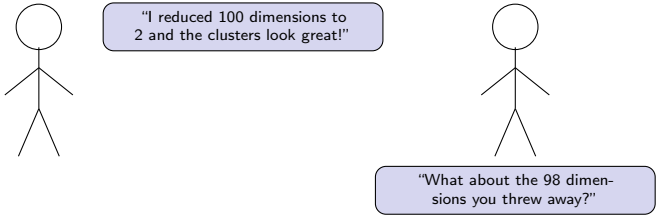
- PCA captures the **directions of maximum spread** – great for compression and preprocessing
- But when the data lives on a curved manifold, PCA's linear projection fails
- **t-SNE** minimizes the mismatch between high-D and low-D *neighborhoods*
- Result: clusters that PCA merges become clearly separated in t-SNE

Coming Up

Next lecture: t-SNE's KL divergence objective, the perplexity parameter, the crowding problem, and UMAP as a modern alternative.

Best practice: run PCA first to reduce to 30–50 dimensions, then apply t-SNE on the PCA output for faster and more stable visualization

The Dimensionality Reduction Dilemma



Always validate: does low-D structure reflect high-D reality? Check reconstruction error and trustworthiness.