ELEC3544 · Fall 2026

Data Science with Foundation Models

Yang Liu

The University of Hong Kong

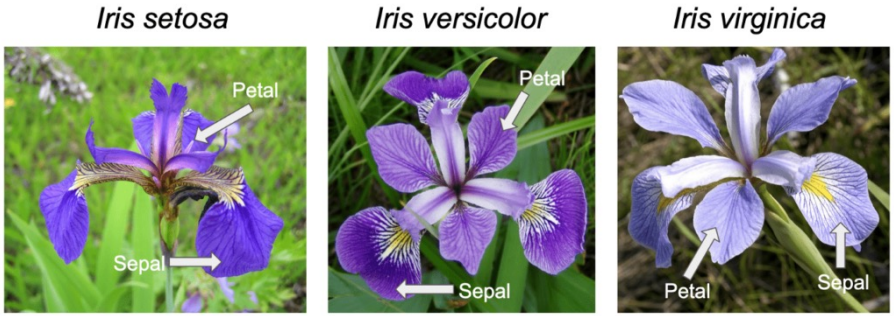


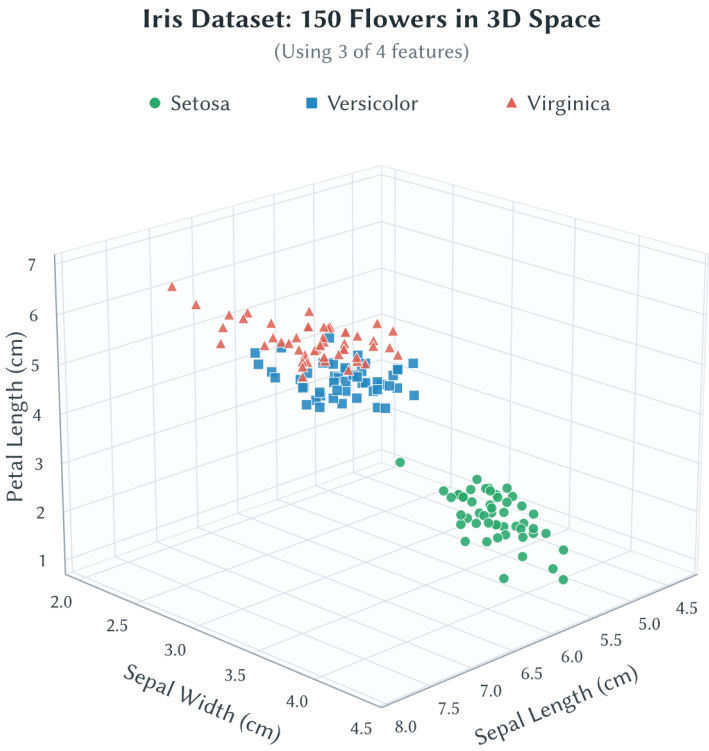
Linear Algebra for Data Science

Seeing Data as Geometry

The Hidden Geometry of Data

"Every dataset is a cloud of points floating in space."





Today we'll learn to see your data this way —
and extract meaning from it.

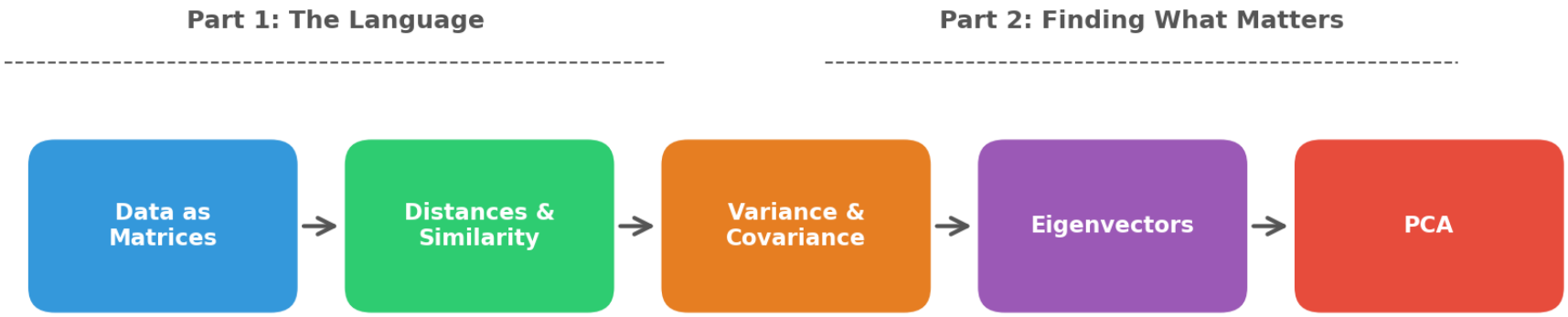
[UCI ML repo] https://archive.ics.uci.edu/dataset/53/iris

What You'll Learn Today

1. Represent datasets as matrices and compute distances/similarities
2. Explain how the covariance matrix captures feature relationships
3. Apply PCA to reduce dimensionality — understand why eigenvectors define components
4. Interpret PCA results by selecting components using explained variance
5. Practice sophisticated CTA collaboration: iterative refinement and verification

Key Insight: Linear algebra transforms data from spreadsheet rows into geometric objects you can measure and manipulate.

The Journey Ahead



Connection to Course

Why does linear algebra matter for data science?

Lecture 3: Probability: Covariance connects to joint distributions

Lecture 5: Regression: Solving normal equations uses linear algebra

Lecture 6: Classification: Linear classifiers have hyperplane boundaries

Lecture 7: Embeddings: Similarity search uses vector distances

🤖 **Your CTA** will help you explore these concepts through demonstrations.

From Spreadsheet to Matrix

The same data, two different lenses

Pandas DataFrame

Index	sepal_len	sepal_wid	petal_len	petal_wid
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
...	...	...	...	...
149	5.9	3.0	5.1	1.8

n = 150 rows (observations)

p = 4 columns (features)

→

Mathematical Matrix

$$X = \begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 4.9 & 3.0 & 1.4 & 0.2 \\ 4.7 & 3.2 & 1.3 & 0.2 \\ \vdots & \vdots & \vdots & \vdots \\ 5.9 & 3.0 & 5.1 & 1.8 \end{bmatrix}$$

n

p

$X \in \mathbb{R}^{n \times p} \rightarrow X \in \mathbb{R}^{150 \times 4}$

Key Insight: A spreadsheet is for humans (labels, types, context).
A matrix is for math (pure numbers, no labels). **Same information, different representation.**

The Data Matrix

Symbol	Domain	Meaning
X	$\mathbb{R}^{n \times p}$	Data matrix — all our data in one object
n	$\mathbb{Z}^{+}$	Number of observations (rows) — how many data points
p	$\mathbb{Z}^{+}$	Number of features (columns) — measurements per point
x_i	$\mathbb{R}^{p}$	The i-th observation — a single data point (row vector)

EQ1: Data Matrix Notation

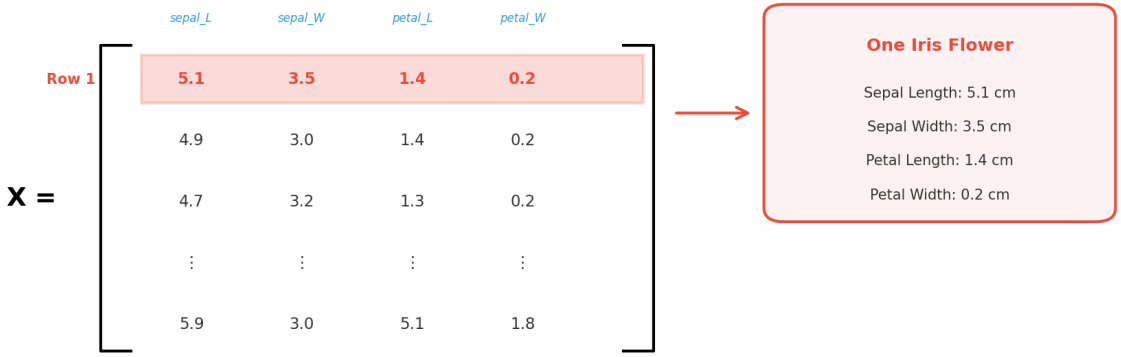
$$X \in \mathbb{R}^{n \times p}$$

Interpretation: Every dataset with n observations and p features is a matrix X.

Example: Iris dataset has $n = 150$ flowers and $p = 4$ measurements $\longrightarrow X \in \mathbb{R}^{150 \times 4}$

Rows as Observations

Each row x_i represents one data point — one observation in our dataset.



Each row $\mathbf{x}_i \in \mathbb{R}^p$ is one observation (one flower)

Row 1: $\mathbf{x}_1 = [5.1, 3.5, 1.4, 0.2]$

Example: Iris Flower #62

x_{62} = [5.9, 3.0, 4.2, 1.5] — One versicolor flower: 5.9cm sepal length, 3.0cm sepal width, 4.2cm petal length, 1.5cm petal width

Columns as Features

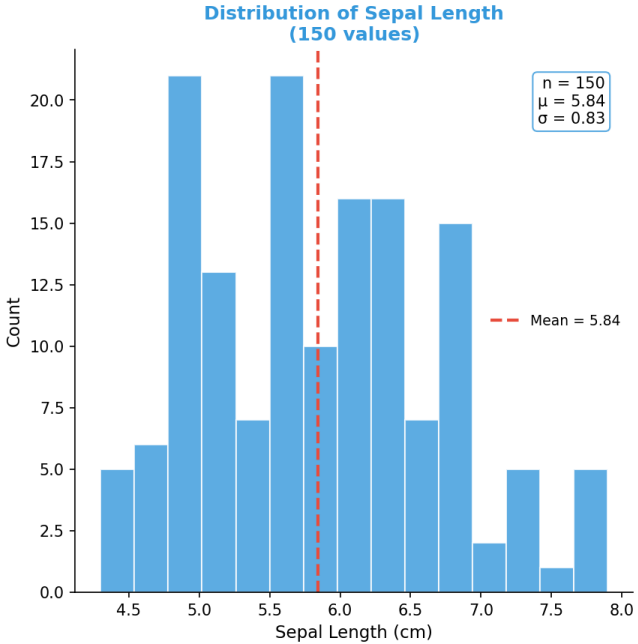
Each column represents one measurement — one feature across all observations.

$X =$

sepal_L	sepal_W	petal_L	petal_W
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
⋮	⋮	⋮	⋮
5.9	3.0	5.1	1.8

Col 1

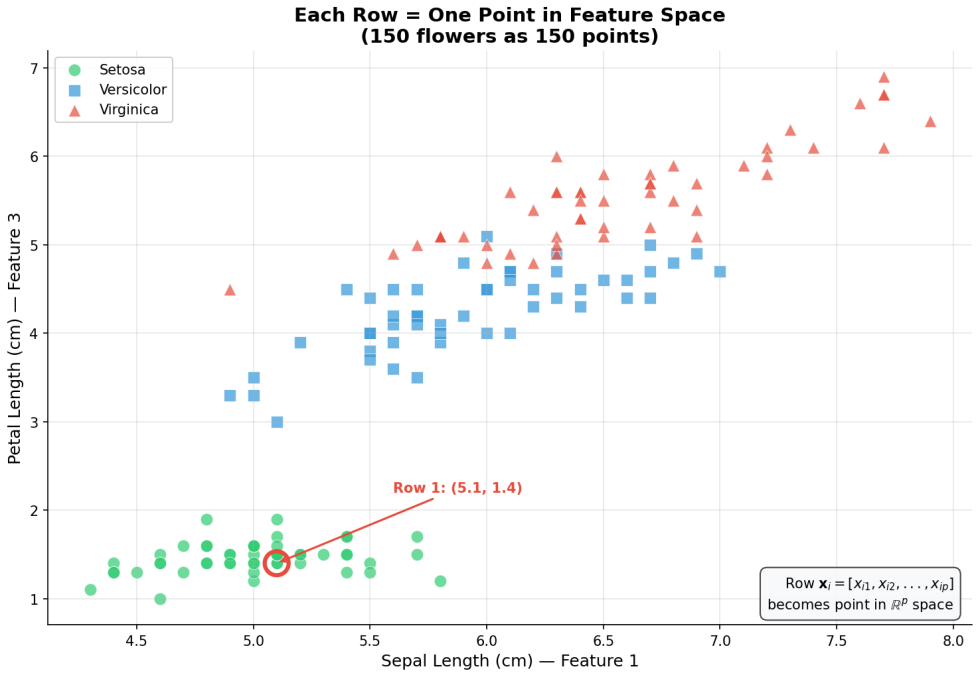
Each column = one feature across all observations



Each column has its own distribution, mean, and variance — this will matter for PCA.

The Geometric View

Key Insight: Each row x_i is a point in p-dimensional space. Your dataset is a cloud of n points.



For Iris: each flower is a point in 4D space. We are seeing a 2D projection above.

Extending to Higher Dimensions

The geometric intuition extends beyond what we can visualize

Iris flowers	4 dimensions	Can project to 2D/3D
MNIST digits	784 dimensions	Each pixel in a 28×28 image is one feature
Gene expression	~20,000 dimensions	Each gene is a dimension
Word embeddings	e.g., 300 dimensions (tokens)	Meaning encoded in high-D space

⚠️ **The Curse of Dimensionality (Preview)**

High dimensions create counter-intuitive effects: distances become similar, data becomes sparse, and nearest neighbors become less meaningful. We need to reduce the dimensionality – *e.g.* using PCA.

Why This View Matters

The geometric view unlocks powerful capabilities:

Measure Distances

Quantify similarity between data points

Find Structure

Identify relationships between features

Reduce Dimensions

Find directions that matter most

Key Insight: Spreadsheet ⟶ Matrix ⟶ Points in Space.
This transformation enables the entire toolkit of linear algebra for data science.

Measuring Relationships

Quantifying how close or similar data points are

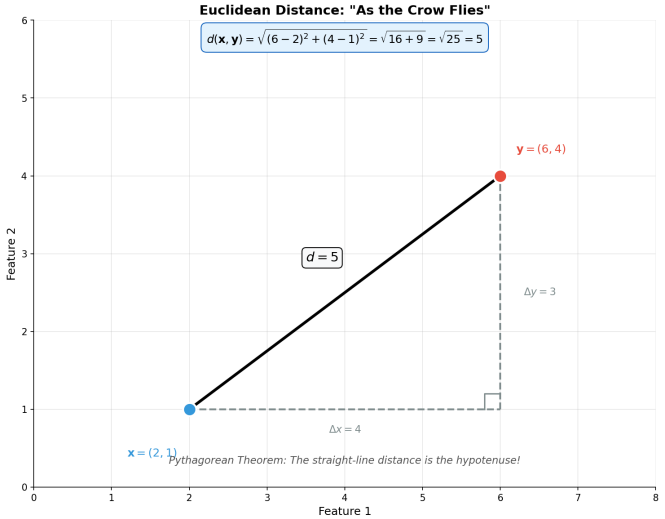
Euclidean Distance: The Straight Line

The most intuitive distance — as the crow flies.

EQ2: Euclidean Distance

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

Sum of squared differences, then take the square root.



Connection to Pythagorean Theorem: In 2D, $d = \sqrt{[(x_2-x_1)^2 + (y_2-y_1)^2]}$. This is just $c = \sqrt{(a^2 + b^2)}$ — the distance formula from geometry, extended to any number of dimensions.

Properties:

- Always ≥ 0 (non-negative)
- $d(x, y) = 0$ only when $x = y$
- Symmetric: $d(x, y) = d(y, x)$
- Sensitive to scale

Euclidean Distance Example

Computing distance between two Iris flowers

Example: Distance Between Two Setosa Flowers

Flower A: $x_A = [5.1, 3.5, 1.4, 0.2]$ (sepal length, sepal width, petal length, petal width)

Flower B: $x_B = [4.9, 3.0, 1.4, 0.2]$

Step 1: Compute differences

$x_A - x_B = [5.1-4.9, 3.5-3.0, 1.4-1.4, 0.2-0.2] = [0.2, 0.5, 0, 0]$

Step 2: Square each difference

$[0.2^2, 0.5^2, 0^2, 0^2] = [0.04, 0.25, 0, 0]$

Step 3: Sum and take square root

$d(A, B) = \sqrt{(0.04 + 0.25 + 0 + 0)} = \sqrt{0.29} \approx 0.539$ cm

Result: These two flowers are 0.539 units apart in 4D feature space.

Cosine Similarity: The Angle

Measures the angle between vectors — ignores magnitude, focuses on direction.

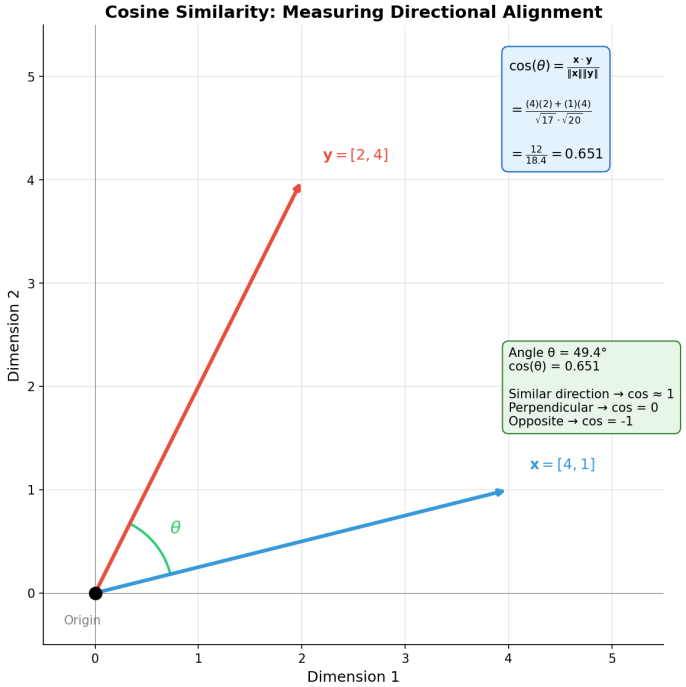
EQ3: Cosine Similarity

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

Dot product divided by product of magnitudes.

Range: [-1, +1]

+1 = same direction
0 = perpendicular
-1 = opposite direction



Key Insight: Cosine similarity only cares about direction, not length.
Two vectors pointing the same way have cosine = 1, regardless of their magnitudes.

Properties:

- Bounded [-1, 1]
- Scale-invariant (2x doesn't change similarity)
- Symmetric
- NOT a true distance metric

Cosine Similarity Example

Same two flowers — now measuring angular similarity

Example: Cosine Similarity of Two Setosa Flowers

Flower A: $x_A = [5.1, 3.5, 1.4, 0.2]$ Flower B: $x_B = [4.9, 3.0, 1.4, 0.2]$

Step 1: Compute dot product ($x_A \cdot x_B$)

$(5.1 \times 4.9) + (3.5 \times 3.0) + (1.4 \times 1.4) + (0.2 \times 0.2) = 24.99 + 10.5 + 1.96 + 0.04 = 37.49$

Step 2: Compute magnitudes

$\|x_A\| = \sqrt{(5.1^2 + 3.5^2 + 1.4^2 + 0.2^2)} = \sqrt{40.26} \approx 6.35$

$\|x_B\| = \sqrt{(4.9^2 + 3.0^2 + 1.4^2 + 0.2^2)} = \sqrt{35.01} \approx 5.92$

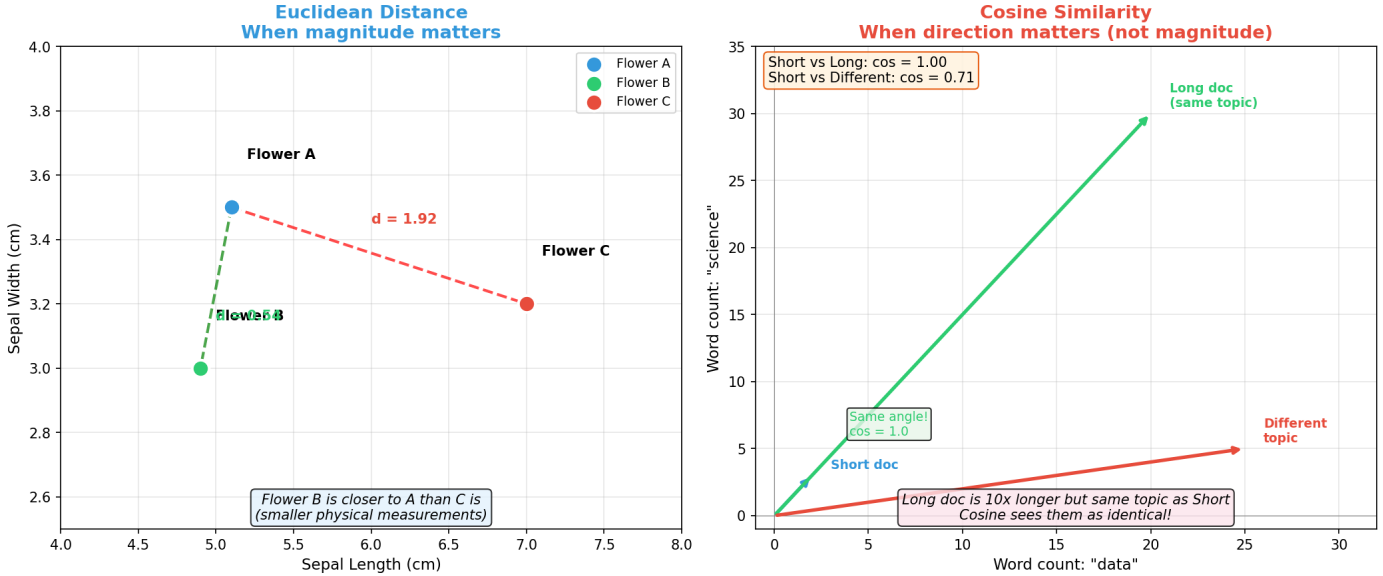
Step 3: Divide

$\cos(A, B) = 37.49 / (\sqrt{40.26} \times \sqrt{35.01}) \approx 0.99858 \approx 0.999$

Result: Cosine similarity ≈ 0.999 — these flowers point in nearly the same direction!

Euclidean vs. Cosine: Visual Comparison

The two metrics can give different answers about which points are "closest"



Key Insight: Euclidean cares about absolute position in space. Cosine cares about direction from origin. Neither is "better" — the right choice depends on what similarity means for your problem.

When to Use Which?

	Euclidean Distance	Cosine Similarity
Measures	Absolute position/magnitude	Direction/orientation
Scale sensitive?	Yes — affected by magnitude	Invariant to positive vector rescaling
Best for	Physical measurements, when scale matters	Text, sparse data, when direction matters
Typical use cases	k-NN, clustering physical data, image pixels	Document similarity, recommendation systems

The choice between metrics is a modeling decision — it reflects what "similar" means in your context.

The Choice Matters

A document similarity example to set up our CTA exploration

Example: Comparing Customer Reviews

Vocabulary: great, product, fast, shipping, terrible, broken, on, arrival.

Review A (short): "Great product, fast shipping" $\longrightarrow$ [1, 1, 1, 1, 0, 0, 0, 0]

Review B (long): A's phrase repeated three times $\longrightarrow$ [3, 3, 3, 3, 0, 0, 0, 0]

Review C (short, different topic): "Terrible, broken on arrival" $\longrightarrow$ [0, 0, 0, 0, 1, 1, 1, 1]

Question: Which review is more similar to A?

- Euclidean prefers C: $d(A,C) = \sqrt{8} < d(A,B) = 4$.
- Cosine prefers B: $\cos(A,B) = 1 > \cos(A,C) = 0$.

🤖 Let's Explore This with the CTA

Rather than memorizing rules, let's build intuition through conversation.
We'll ask the CTA when to use each metric and verify with code.

CTA Demo 1: Iterative Refinement

Building understanding through progressive questioning

🤖 What We'll Explore

When should you use Euclidean distance vs. cosine similarity? We'll start broad, get specific, ask for examples, and verify with code — building understanding through conversation, not memorization.

Key Questions We'll Ask:

1. "When should I use Euclidean distance vs. cosine similarity?"
2. "Give me a concrete example where they rank pairs differently."
3. "For text data specifically — which would you recommend?"
4. "Let's verify with code: create vectors where they disagree."

Collaboration Pattern: Iterative Refinement

VERIFY: Examine, Investigate, Validate

🤖 CTA Demo: Exploring Distance Metrics

LIVE DEMO

Watch the iterative refinement pattern in action:

Start broad → Get specific → Request examples → Verify with code

Notice: We don't need the perfect prompt. We have a conversation.

Pattern: Iterative Refinement

Demo 1 Takeaways

Content Insight: Distance Metrics

- Euclidean: Use when absolute magnitude matters (physical measurements, images)
- Cosine: Use when direction matters more than scale (text, sparse data)
- The choice is a modeling decision — it defines what "similar" means

🤖 Collaboration Pattern: Iterative Refinement

- Start with a broad question — don't stress about the "perfect" prompt
- Follow up to get specific: "Give me a concrete example..."
- Verify understanding with code or computation
- Build understanding through conversation, not memorization

✓ VERIFY Framework Connection

Examine: We checked reasoning with specific examples
Investigate: We explored the text data edge case
Validate: We verified with working code

Relationships Between Features

Transitioning from measuring between rows to measuring between columns

Distances & Similarities

How similar are two observations?
(Comparing rows)

→

Covariance Matrix

How do features relate to each other?
(Comparing columns)

The Key Question:

When one feature increases, what happens to other features? Do they increase together? Decrease? Move independently?

Why this matters: The covariance matrix captures the "shape" of your data cloud — essential for PCA.

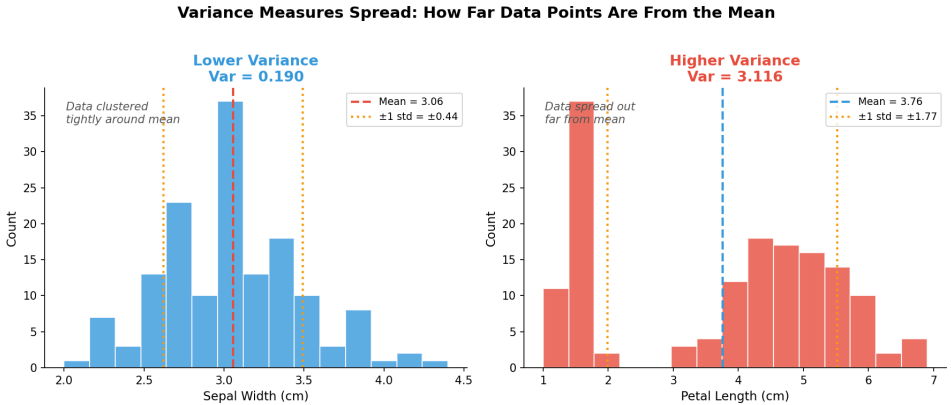
Variance: Spread of One Feature

Before covariance, let's recall variance — how spread out a single feature is.

EQ4: Variance (Sample)

$$\text{Var}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Average squared deviation from the mean.
Larger variance = more spread.



Intuition: Variance measures how much a feature "varies" around its mean.
If all values are the same, variance = 0. More spread = higher variance.

Why *n*-1 instead of *n*? This is Bessel's correction for sample variance. We use *n*-1 because we estimated the mean from the same data. The degrees of freedom is ***n*-1**. **This makes our variance estimate unbiased**.

Why Divide by $n-1$?

$$\text{Bias(estimate)} = \text{E[estimate]} - \text{true value}$$

Expectation averages estimates over repeated samples of the same size.

Why the correction works

i.i.d. observations; mean μ, finite variance σ^2; $n \geq 2$.

$$R = \sum_{i=1}^{n} (X_i - \bar{X})^2$$

$$R = \sum (X_i - \mu)^2 - n(\bar{X} - \mu)^2$$

$$\mathrm{E}[R] = n\sigma^2 - n\left(\frac{\sigma^2}{n}\right) = (n-1)\sigma^2$$

Divide by n

$$\mathrm{E}[R/n] = \frac{n-1}{n}\,\sigma^2$$

Divide by $n-1$

$$\mathrm{E}[R/(n-1)] = \sigma^2$$

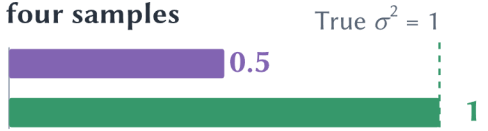
The deviations sum to zero: one constraint.
That leaves $n-1$ degrees of freedom.

Exact repeated sampling

$X = -1$ or $+1$ with probability 1/2 each.
$\mu = 0$, $\sigma^2 = 1$; $n = 2$ (each sample: probability 1/4).

Sample	R	R/n	$R/(n-1)$
$(-1, -1)$	0	0	0
$(-1, +1)$	2	1	2
$(+1, -1)$	2	1	2
$(+1, +1)$	0	0	0

Average over all four samples



Unbiased means correct on average across repeated samples.
A single estimate can still be too high or too low.

Covariance: Two Features Together

Extending variance to capture how two features vary together.

EQ5: Covariance

Product of deviations from each mean, averaged.

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

- $\mathrm{Cov}(X, Y)$ = sample covariance between features X and Y
- x_i, y_i = the i-th observation of features X and Y
- $\bar{x}, \bar{y}$ = sample means of X and Y

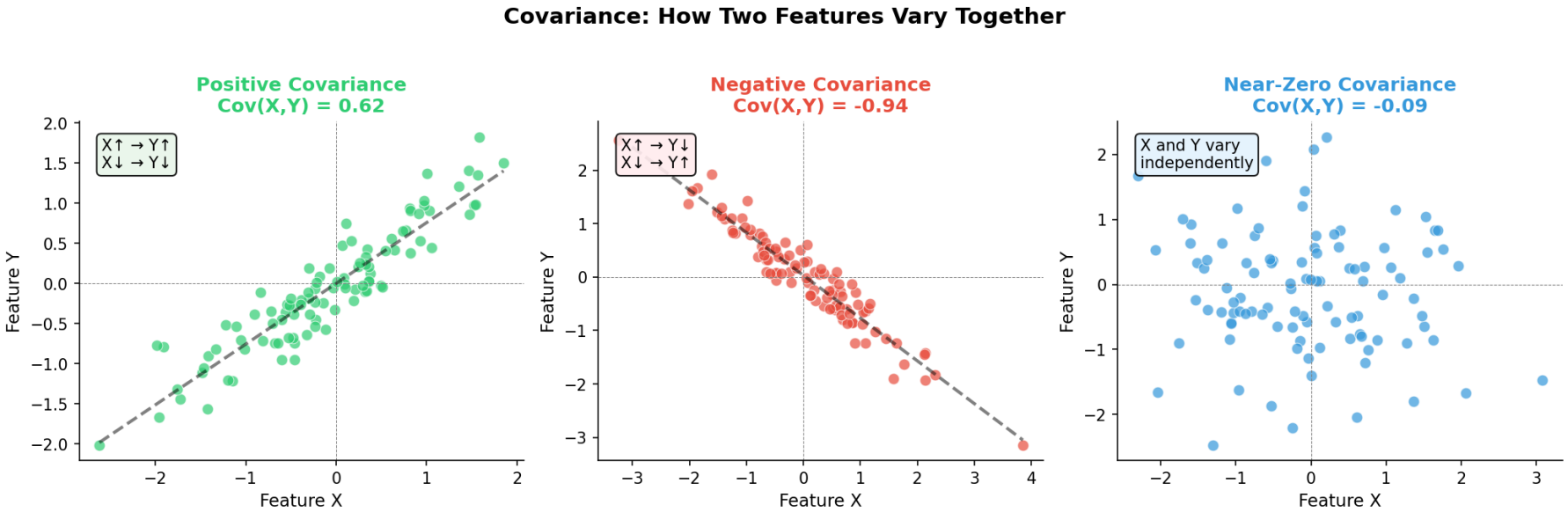
Interpretation:

Cov(*X*, *Y*) > 0	Positive covariance	When *X* is above its mean, *Y* tends to be above its mean too
Cov(*X*, *Y*) < 0	Negative covariance	When *X* is above its mean, *Y* tends to be below its mean
Cov(*X*, *Y*) ≈ 0	Near-zero covariance	Little linear association does not imply independence.

Note: Cov(X,X) = Var(X) — covariance of a feature with itself is just its variance.

Visualizing Covariance

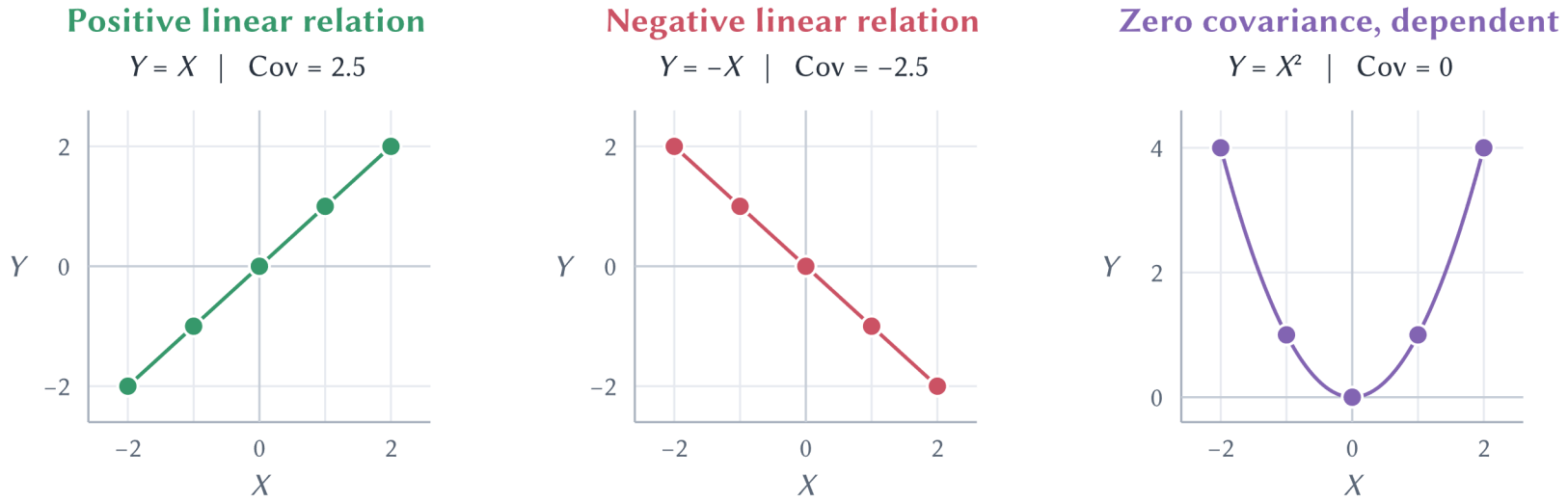
Covariance measures linear co-variation; its magnitude depends on units.



Key Insight: Zero covariance does not imply independence.
In the third panel, Y is completely determined by X, yet covariance is zero.

Visualizing Covariance

Let's see a simpler case!



Synthetic data • 5 observations per panel • sample covariance uses n − 1

Key Insight: Zero covariance does not imply independence.
In the third panel, Y is completely determined by X, yet covariance is zero.

The Covariance Matrix

Collecting all pairwise covariances into one matrix.

EQ6: Covariance Matrix

A $p \times p$ matrix containing all pairwise covariances between features.

$$\Sigma \in \mathbb{R}^{p \times p}, \quad \Sigma_{ij} = \mathrm{Cov}(X_i, X_j)$$

- Σ = the covariance matrix (capital sigma)
- p = number of features
- Σ_{ij} = element in row i, column j of Σ

Structure of the Covariance Matrix:

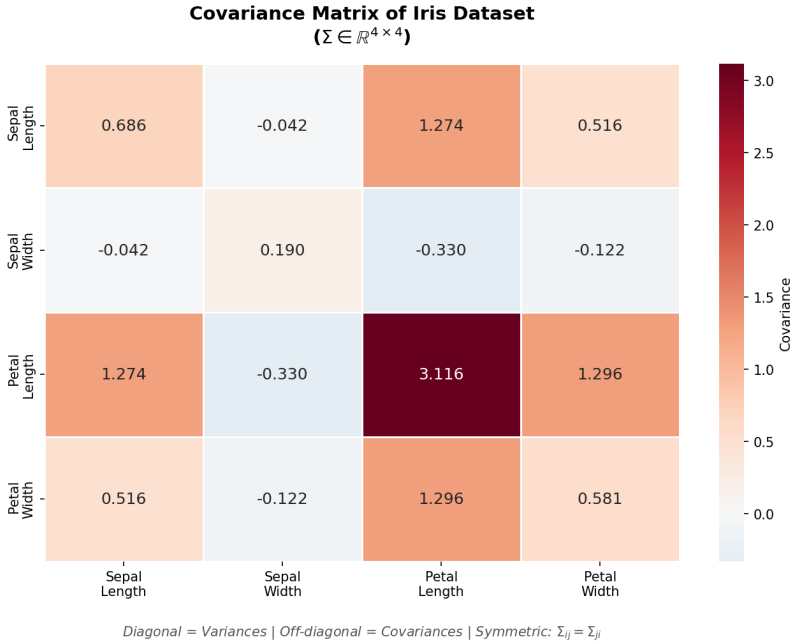
	Feature 1	Feature 2	Feature 3
Feature 1	$\mathrm{Var}(X_1)$	$\mathrm{Cov}(X_1,X_2)$	$\mathrm{Cov}(X_1,X_3)$
Feature 2	$\mathrm{Cov}(X_2,X_1)$	$\mathrm{Var}(X_2)$	$\mathrm{Cov}(X_2,X_3)$
Feature 3	$\mathrm{Cov}(X_3,X_1)$	$\mathrm{Cov}(X_3,X_2)$	$\mathrm{Var}(X_3)$

Key Properties:

- Diagonal = variances of each feature
- Off-diagonal = covariances between features
- Symmetric: $\Sigma_{ij} = \Sigma_{ji}$ (order does not matter)
- Positive semidefinite (all eigenvalues ≥ 0)

Reading the Covariance Matrix

Interpreting the Iris dataset covariance matrix as a heatmap.



What We See:

- Petal length & width have high positive covariance (bright)
- Sepal width has low covariance with petals (dim)
- Diagonal shows variances — petal length has highest

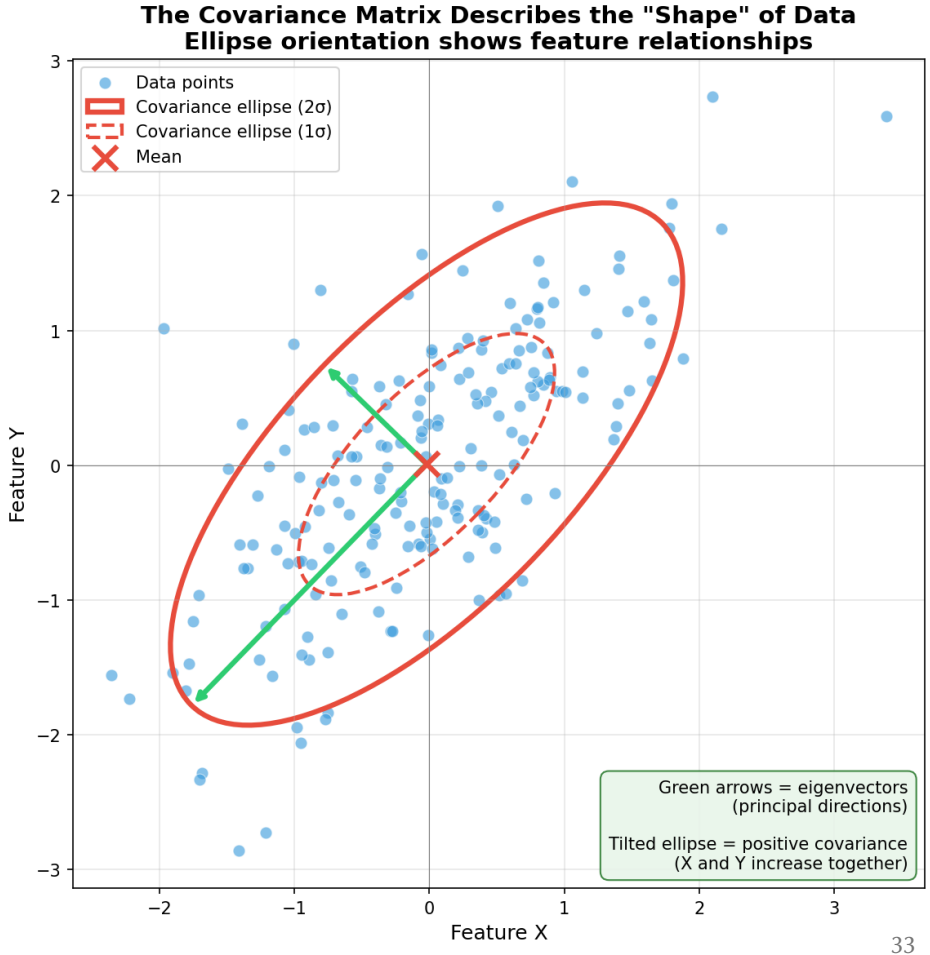
Key Insight: The covariance matrix reveals which features "move together" — this is exactly what PCA will exploit to find principal components.

The Shape of Data

The covariance matrix describes the ellipsoid shape of your data cloud.

Key Insight: The Bridge to PCA

The covariance matrix encodes the "shape" of your data cloud. We'll see that the eigenvectors of this matrix point along the principal axes of this ellipsoid — these are the principal components!



CTA Demo 2: Verifying Code from the CTA

Asking the CTA to write code, then verifying it actually works

What We'll Do

We'll ask the CTA to write Python code that computes the covariance matrix for the Iris dataset. Then we'll run the code ourselves to verify it works — and check if the output makes sense.

Our Prompt to the CTA:

"Write Python code that loads the Iris dataset and computes its covariance matrix. Explain what each step does and what the output tells us about feature relationships."

Our Verification Process:

1. Run the code — does it execute without errors?
2. Check the output — is it a 4×4 symmetric matrix? Are diagonal values positive?
3. Sanity check — do high covariances match what we expect from the scatter plots?

Pattern: Verifying CTA-Generated Code

VERIFY: Validate, Examine, Investigate

🤖 CTA Demo: Verifying Covariance

LIVE DEMO

Watch the verification through computation pattern:

Ask for step-by-step work → Verify independently → Investigate discrepancy → Learn!

Pattern: Verification Through Computation

Demo 2 Takeaways

Content Insight: *n* vs. *n*-1 Matters!

- Population covariance uses *n*; sample covariance uses *n*-1 (Bessel's correction)
- NumPy's `np.cov()` uses *n*-1 by default (sample covariance)
- This isn't a technicality — it affects statistical inference. Always know which you're using!

🤖 Collaboration Pattern: Verification Through Computation

- Ask for step-by-step work — makes verification possible
- Always verify independently with trusted tools (NumPy, calculator, etc.)
- Discrepancies aren't failures — they're learning opportunities
- The goal is understanding, not just getting an answer

✓ VERIFY Framework Connection

Validate: We checked against NumPy | Examine: We investigated the discrepancy | Flag: We noted the unstated *n* vs. *n*-1 assumption

Meta-Lesson: Verification isn't about distrust — it's about understanding.
The discrepancy taught us more than a "correct" answer would have.

Part 1 Recap

Three transformations that changed how we see data

Spreadsheet → Matrix

Data becomes a mathematical object we can manipulate

Matrix → Points in Space

Each row is a point in p-dimensional space

Points → Covariance Structure

The covariance matrix captures how features relate

Tools Acquired:

Euclidean distance & cosine similarity (measuring between points) • Variance & covariance (measuring feature relationships) • The covariance matrix (capturing data shape)

Looking Ahead: Part 2

Finding the directions that matter most

The Central Question of Part 2:

The covariance matrix captures "shape" — but which directions capture the MOST variance?
How do we find them? And how do we use them to reduce dimensionality?

Coming Up:

- **Eigenvectors & Eigenvalues:** Special directions where matrices just stretch
- **PCA Algorithm:** Center → Covariance → Eigenvectors → Project
- **Interpreting Components:** What do principal components "mean"?
- **Selecting Components:** How many to keep? The 90% rule and beyond

🤖 CTA: Debugging Understanding + Challenging for Completeness

Finding What Matters

Principal Component Analysis (PCA)

The Problem with High Dimensions

Real-world data often has far more features than we can visualize or efficiently process.

Examples of High-Dimensional Data:

Images (MNIST)	784 dimensions	28×28 pixels, each is a feature
Gene Expression	~20,000 dimensions	One dimension per gene
Text (Bag of Words)	10,000+ dimensions	One dimension per word in vocabulary
Sensor Networks	100s-1000s dimensions	One dimension per sensor reading
Financial Data	100s dimensions	Multiple indicators per asset

The Question: How do we work with data when feature dimension $p >> 3$ (or even feature dimension $p >>$ sample dimension n)?

The Curse of Dimensionality

High dimensions create counter-intuitive problems that break our low-dimensional intuitions.

Distances Converge

In many high-dimensional distributions, nearest and farthest distances become similar relative to their size. This depends on the data and metric.

Data Becomes Sparse

Ten intervals per feature create 10^p cells. Maintaining the same coverage can require exponentially more data.

Nearest Neighbors Can Lose Contrast

When all distances are similar, "nearest" loses meaning. Algorithms relying on local structure struggle.

Key Insight: The curse isn't about computation — it's about the geometry of high-dimensional space itself. We need methods that find lower-dimensional structure.

The Need to Reduce

Three motivations for dimensionality reduction

👁️ Visualization

We can only see in 2D or 3D. To visually explore high-dimensional data, we must project it down while preserving structure.

⚡ Computation

Many algorithms scale poorly with dimensions ($O(p^2)$ or worse). Reducing p before modeling can dramatically speed up training.

🎯 Generalization

More features = more parameters = more risk of overfitting.
Reducing dimensions can improve model generalization.

But how do we reduce dimensions without losing important information?

The Goal: Preserve What Matters

PCA's big idea: Find the directions that capture the most variance.

The Core Insight of PCA

Not all directions in feature space are equally important. Some directions capture lots of variation in the data; others capture almost none.

If we project onto the high-variance directions and ignore the low-variance ones, we can reduce dimensions while preserving most of the "signal" in the data.

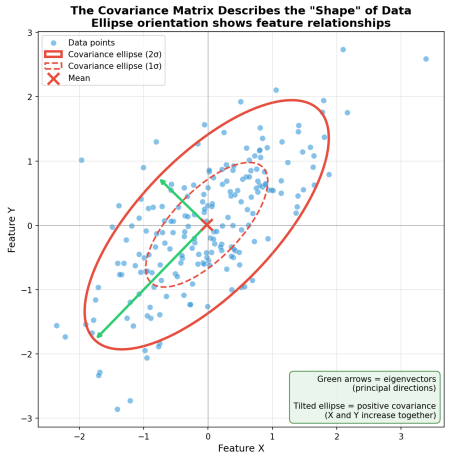
Visual Intuition:

Imagine a cigar-shaped data cloud

The long axis captures most variance.
The short axis captures little.

PCA finds these axes automatically.

Next question: How do we find these "maximum variance" directions mathematically?



The Key Question

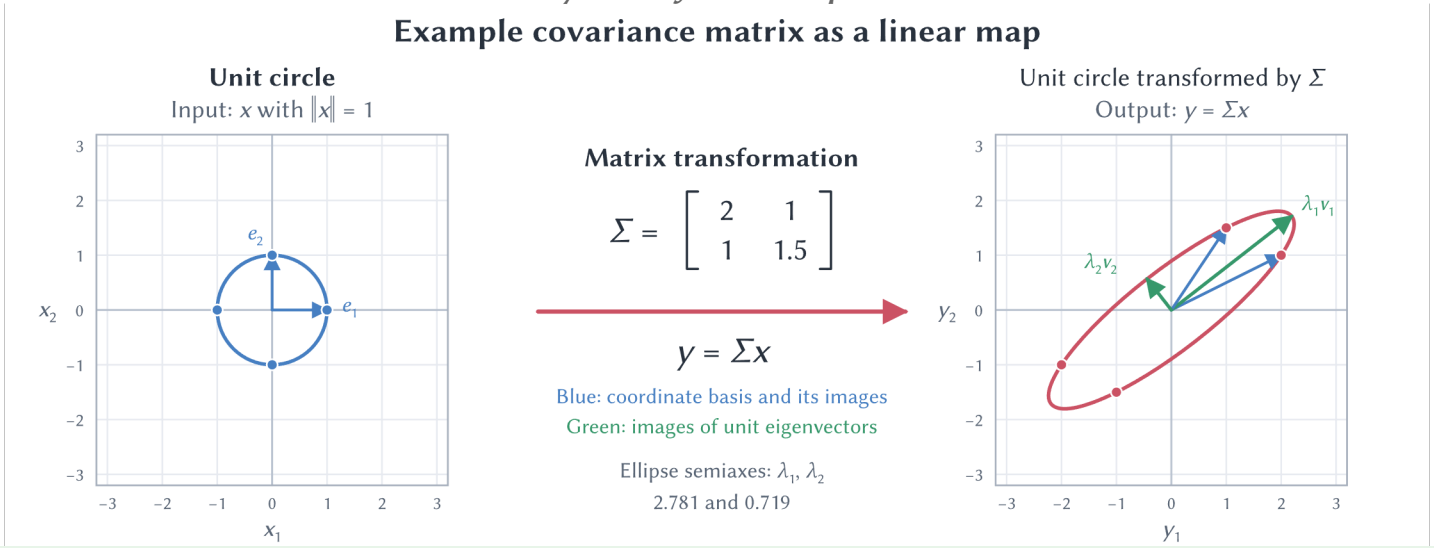
> *"Which directions in feature space capture the most variance?"*
>
> This is a mathematical optimization problem. The answer comes from linear algebra: eigenvectors of the covariance matrix.

Our Path to the Answer:

1. **Matrix Transformations:** Understand what matrices "do" to vectors
2. **Eigenvectors & Eigenvalues:** Find special directions where matrices just stretch
3. **Connection to Covariance:** See why eigenvectors of Σ give max-variance directions
4. **The PCA Algorithm:** Put it all together into a practical procedure

Matrix Transformations

Matrices stretch and rotate vectors — they transform shapes.



Key Observation

Most vectors get both stretched AND rotated by a matrix. But some special vectors only get stretched (or compressed) — their direction doesn't change. These are the eigenvectors.

Special Directions: Eigenvectors

Eigenvectors are directions where a matrix transformation only stretches, without rotating.

EQ7: Eigenvalue Equation

$$\Sigma \mathbf{v} = \lambda \mathbf{v}$$

When the covariance matrix Σ multiplies v, the result is λv.
No rotation — just stretching (or compressing if $|\lambda| < 1$).

Where:

- Σ = square matrix (here, the covariance matrix)
- $\mathbf{v}$ = eigenvector (non-zero vector)
- λ = eigenvalue (scalar)

Interpretation: When Σ acts on $\mathbf{v}$, the result is $\mathbf{v}$ scaled by λ.

Intuition: Eigenvectors are the "natural axes" of the transformation — the directions along which the matrix behaves simply.

Eigenvalues: The Stretch Factor

The eigenvalue λ tells us how much the matrix stretches vectors in the eigenvector direction.

Interpreting Eigenvalues:

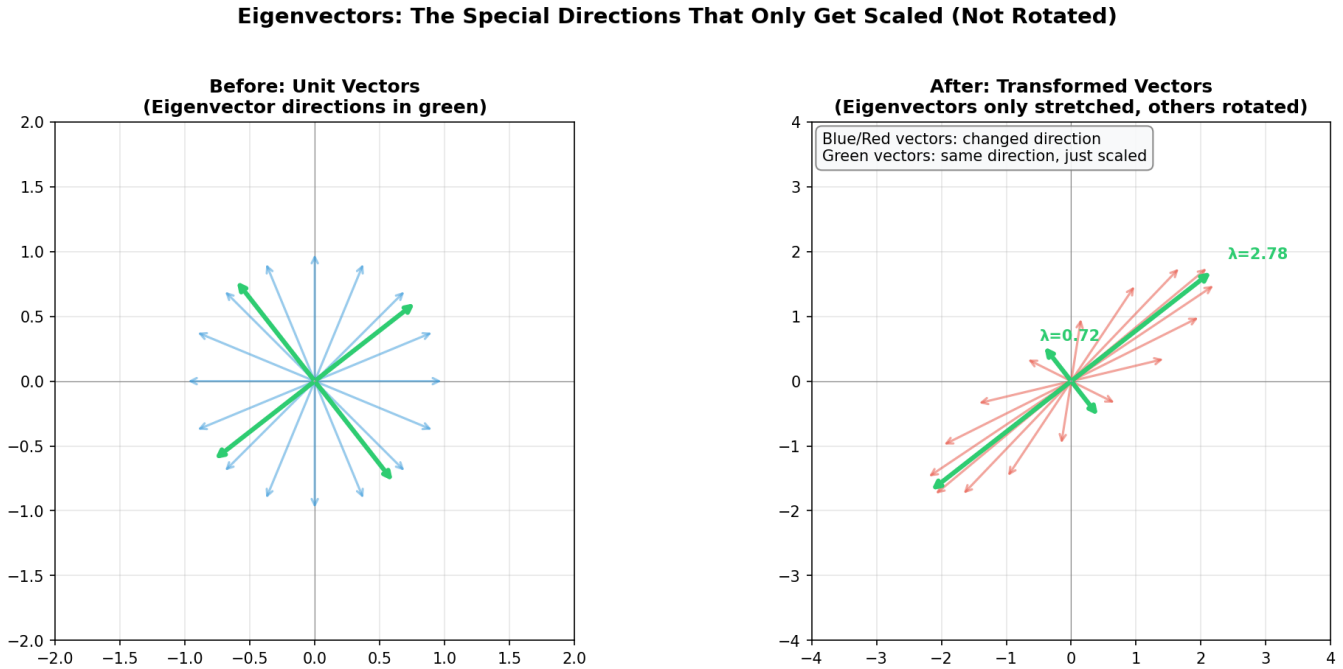
$\lambda > 1$	Stretching	Vectors in this direction get longer
$\lambda = 1$	No change	Vectors in this direction stay same length
$0 < \lambda < 1$	Compression	Vectors in this direction get shorter
$\lambda = 0$	Collapse	This direction gets squashed to zero
$\lambda < 0$	Flip + scale	Direction reverses (impossible for an exact covariance matrix)

For PCA: Larger eigenvalue = more variance along that direction

When we compute eigenvectors of the covariance matrix, the eigenvalue tells us how much variance is captured in that direction. Sort by eigenvalue ⟶ get principal components in order of importance.

Visual: Eigenvectors in Action

Most vectors rotate when transformed. Eigenvectors only stretch.



Notice: The eigenvectors point along the "natural axes" of the transformation — the directions the ellipse is stretched along.

Eigenvectors of the Covariance Matrix

The covariance matrix has special properties that make its eigenvectors particularly useful.

Special Properties of Covariance Matrix Eigenvectors:

Real eigenvalues

All eigenvalues are real numbers (not complex) — because Σ is symmetric.

Non-negative eigenvalues

All eigenvalues are ≥ 0 — because Σ is positive semi-definite. (You can't have negative variance!)

Orthogonal eigenvectors

Eigenvectors are perpendicular to each other — they form a natural coordinate system.

Why This Matters for PCA:

These properties guarantee that we can always find a set of perpendicular axes (eigenvectors) with non-negative importance values (eigenvalues). This is exactly what we need for dimensionality reduction!

The Connection to Variance

Here's the key theorem that makes PCA work.

EQ8: Variance Along a Direction

$$\text{Var}_{\mathbf{v}}(X) = \mathbf{v}^T \Sigma \mathbf{v} = \lambda$$

The variance of data projected onto an eigenvector equals the corresponding eigenvalue!

This means eigenvector with largest eigenvalue = direction of maximum variance.

- $\text{Var}_{\mathbf{v}}(X)$ = variance of data projected onto direction $\mathbf{v}$
- $\mathbf{v}$ = unit eigenvector ($\|\mathbf{v}\| = 1$)
- $\mathbf{v}^T$ = transpose of $\mathbf{v}$ (row vector)
- Σ = covariance matrix
- λ = eigenvalue corresponding to $\mathbf{v}$

Key insight: The eigenvalue λ equals the variance in direction $\mathbf{v}$.

Key Insight: Eigenvalues ARE Variances

λ_1 = variance captured by first principal component

λ_2 = variance captured by second principal component

... and so on. The eigenvalues tell us exactly how much information each direction contains.

This completes the connection: eigenvectors of Σ = principal components, eigenvalues = their importance.

Eigenvectors Define Principal Directions

The eigenvectors of Σ define the principal axes; projections give PC scores.

Eigenvector Concept	PCA Interpretation
v_1 (eigenvector with largest λ)	**PC1 — first principal component**
v_2 (eigenvector with 2nd largest λ)	**PC2 — second principal component**
λ_1 (largest eigenvalue)	**Variance explained by PC1**
$\lambda_1 / \Sigma_j \lambda_j$ (proportion)	**Fraction of total variance in PC1**
$v_i \perp v_j$ (orthogonal eigenvectors)	**Projected PC scores are uncorrelated**

Summary

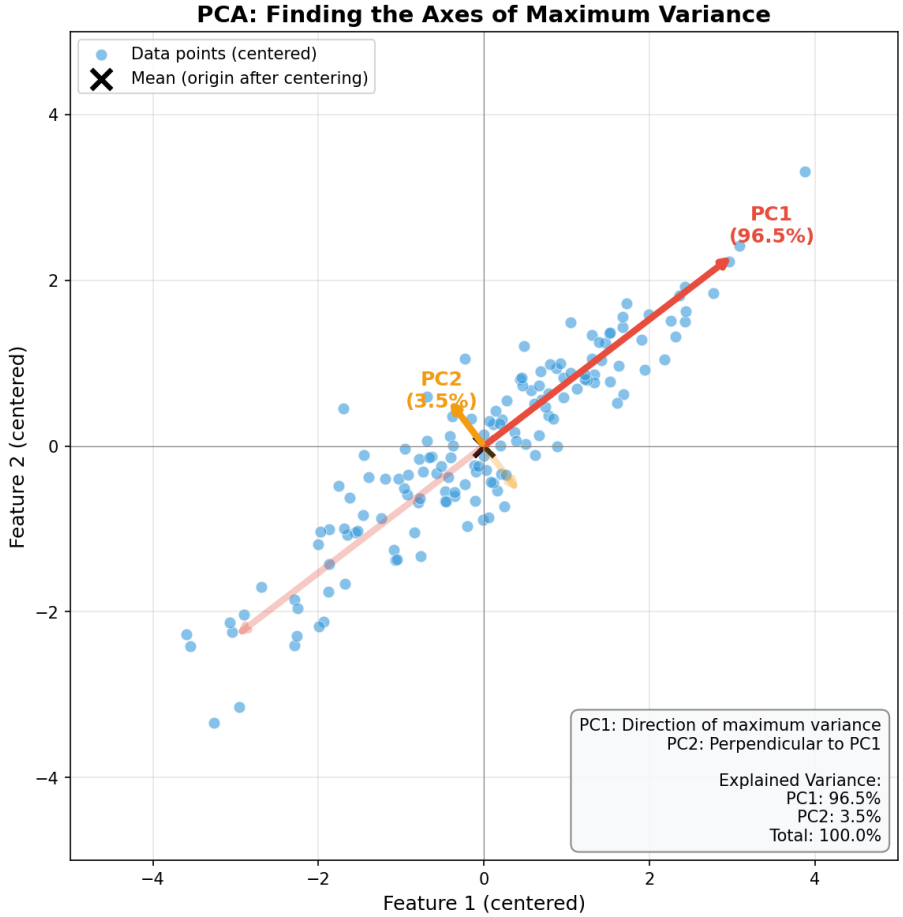
PCA reduces dimensionality by projecting data onto the top k eigenvectors of the covariance matrix. These are the directions that capture the most variance — the directions that "matter most."

Visual: PCA on 2D Data

The canonical visualization: eigenvectors as new axes aligned with data structure.

What to See:

PC1 points along the direction of maximum spread.
PC2 is perpendicular.
Together they form a new coordinate system aligned with the data's natural structure, not the original feature axes.



Eigenvectors as a New Coordinate System

The eigenvectors form new axes aligned with the data's structure.

Original Coordinates

- Axes defined by features
- Often arbitrary (which measurement comes first)
- Features may be correlated
- Not aligned with data structure

→

PC Coordinates

- Axes defined by eigenvectors of Σ
- Aligned with directions of max variance
- Components are uncorrelated
- Ordered by importance ($\lambda_1 \geq \lambda_2 \geq \ldots$)

What We've Learned:

✓ Matrices transform vectors by stretching and rotating

✓ Eigenvectors are special directions where matrices only stretch (no rotation)

✓ Eigenvalues tell us how much stretching occurs

✓ For the covariance matrix: eigenvectors = principal components, eigenvalues = variances

✓ PCA gives us a new coordinate system aligned with the data's natural structure

Next: The PCA algorithm — putting this into practice.

The PCA Algorithm

From theory to practice: the four-step recipe

The Four Steps of PCA

A simple algorithm with deep mathematical foundations.

Step 1: Center the Data

Subtract the mean of each feature so data is centered at origin

Step 2: Compute Covariance Matrix

Calculate $\Sigma = X^T X / (n - 1)$, where X is the centered $n \times p$ data matrix.

Step 3: Find Eigenvectors

Compute eigenvectors and eigenvalues of Σ, sort by eigenvalue

Step 4: Project the Data

Multiply centered data by top k eigenvectors to get k-dimensional representation

That's the entire algorithm. Let's walk through each step in detail.

Step 1: Center the Data

Subtract the mean of each feature so the data is centered at the origin.

EQ9: Centering

$$X_{\text{centered}} = X - \boldsymbol{\mu}, \quad \text{where } \mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

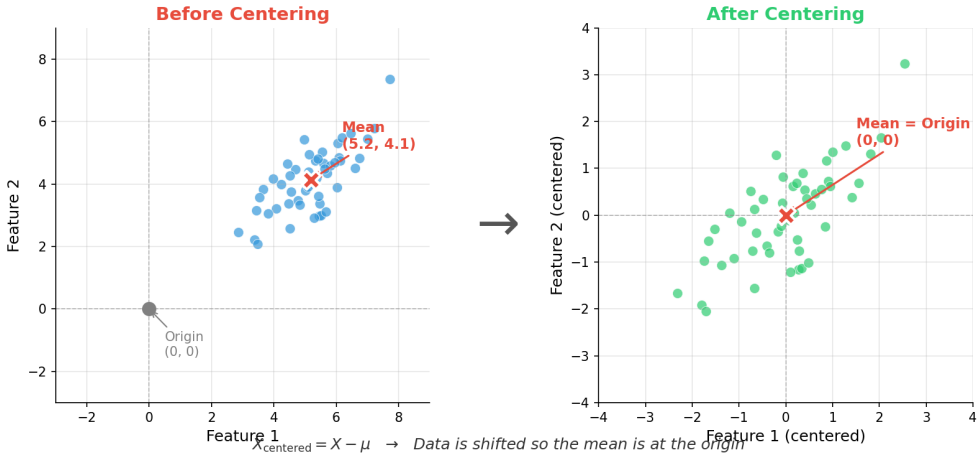
Why Center?

Definitional necessity: Covariance is defined relative to the mean, so computing it correctly requires centering.

Separates location from shape: Without centering, Uncentered PCA mixes the mean's location with variation around it.

Translation invariance: PCA directions should depend only on the data's internal structure, and agnostic to coordinate choices.

Removes the mean contribution: The uncentered product includes the mean's outer product, which can change the principal directions.



Step 2: Compute Covariance Matrix

Calculate how features vary together — this captures the data's "shape".

EQ10: Covariance Matrix (for centered data)

$$\Sigma = \frac{1}{n-1} X_{\text{centered}}^T X_{\text{centered}}$$

For centered X ($n \times p$), Σ is $p \times p$. Each entry Σ_{ij} = Cov(feature i, feature j).

Recall: The covariance matrix Σ encodes the "shape" of your data cloud.
Diagonal = variances, off-diagonal = covariances between features.

In Python:

```
import numpy as np

X_centered = X - X.mean(axis=0)  # Center each feature
# .T: np.cov expects features as rows by default
cov_matrix = np.cov(X_centered.T)
```

Step 3: Find Eigenvectors

Compute the eigenvectors and eigenvalues, then sort by decreasing eigenvalue.

What We Compute: $\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \text{for } i = 1, \ldots, p$

Eigenvectors (v_1, v_2, ..., v_p)

The directions of maximum variance.
These become our principal components.
Each is a unit vector (length = 1).

Eigenvalues (λ_1, λ_2, ..., λ_p)

Variance captured by each principal direction.
Larger eigenvalue = more variance.
Sum of all λ_i = total variance.

Critical: Sort eigenvectors by their eigenvalues (largest first). This ordering defines PC1, PC2, etc. The eigenvector with the largest eigenvalue is PC1.

In Python:

```
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
sorted_idx = np.argsort(eigenvalues)[::-1]   # Sort descending
eigenvalues = eigenvalues[sorted_idx]
eigenvectors = eigenvectors[:, sorted_idx]   # Columns are eigenvectors
```

Step 4: Project the Data

Transform data into the new coordinate system defined by the principal components.

EQ11: Projection

$$Z = X_{\text{centered}} \cdot W_k, \quad \text{where } W_k = [v_1 \mid v_2 \mid \cdots \mid v_k]$$

X_{centered} is $n \times p$, W_k is $p \times k$, so Z is $n \times k$. We have reduced from p to k dimensions!

What This Means:

- Each row of Z is the original data point expressed in PC coordinates
- `Z[:,0]` contains PC1 scores (projections onto first eigenvector)
- `Z[:,1]` contains PC2 scores, and so on
- If we keep $k < p$ components, we've reduced dimensionality from p to k

In Python:

```
k = 2                                    # Keep top 2 components
W_k = eigenvectors[:, :k]                # First k eigenvectors
Z = X_centered @ W_k                     # Project! Now n×k
```

Complete PCA Implementation

All four steps in one code block — it's surprisingly simple!

```
import numpy as np

def pca(X, k):
    """Perform PCA, reducing X from n×p to n×k."""

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

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

    # Step 3: Find eigenvectors, sort by eigenvalue
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
    sorted_idx = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_idx]
    eigenvectors = eigenvectors[:, sorted_idx]

    # Step 4: Project onto top k eigenvectors
    W_k = eigenvectors[:, :k]
    Z = X_centered @ W_k

    return Z, eigenvalues, eigenvectors
```

In practice, use `sklearn.decomposition.PCA` — but understanding this code is essential!

PCA on Iris: Step by Step

PCA on Iris after standardizing all four features.

Example: Iris Dataset (150 samples × 4 features)

Step 1: Standardize → center and scale each feature

X_{scaled}: 150×4 matrix with feature means ≈ 0 and population standard deviations = 1

Step 2: Covariance matrix → 4×4 symmetric matrix

Σ captures how each pair of features varies together

Step 3: Eigenvectors → 4 eigenvectors, 4 eigenvalues

$\lambda_1 = 2.938$ (72.96%), $\lambda_2 = 0.920$ (22.85%), $\lambda_3 = 0.148$ (3.67%), $\lambda_4 = 0.021$ (0.52%)
PC1 captures 72.96% of variance. PC1+PC2 capture 95.81%.

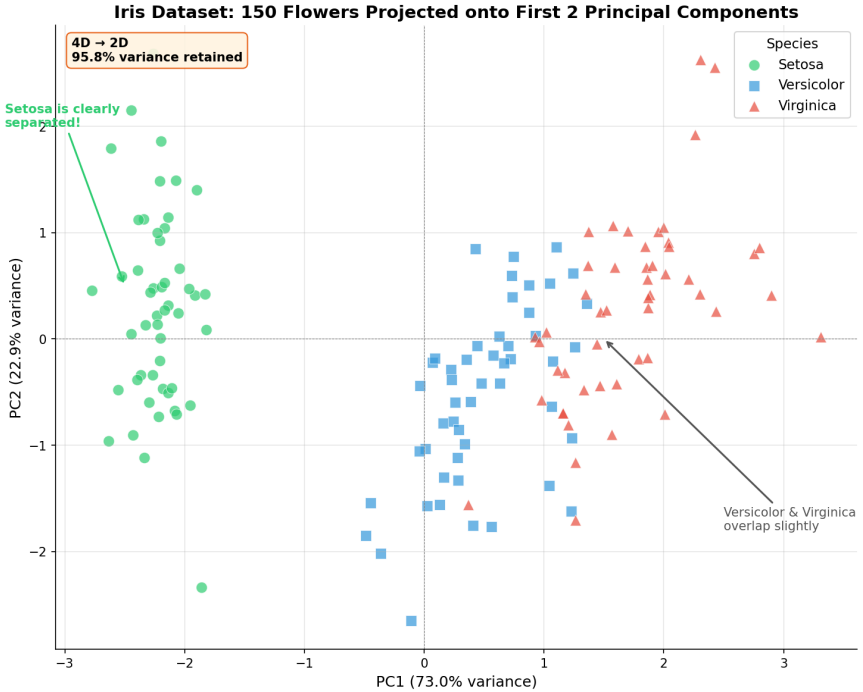
Step 4: Project → keep top 2 components

$Z = X_{\text{scaled}} \cdot W_2 \rightarrow$ 150×2 matrix. We went from 4D to 2D while retaining 95.81% of standardized variance.

Result: We can now visualize the 4D Iris data in 2D!

Iris PCA Visualization

150 flowers, four standardized features, projected onto two principal axes.



Key Insight: The species cluster structure was "hidden" in 4D. PCA reveals it by finding the 2D projection that retains the most variance among 2D linear projections.

CTA Demo 3: Debugging Understanding

Using the CTA to clarify confusions and fix misconceptions

🤖 What We'll Explore

PCA involves several concepts that are easy to confuse. We'll deliberately surface common misconceptions and use the CTA to debug our understanding — treating confusion as a feature to help learning, not a bug.

Common Confusions We'll Address:

- "Are principal components the same as original features?"
- "Does PCA pick features to drop, or does it create new ones?"
- "Why do we center but not always standardize?"
- "Is explained variance the same as accuracy?"

Collaboration Pattern: Debugging Understanding

VERIFY: Examine, Refine, Yield

🤖 CTA Demo: Debugging PCA Understanding

LIVE DEMO

The debugging understanding pattern:

State confusion explicitly → Ask for clarification → Request analogies → Test understanding

Example: "I'm confused about whether PCA picks existing features or creates new ones. Can you clarify?"

Pattern: Debugging Understanding

Demo 3 Takeaways

Content Insight: PCA Creates, Doesn't Select

- PCA creates NEW features (linear combinations of originals), not subsets
- Principal components are weighted sums of all original features
- "Feature selection" is a different technique that picks a subset of original features
- Covariance PCA uses centered features; choose scaling based on units and task

🤖 Collaboration Pattern: Debugging Understanding

- State your confusion explicitly — "I'm confused about X vs Y"
- Ask for clarification with specifics — "Does this mean...?"
- Request analogies — "Can you give me an intuitive example?"
- Test your understanding — "So if I understand correctly..."

✓ VERIFY Framework Connection

Examine: We surfaced hidden assumptions | Refine: We iterated until clarity | Yield: We emerged with corrected mental models

Meta-Lesson: Confusion is valuable data. Stating it explicitly to the CTA lets you fix misconceptions early, before they compound into larger errors.

Selecting Components

How many principal components should we keep?

The Trade-off: Fewer components = simpler model, but more information lost.
More components = more information retained, but less compression.

Explained Variance Ratio

Measuring how much information each component captures.

EQ13: Explained Variance Ratio

$$\text{EVR}_i = \frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}$$

The proportion of total variance captured by component *i*.
Each eigenvalue divided by the sum of all eigenvalues.

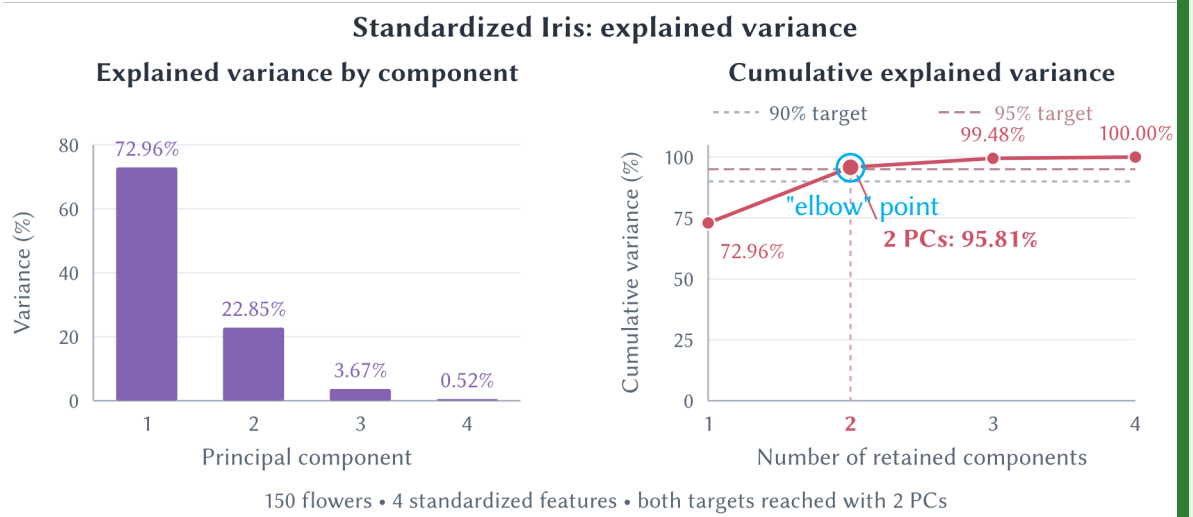
How to Interpret:

$\text{EVR}_1 = 0.73$	PC1 captures 73% of the total variance
$\text{EVR}_2 = 0.23$	PC2 captures 23% of the total variance
$\text{EVR}_1 + \text{EVR}_2 = 0.96$	Together, first 2 PCs capture 96% of variance

Key: All EVRs sum to 1.0 (100%). Choose *k* components to capture "enough" variance.

The Scree Plot

Visualizing explained variance to find the "elbow" point.



Reading the Scree Plot

- Named after "scree" — rocky debris at the base of a cliff
- Look for the "elbow" where the curve flattens (the slope changes)
- Components before the elbow capture most variance
- Components after the elbow add little new information

Elbow Method: Use the elbow as a heuristic. For standardized Iris, 2 components retain 95.81% of variance.

Cumulative Explained Variance

Standardized Iris: adding up variance to see how many components are "enough".

Component	Eigenvalue (λ)	EVR (%)	Cumulative (%)
PC1	2.938	72.96%	**72.96%**
PC2	0.920	22.85%	**95.81%**
PC3	0.148	3.67%	**99.48%**
PC4	0.021	0.52%	**100.0%**

Interpretation for Iris:

- PC1 alone captures 72.96% of standardized feature variance
- PC1 + PC2 capture 95.81% of standardized variance.
- PC3 and PC4 add 4.19%. Check whether this matters for your task.

The 90% Rule of Thumb

A common heuristic for choosing the number of components.

The Rule

Keep enough components to explain at least 90% of the total variance.
Or 95% if you want to be conservative. Or 80% if you prioritize compression.

In Python:

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

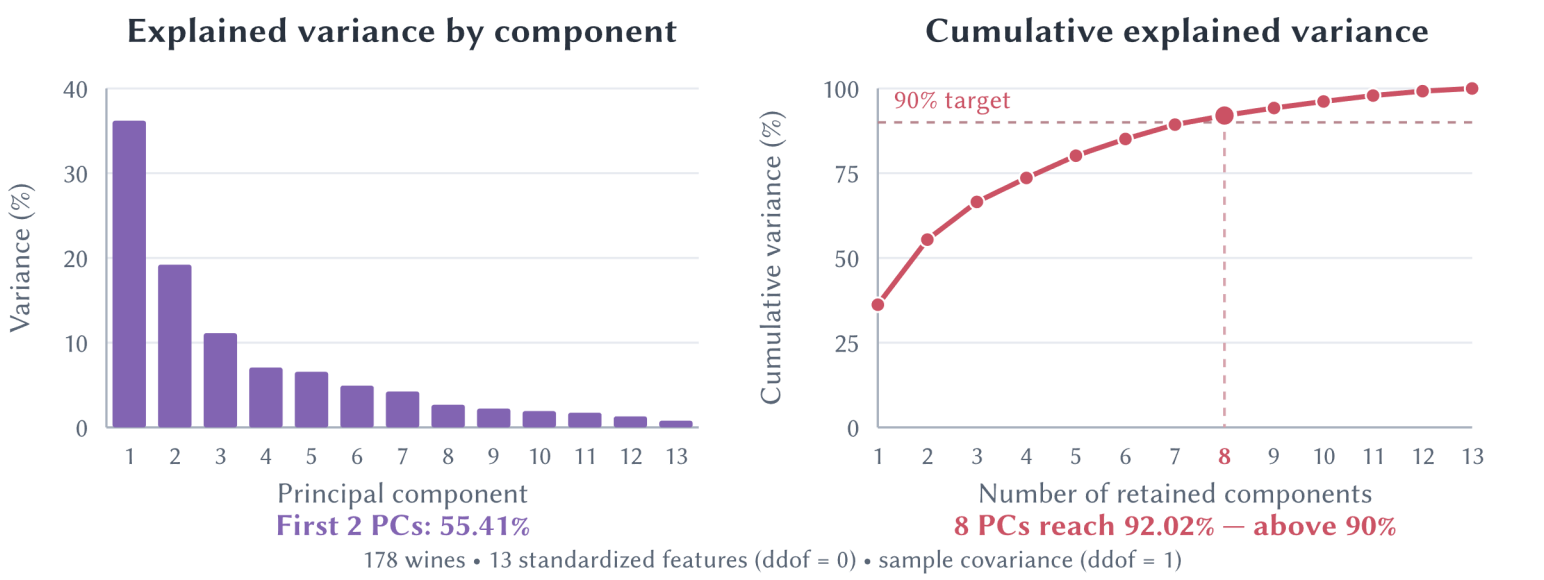
# Method 1: Let sklearn pick automatically
X_scaled = (X - X.mean(axis=0)) / X.std(axis=0, ddof=0)
pca = PCA(n_components=0.90, svd_solver="full")  # Keep 90% variance
X_reduced = pca.fit_transform(X_scaled)
print(f"Components needed: {pca.n_components_}")

# Method 2: Compute manually
pca_full = PCA(svd_solver="full").fit(X_scaled)
cumsum = np.cumsum(pca_full.explained_variance_ratio_)
k = np.searchsorted(cumsum, 0.90, side="right") + 1  # First index above 90%
```

Caution: 90% is arbitrary! The right threshold depends on your application. For visualization, 80% might suffice. For prediction, select the component count using task validation.

Applying to Real Data: Wine Dataset

Standardized Wine: 178 observations and 13 features. How many components?



Wine Dataset Results:

- Original: 13 features ⟶ With 90% threshold: 8 components needed
- First 2 PCs retain 55.41% (standardized Iris: 95.81%) — variance is spread across more PCs
- Lesson: The "right" number of components is data-dependent!

But Is Variance Always Right?

A critical question: variance ≠ importance for your task.

⚠ The Variance Limitation

PCA maximizes retained variance; usefulness depends on the task.

But retained variance may not serve your task. Consider:

- What if the signal you care about has low variance?
- What if high-variance directions are just noise?
- What if your downstream task needs those "unimportant" components?

Example: Classification Task

Imagine: PC1 captures 90% of variance but doesn't separate classes at all.
PC3 captures only 2% of variance but perfectly separates two classes.

If your goal is classification, PC3 is more valuable than PC1!

Key Insight: Always validate PCA on your actual task (classification accuracy, reconstruction error, etc.), not just explained variance. Explained variance is a proxy, not a guarantee.

CTA Demo 4: Challenging for Completeness

Pushing the CTA to explore edge cases and limitations

🤖 What We'll Explore

We've learned that variance-based selection has limitations. Now we'll challenge the CTA to give us a more complete picture: When does PCA fail? What are the alternatives? This pattern — challenging for completeness — helps avoid oversimplified mental models.

Questions We'll Ask:

- "When would keeping 99% variance still be a bad idea?"
- "What if my data has nonlinear structure?"
- "Are there alternatives to PCA for dimensionality reduction?"

Collaboration Pattern: Challenging for Completeness

VERIFY: Validate, Investigate, Yield

Goal: Build a nuanced understanding that acknowledges when standard advice doesn't apply.

🤖 CTA Demo: When Does Variance Fail?

LIVE DEMO

Watch the challenging for completeness pattern:

Accept standard answer → Push for exceptions → Ask for alternatives → Synthesize limitations

"I know PCA maximizes variance, but when would that be exactly the wrong thing to do? Give me concrete examples."

Pattern: Challenging for Completeness

Demo 4 Takeaways

Content Insight: PCA's Limitations

- PCA uses a linear subspace; curved structure may need more components
- PCA maximizes variance; task-relevant signal can have low variance
- PCA is unsupervised — doesn't know your labels (consider LDA for classification)
- PCA is sensitive to scaling — choose feature scaling for the data and task

🤖 Collaboration Pattern: Challenging for Completeness

- Don't accept the first answer as the complete picture
- Ask "when does this fail?" and "what are the alternatives?"
- Push for edge cases and exceptions — this builds robust understanding
- The goal isn't to find problems, but to know the boundaries of your tools

✓ VERIFY Framework Connection

Validate: We questioned the "always use 90% variance" rule
Investigate: We explored when PCA assumptions break down
Yield: We emerged with a nuanced view of when PCA is/isn't appropriate

Meta-Lesson: Every technique has assumptions.
Challenging the CTA to reveal them builds the judgment to know when tools apply — and when they don't.

Key Takeaways: Concepts

Five core ideas you can now explain and apply.

1. Data as Matrices

Numeric tabular data can be represented as an $n \times p$ matrix — rows are observations, columns are features. This perspective unlocks all of linear algebra.

2. Geometric Intuition

Rows are points in p-dimensional space. Distances and similarities measure relationships between observations.

3. Covariance & Correlation

The covariance matrix captures feature relationships. It encodes the "shape" of your data cloud.

4. Eigenvectors as Axes

Eigenvectors of the covariance matrix point along directions of maximum variance — the "natural axes" of your data.

5. PCA for Reduction

Project onto top eigenvectors to reduce dimensions while preserving maximum variance: center, covariance, eigenvectors, project.

Key Takeaways: CTA Collaboration

Four patterns for effective AI-assisted learning you practiced today.

Pattern	Demo	When to Use
Exploring Concepts	Demo 1	Building intuition for new ideas; asking "what does this mean?"
Verifying Intuitions	Demo 2	Checking if your understanding is correct; testing predictions
Debugging Understanding	Demo 3	Fixing misconceptions; clarifying confusions
Challenging for Completeness	Demo 4	Finding edge cases and limitations; building robust knowledge

The Meta-Skill

These patterns aren't just for this course — they're transferable skills for learning anything with AI assistance.
The key is knowing which pattern fits your current learning need, and always verifying the AI's outputs.

The VERIFY Habit

A framework for responsible AI-assisted learning.

✓ VERIFY

V	**Validate:**	Check that the AI's response addresses your actual question
E	**Examine:**	Scrutinize claims and reasoning — does it make sense?
R	**Refine:**	Ask follow-up questions to fill gaps or clarify ambiguities
I	**Investigate:**	Test with examples, edge cases, or alternative approaches
F	**Find Sources:**	Cross-reference with textbooks, documentation, or trusted resources
Y	**Yield:**	Synthesize into your own understanding — can you explain it yourself?

Make VERIFY a habit. Every interaction with the CTA is a chance to practice critical engagement.

Technically Intriguing Questions

Explore these with the CTA to deepen your understanding.

💡 What happens if you don't center the data before PCA?

Ask the CTA for an example where centering matters — have it show you the difference!

💡 Is there a version of PCA that's robust to outliers?

Explore Robust PCA and RANSAC-like approaches. When might you need these?

💡 How does PCA relate to SVD (Singular Value Decomposition)?

There's a deep mathematical connection. `sklearn` *can use SVD or covariance eigendecomposition.*

💡 Can you do PCA on categorical features?

Explore Multiple Correspondence Analysis. What goes wrong with one-hot encoding?

These questions are designed to reward curiosity. Pick one and spend 15 minutes exploring with the CTA!

Questions?

CONTACT INFORMATION



yangliu.mit.edu

Instructor: Prof. Yang Liu

Email: yliu12@hku.hk

Office Hours: 5:00 - 5:50 PM, Thursdays

Location: CB-714, Chow Yei Ching Building

Let me know if you want remote!

🤖 **CTA is your 24/7 interactive and responsive learning partner!**