COM SCI M148, Lecture 8:
Unsupervised Learning (Part 1)

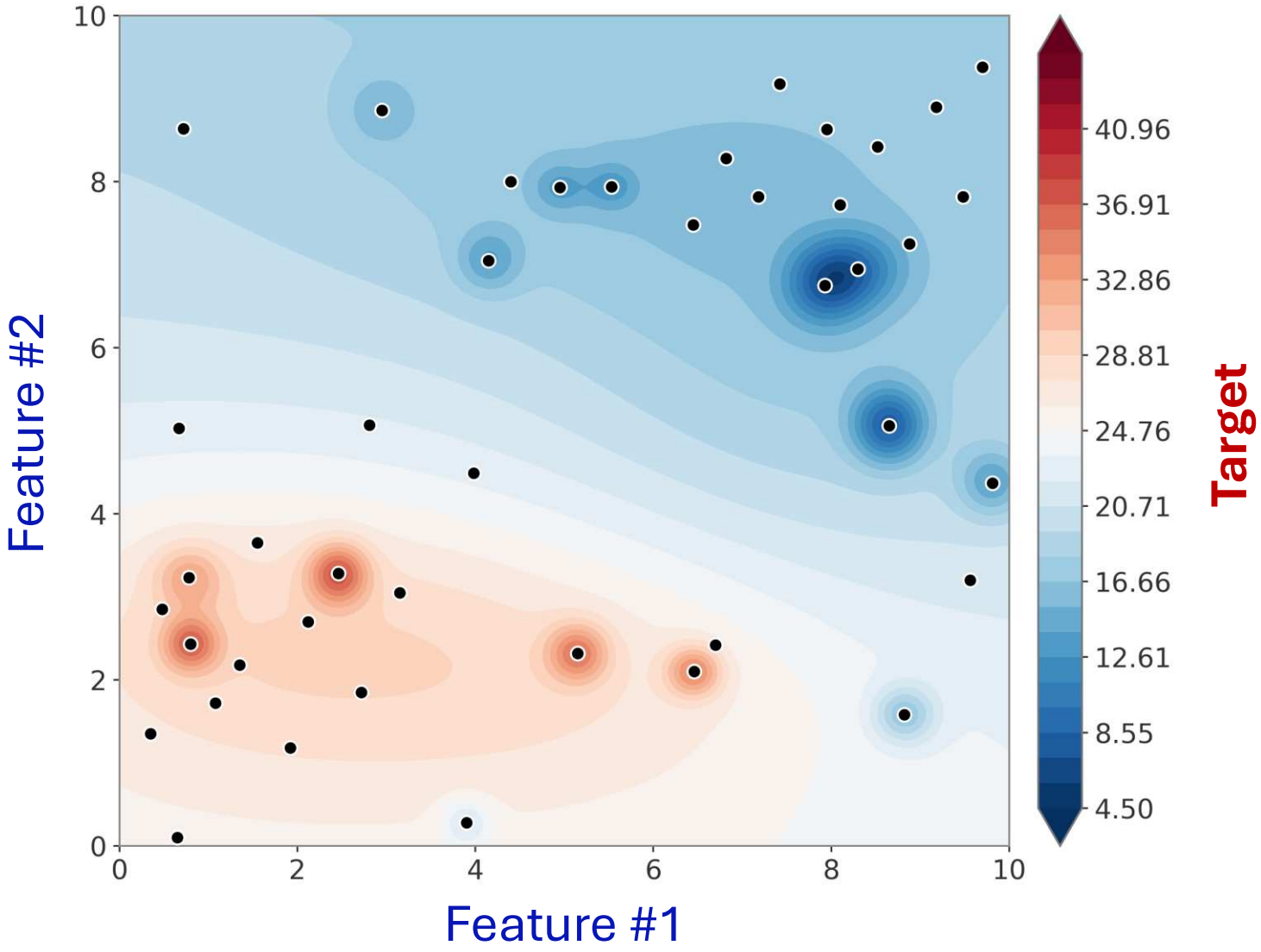
***Bonus content!** 

Prof. Nathan Szymanski
nszymanski@seas.ucla.edu
Department of Materials Science

UCLA

Samueli School of Engineering

Recap: Models for regression



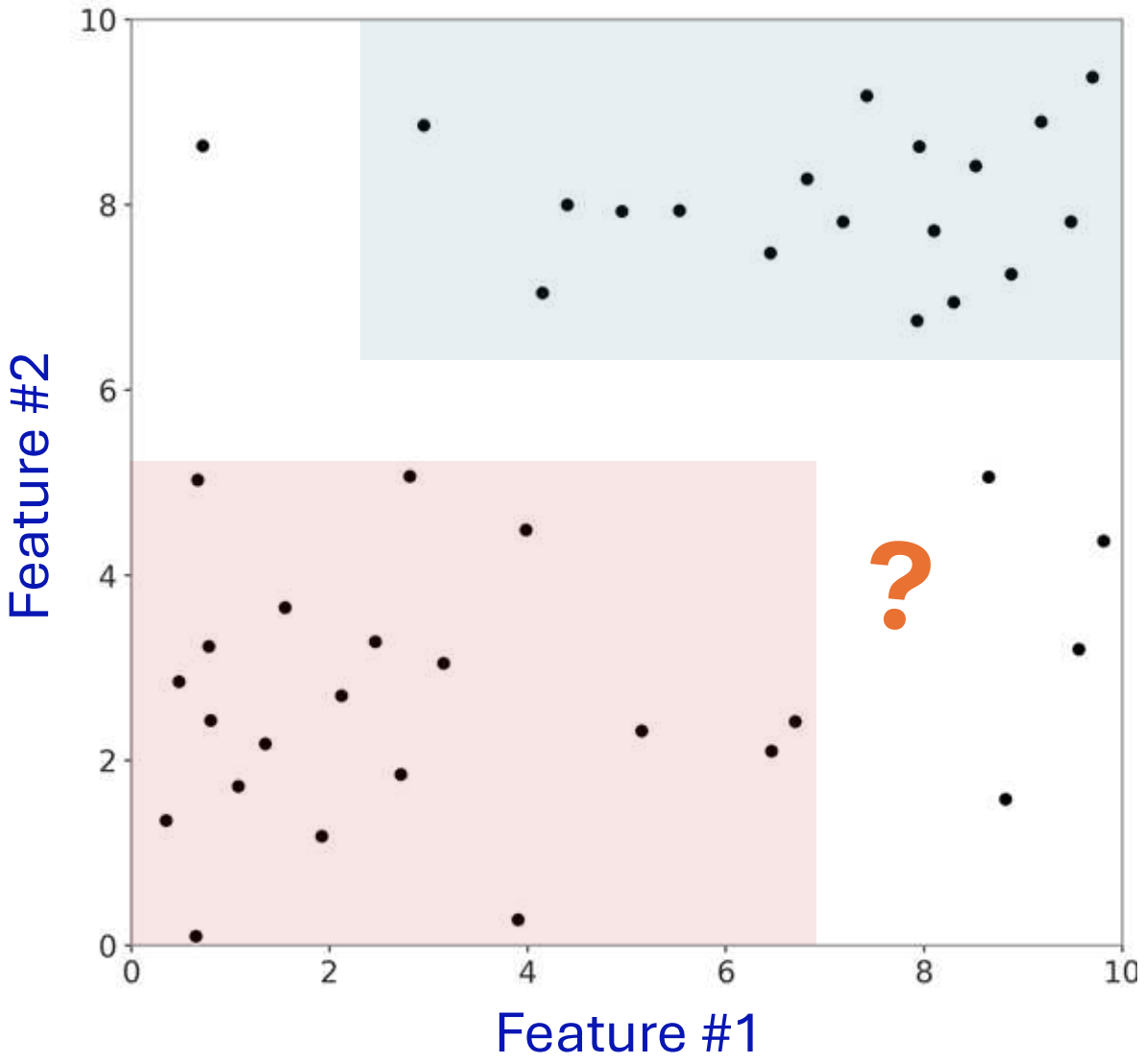
$$y = f(x_1, x_2, \dots)$$

We've covered a few methods that learn to **map features (x_i) onto a target (y)**

- Linear & polynomial regression
- Ridge/Lasso regularization
- K-Nearest Neighbors

These assume we have a set of samples with **known output**, which we can train on
→ **supervised learning**

Recap: Models for regression



$$y = f(x_1, x_2, \dots)$$

But what if we **do *not* have the target** values for any of these samples? (*i.e.*, **unlabeled data**)

Can we still learn anything about the **structure of the data**, solely from their inputs, and how they are distributed?
→ ***unsupervised* learning**

EX: perhaps we suspect these two **groups of data** will have different outputs

In the real world, most data is unlabeled!

Example:
Most images don't have labels



But in principle, we can still **organize them by similarity** **without having any knowledge of the true labels (cat/dog)**





In the real world, most data is unlabeled!

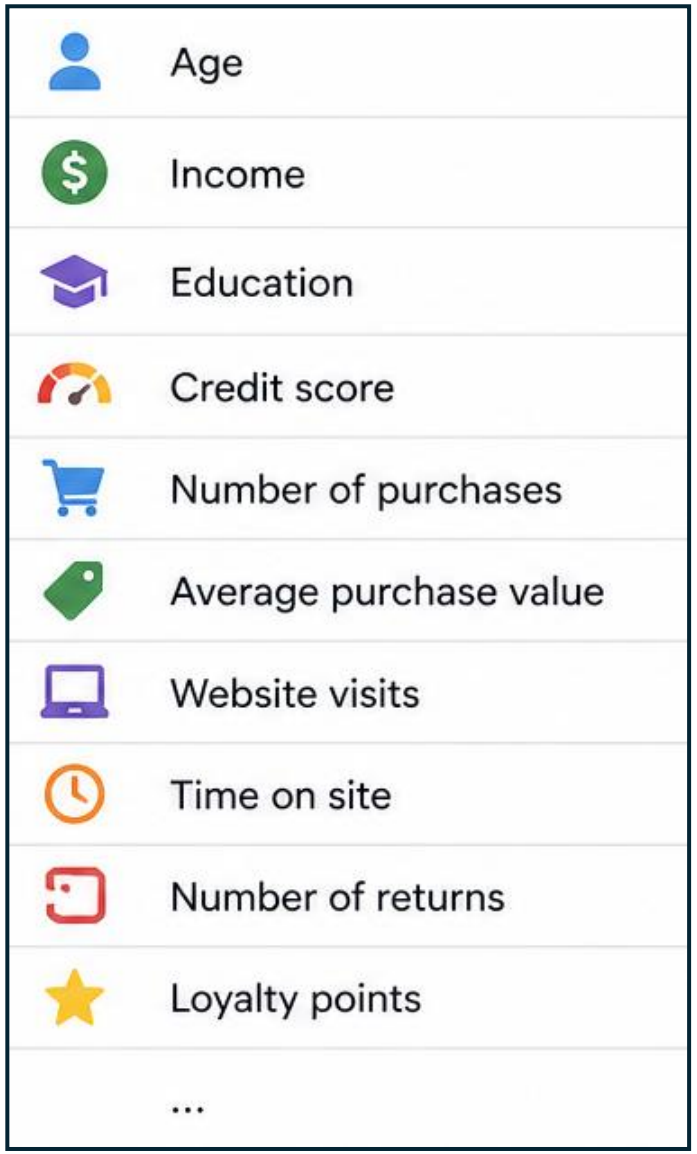


Another example: Audio

Even without labels identifying who is speaking, we could probably **distinguish two speakers by consistent differences in their voices.**



We may also want to visualize trends

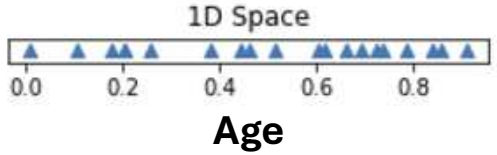


Many features!

Suppose we collect data on many features – *e.g.*, consumer data – **how do we visualize trends beyond three dimensions?**

We may also want to visualize trends

- Age
- Income
- Education
- Credit score
- Number of purchases
- Average purchase value
- Website visits
- Time on site
- Number of returns
- Loyalty points
- ...



Suppose we collect data on many features – *e.g.*, consumer data – **how do we visualize trends beyond three dimensions?**

We may also want to visualize trends

- Age
- Income
- Education
- Credit score
- Number of purchases
- Average purchase value
- Website visits
- Time on site
- Number of returns
- Loyalty points
- ...

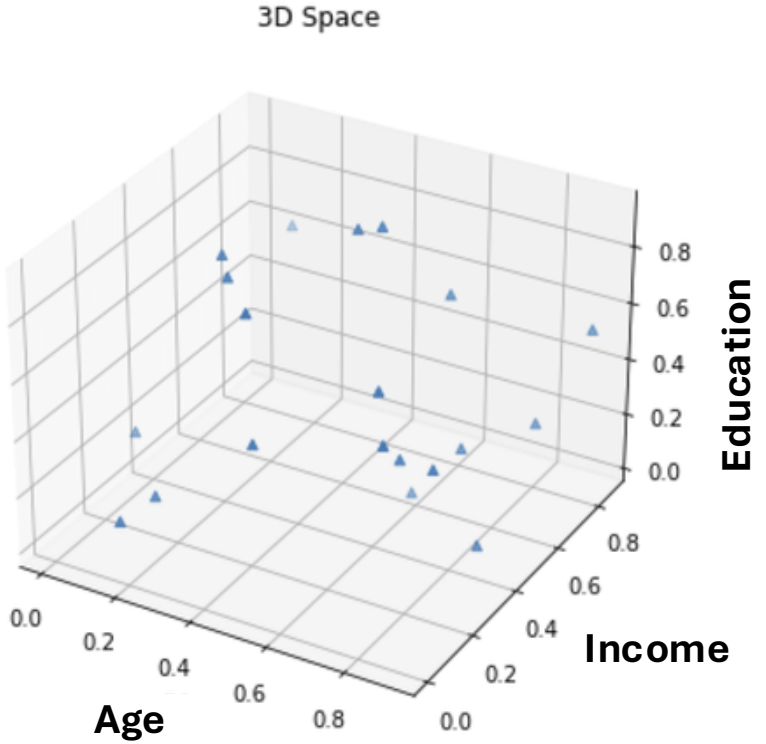


Suppose we collect data on many features – *e.g.*, consumer data – **how do we visualize trends beyond three dimensions?**

We may also want to visualize trends

- Age
- Income
- Education
- Credit score
- Number of purchases
- Average purchase value
- Website visits
- Time on site
- Number of returns
- Loyalty points
- ...





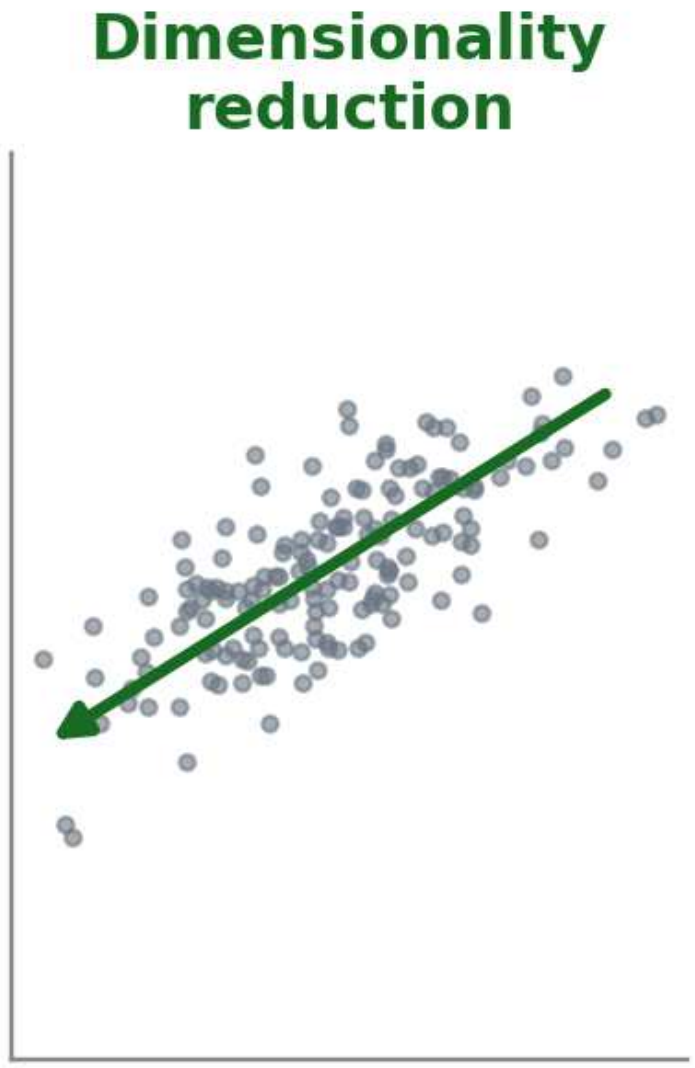
Visualization is tough beyond this…

Suppose we collect data on many features – *e.g.*, consumer data – **how do we visualize trends beyond three dimensions?**

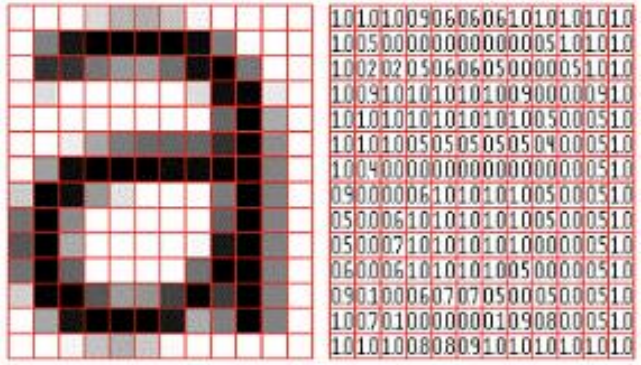
Three unsupervised learning problems



Today's focus: Dimensionality reduction



In this example, each data point has two features (x, y). Can we **compress this data** into just one feature? In a way that **loses as little information as possible?**

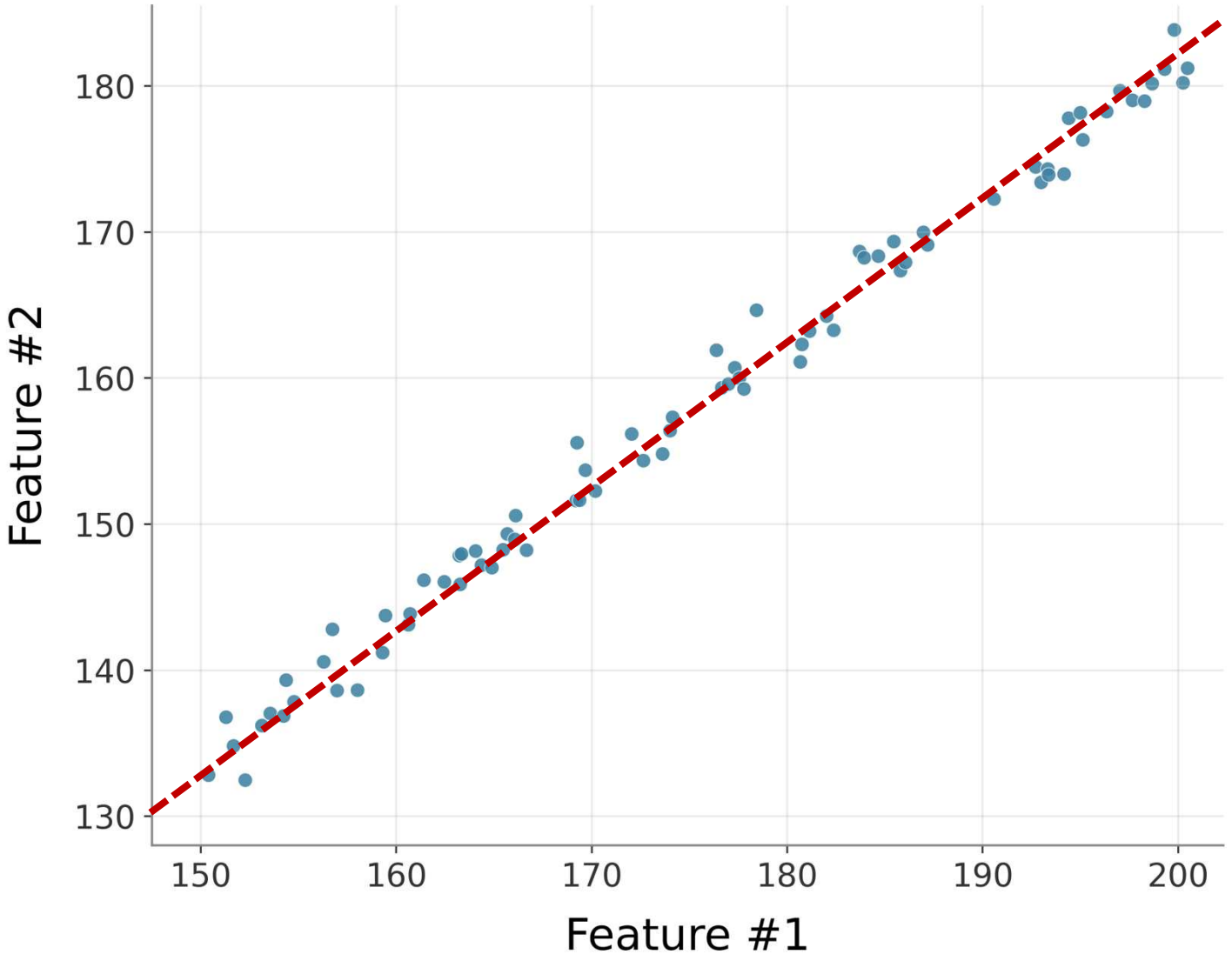




Mega Pixels

This could be very useful when dealing with high-dimensional data…For example:
Images have millions of features!

Features are often redundant



Do we really need both features to describe each sample?

If I gave you feature #1, you could reasonably predict feature #2 (this is exactly what we've done with linear regression)

Why not just describe each point by its position along this line?

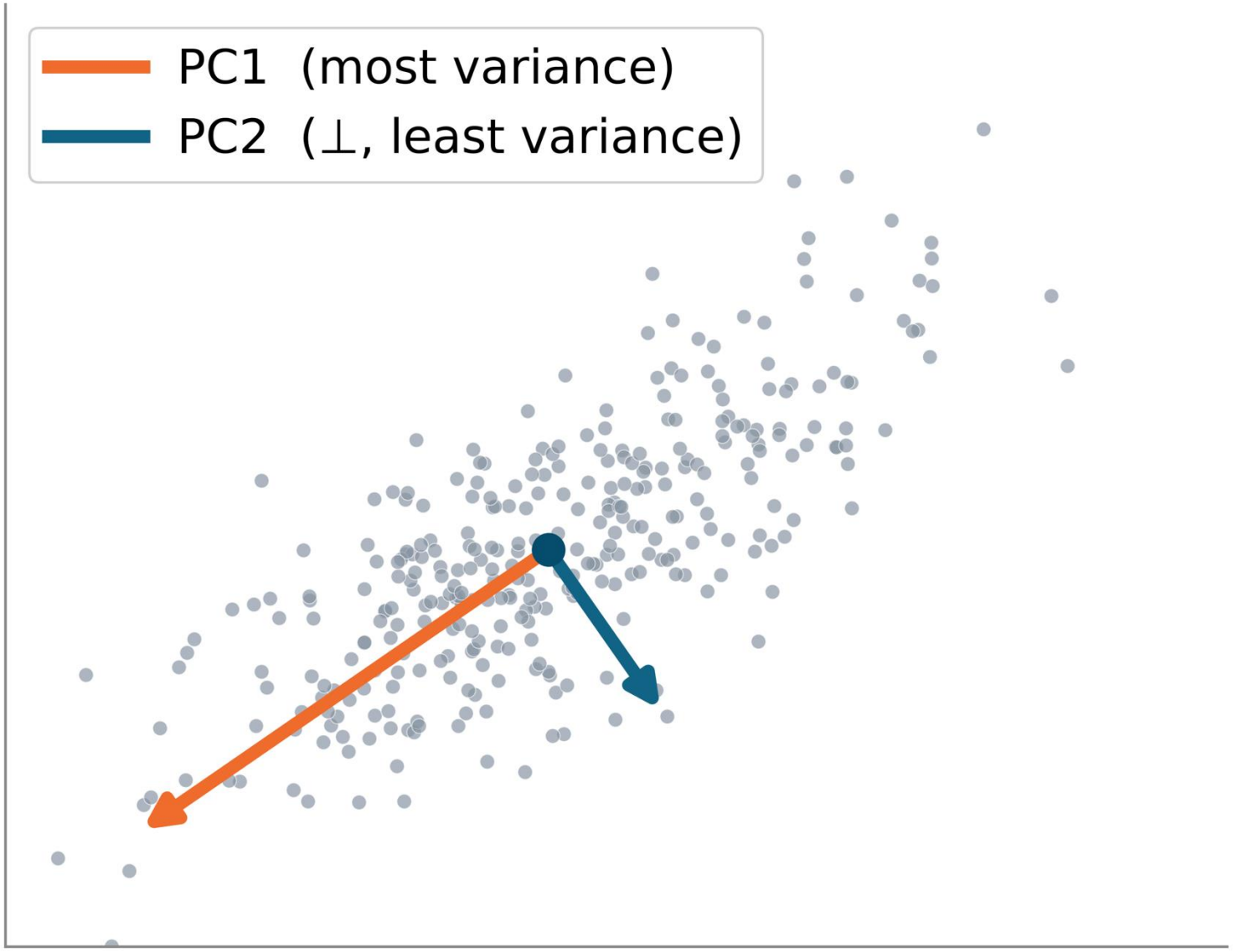


This is the idea behind PCA!

...Find the best line...

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

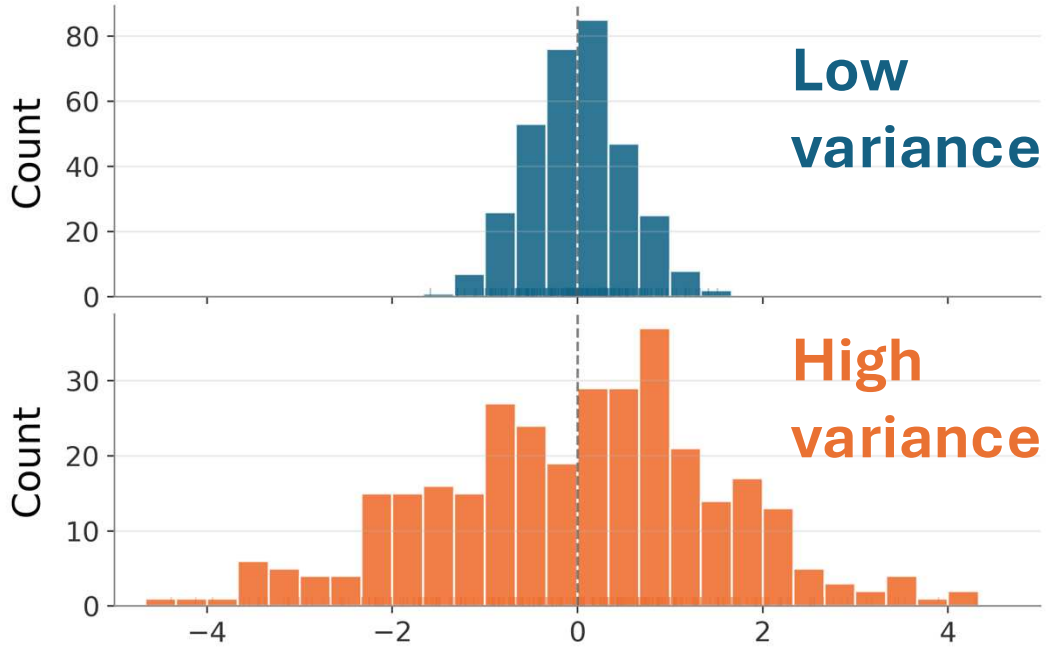


PCA finds **new axes** (called principal components) ordered so that **the first captures the most *variance* in the data**, while each subsequent component captures as much of the remaining variance as possible and is ***orthogonal*** to all previous components.

(In 2D, this is trivial)

Remember…Variance = spread in data

$$\mathbf{Var}(x) = \frac{1}{n-1}\sum_{i=0}^{n}(x_i - \bar{x})^2$$

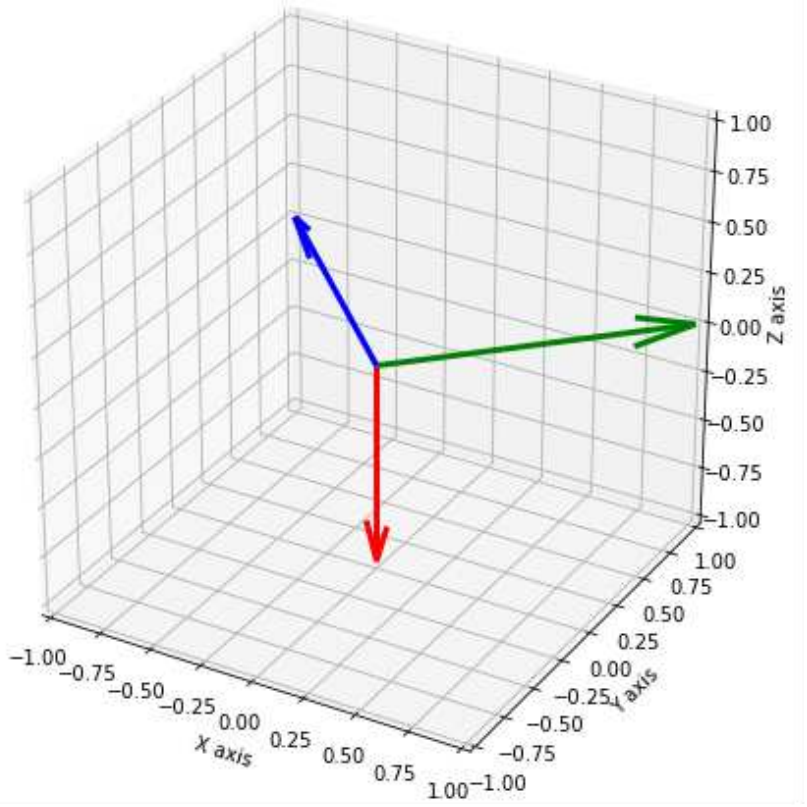


PCA finds **new axes** (called principal components) ordered so that **the first captures the most *variance* in the data**

This is just the square of the standard deviation along some direction

First component should **maximize the spread in the data** along that direction

Subsequent components are orthogonal



$$u = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

$$v = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

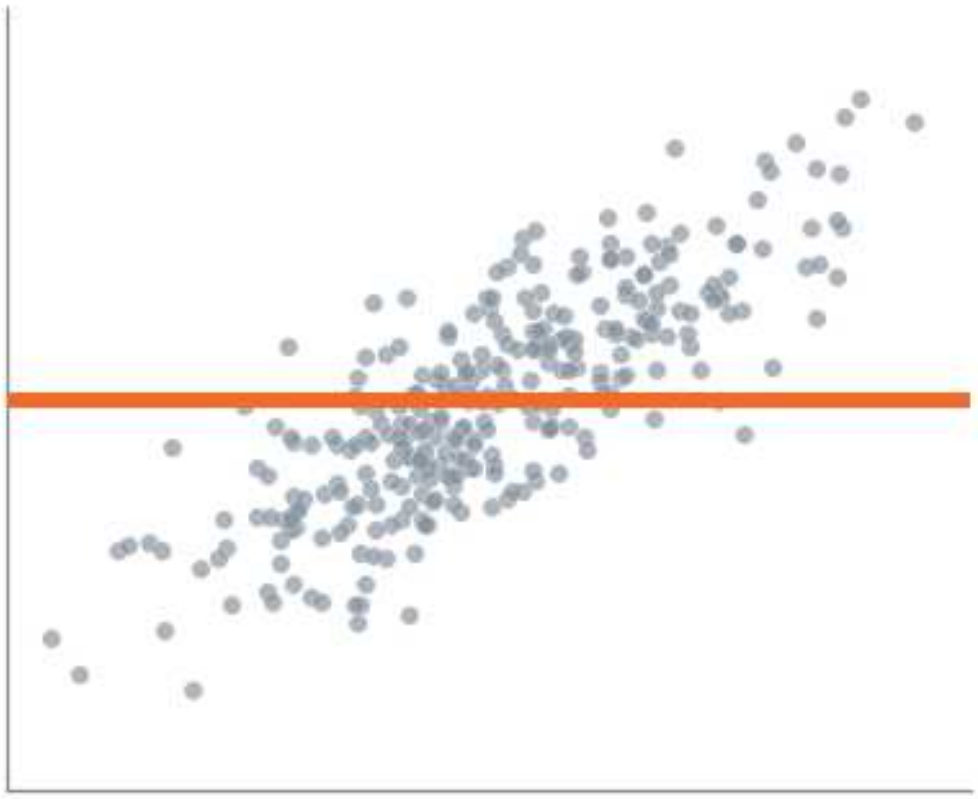
$$u^T v = 0$$

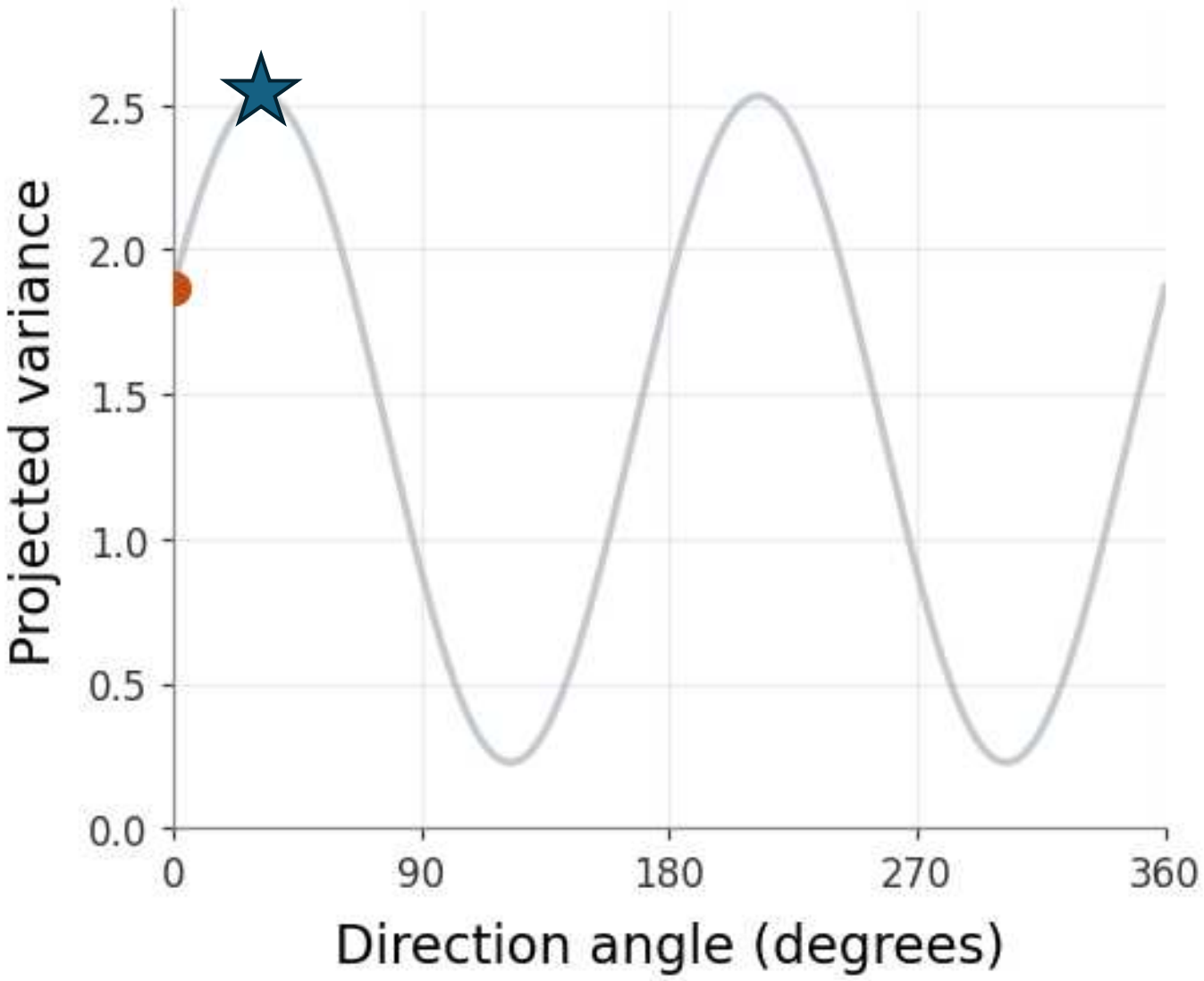
Their dot product is zero

PCA finds **new axes** (called principal components) ordered so that **the first captures the most *variance* in the data**, while each subsequent component captures as much of the remaining variance as possible and is ***orthogonal*** to all previous components.

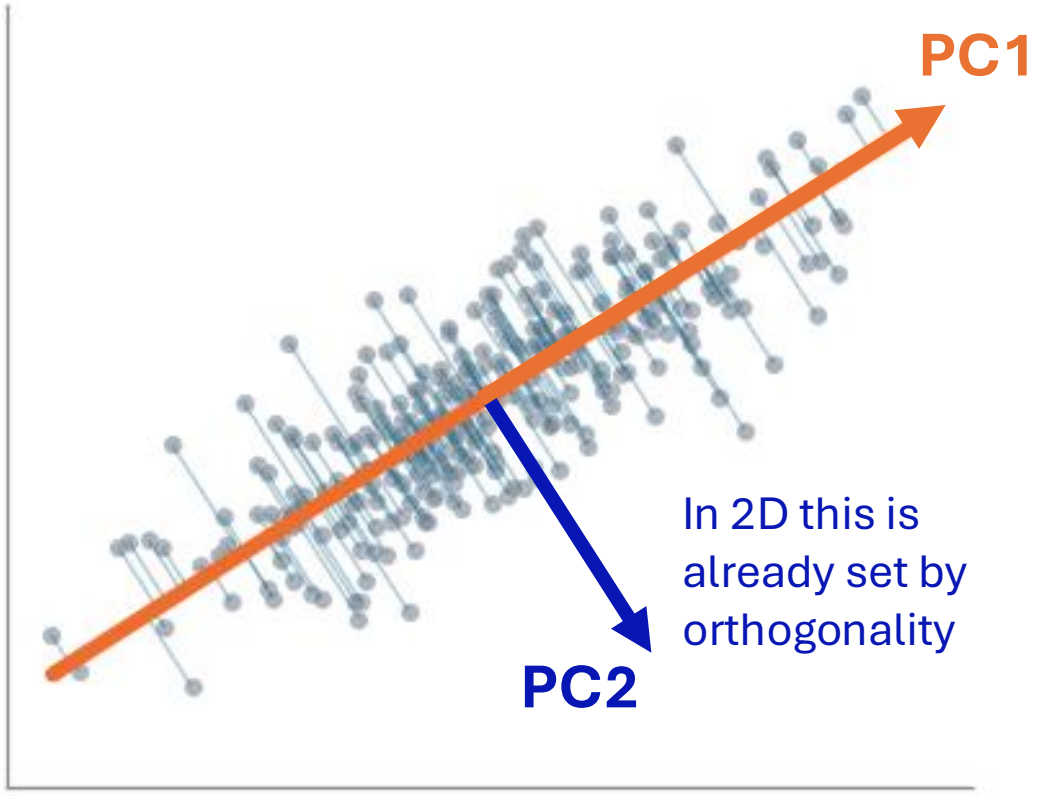
(In 2D, this is trivial)
(not trivial in higher dimensions)

PCA = finding directions of max variance





PCA = then projecting onto those directions



To reduce dimensions, all we need to do is **project our data onto the principal component with maximal variance (PC1)**

If we do this projection, we essentially get **rid of information*** along the orthogonal component with minimal variance (PC2)

Now let's walk through the math…

*We'll come back to this…

PCA: Four steps

1. Center the data around the origin (subtract the mean of each feature)

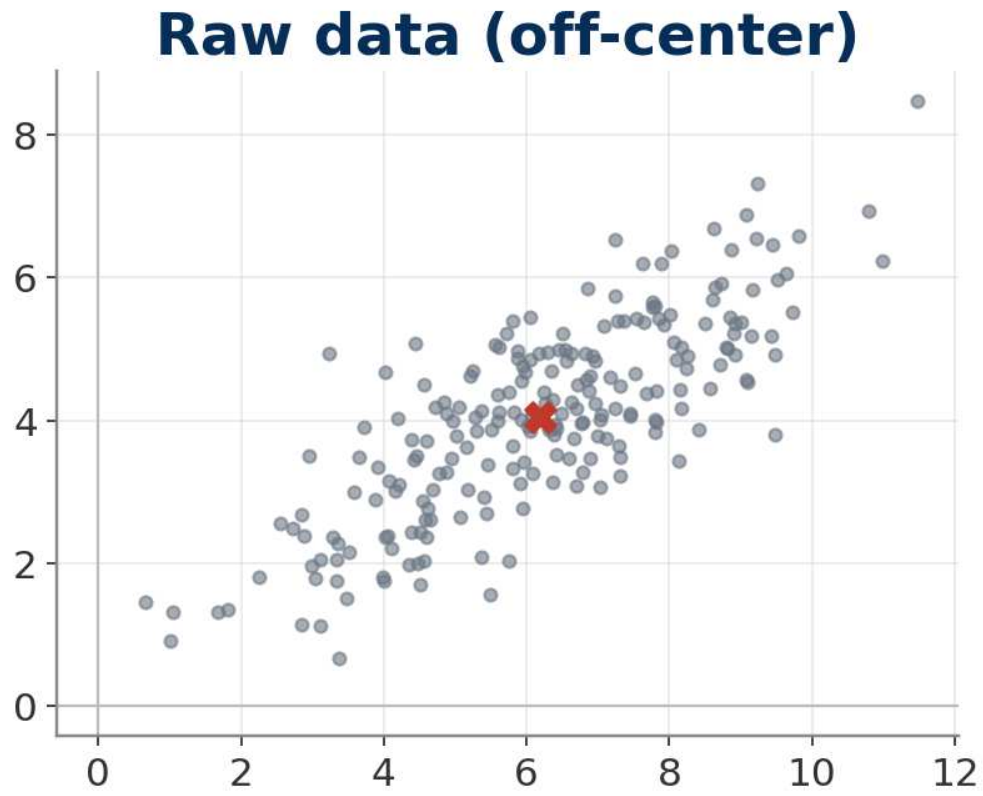
2. Compute the covariance matrix (Σ) of the data

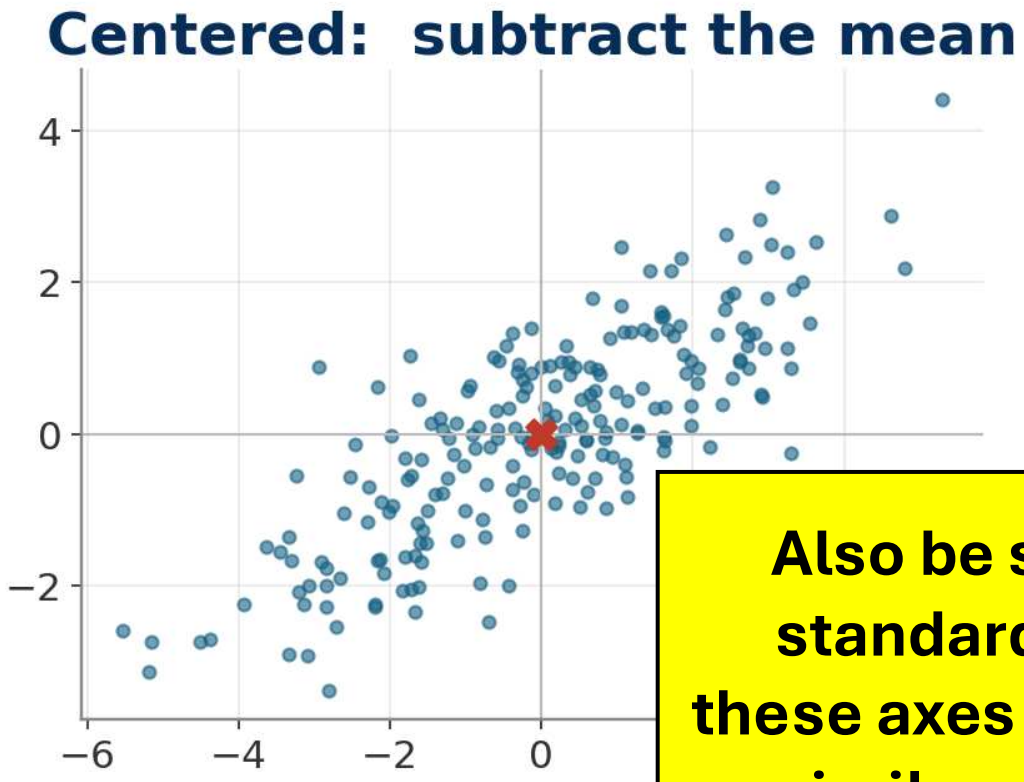
3. Compute eigenvalues and eigenvectors of the covariance matrix

4. Keep the top-*k* eigenvectors and project the data onto them

Step 1: Centering the data

1. Center the data around the origin (subtract the mean of each feature)





Also be sure to standardize so these axes are on similar scales!

Step 1: Centering the data

1. Center the data around the origin (subtract the mean of each feature)

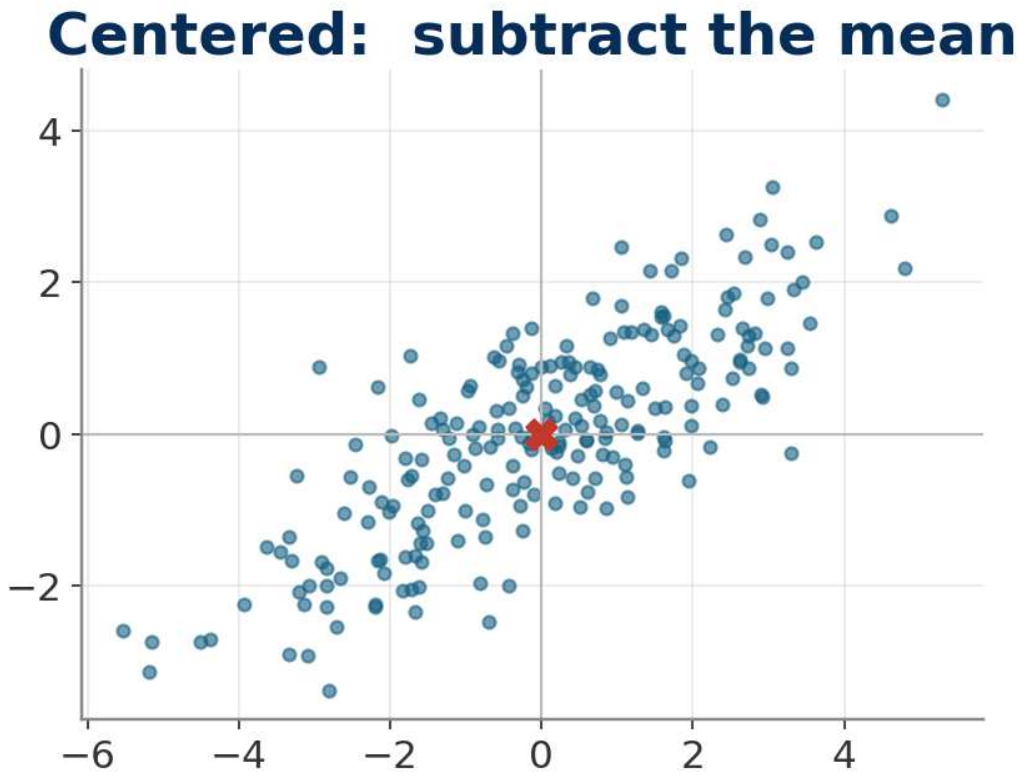
We describe this using the **centered data matrix (X):**

$$X = \begin{pmatrix} -2.1 & -0.9 \\ 0.3 & 0.5 \\ -3.4 & -1.1 \\ \dots & \dots \\ 3.8 & 2.1 \end{pmatrix}$$

Sample 1
Sample 2

Feature #1 Feature #2

There is no intercept column!



Step 2: The covariance matrix

2. Compute the covariance matrix (Σ) of the data

$$\Sigma = \frac{1}{n-1} X^T X$$

Where n is the total number of samples

Recall: Matrix transpose

2	4	-1
-10	5	11
18	-7	6



2	-10	18
4	5	-7
-1	11	6

In two dimensions (x and y), this is:

$$\Sigma = \begin{pmatrix} \mathbf{Var}(x) & \mathbf{Cov}(x,y) \\ \mathbf{Cov}(x,y) & \mathbf{Var}(y) \end{pmatrix}$$

Where...

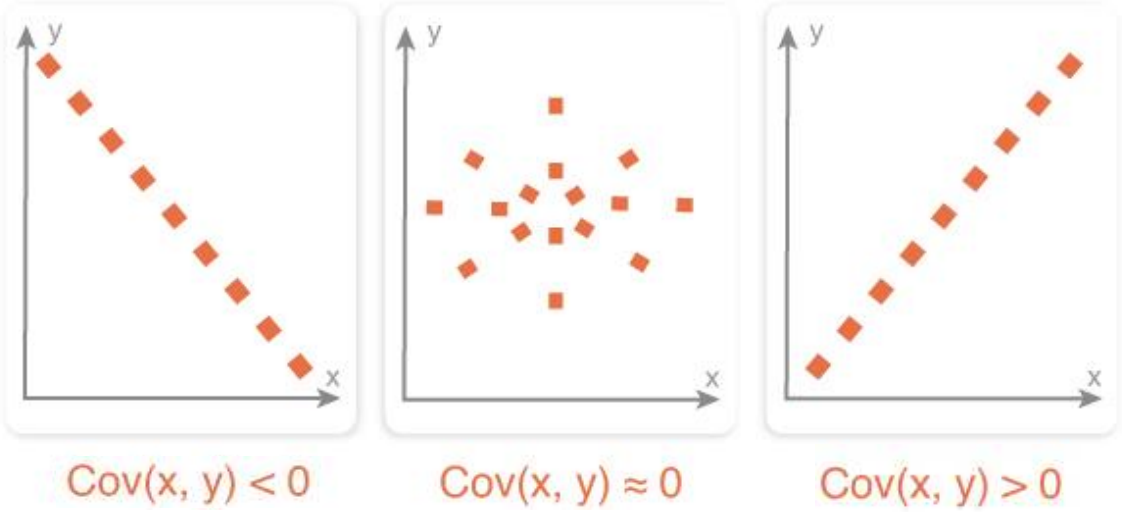
$$\mathbf{Cov}(x,y) = \frac{1}{n-1} \sum_{i=0}^{n} (x_i - \bar{x})(y_i - \bar{y})$$

Step 2: The covariance matrix

2. Compute the covariance matrix (Σ) of the data

Remember:

- *Variance* describes spread in 1 variable
- *Covariance* measures how closely 2 variables change together



In two dimensions (x and y), this is:

$$\boldsymbol{\Sigma} = \begin{pmatrix} \mathbf{Var}(\boldsymbol{x}) & \mathbf{Cov}(\boldsymbol{x},\boldsymbol{y}) \\ \mathbf{Cov}(\boldsymbol{x},\boldsymbol{y}) & \mathbf{Var}(\boldsymbol{y}) \end{pmatrix}$$

Where…

$$\mathbf{Cov}(\boldsymbol{x},\boldsymbol{y}) = \frac{\mathbf{1}}{\boldsymbol{n}-\mathbf{1}} \sum_{\boldsymbol{i}=\mathbf{0}}^{\boldsymbol{n}} (\boldsymbol{x_i} - \overline{\boldsymbol{x}})(\boldsymbol{y_i} - \overline{\boldsymbol{y}})$$

Step 2: The covariance matrix

2. Compute the covariance matrix (Σ) of the data

Remember:

- *Variance* describes spread in 1 variable
- *Covariance* measures how closely 2 variables change together

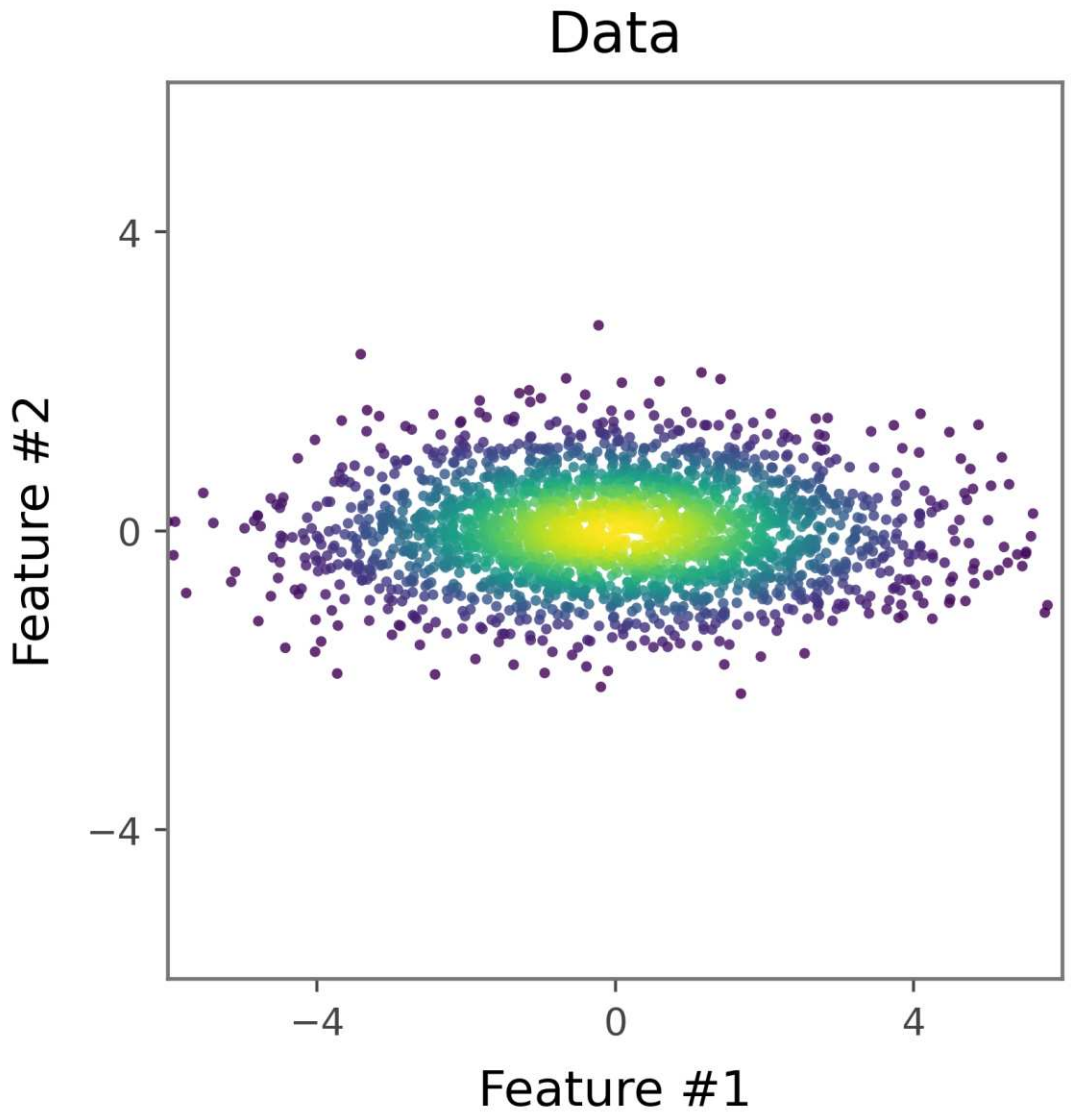
In two dimensions (x and y), this is:

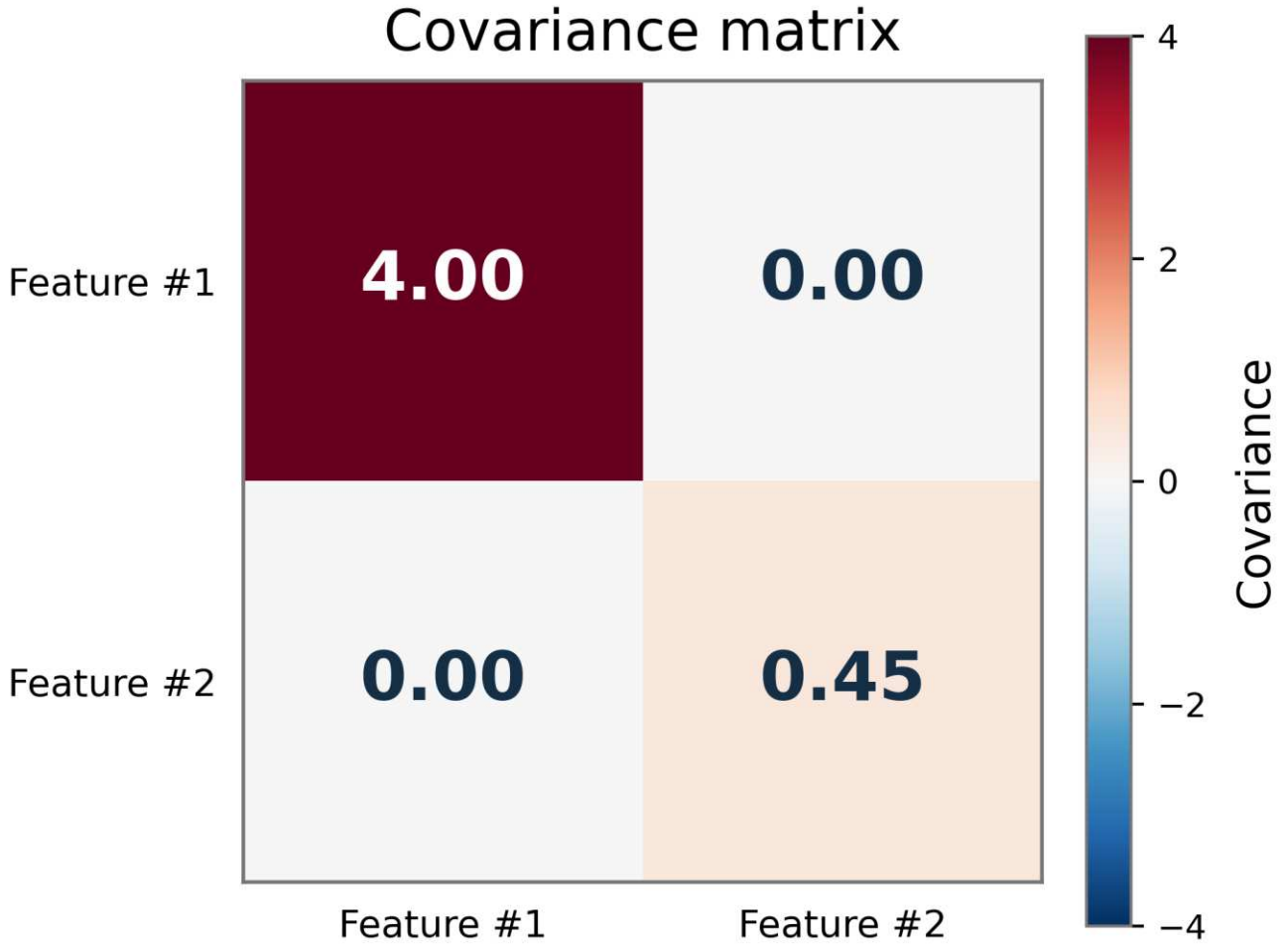
$$\Sigma = \begin{pmatrix} \mathbf{Var}(x) & \mathbf{Cov}(x,y) \\ \mathbf{Cov}(x,y) & \mathbf{Var}(y) \end{pmatrix}$$

Σ_{ij} = how features i and j vary together

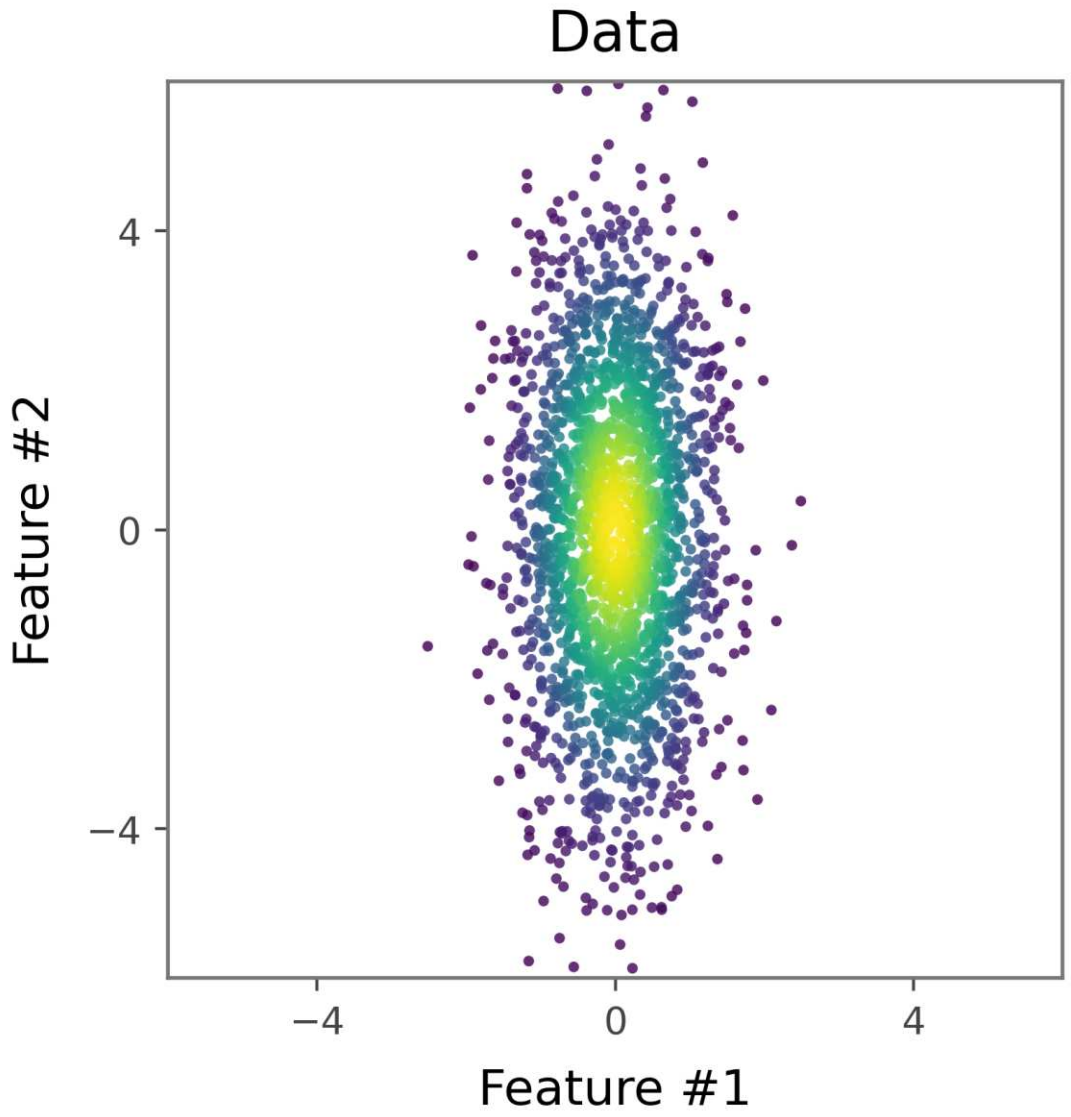
If off-diagonals are large, features are correlated, and there is room to reduce!

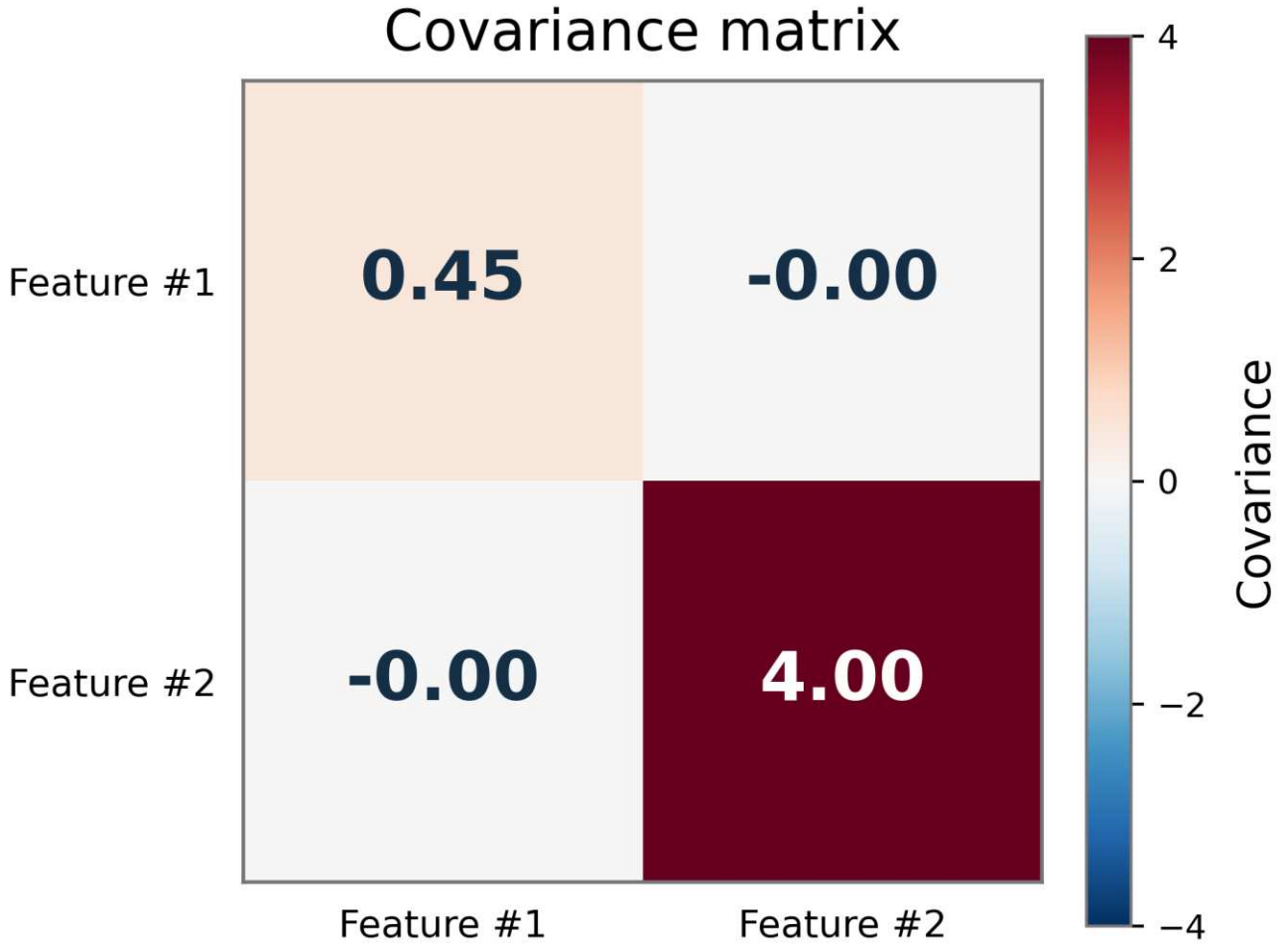
Covariance matrix examples



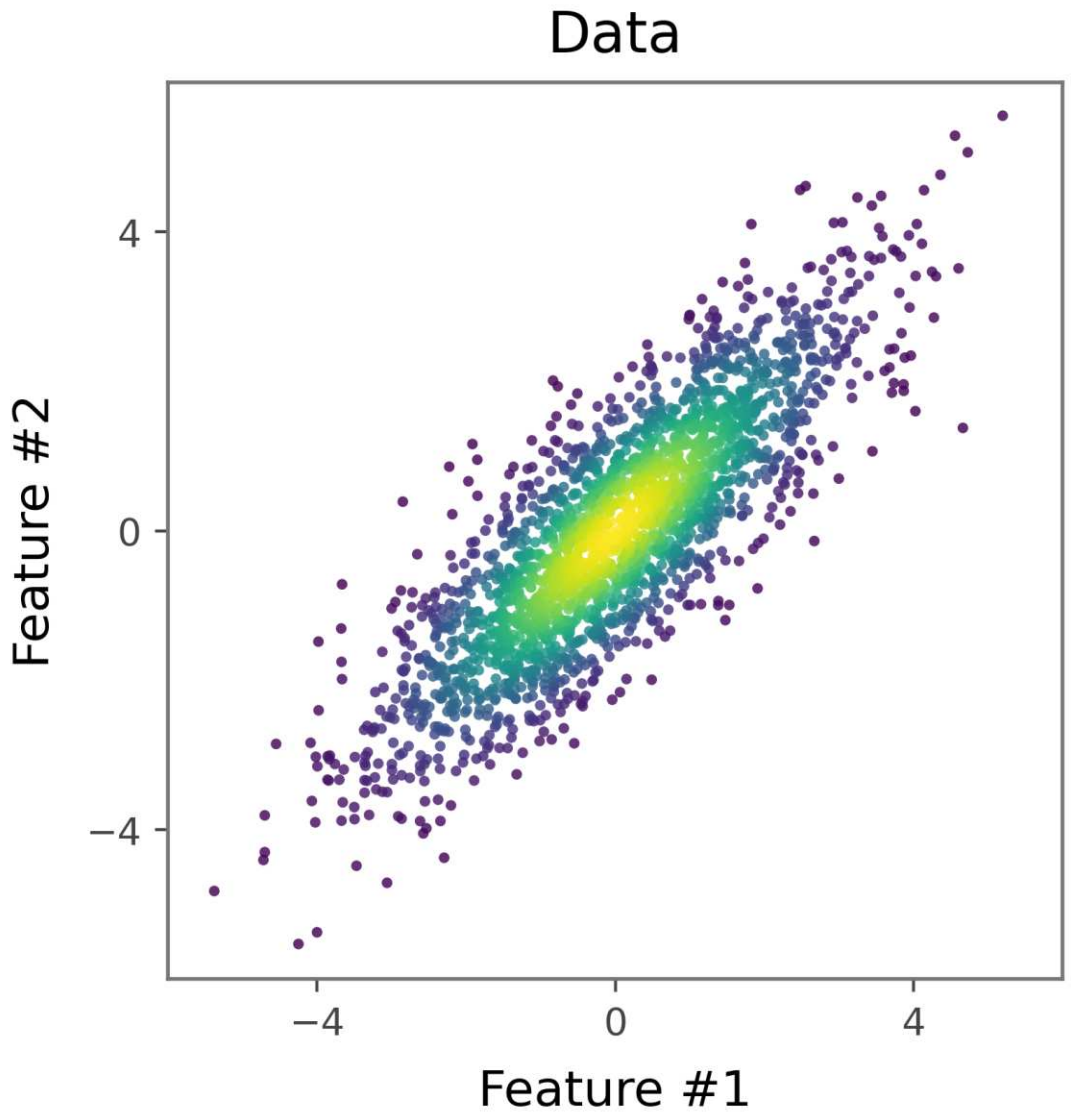


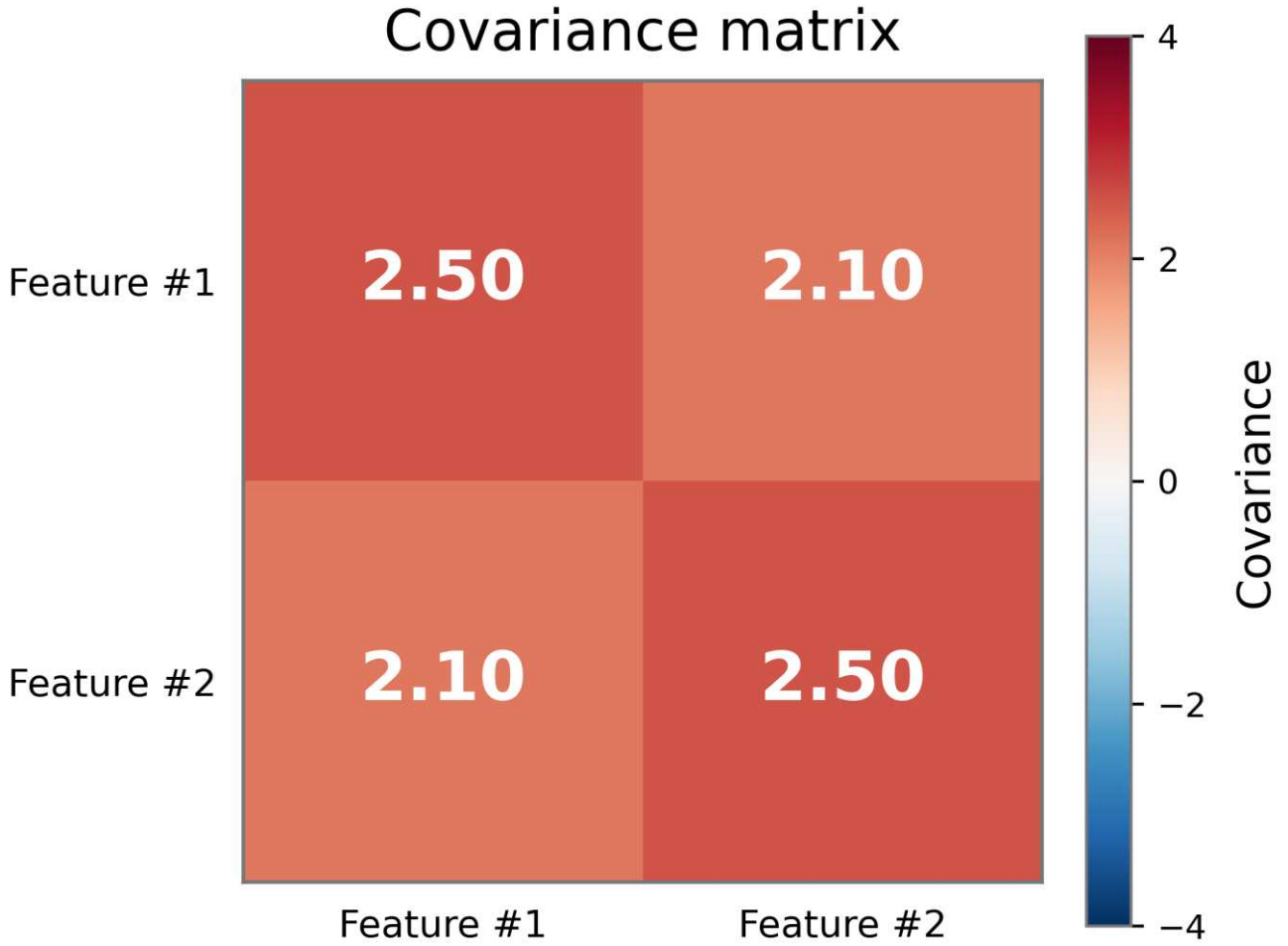
Covariance matrix examples



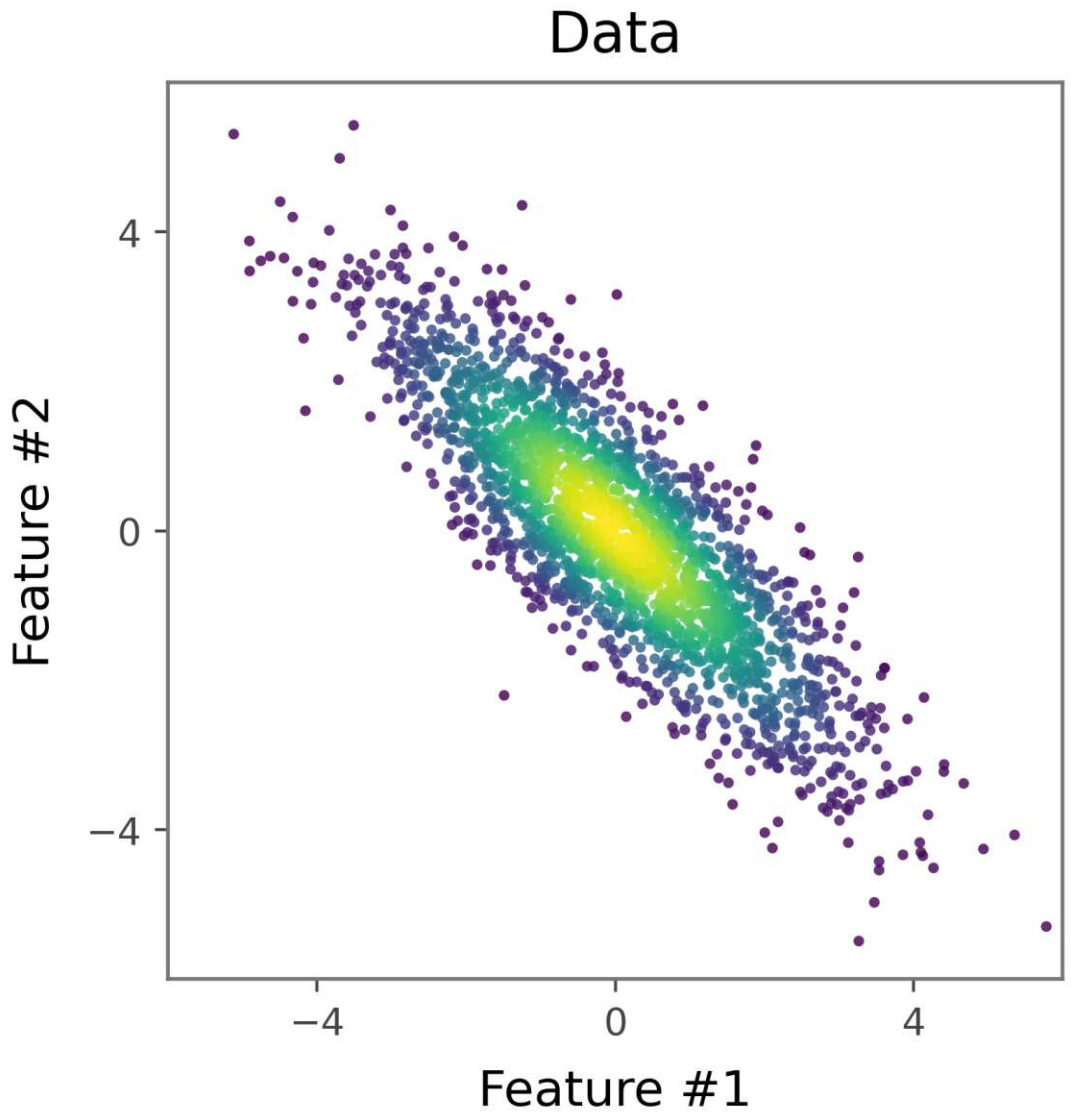


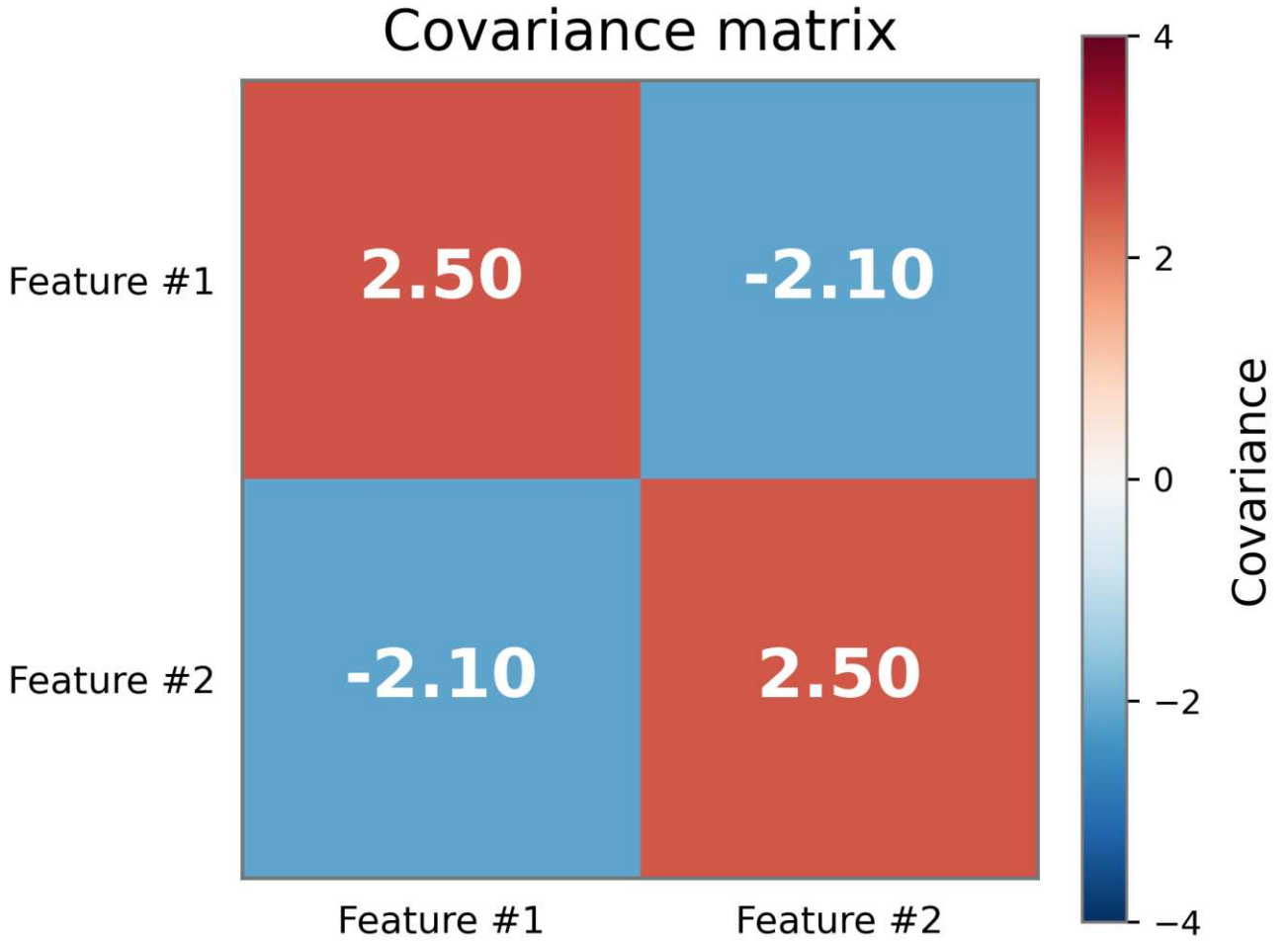
Covariance matrix examples





Covariance matrix examples





Questions?

PCA: Four steps

1. Center the data around the origin (subtract the mean of each feature)

2. Compute the covariance matrix (Σ) of the data

3. Compute eigenvalues and eigenvectors of the covariance matrix

4. Keep the top-*k* eigenvectors and project the data onto them

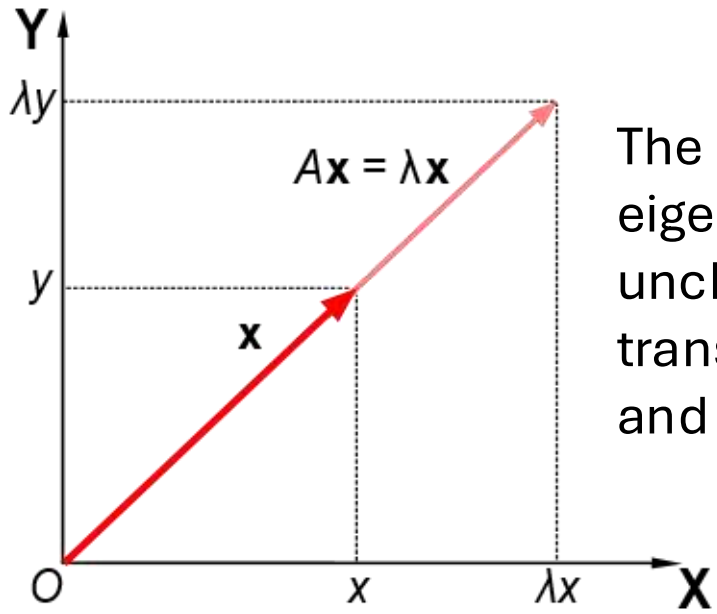
Step 3: Eigenvalues and vectors

3. Compute eigenvalues and eigenvectors of the covariance matrix

Eigenvalues of Σ

Covariance matrix → $\Sigma v_i = \lambda_i v_i$

Eigenvectors of Σ



The direction of eigenvector ***x*** is unchanged by the transformation *A*, and is scaled by λ

Recall: An eigenvector is a direction that is unchanged by a linear transformation (except for scaling), and the corresponding eigenvalue gives the amount of that scaling.

Step 3: Eigenvalues and vectors

3. Compute eigenvalues and eigenvectors of the covariance matrix

Eigenvalues of Σ

Covariance matrix → $\Sigma v_i = \lambda_i v_i$

Eigenvectors of Σ

Solve for the eigenvectors and values:

$$\det(\Sigma - \lambda I) = 0$$

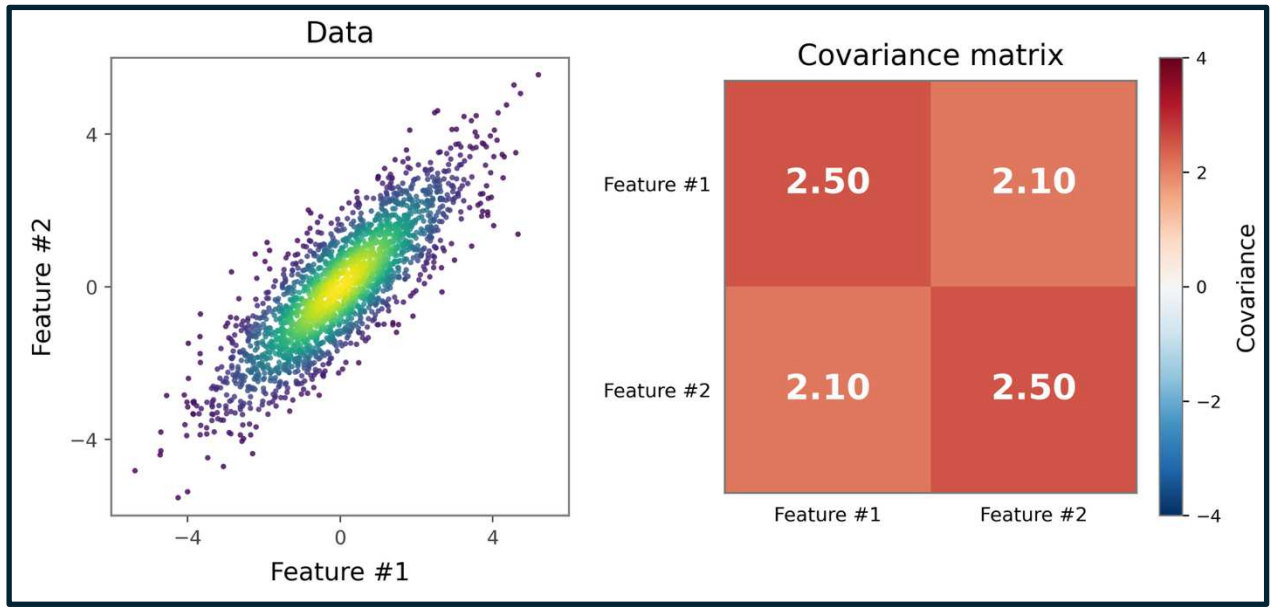
Where*

$$\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

*I will only ask you about 2×2 matrices ☺

Recall: An eigenvector is a direction that is unchanged by a linear transformation (except for scaling), and the corresponding eigenvalue gives the amount of that scaling.

Eigenvalues and vectors: Calculation EX



$$\Sigma - \lambda I = \begin{pmatrix} 2.5 - \lambda & 2.1 \\ 2.1 & 2.5 - \lambda \end{pmatrix}$$

$$\det(\Sigma - \lambda I) = (2.5 - \lambda)^2 - 2.1^2 = 0$$

$$(2.5 - \lambda)^2 = 2.1^2 \rightarrow 2.5 - \lambda = \pm 2.1$$

Eigenvalues $\lambda = \{0.4, 4.6\}$

$$\Sigma v_i = \lambda_i v_i \rightarrow (\Sigma - \lambda_i I) v_i = 0$$

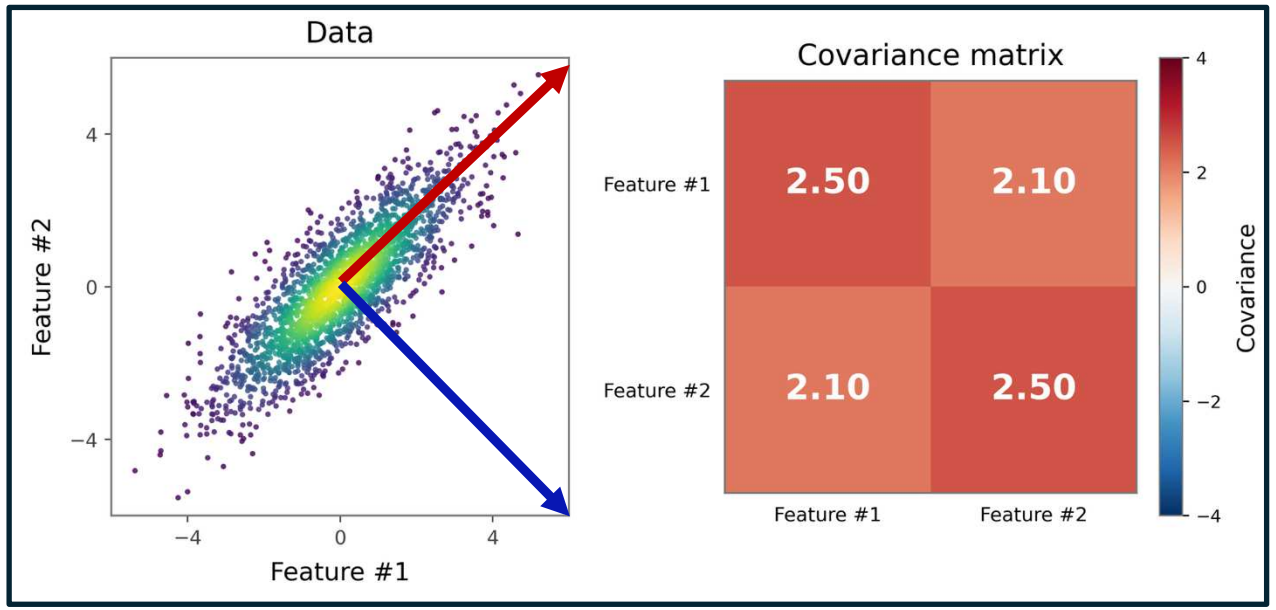
$$\begin{pmatrix} 2.5 - 0.4 & 2.1 \\ 2.1 & 2.5 - 0.4 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = 0$$

$$\left.\begin{aligned} 2.1x_1 + 2.1y_1 = 0 \\ 2.1x_1 + 2.1y_1 = 0 \end{aligned}\right\} \; x_1 = -y_1$$

$$\begin{pmatrix} 2.5 - 4.6 & 2.1 \\ 2.1 & 2.5 - 4.6 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = 0$$

$$\left.\begin{aligned} -2.1x_2 + 2.1y_2 = 0 \\ 2.1x_2 - 2.1y_2 = 0 \end{aligned}\right\} \; x_2 = y_2$$

Eigenvalues and vectors: Calculation EX



Notice how these look on the plot! They are the directions of max and min variance.

$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$v_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

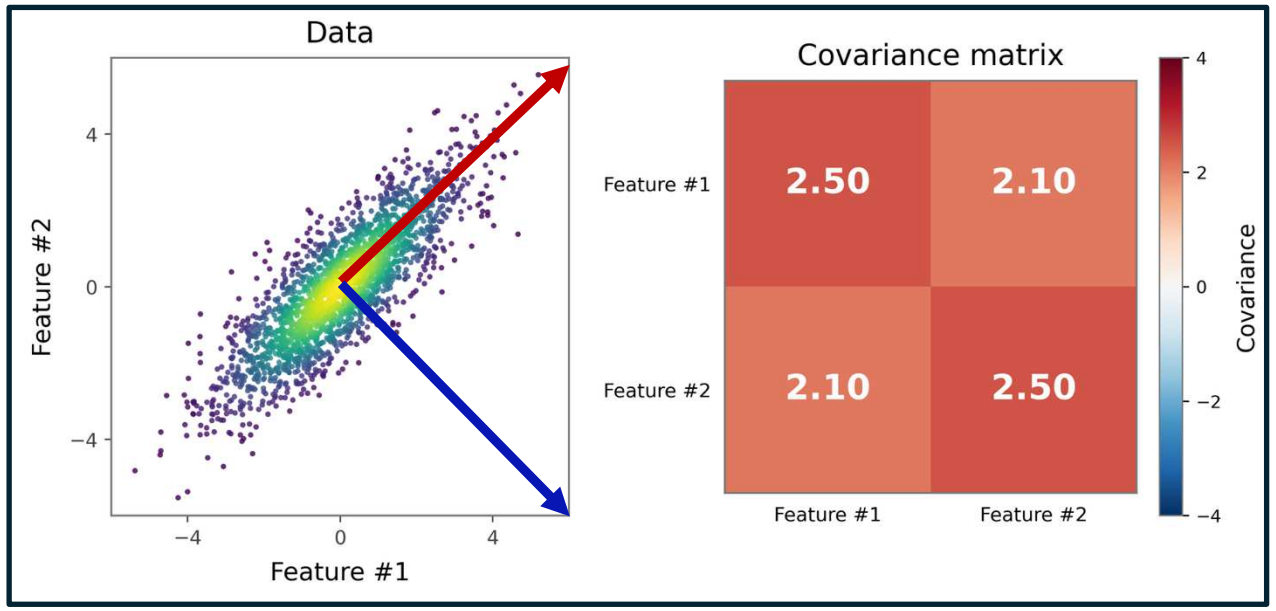
Eigenvectors

$$\Sigma v_i = \lambda_i v_i \rightarrow (\Sigma - \lambda_i I) v_i = 0$$

$$\begin{pmatrix} 2.5 - 0.4 & 2.1 \\ 2.1 & 2.5 - 0.4 \end{pmatrix}\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = 0 \qquad \left.\begin{array}{l} 2.1x_1 + 2.1y_1 = 0 \\ 2.1x_1 + 2.1y_1 = 0 \end{array}\right\} \; x_1 = -y_1$$

$$\begin{pmatrix} 2.5 - 4.6 & 2.1 \\ 2.1 & 2.5 - 4.6 \end{pmatrix}\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = 0 \qquad \left.\begin{array}{l} -2.1x_2 + 2.1y_2 = 0 \\ 2.1x_2 - 2.1y_2 = 0 \end{array}\right\} \; x_2 = y_2$$

Eigenvalues and vectors: Calculation EX



Notice how these look on the plot! They are the directions of max and min variance.

$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

$$v_2 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

Eigenvectors

Normalization: Divide by the norm

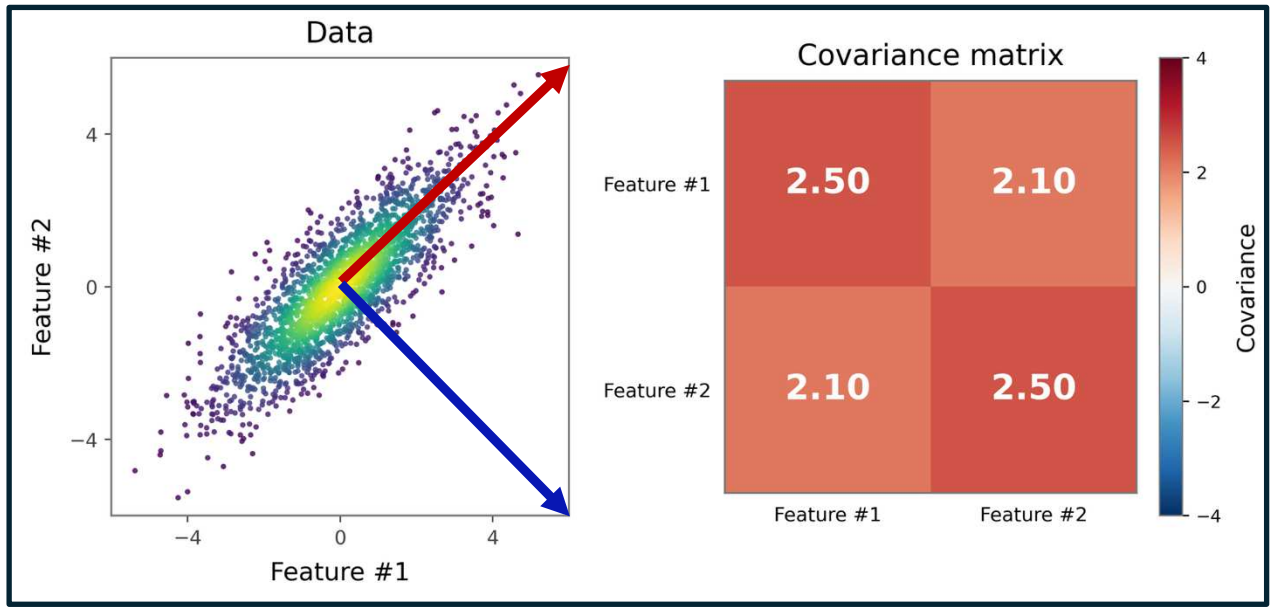
$$v'_1 = \begin{pmatrix}1\\-1\end{pmatrix} \rightarrow |v'_1| = \sqrt{1^2 + (-1)^2} = \sqrt{2}$$

$$v_1 = \frac{1}{|v'_1|}\begin{pmatrix}1\\-1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix}$$

$$x_1 = -y_1$$

$$x_2 = y_2$$

Eigenvalues and vectors: Calculation EX



Notice how these look on the plot! They are the directions of max and min variance.

$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$v_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Eigenvectors

It is typical to re-order these so **first component has the largest variance:**

$$\mathbf{PC}_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad \mathbf{PC}_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

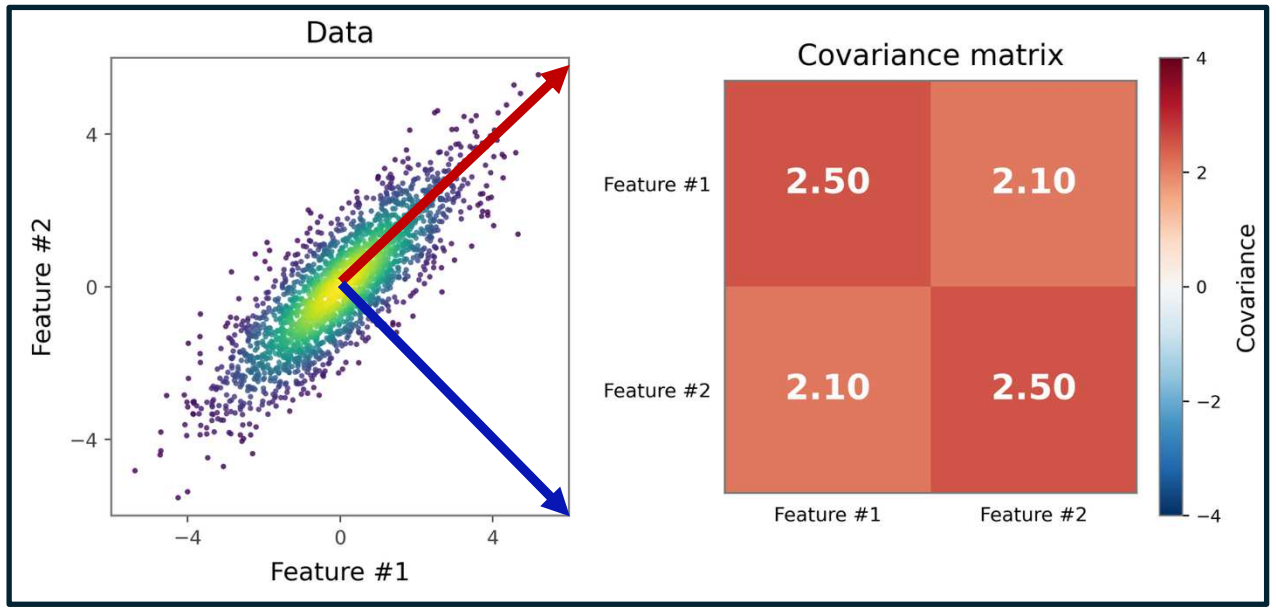
Eigenvalues describe the variance in our data long each direction!

$$\lambda_1 = 4.6$$

$$\lambda_2 = 0.4$$

Large λ means high variance

Eigenvalues and vectors: Calculation EX



Notice how these look on the plot! They are the directions of max and min variance.

$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$v_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Eigenvectors

$f_1 = \frac{4.6}{4.6+0.4} = 0.92 \rightarrow$ **quite large! Let's get rid of PC_2 ...**

The fraction of the total variance explained by PC_i is $\frac{\lambda_i}{\sum_i \lambda_i}$

Eigenvalues describe the variance in our data long each direction!

$\lambda_1 = 4.6$ $\lambda_2 = 0.4$

PCA: Four steps

1. Center the data around the origin (subtract the mean of each feature)

2. Compute the covariance matrix (Σ) of the data

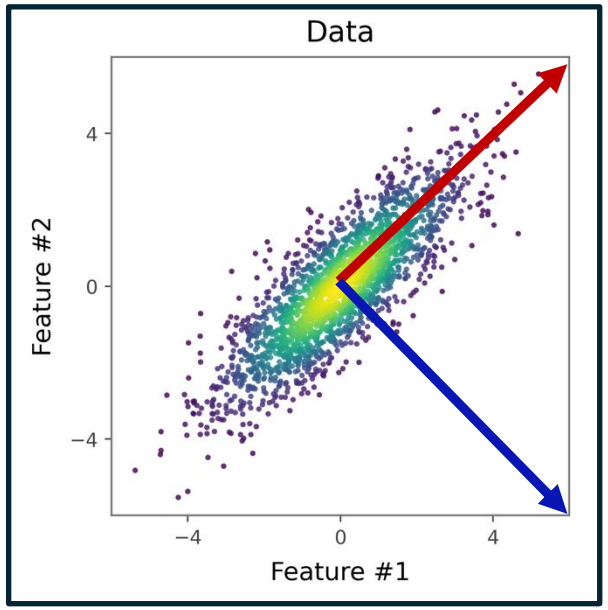
3. Compute eigenvalues and eigenvectors of the covariance matrix

4. Keep the top-*k eigenvectors and project the data onto them**

*sorry for the many definitions of *k* throughout the course

Step 4: Projection onto principal component(s)

4. Keep the top-*k* eigenvectors and project the data onto them

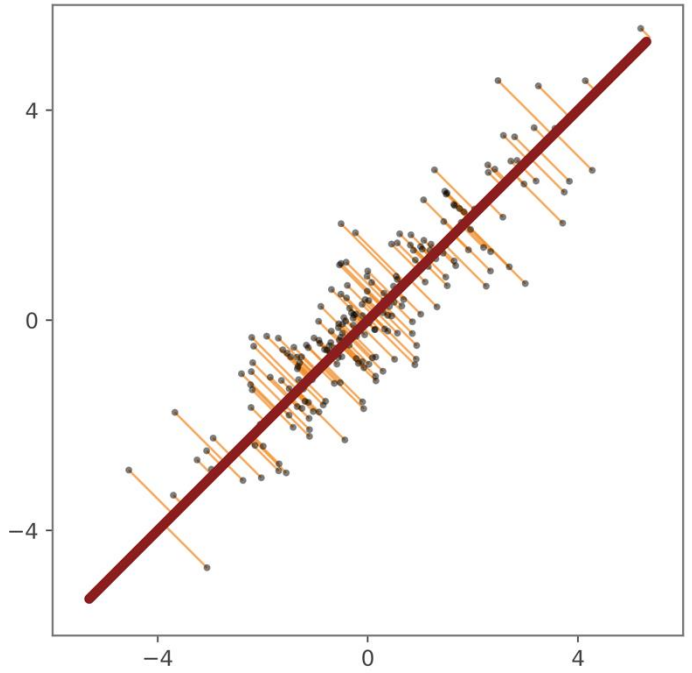


If our original data is 2D, we compress down to 1D by projecting the data onto principal component #1

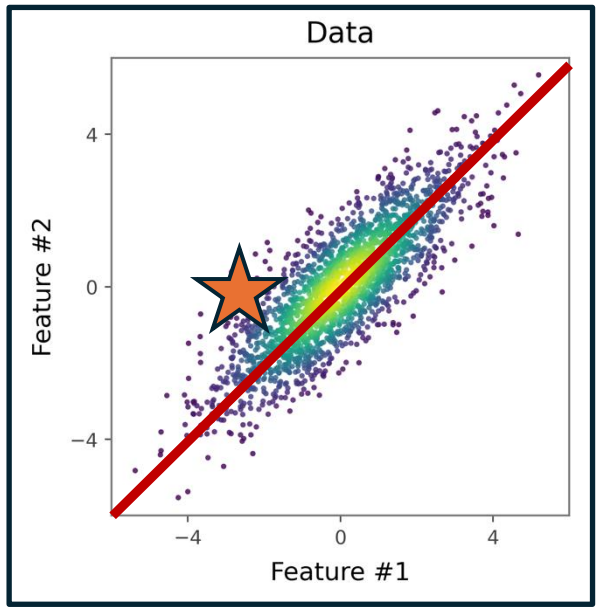
$$z_k = x^T v_k$$

This is a **scalar** representing the **coordinate along PC1.**

Multiply the transpose of each sample in the centered data matrix by the kth (often 1st) eigenvector



Step 4: Projection onto principal component(s)



$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

$$z_1 = x^T v_1$$

$$X = \begin{pmatrix} -3.5 & -0.1 \\ 0.3 & 0.5 \\ -3.4 & -1.1 \\ \dots & \dots \\ 3.8 & 2.1 \end{pmatrix}$$

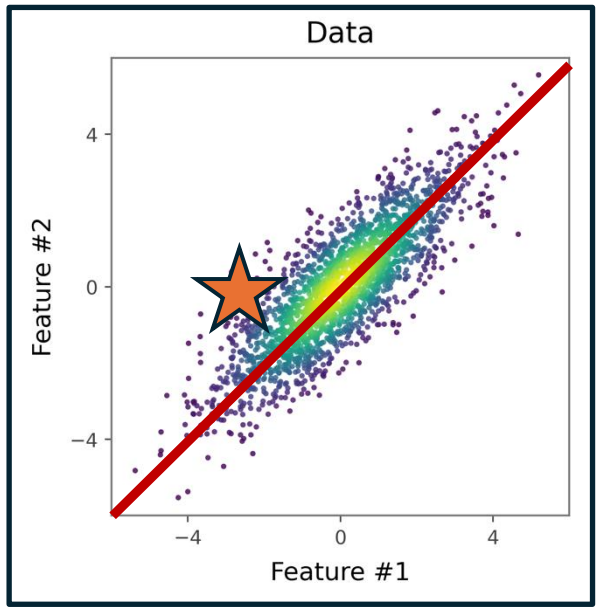
Remember: **each row is one sample.** Let's see how the 1st sample **gets projected onto PC1**

$$x^T v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}-3.5 & -0.1\end{pmatrix}\begin{pmatrix}1\\1\end{pmatrix}$$

$$= \frac{-3.5 - 0.1}{\sqrt{2}} \approx -2.54$$

Note: You can leave it as row (transpose of a vector)

Step 4: Projection onto principal component(s)



$$v_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

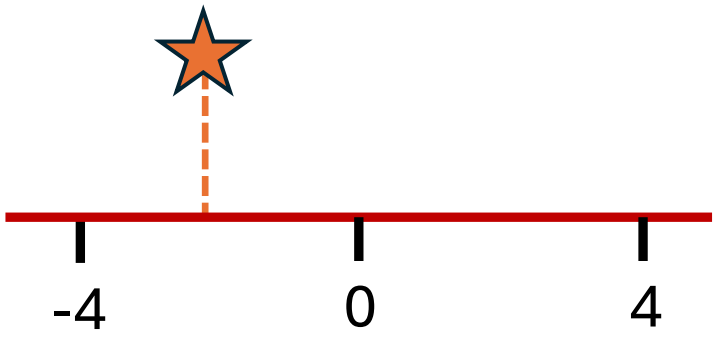
$$z_1 = x^T v_1$$

$$X = \begin{pmatrix} -3.5 & -0.1 \\ 0.3 & 0.5 \\ -3.4 & -1.1 \\ \dots & \dots \\ 3.8 & 2.1 \end{pmatrix}$$

Remember: **each row is one sample.** Let's see how the 1st sample **gets projected onto PC1**

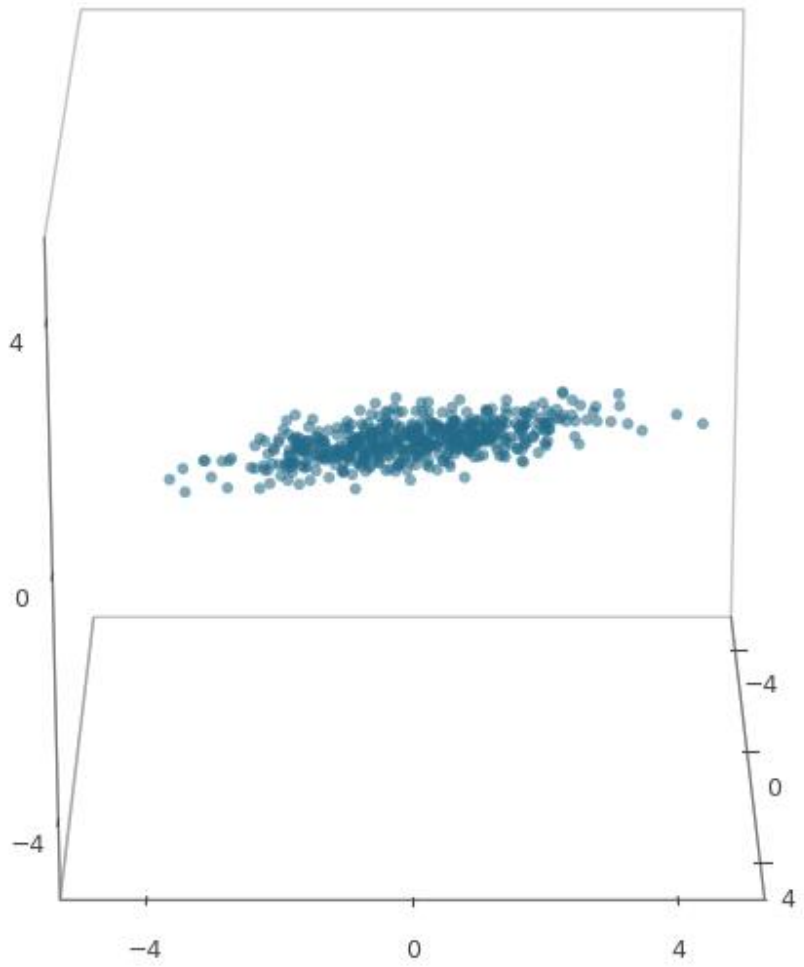
$$x^T v_1 = \frac{1}{\sqrt{2}}(-3.5 \quad -0.1)\begin{pmatrix}1\\1\end{pmatrix}$$

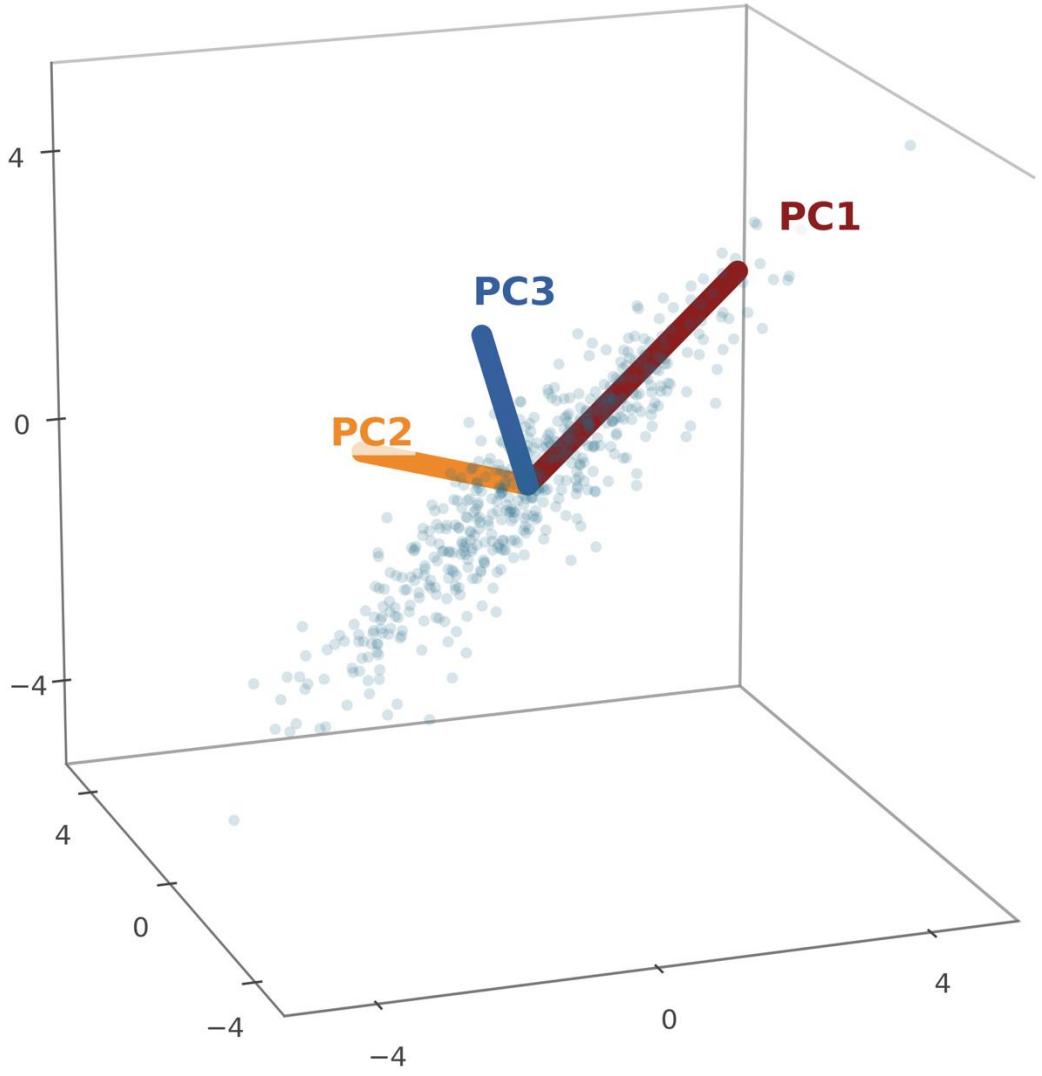
$$= \frac{-3.5 - 0.1}{\sqrt{2}} \approx -2.54$$



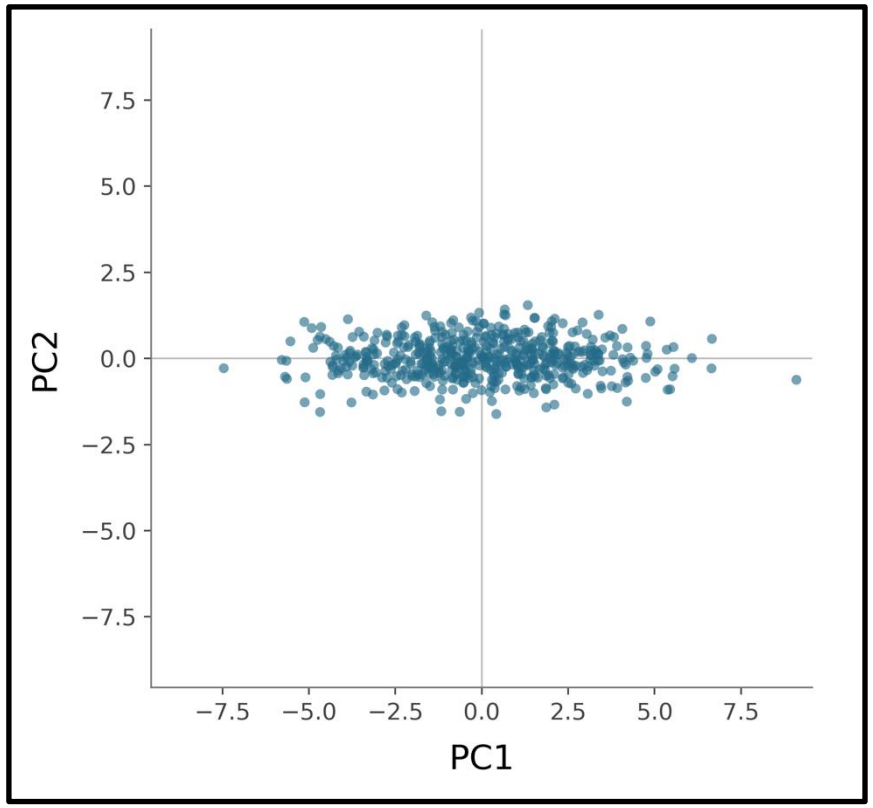
Questions?

PCA in k dimensions (up to k components)

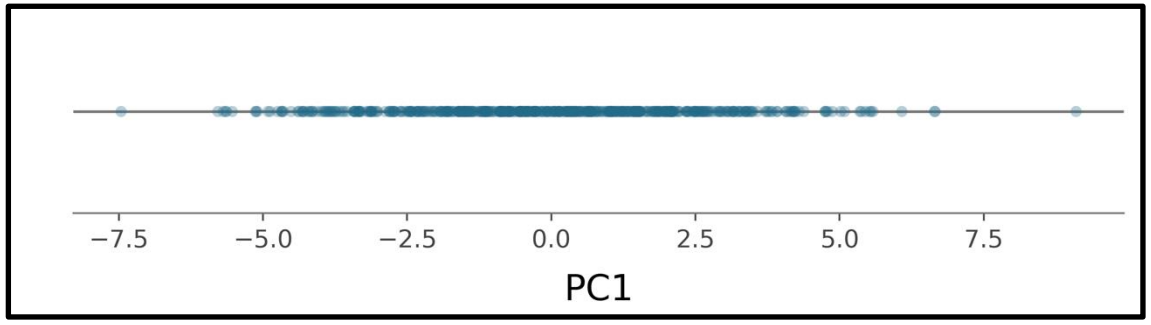


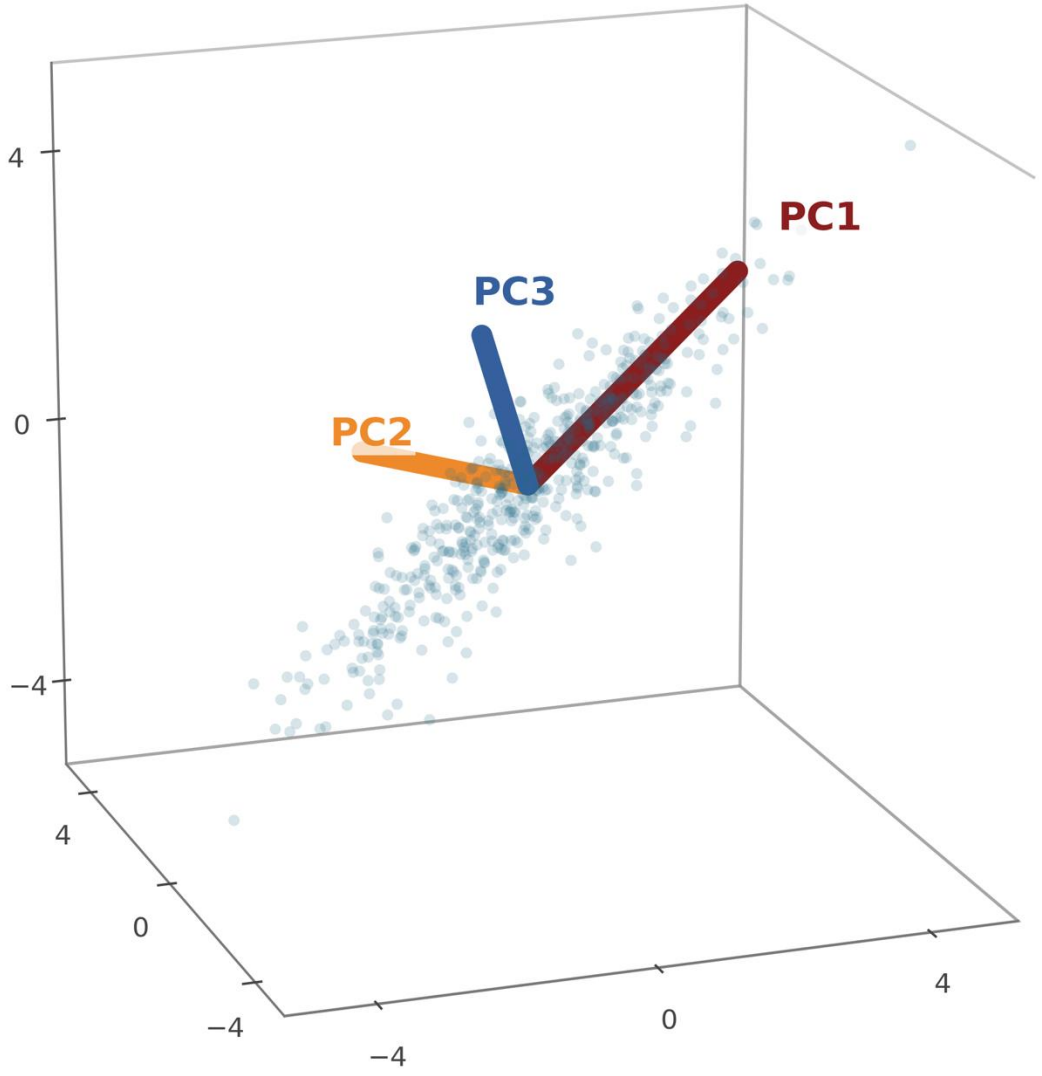


How many principal components to keep?



Compress further

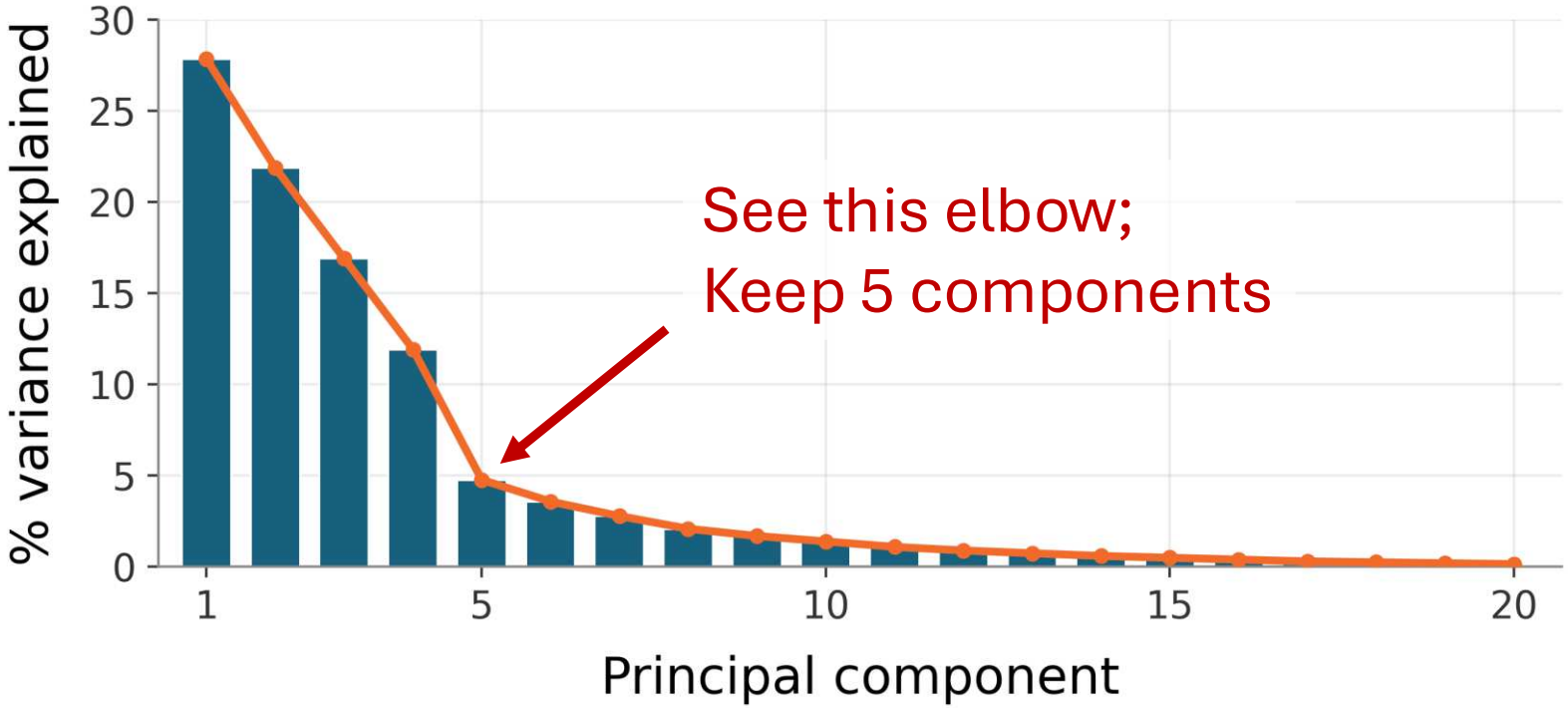




The proportion of variance

The fraction of the total variance explained by PC_i is $\frac{\lambda_i}{\sum_i \lambda_i}$

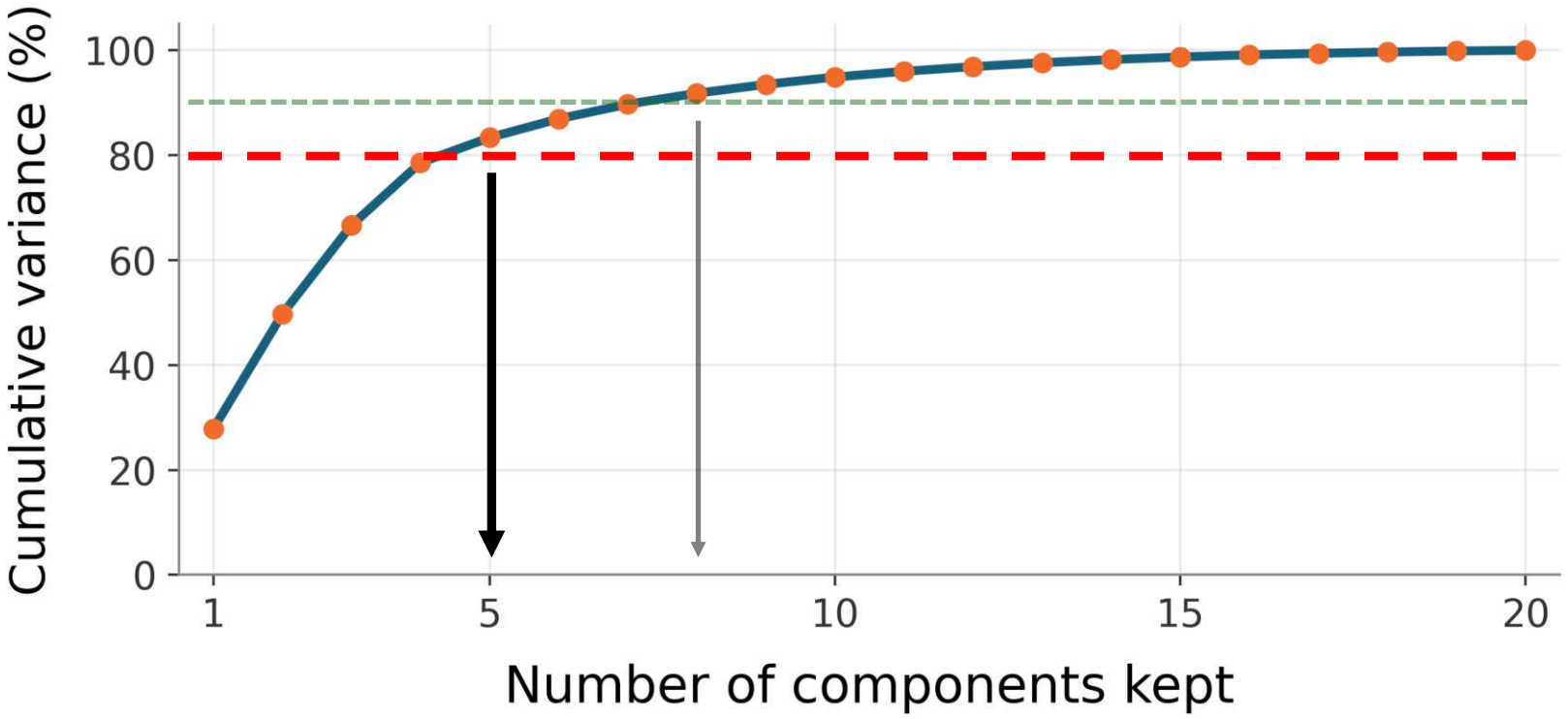
Each subsequent component accounts for a smaller fraction of the total variance



We can visualize this using a **"Scree plot"**

Look for the **"elbow"** to decide how many components to keep (though, it is not always so obvious…)

The cumulative variance



$$\mathbf{Var}_k^{\mathbf{cumul.}} = \frac{\lambda_1 + \lambda_2 + \cdots \lambda_k}{\sum_i \lambda_i}$$

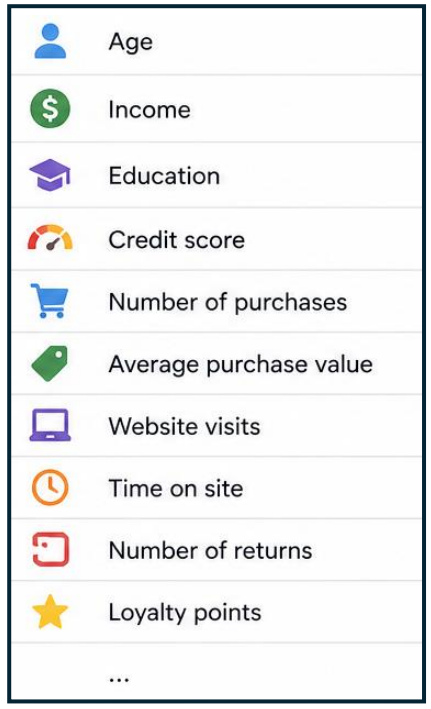
This accounts for how much variance is explained by the first k components

There is a tradeoff between accuracy and simplicity/interpretability

- **Fewer components → more interpretable but some information loss**

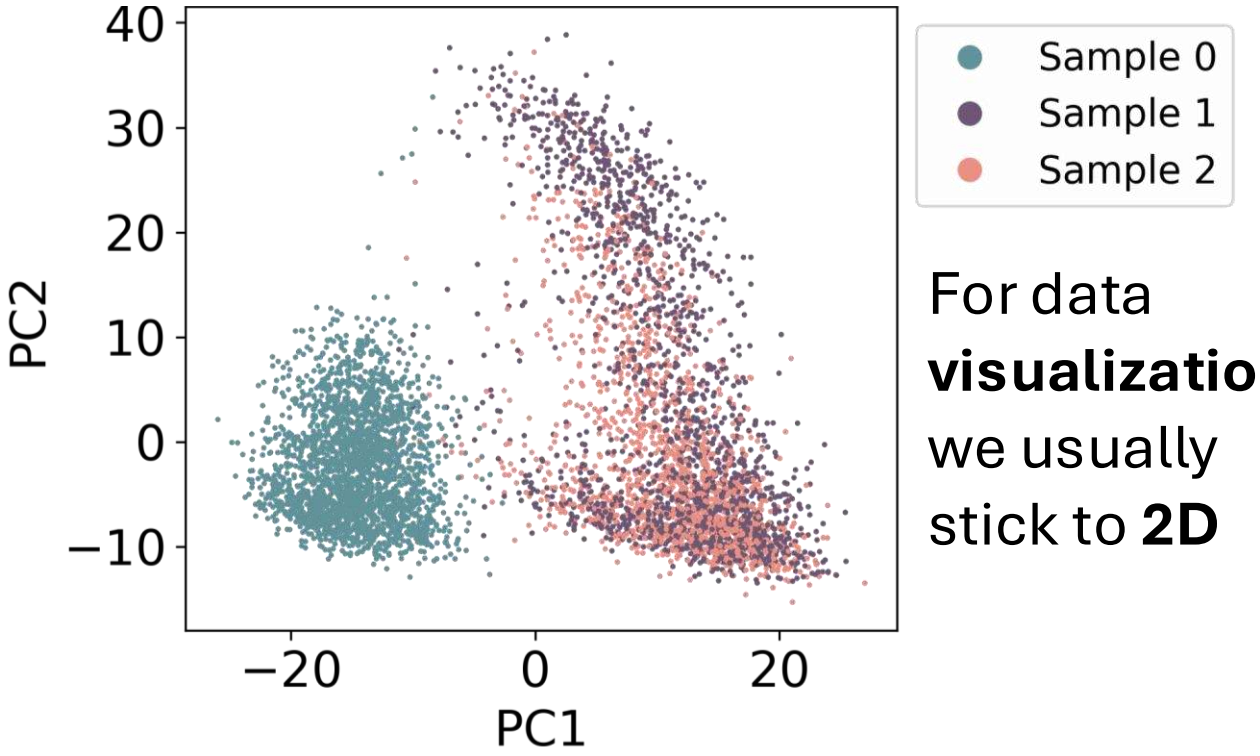
In general, aim for *at least* 80% variance accounted for, ideally > 90%

How many components? Depends on your goal



Model

If our goal is **feature downselection** before we **build a model**, then it is best to keep more components



For data **visualization**, we usually stick to **2D**

There is a tradeoff between accuracy and simplicity/interpretability

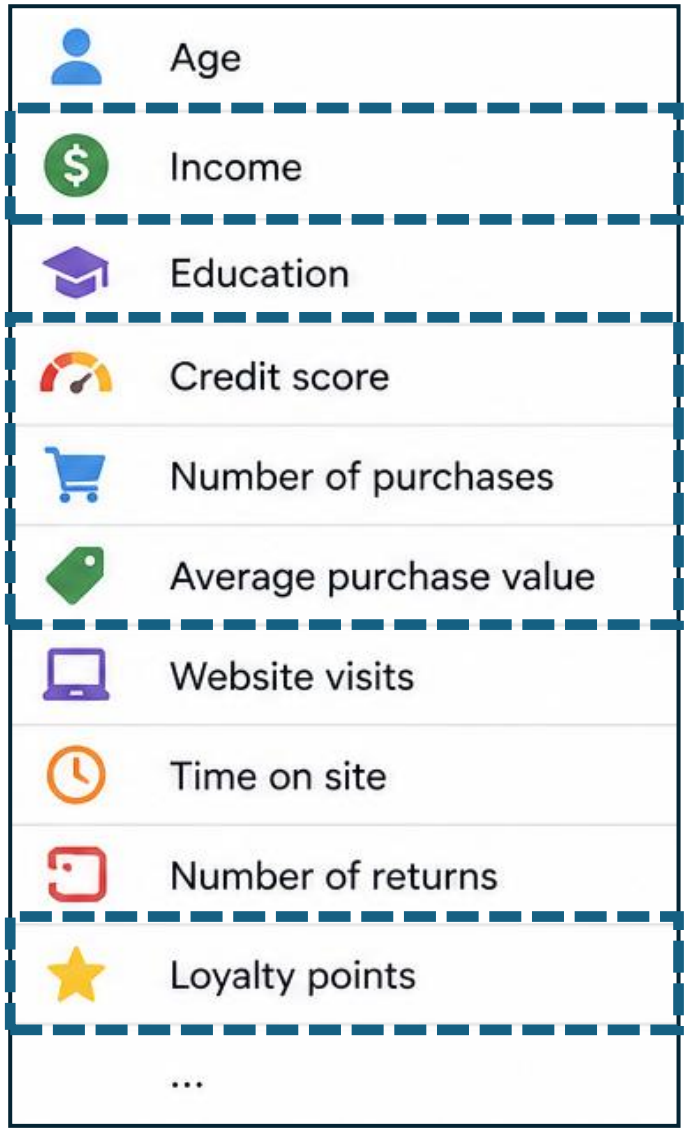
- **Fewer components → more interpretable but some information loss**

In general, aim for *at least* 80% variance accounted for, ideally > 90%

Questions?

Applications of PCA

Interpretable feature *combinations*



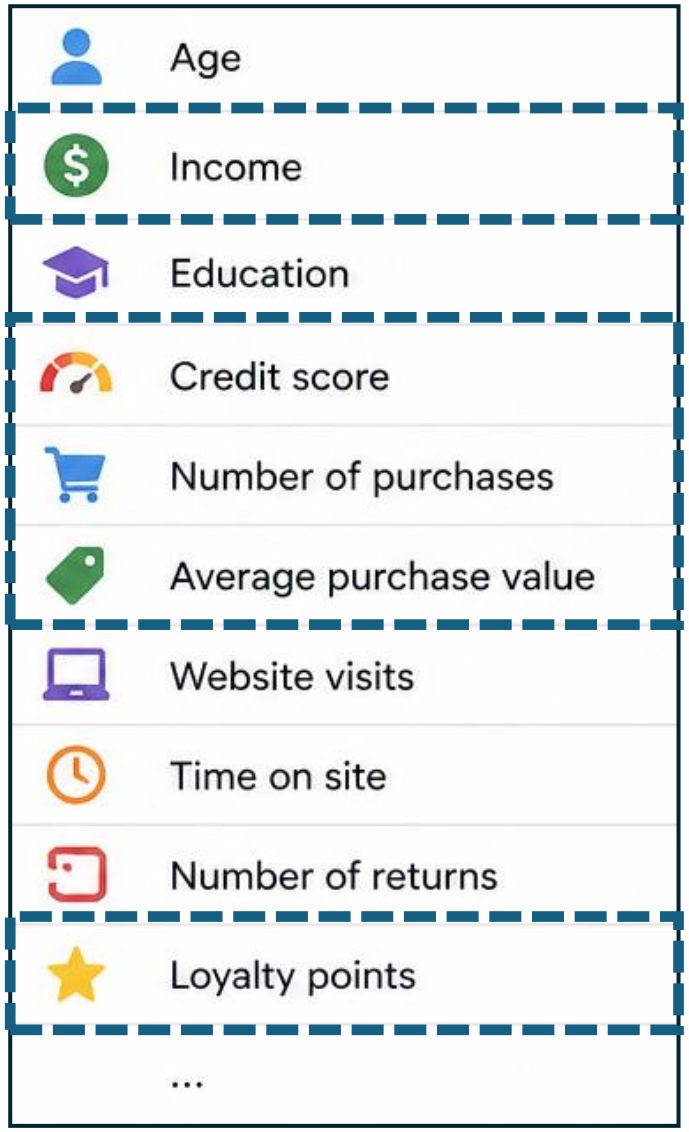
Each principal component is a **linear combination** of the original features → good interpretability!

$$\mathbf{PC}_1 = \begin{pmatrix} 0 \\ 0.32 \\ 0 \\ 0.10 \\ 0.25 \\ 0.21 \\ 0 \\ 0 \\ 0 \\ 0.12 \\ \dots \end{pmatrix}$$

Suppose our first principal component looked something like this

(Note: I got lazy, so this is not normalized)

Interpretable feature *combinations*

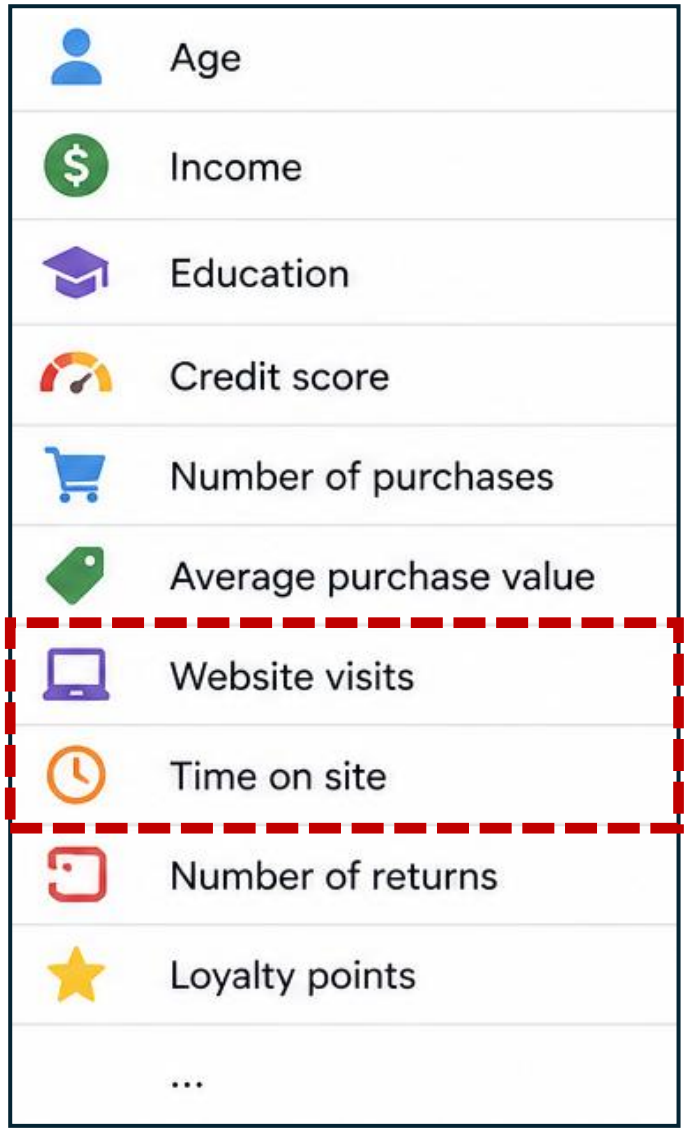


Each principal component is a **linear combination** of the original features → good interpretability!

$$\mathbf{PC_1} = \mathbf{0.32(income)} + \mathbf{0.10(credit\ score)} + \mathbf{0.25(number\ of\ purchases)} + \mathbf{0.21(average\ purchase\ value)} + \mathbf{0.12(loyalty\ points)}$$

We might think of this 1st component as **"purchasing power"**

Interpretable feature *combinations*



Each principal component is a **linear combination** of the original features → good interpretability!

$$\mathbf{PC_2} = \mathbf{0.62(website\ visits) + 0.38(time\ on\ site)}$$

We might think of this 2nd component as **"website engagement"**

Plotting by PCs reveal natural clusters

- Age
- Income
- Education
- Credit score
- Number of purchases
- Average purchase value
- Website visits
- Time on site
- Number of returns
- Loyalty points
- ...

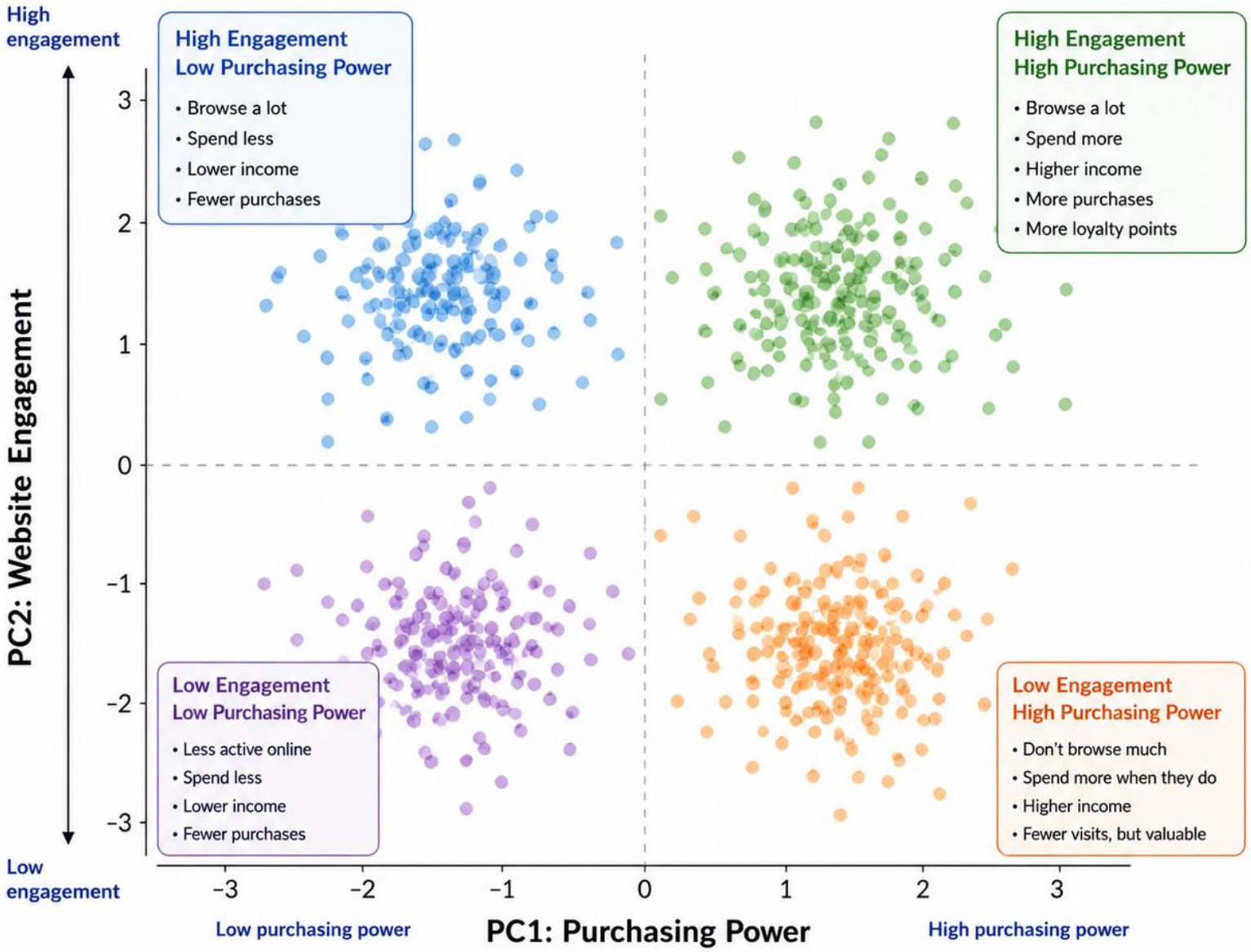
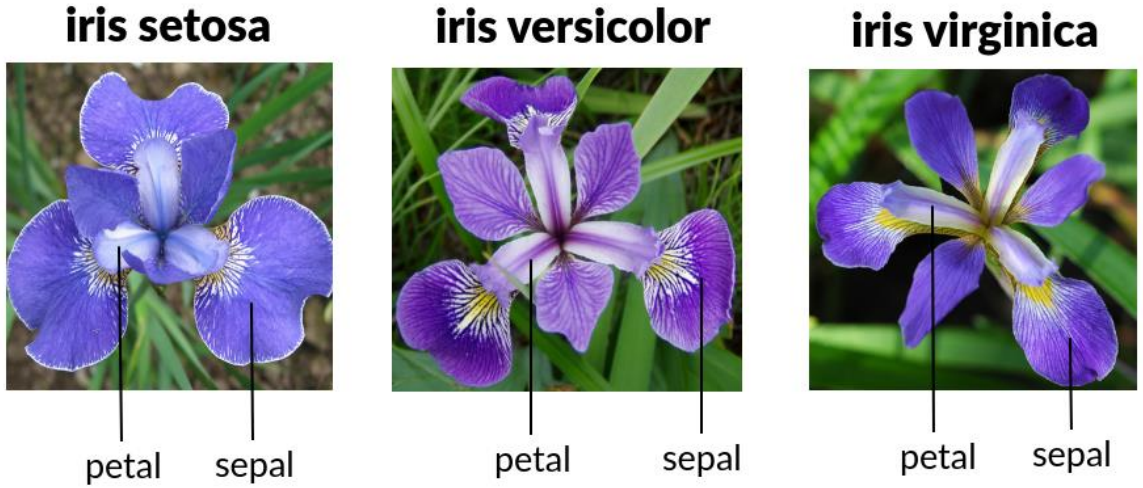
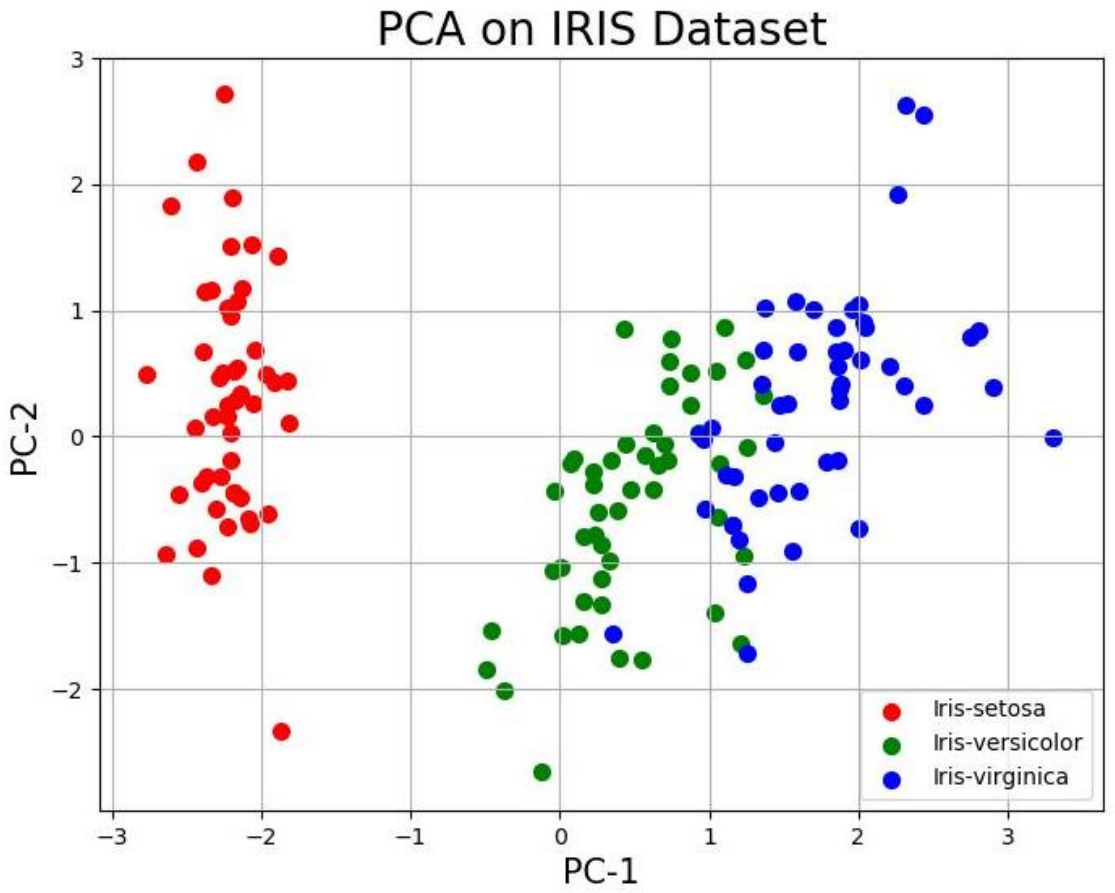


Image from ChatGPT

Clustering of samples with similar features

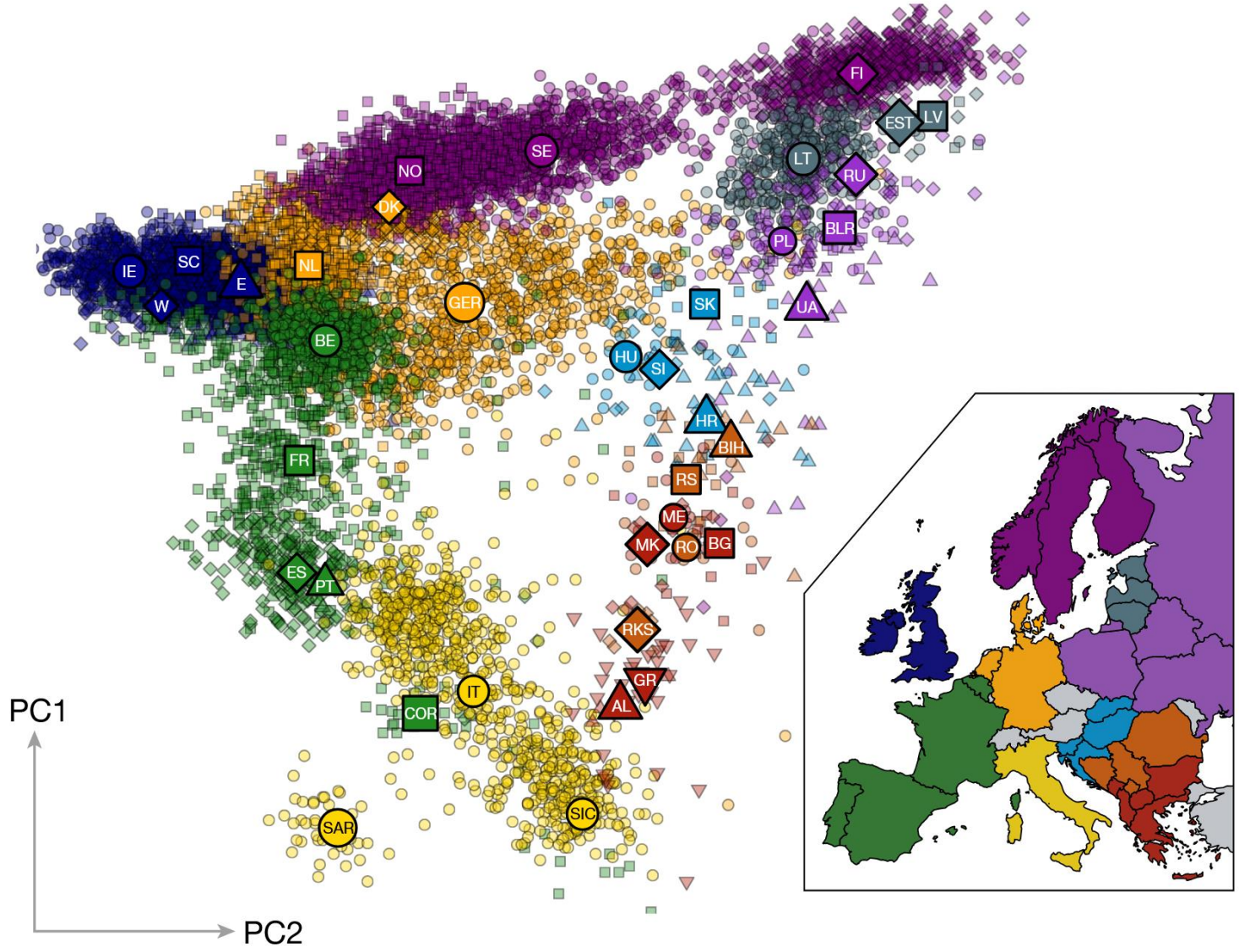




- Contains data on 150 flowers, with 50 from each species, described using **four descriptors:**
  - Sepal length (cm)
  - Sepal width (cm)
  - Petal length (cm)
  - Petal width (cm)

PCA leads to natural clustering of species!
Without ever seeing the labels

Clustering of samples with similar features

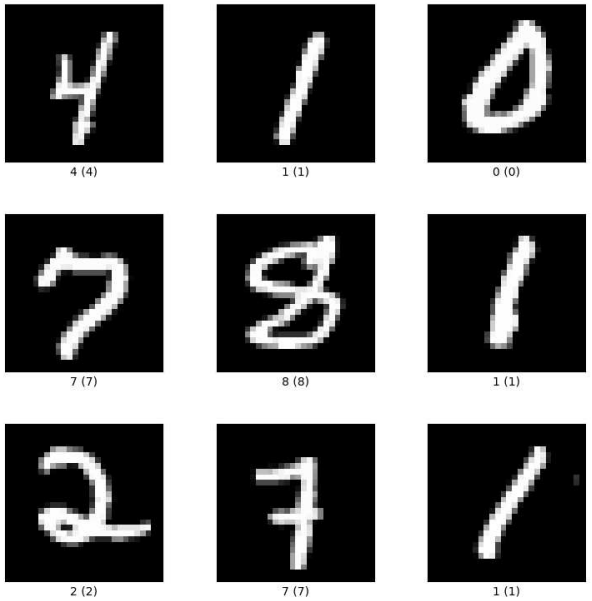


- Each individual is represented by hundreds of *thousands* of genetic variants (SNPs), encoded numerically according to which allele they carry.
- **PCA shows a clear separation of genetic variants by the people's country of origin!**

1000s → 2D

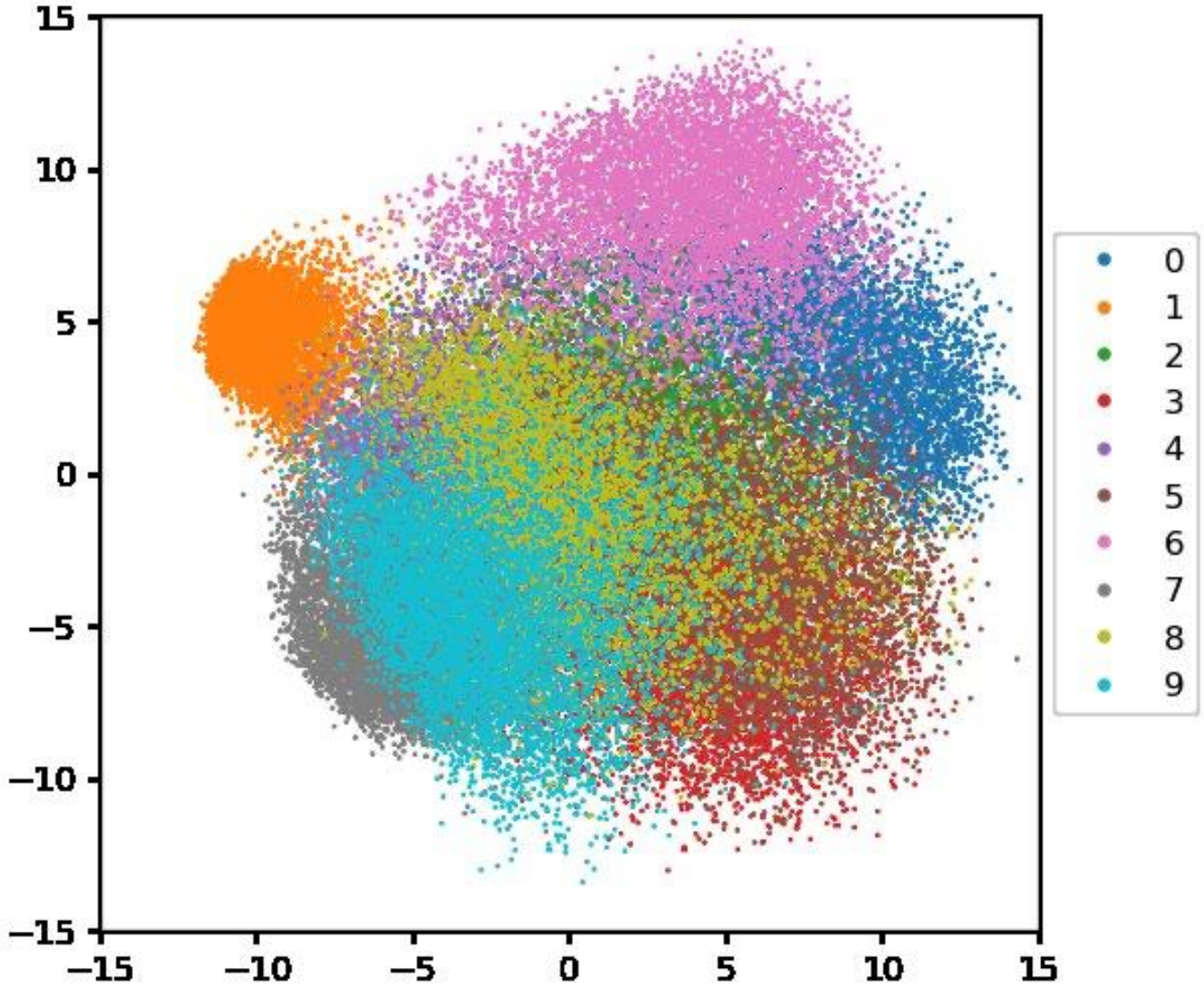
nature.com/articles/s41586-025-09437-6

Clustering of samples with similar features

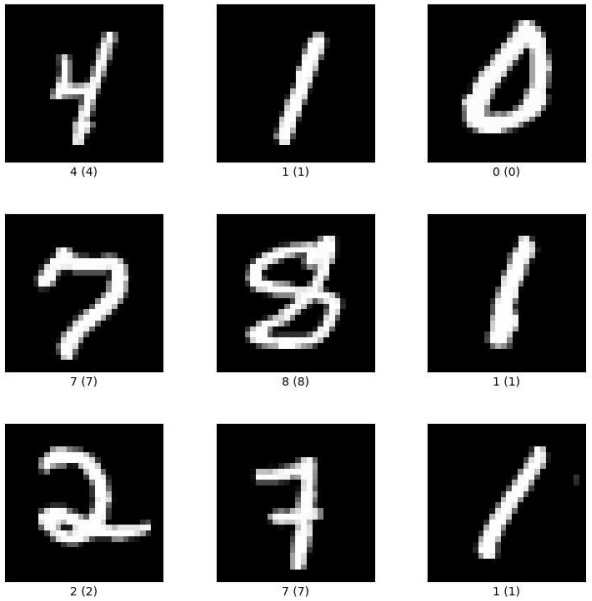


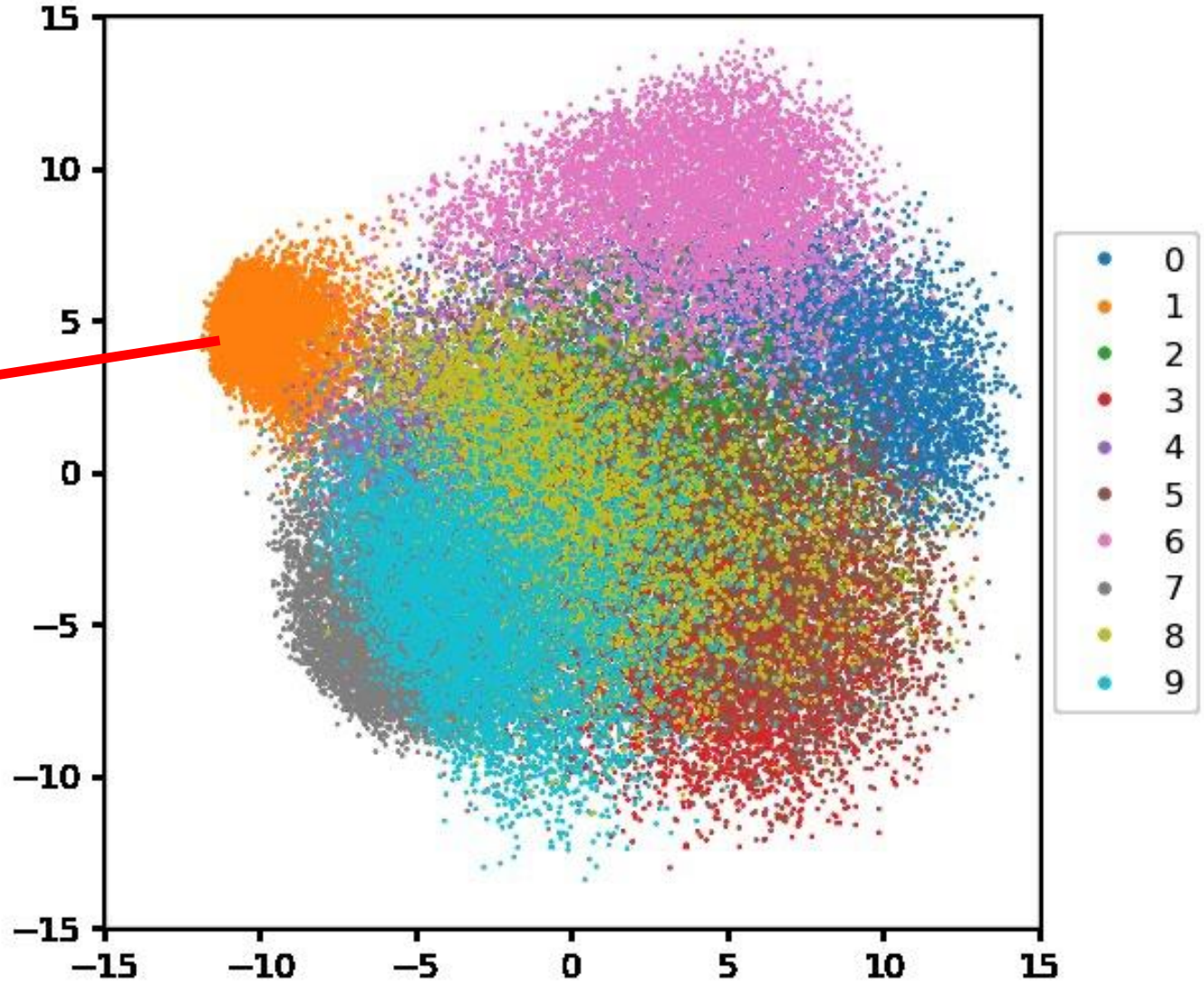
MNIST dataset: Thousands of grayscale images of handwritten digits, each with 784 pixels

PCA compresses these 784-pixel images into a 2D space, revealing *some* clustering by digits (again, this is done without any knowledge of their labels!) Still quite messy though…We'll revisit this problem later



Clustering of samples with similar features

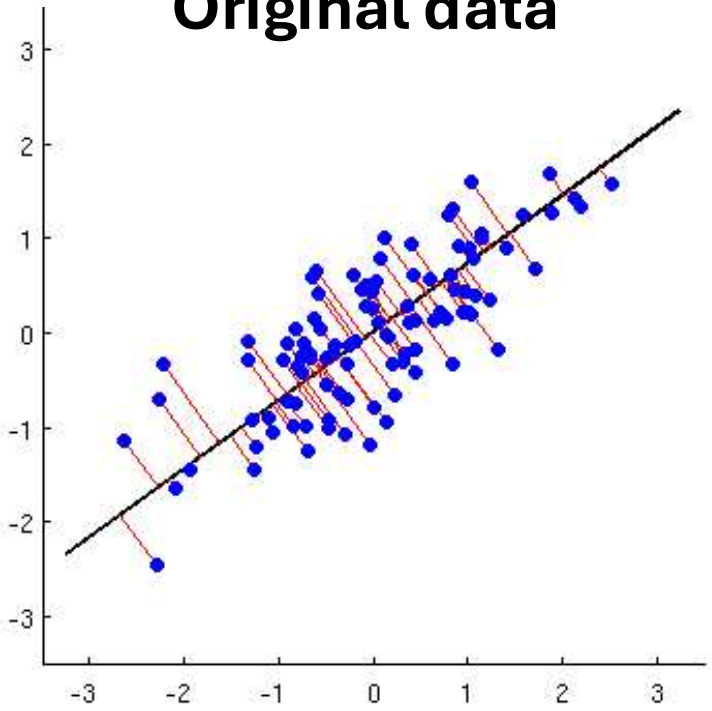




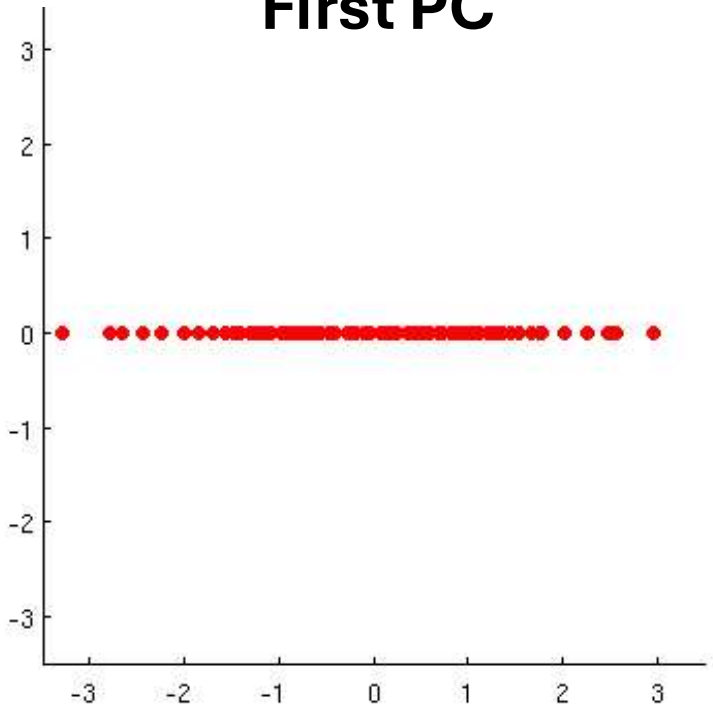
Question: Can we choose a point from the PCA projection and transform it back into its full original image?
(from 2D back to full-res)

Reconstruction from PCA

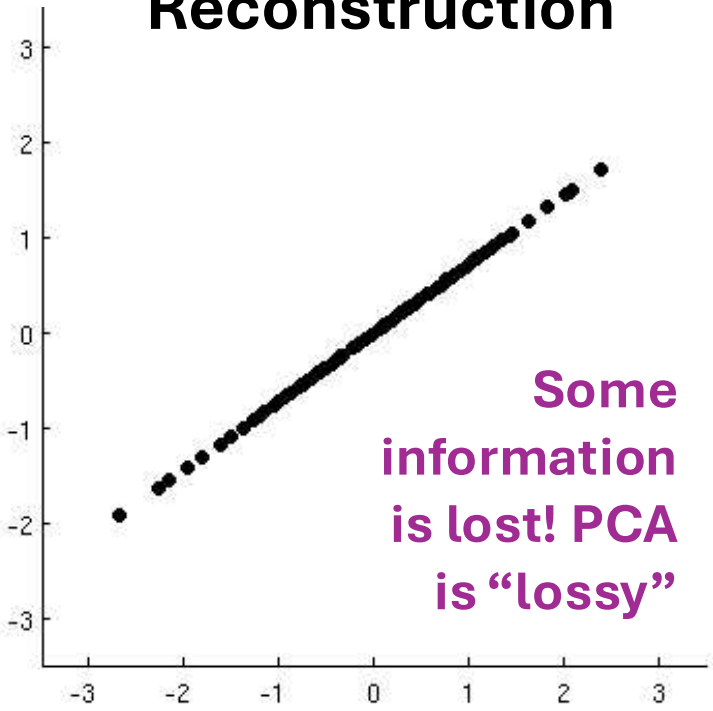
Original data



First PC



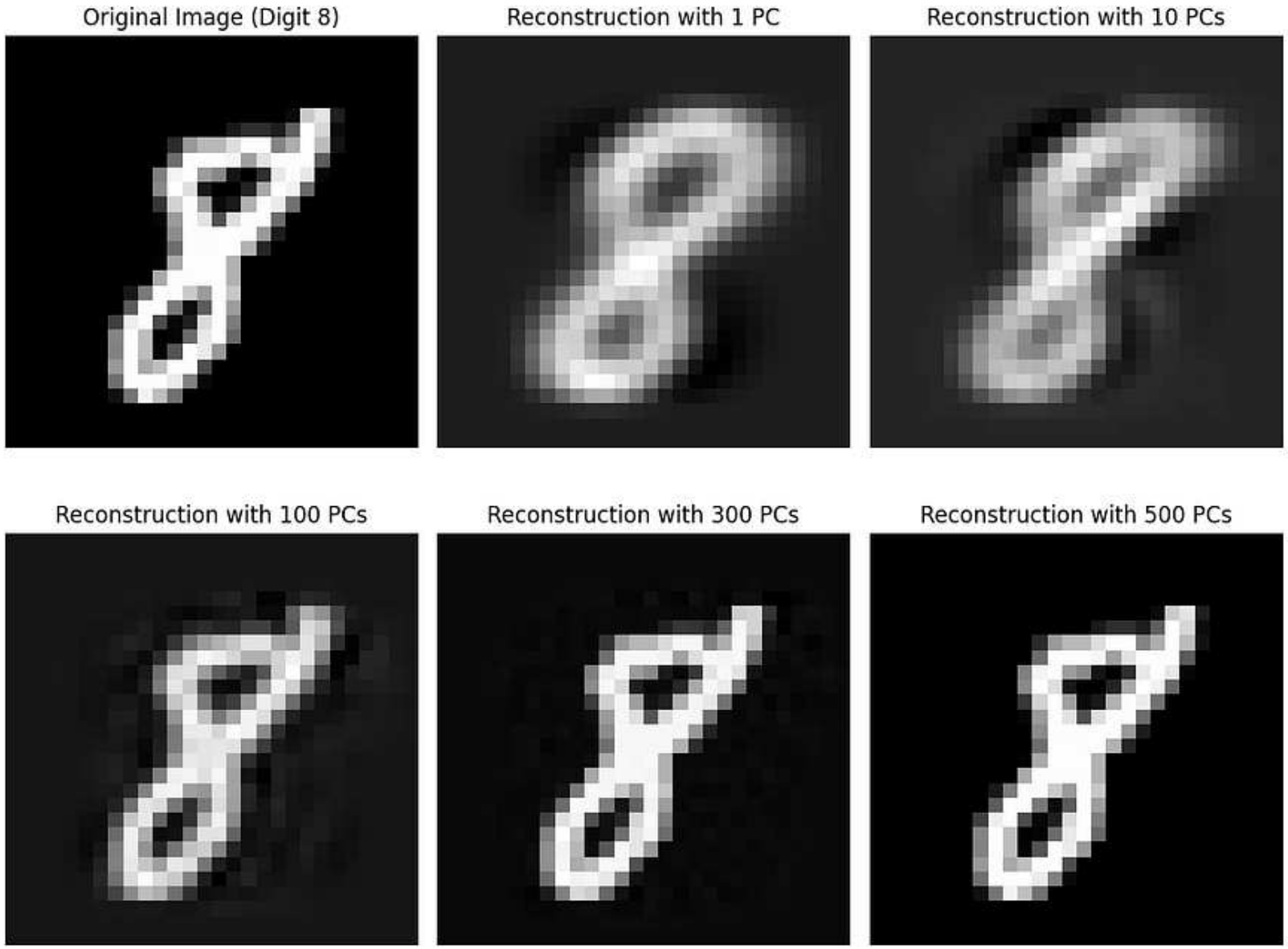
Reconstruction



Some information is lost! PCA is "lossy"

Every step in the PCA process involves **matrix multiplication**, which can be done **"in reverse"** to project the low-dimensional data back into the original dimensions (***reconstruction***), with some loss of information

Image compression and reconstruction

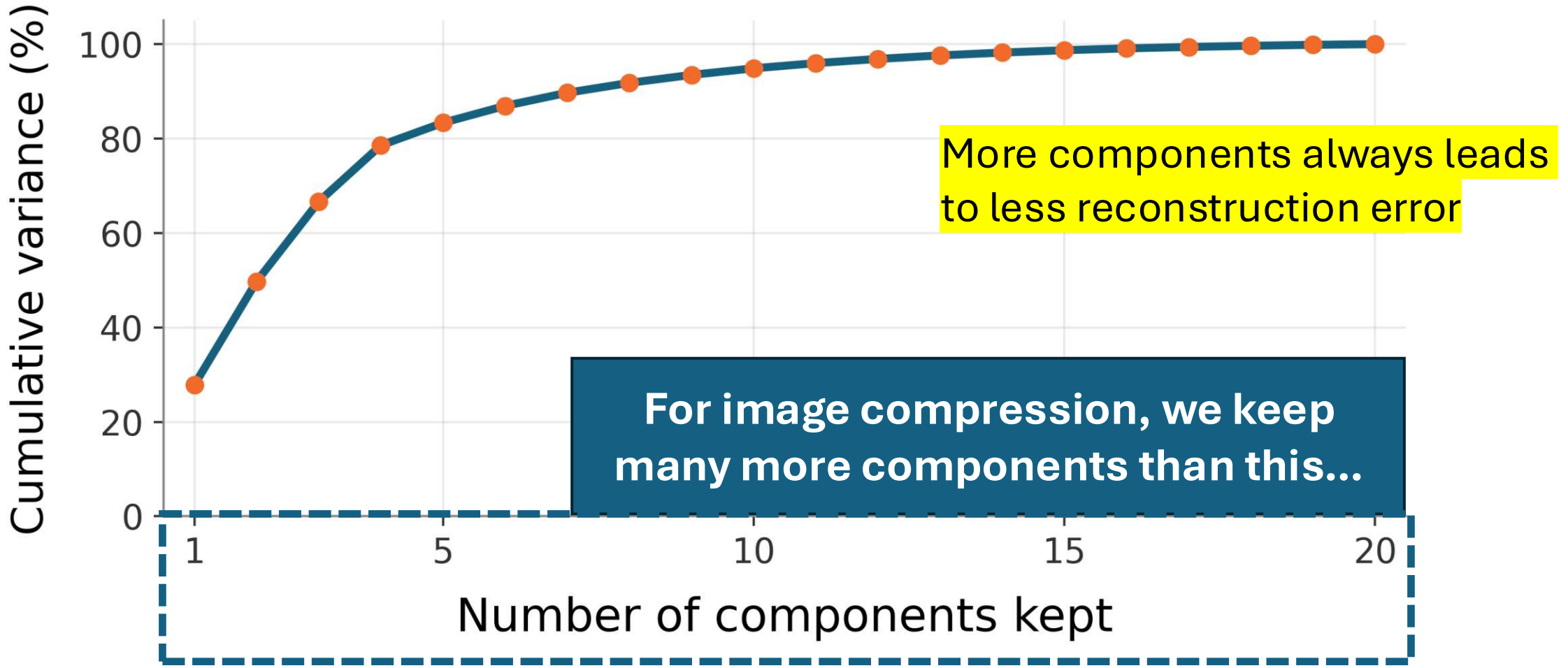


Early on, PCA was used as a compression algorithm, making it easier to store and transmit data…
But with some loss

If you want less information loss, keep more principal components!

Image compression and reconstruction

The loss is described by: 100% – cumulative variance



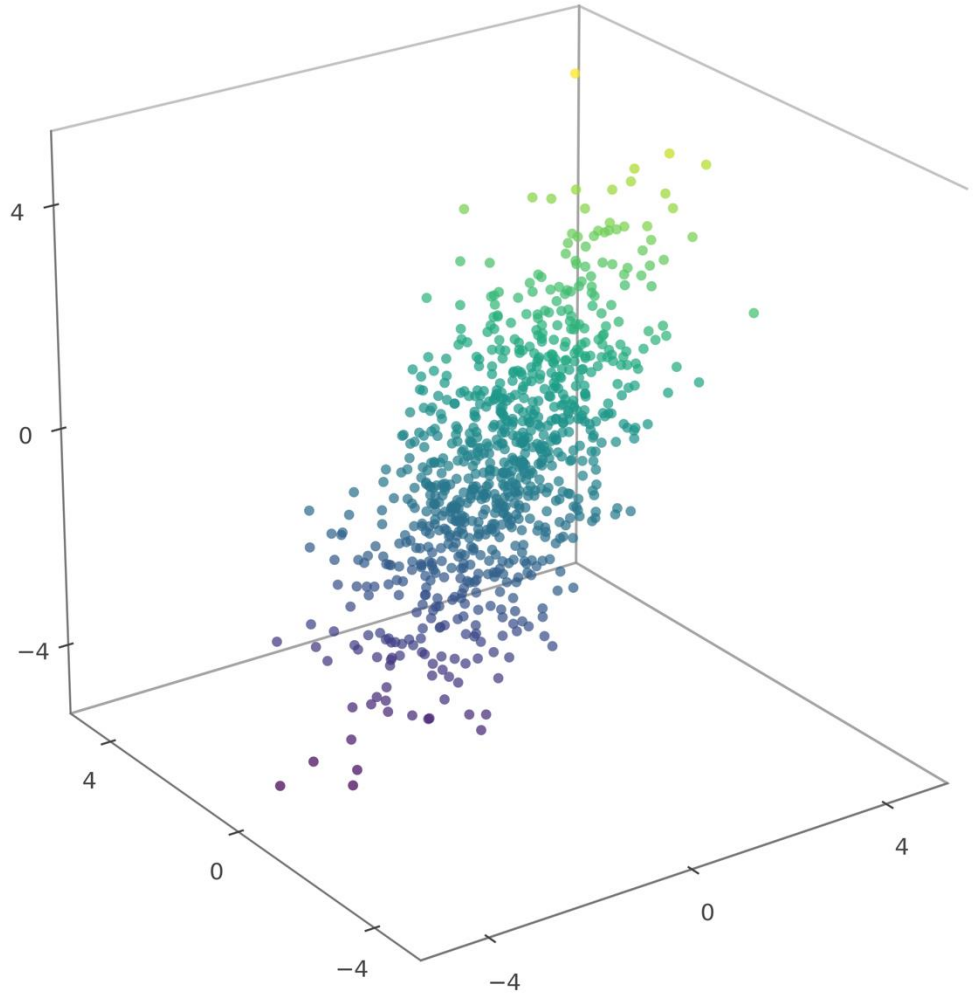
Questions?

t-SNE & UMAP

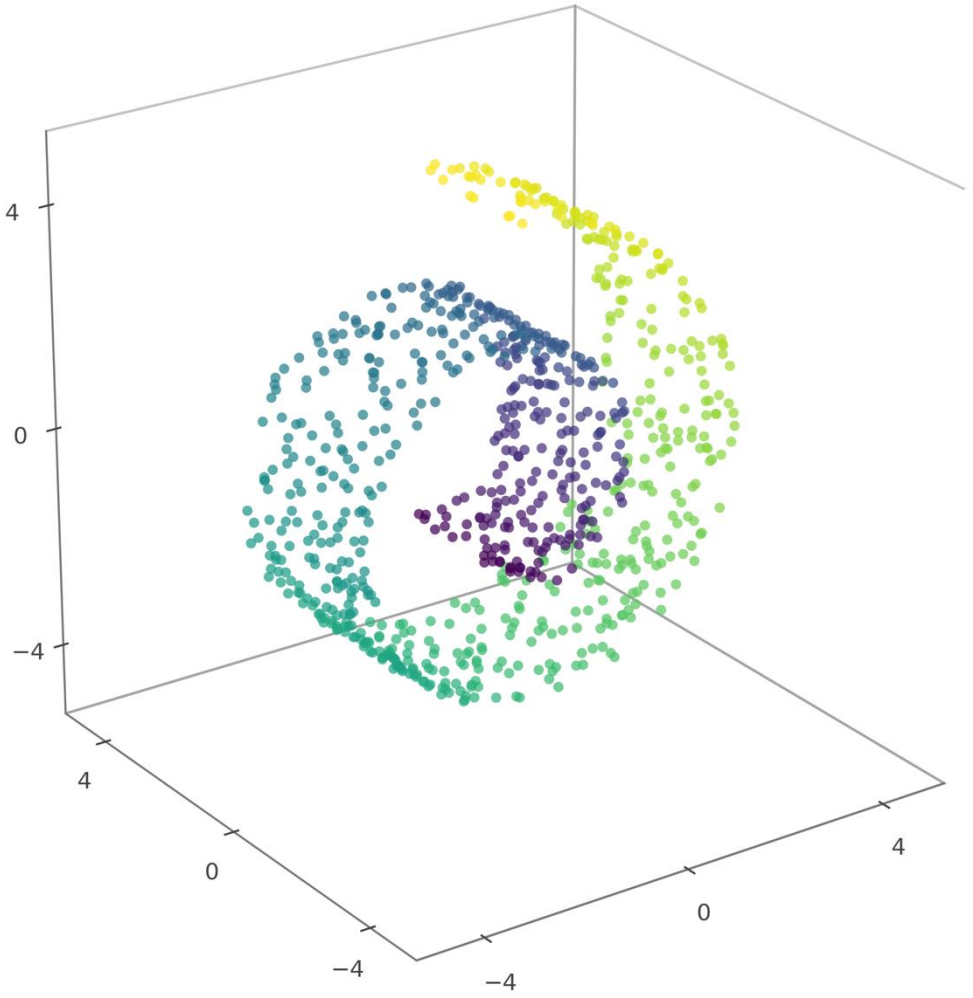
Nonlinear methods for dimensionality reduction & visualization

Non-linear data manifolds

Well suited for PCA ✓



***Not* well suited for PCA** ✗



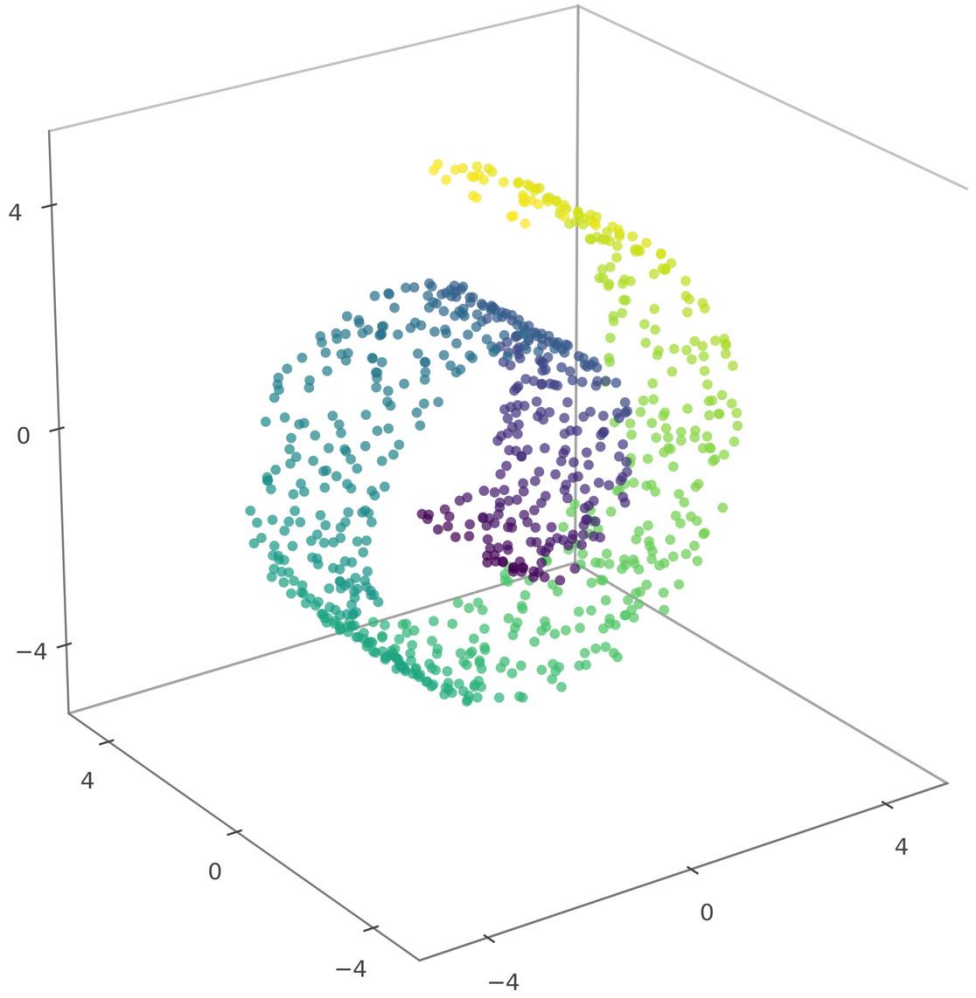
Non-linear data manifolds

- But clearly there is still some sort of structure in this data
- Imagine **projecting the data onto a curved surface** rather than a flat one

If we choose the right 2D surface (like this Swiss roll), the data can be compressed with very little information loss

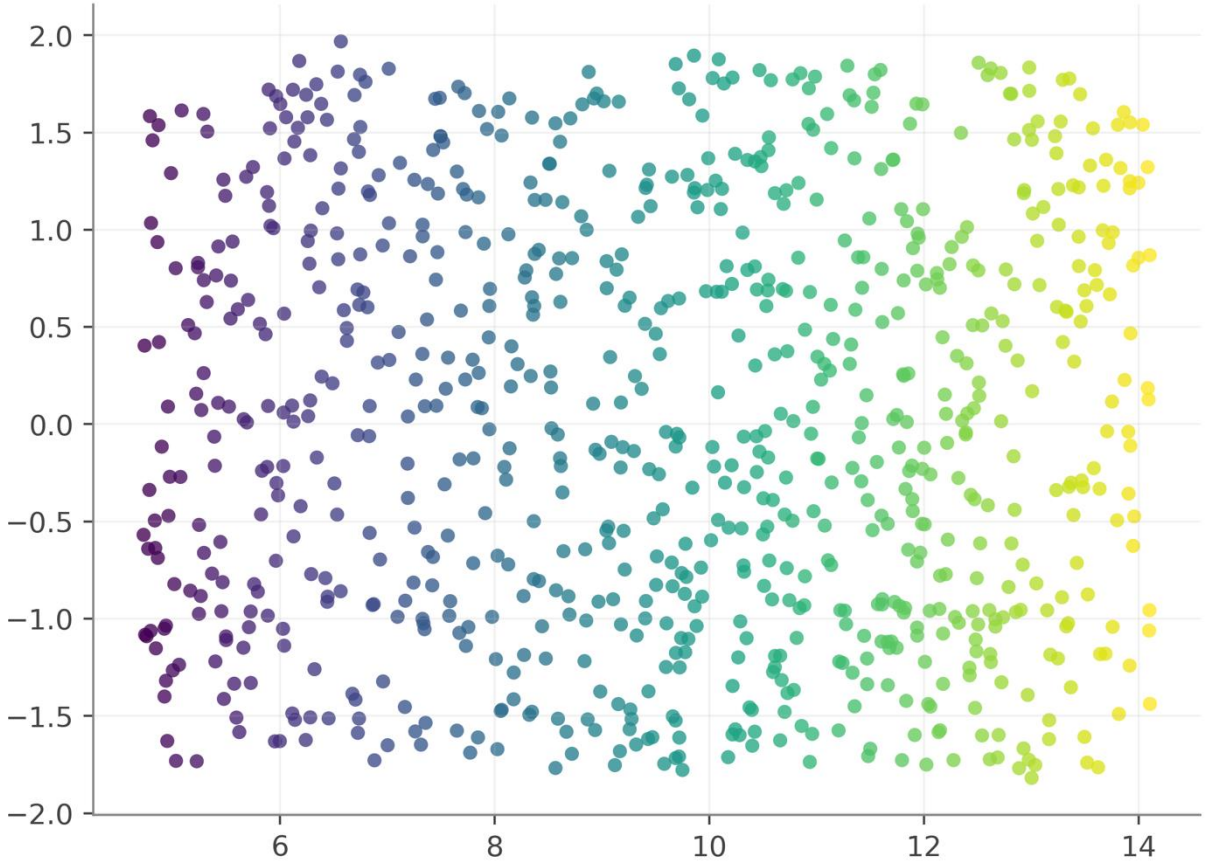


***Not* well suited for PCA** ✗

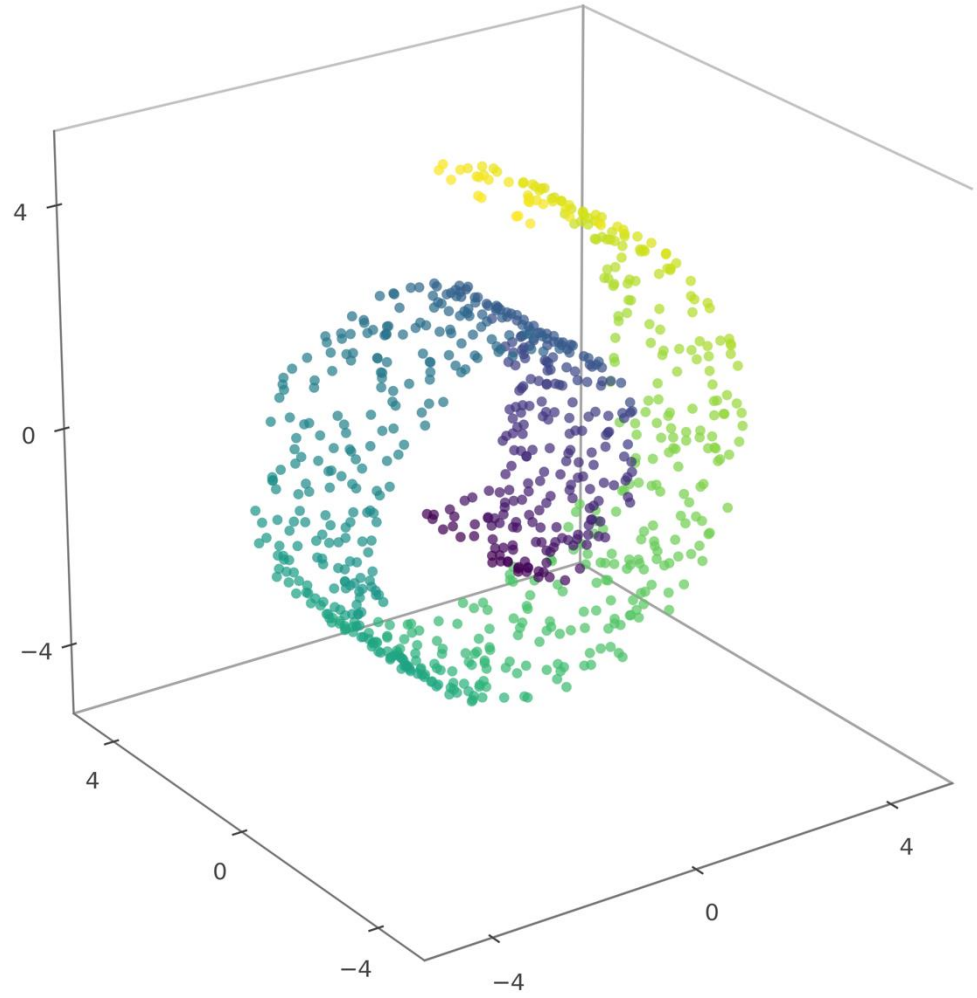


Non-linear data manifolds

Projected onto the “Swiss roll”

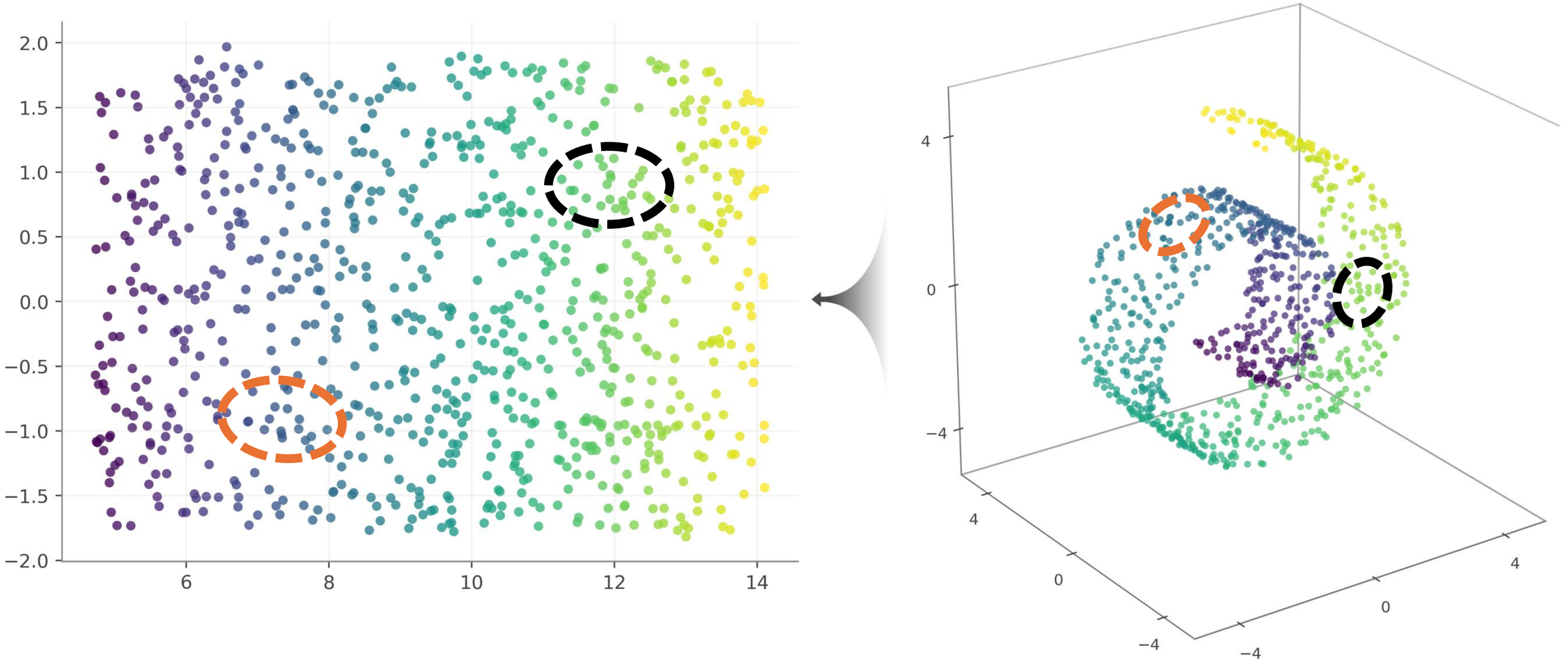


Not well suited for PCA



Non-linear data manifolds

Key idea: Neighborhoods are preserved before/after the transformation



t-SNE: Stochastic Neighbor Embedding

Consider two probability distributions

The original data, before transforming it into lower dimensions:

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

Distance between points i and j

Distance between i and all other points

t-SNE: Stochastic Neighbor Embedding

Consider two probability distributions

The original data, before transforming it into lower dimensions:

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

Distance between points i and j

Distance between i and all other points

The transformed data in lower dimensions:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

Similarity between points i and j **in lower-*d***

Similarity between i and all other points...

t-SNE: Stochastic Neighbor Embedding

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

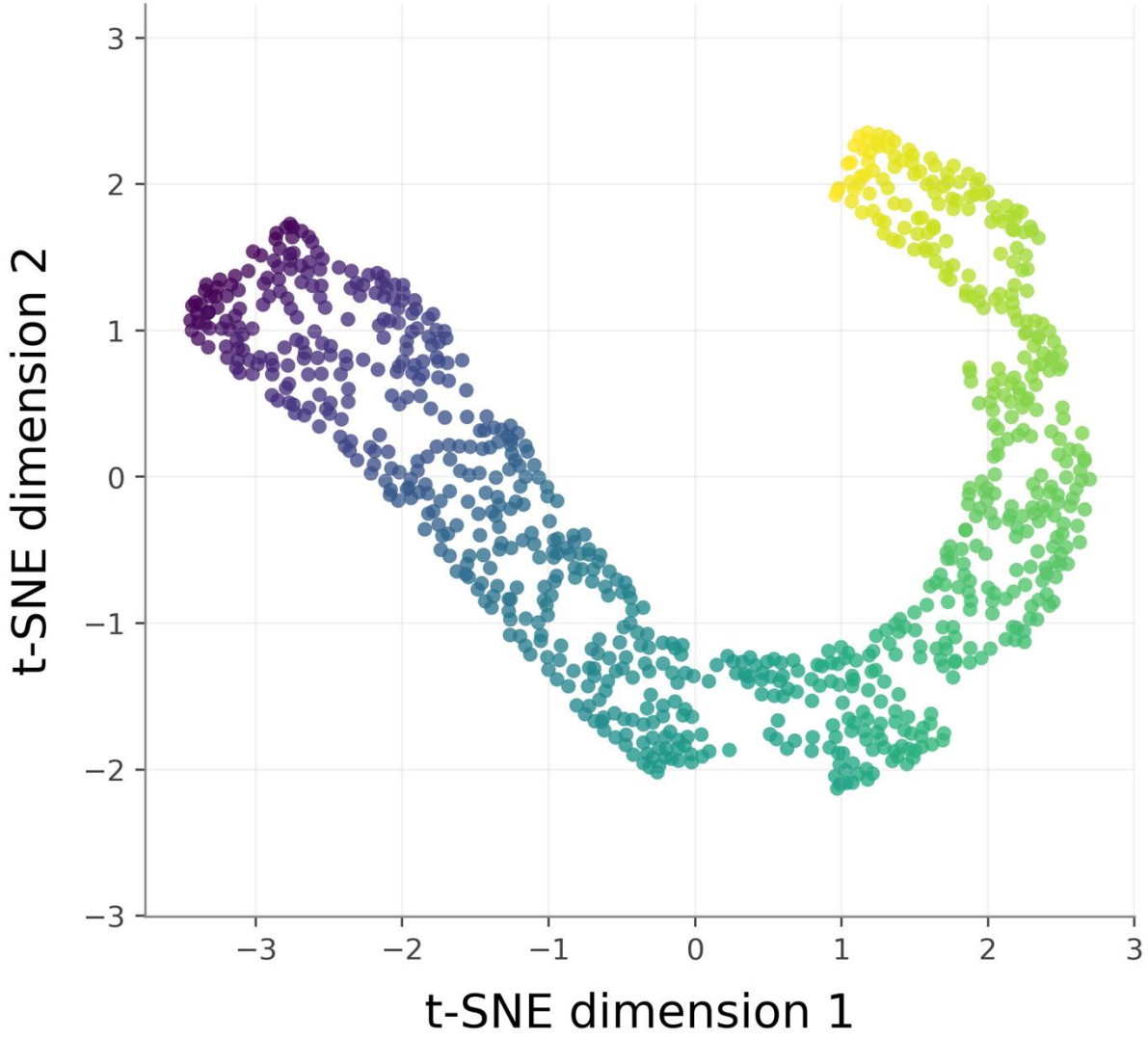
$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

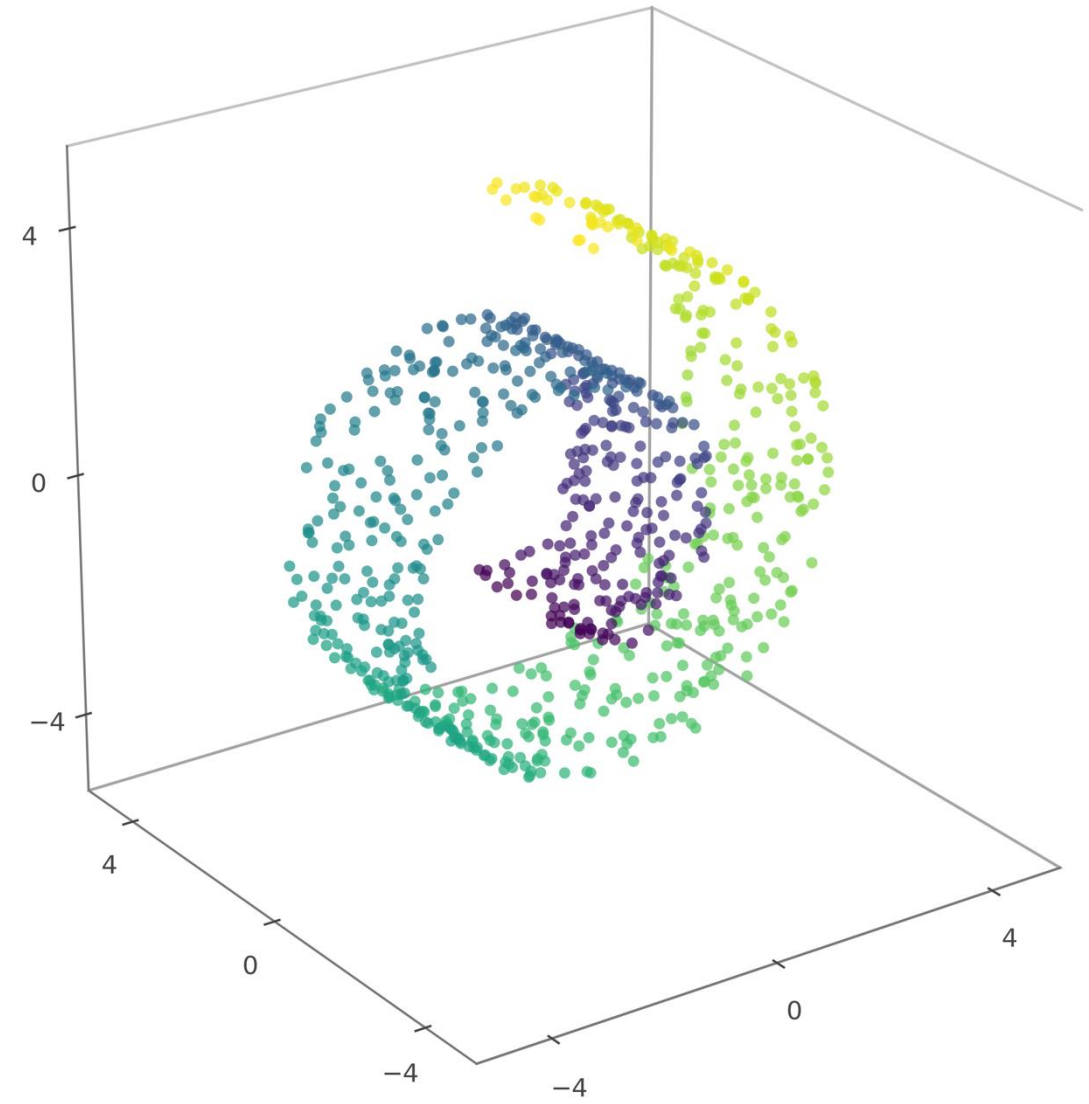
t-SNE attempts to learn a mapping that minimizes:

$$\sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

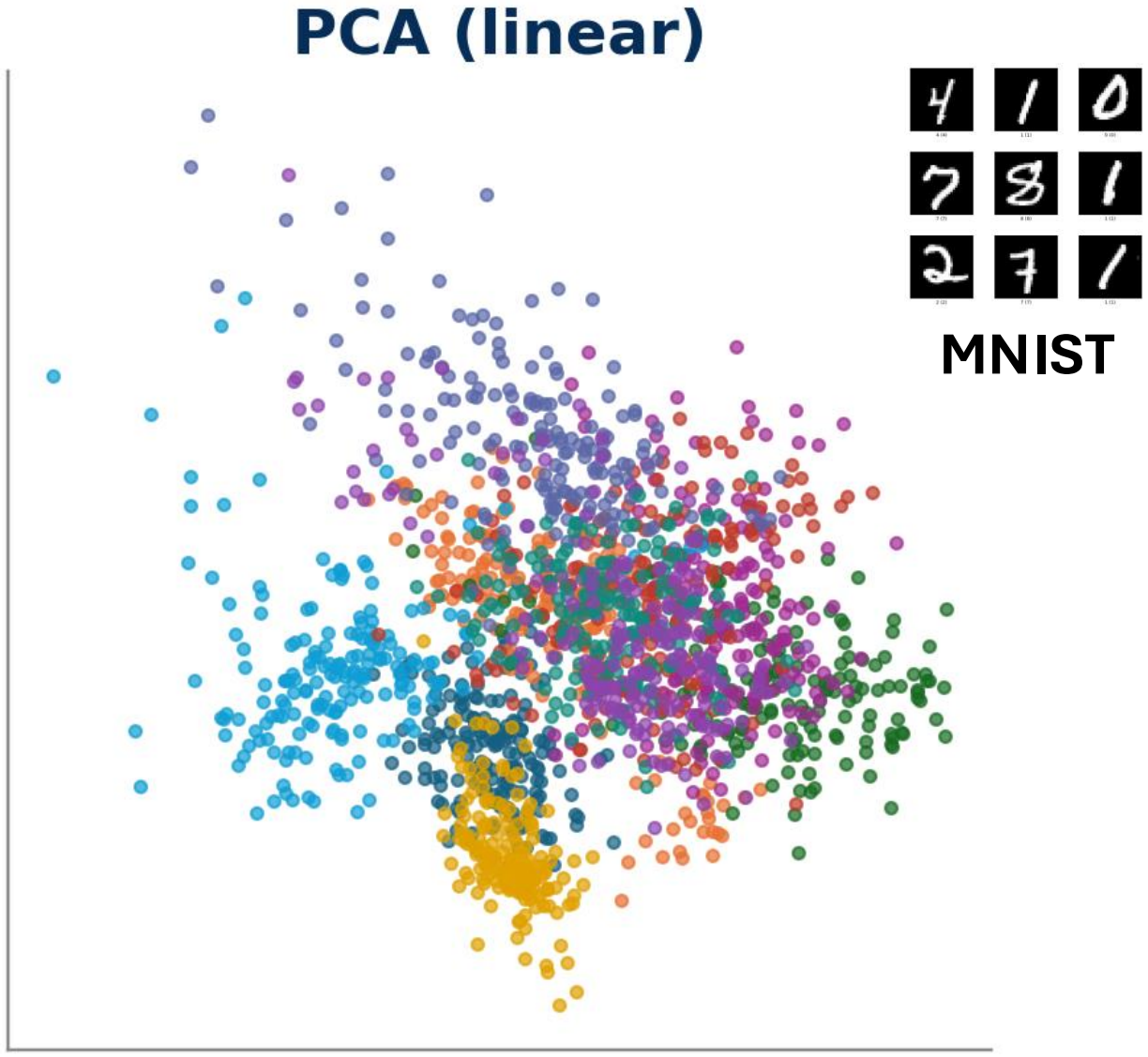
It forces p_{ij} and q_{ij} to be as similar as possible, since $\log(1) = 0$, which means neighbors are preserved

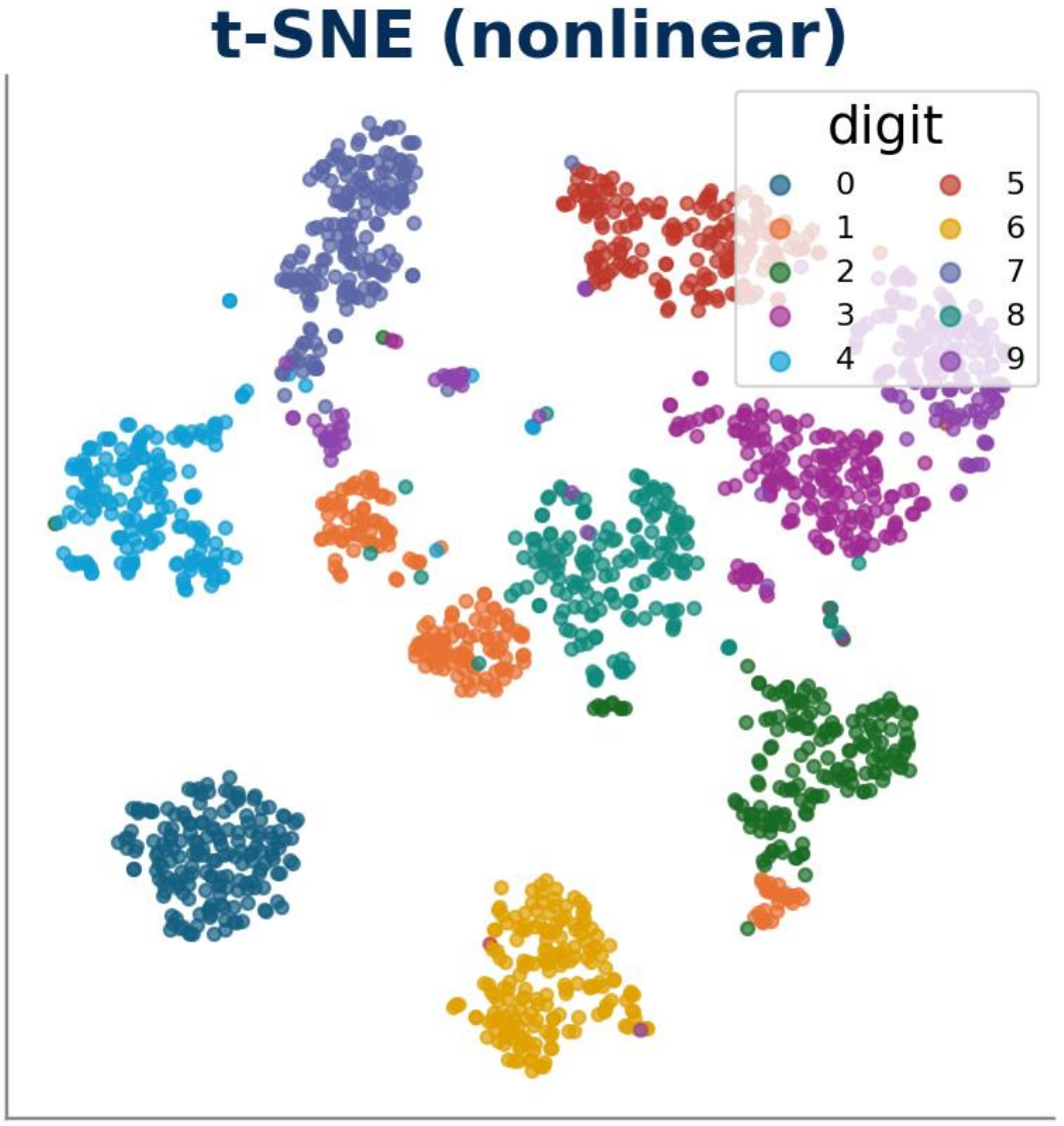
t-SNE: Stochastic Neighbor Embedding





t-SNE: Stochastic Neighbor Embedding





t-SNE: Stochastic Neighbor Embedding

σ determines the size of each neighborhood around which nearby points are considered "close"

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}$$

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|y_k - y_l\|^2)^{-1}}$$

t-SNE attempts to learn a mapping that minimizes:

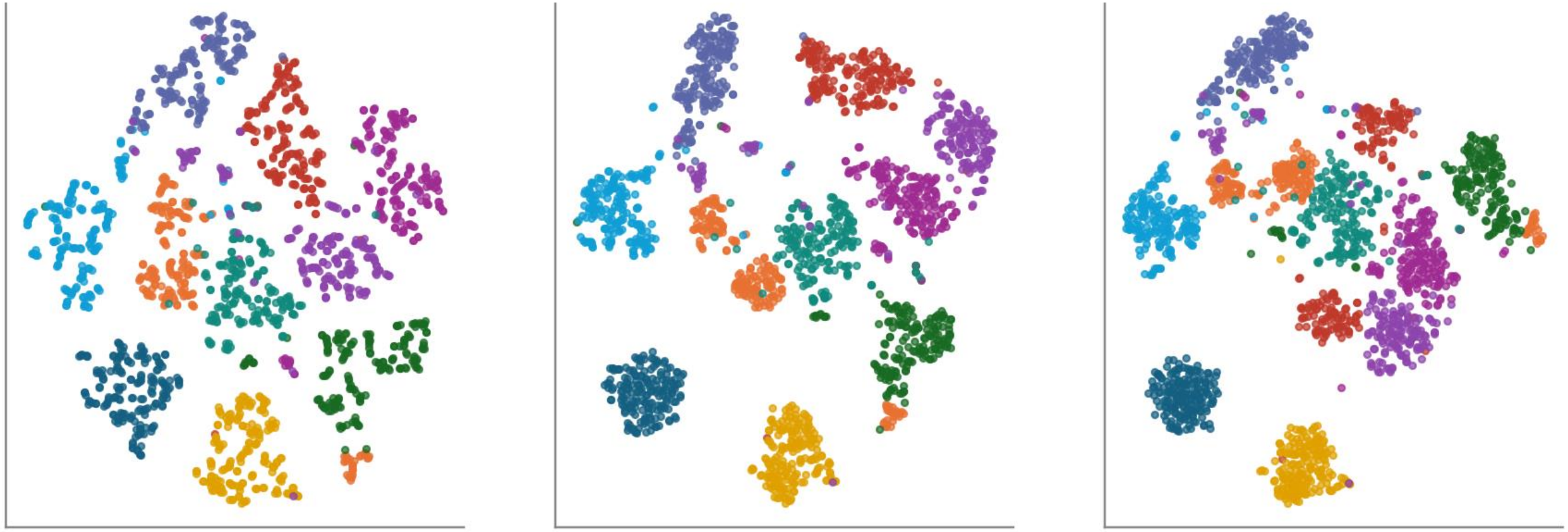
$$\sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

It forces p_{ij} and q_{ij} to be as similar as possible, since $\log(1) = 0$, which means neighbors are preserved

t-SNE: Stochastic Neighbor Embedding

Small $\sigma \rightarrow$ small neighborhood
Less close neighbors

Large $\sigma \rightarrow$ large neighborhood
More close neighbors



In reality, we choose **perplexity**, which affects σ, but there are no hard rules
Try several values and see how it affects the result

t-SNE: Stochastic Neighbor Embedding

Cluster SIZES mean nothing	DISTANCES between clusters mean little	Great for SEEING, not for modeling
t-SNE inflates dense clusters and shrinks sparse ones	far-apart blobs are not necessarily more different	use it to explore; don't feed its 2-D output to a model

Remember: **Cluster sizes** and **distance between clusters** are ***NOT*** **meaningful!**

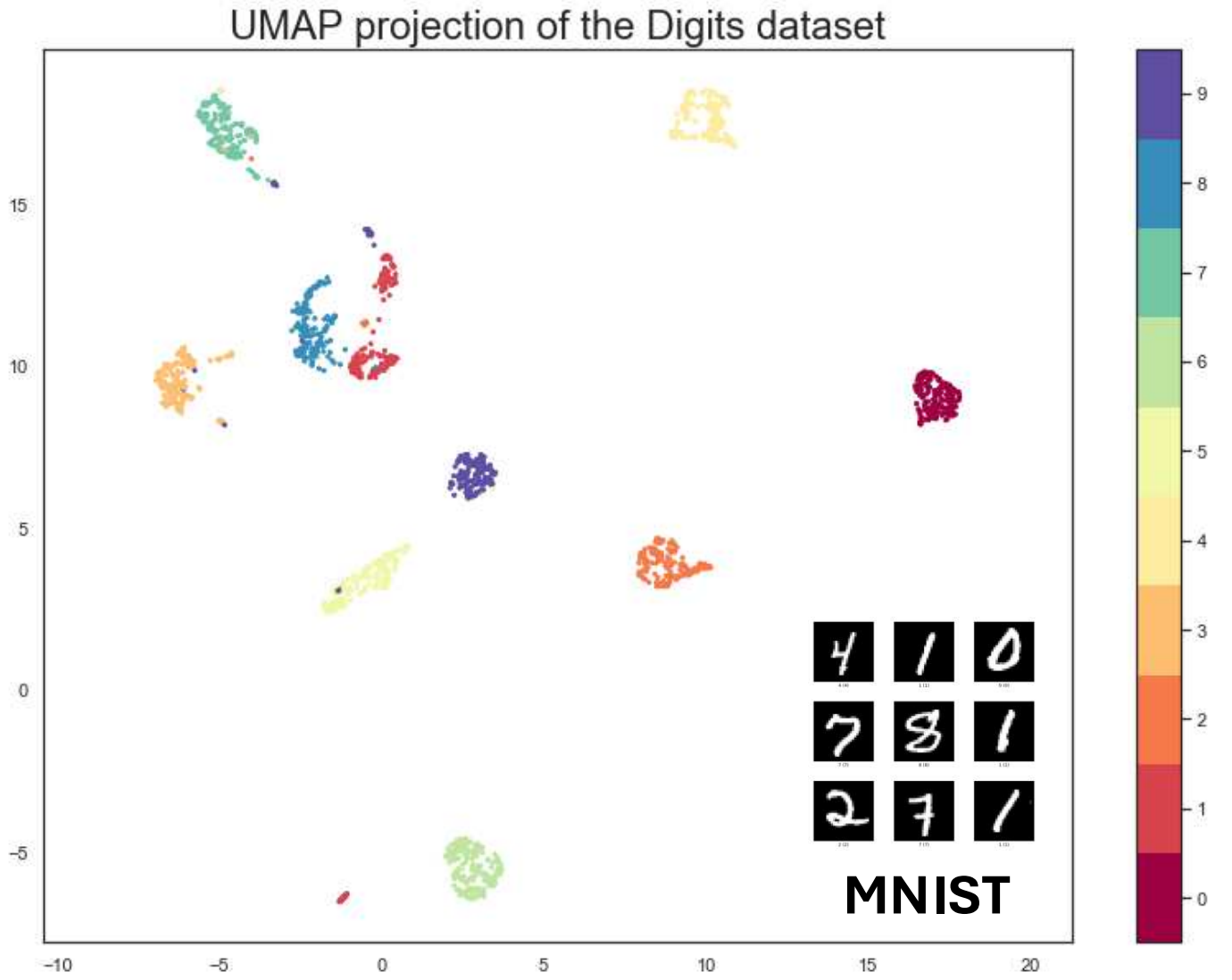
Do not over-interpret your t-SNE results. This is a highly stochastic process!

Very briefly: UMAP

The Uniform Manifold Approximation and Projection (UMAP) is another popular **nonlinear dimensionality reduction technique**

It constructs a graph connecting neighboring points in the original space, then optimizes a low-dim. embedding whose graph structure is as similar as possible to original one

Usually more effective than t-SNE (and faster) on large datasets



When to use each method

	PCA	t-SNE	UMAP
Type	linear	nonlinear	nonlinear
Best for	variance, compression	visualizing local clusters	visualizing local + some global
Speed	very fast	slow	fast
Deterministic?	yes	no (random)	no (random)

- Use PCA if you want to good interpretability, compression, or feature downselection
- Use t-SNE or UMAP if all you care about is data visualization

Questions?

What to know for HW + Exams

- Difference between **supervised *vs.* unsupervised learning**
- How to do **PCA – know the math!**
- What does the **covariance matrix** tell you about your data?
- The **meaning of eigenvalues and proportion of variance**
- When to use PCA *vs.* t-SNE or UMAP (math not needed)