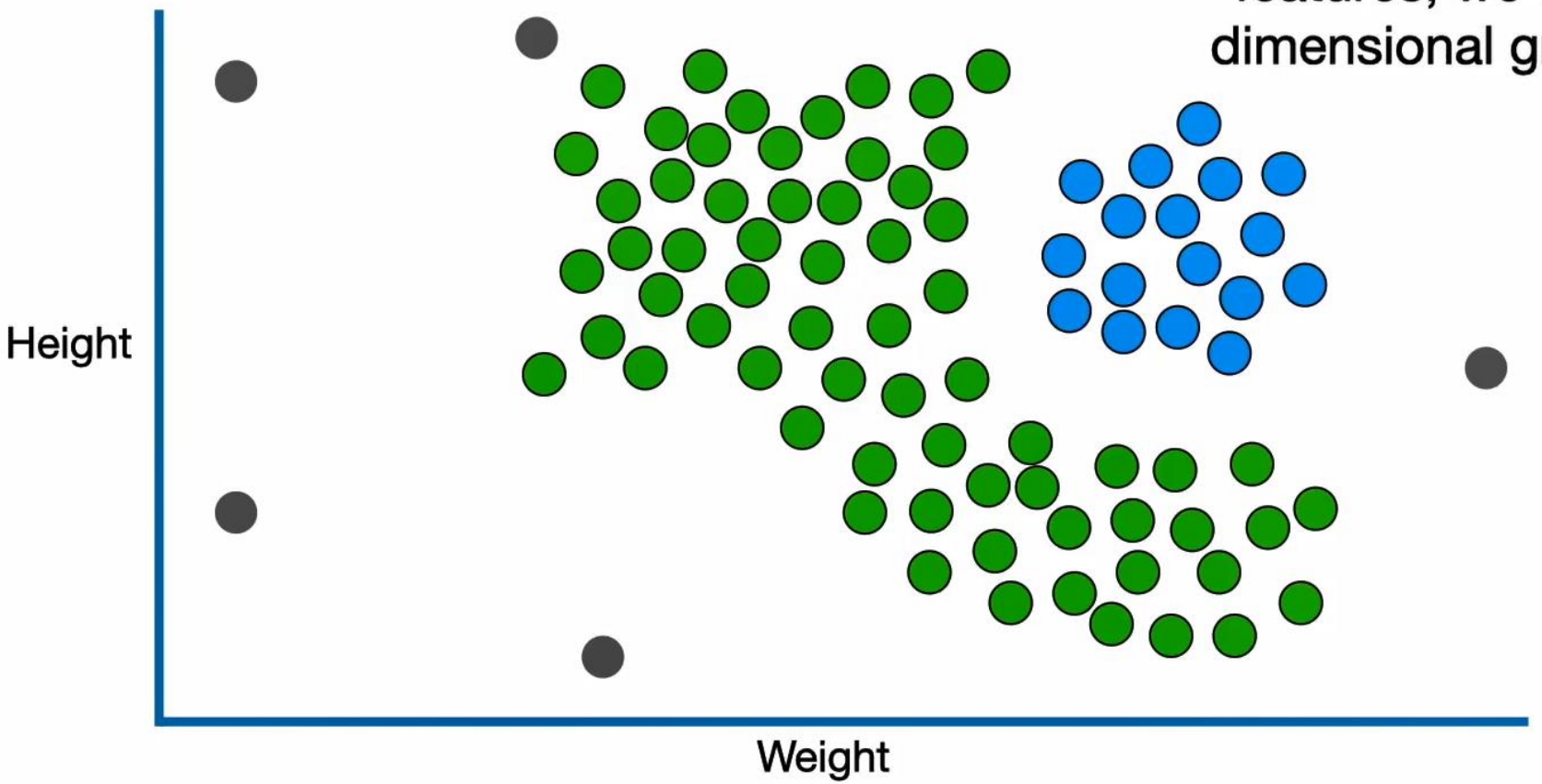
¿What is Dimensionality Reduction?

A lot of Machine Learning problems consist of hundreds to thousands of features. This issue is known as Dimensionality Reduction as the process of reducing the number of random variables under consideration by obtaining a set of principal variables.

So looking at data can be very useful, and generally speaking, is one of the first steps in any data analysis.

However, if we wanted to include a lot more features, we'd need to draw a **4** or more-dimensional graph, and that's *not possible*.



Well, one good option is **Principal Component Analysis (PCA)**…

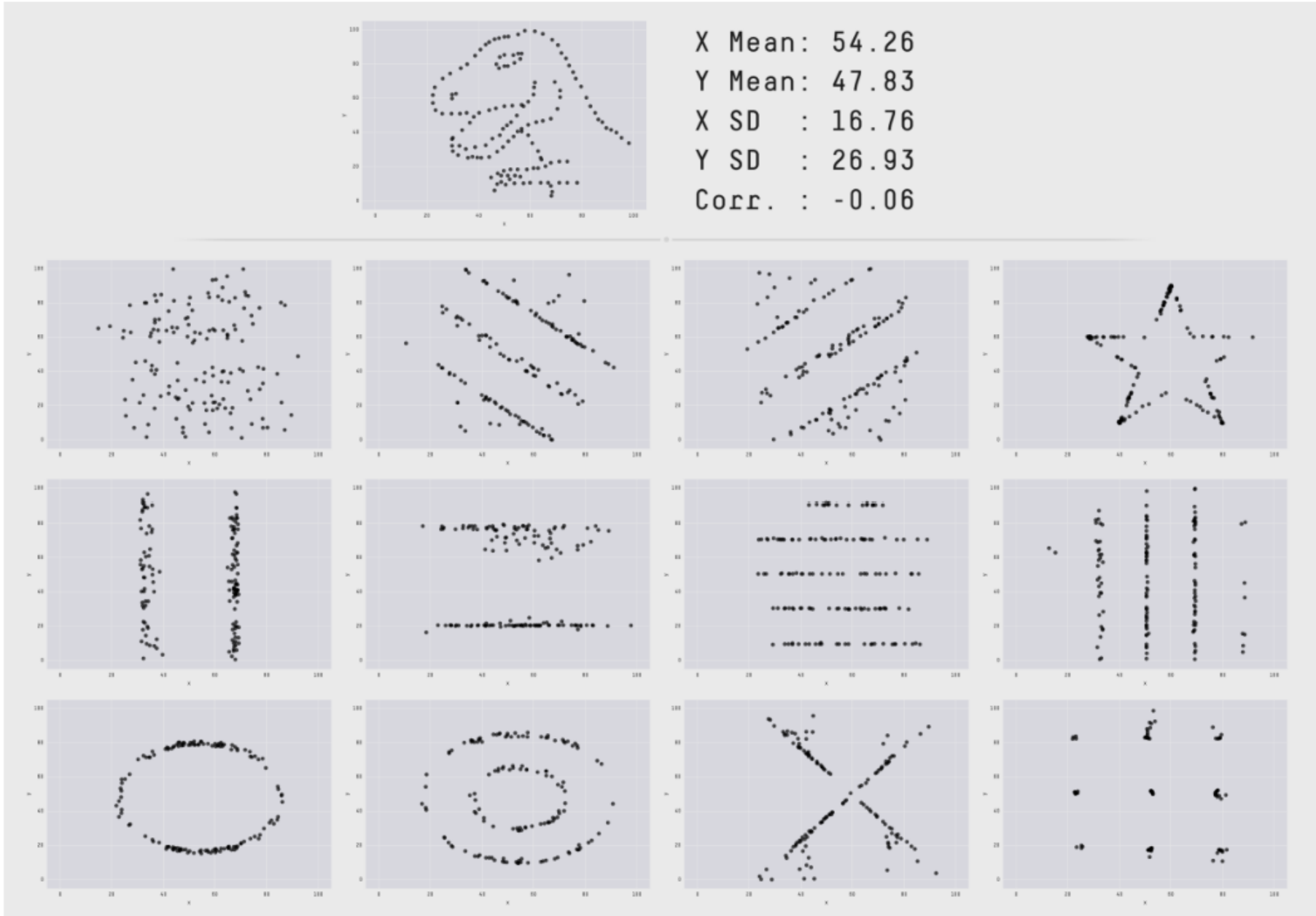
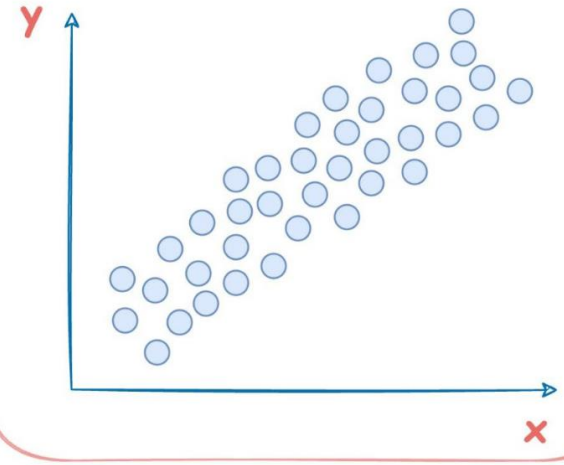


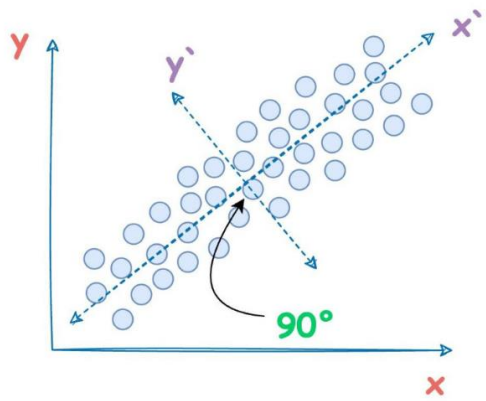
Fig. 1: Same Stats, Different Graphs

High dimensional data



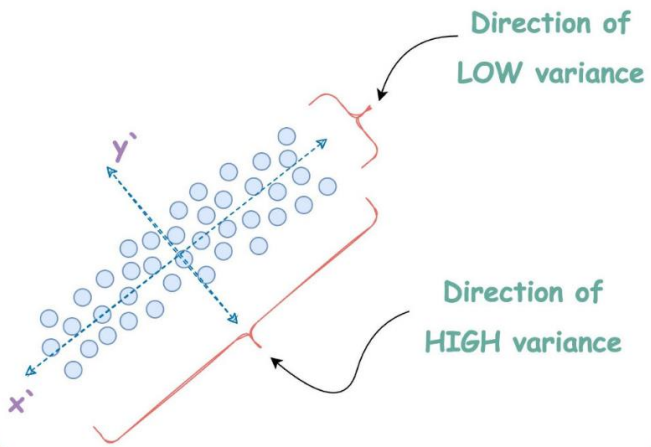
Step 1

Determine a system of UNCORRELATED axes (x`, y`) to represent the data



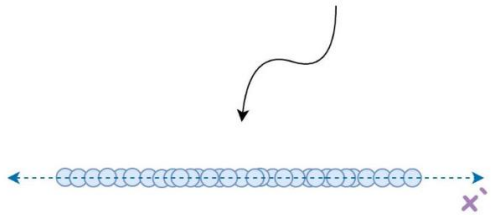
Step 2

Find the variance of data along all uncorrelated axes

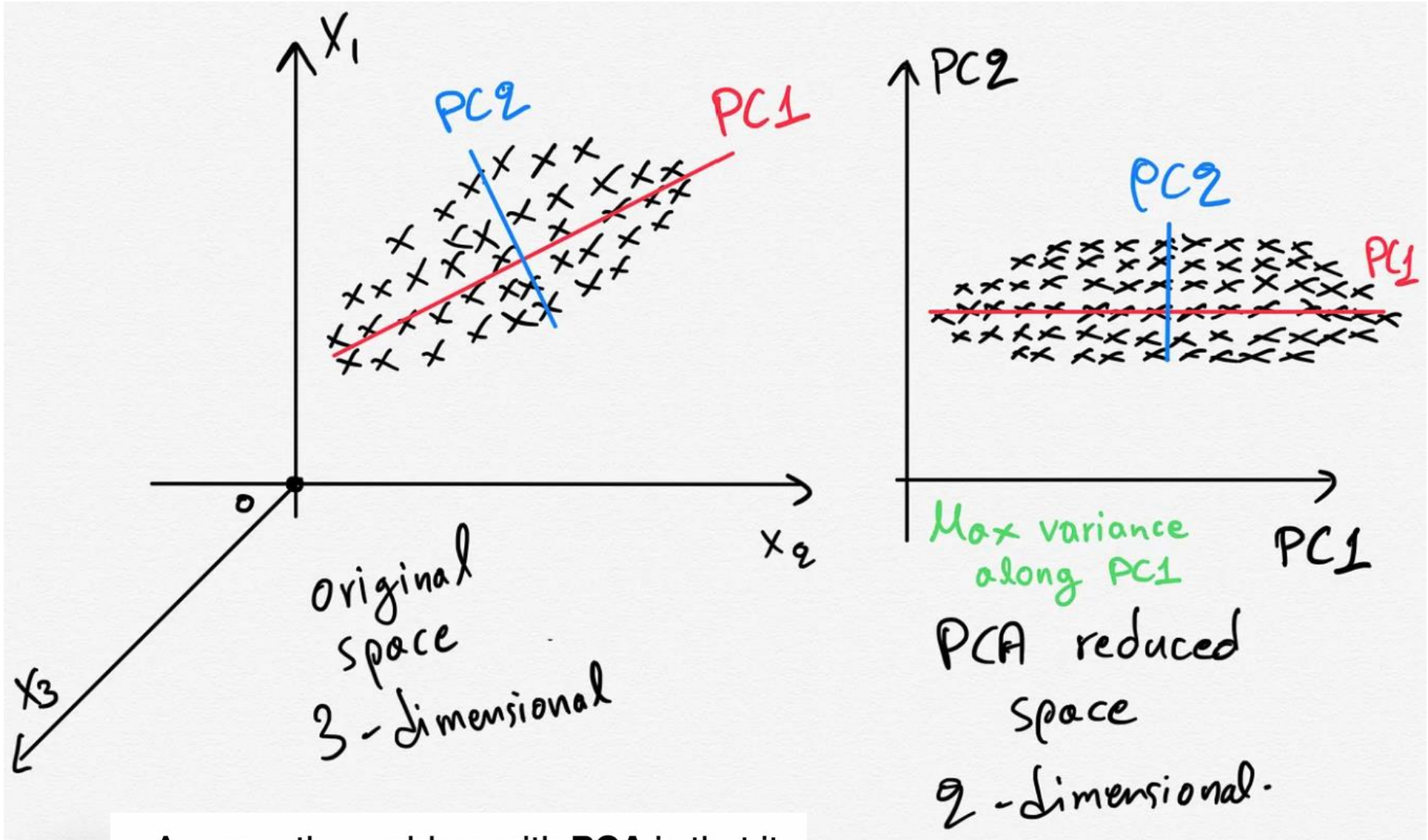


Step 3

Discard directions of LOW variance (y`) and project the data along directions of HIGH variance (x`)



FINAL DATA WITH REDUCED DIMENSIONS



Anyway, the problem with **PCA** is that it only works well when the first **2 Principal Components** account for most of the variation in the data.

Principal Component Analysis (PCA)

- We assume that the attributes follow gaussian distributions.
- Global variance $\sum_{d=1}^{D} Var(x_d) = \sum_{d=1}^{D} Var(y_d)$
- The data is projected onto a set of orthogonal dimensions (components) that are linear combinations of the original attributes.

$$y_i = \sum_{d=1}^{D} w_{id} x_d, \;\; ; \;\; y \in \mathbb{R}^{N \times D} \quad \forall i, j \;\; w_i \perp w_j$$

- There are several ways to derive the method to calculate PCA
- We are going to explain the method based on the maximization of variance
- We will consider a data set $X = \{x_1, \ldots, x_N\}$ $x_n \in \mathbb{R}^D$ with that has been standardized (all features are centered and have unit variance), with a covariance defined as:

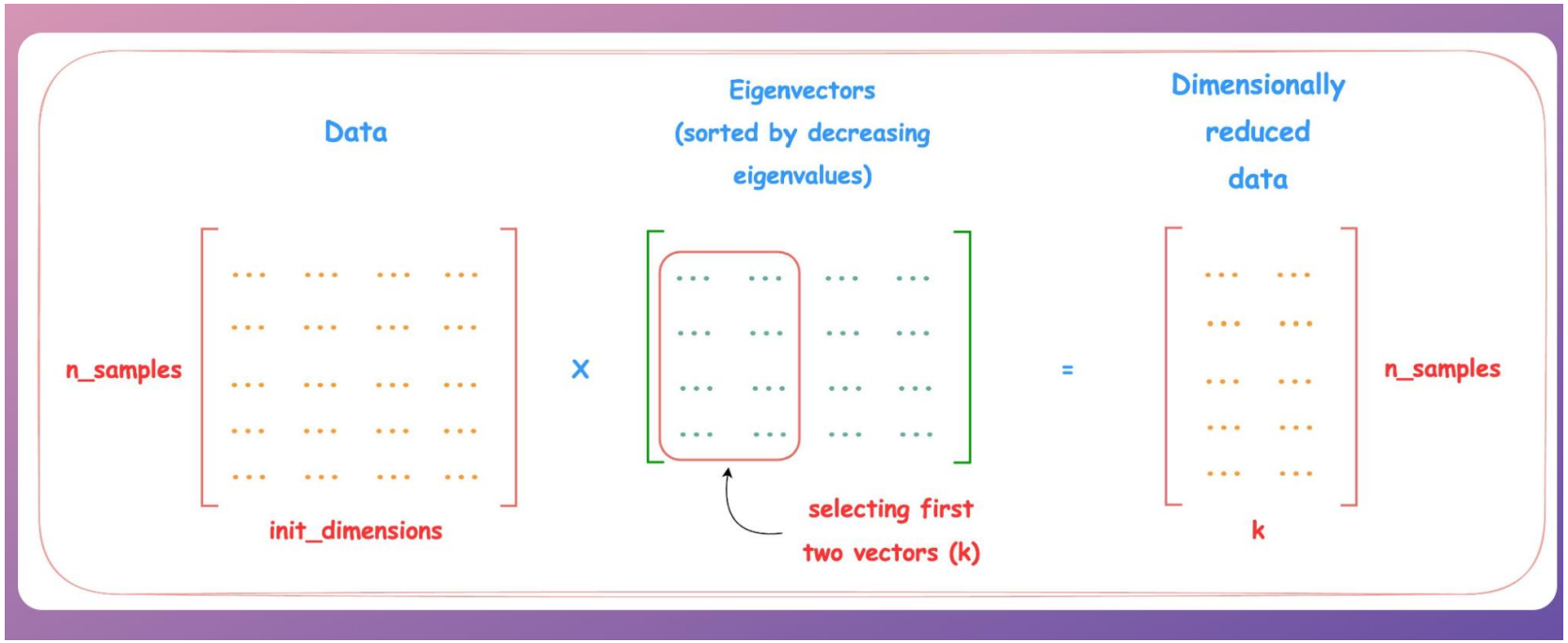
$$\Sigma = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^\top = \frac{1}{N} X X^\top, \quad \Sigma \in \mathbb{R}^{D \times D}$$

Eigenvector and Eigenvalue

If T is a linear transformation from a vector space V and v is a nonzero vector in V, then v is an eigenvector of T if $T(v)$ is a scalar multiple of v. This can be written as

$$T(\mathbf{v}) = \lambda\mathbf{v}$$

where λ is a scalar, known as the eigenvalue, associated with v.



- This means that to maximize the variance of the first vector b_1 we must choose the largest eigenvalue of the covariance matrix and use its first eigenvector (**first principal component**)

- To find the rest of the components we can proceed iteratively subtracting the effect of the previous $m - 1$ components and imposing that the new component must be orthogonal

- In the end, we obtain that the components correspond to the eigenvectors of the covariance matrix and the sum of the eigenvalues is the total variance

$$\Sigma = B\Lambda B^{\top}$$

- With Λ a diagonal matrix with the eigenvalues and B the eigenvectors

PCA: Dimensionality Reduction

- Since the eigenvalues correspond to the total variance, we can calculate the proportion that each component represents

$$\%Var(b_i) = \frac{\lambda_i}{\sum_{d=1}^{D} \lambda_d}$$

- We can decide a threshold on how much variance we want to keep (for example, 90%) and discard the remaining components

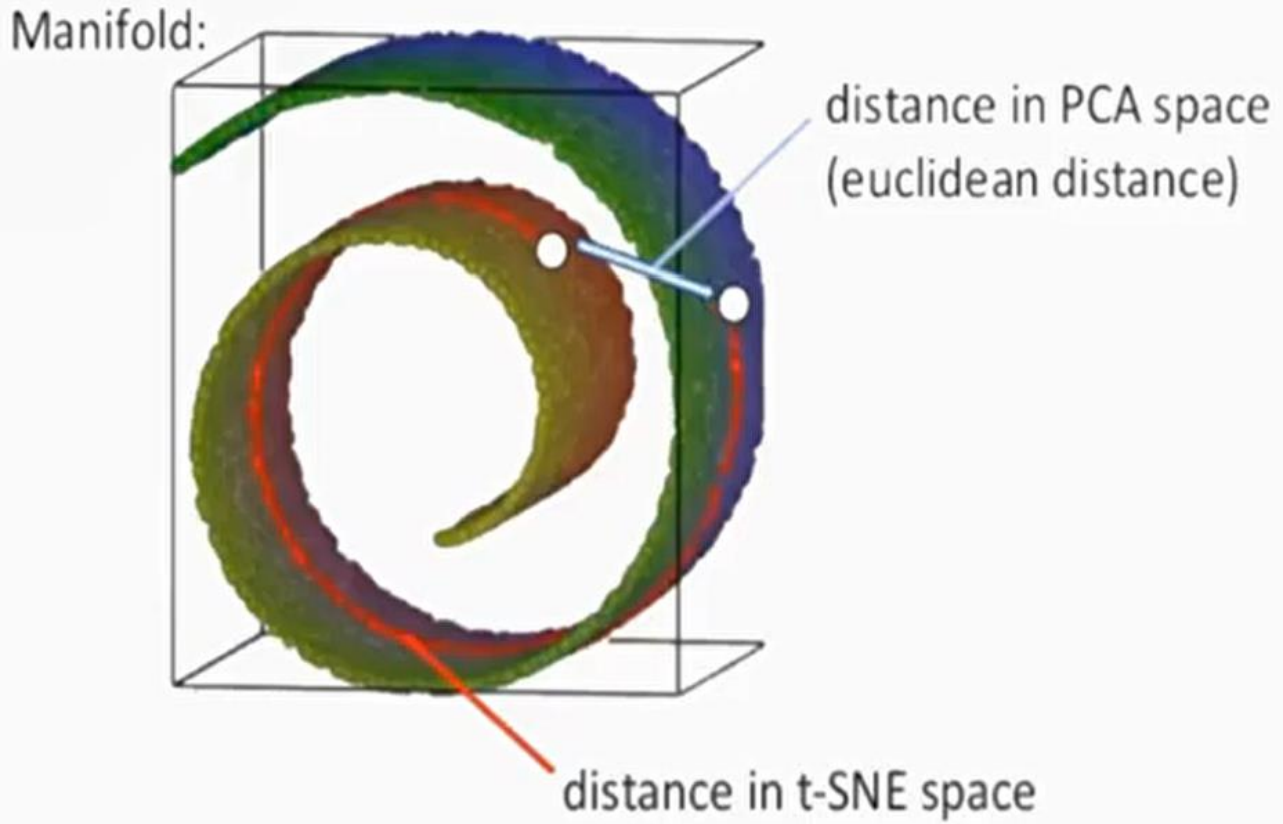
- We can represent the variance of the components in decreasing form (scree plot) and look for a jump in the value of the variance

- Taking **2-3** components we can **visualize** the data (the quality of the visualization depends on the amount of variance of those components)

Other dimension reduction methods: used later for visualisation

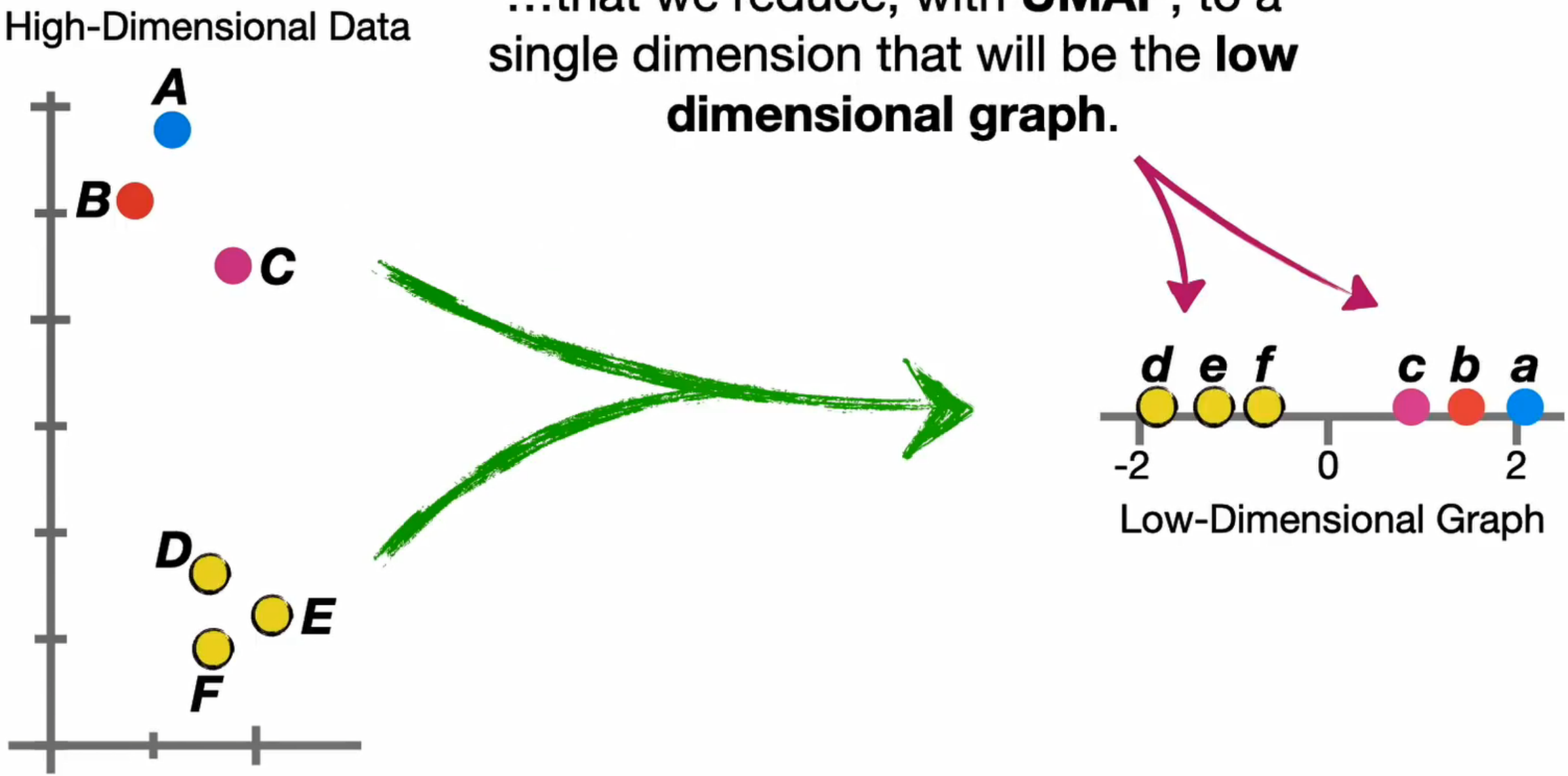
- Graph-based, non-linear methods like tSNE and UMAP
- PCA, tSNE and UMAP available as options in most tools
- We use PCA for dimension reduction before clustering, and tSNE or UMAP for visualisation

(Uniform Manifold Approximation and Projection)

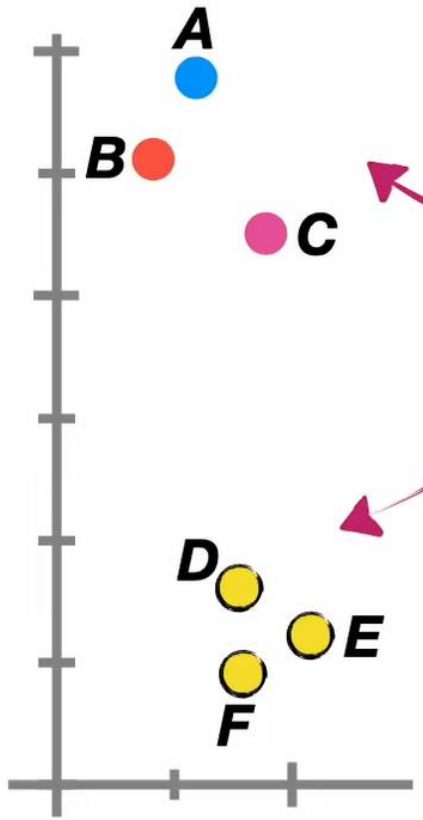


- Projects the data into a lower dimensional space trying to preserve the distances between neighbors
- Assume that each example can be reconstructed by a linear combination of its neighbors (weights)
- With these weights, a new data set is calculated that preserves the reconstruction in a space with fewer dimensions.
- Parameters:
  - The number k of neighbors to calculate the locality
  - The D value for the number of target dimensions ($D < k - 1$)

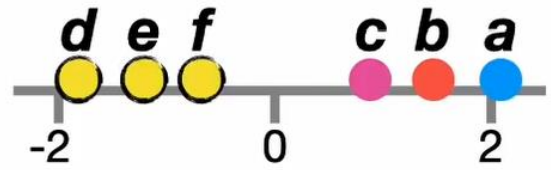
…that we reduce, with **UMAP**, to a single dimension that will be the **low dimensional graph**.



High-Dimensional Data



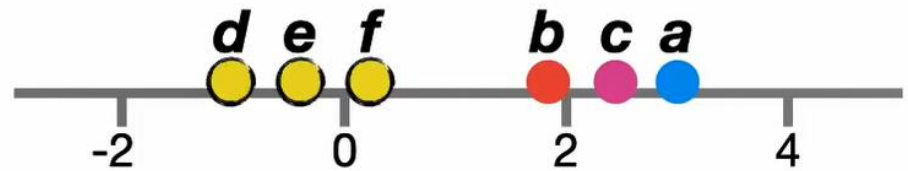
The goal of **UMAP** is to create a low-dimension graph of this data that preserves these high-dimensional clusters and their relationship to each other.



Low-Dimensional Graph

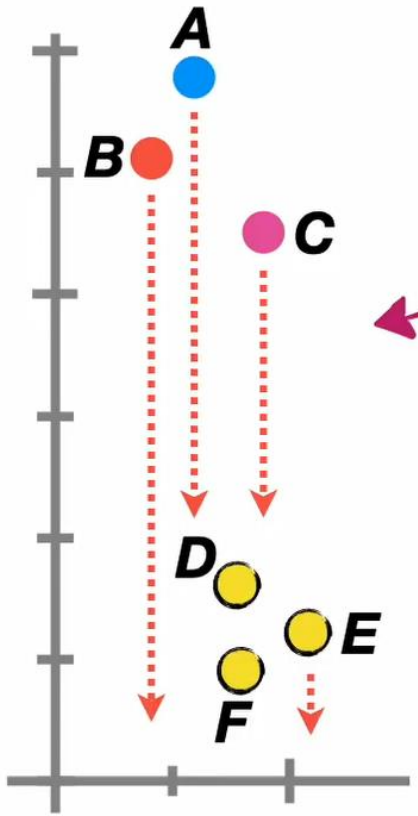
…**UMAP** wants the low-dimensional points ***a***, ***b*** and ***c*** to be close to each other…

…and relatively far from the other cluster.



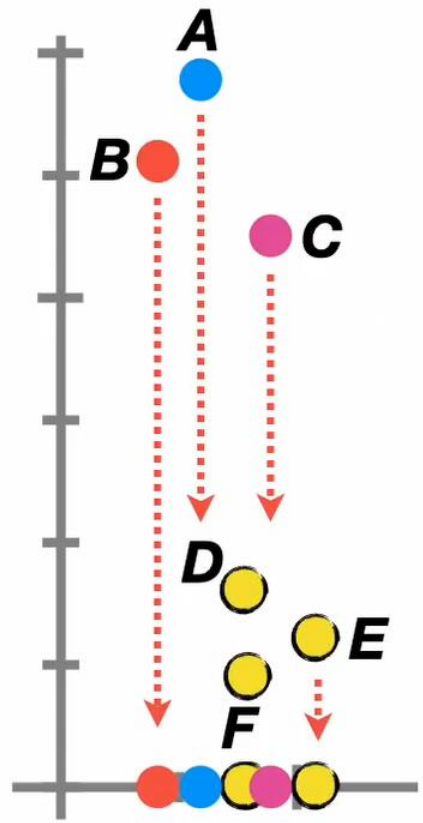
Low-Dimensional Graph

High-Dimensional Data



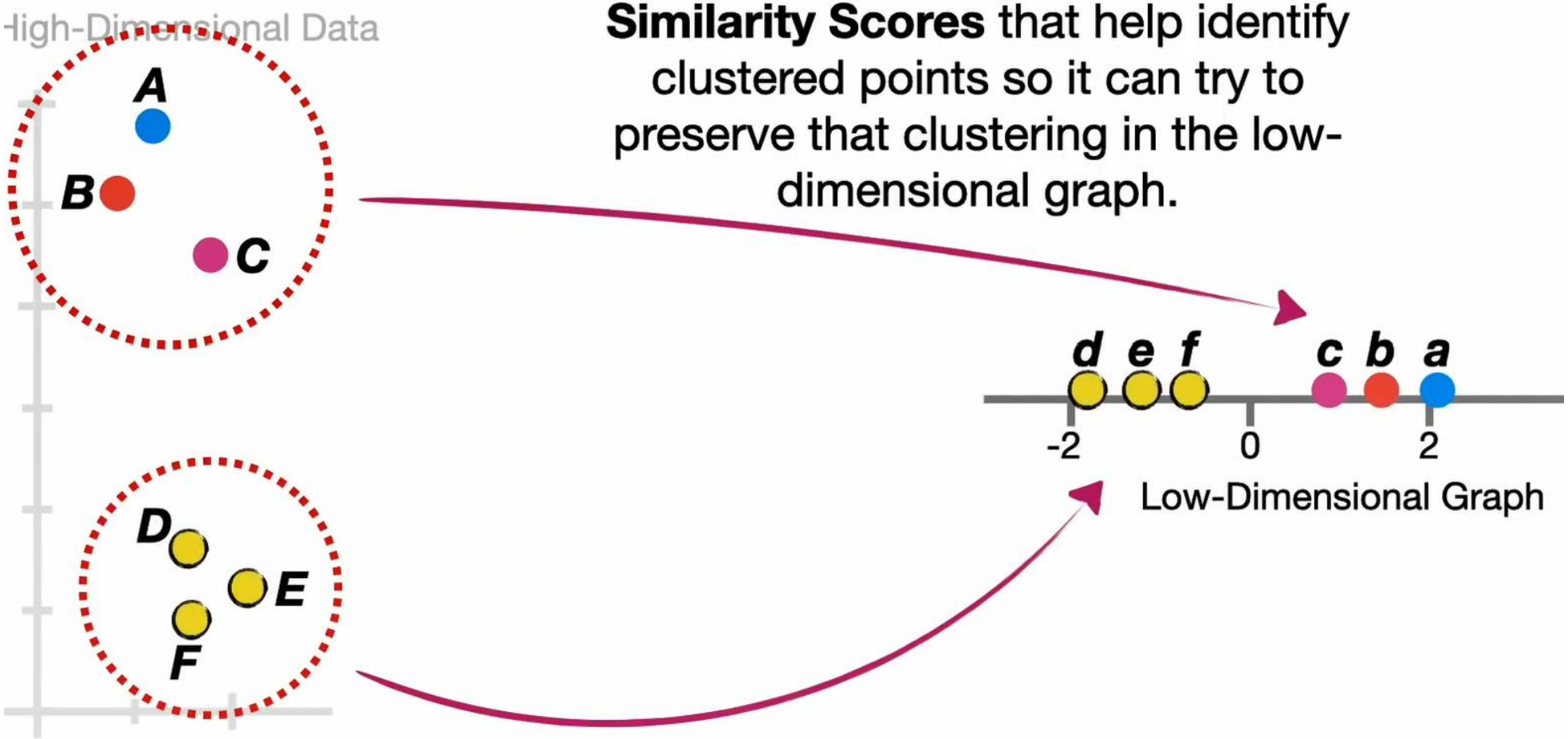
NOTE: If **UMAP** simply projected all of the data onto the x-axis…

High-Dimensional Data



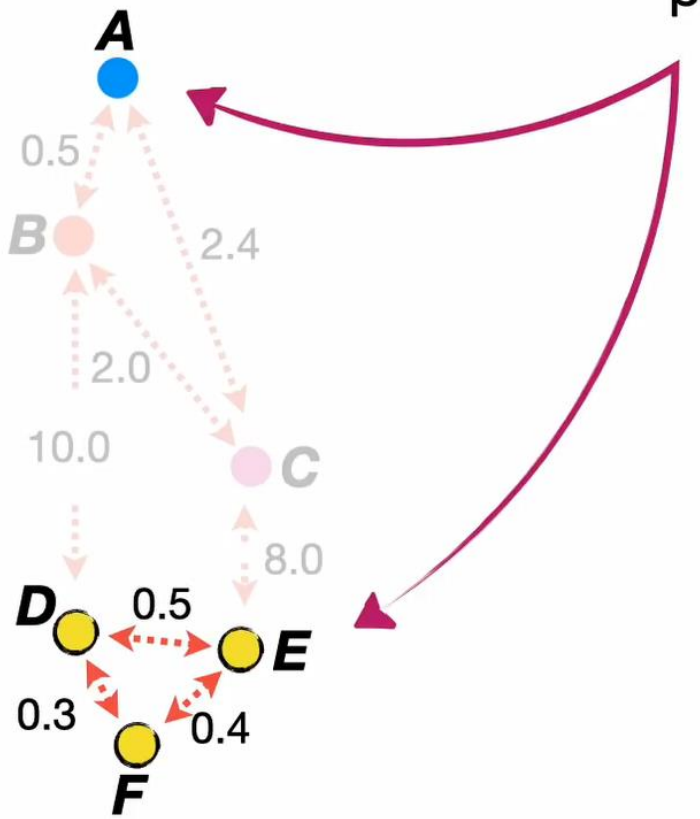
…then, instead of two distinct clusters, we'd see a mishmash of points.

So what **UMAP** does is calculate **Similarity Scores** that help identify clustered points so it can try to preserve that clustering in the low-dimensional graph.

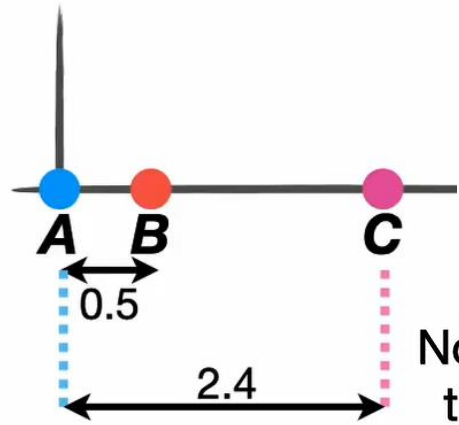


So the first thing that **UMAP** does is calculate the distances between each pair of high-dimensional points…

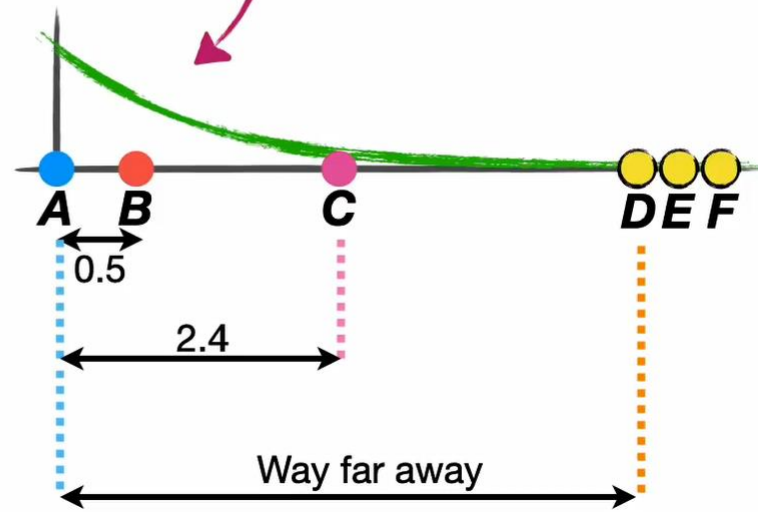
Raw Distances



And because the remaining points, ***D***, ***E*** and ***F***, are all *way far away* from ***A***…



Now we draw a curve over the data to calculate the **Similarity Scores**.



$\log_2$(num. neighbors)

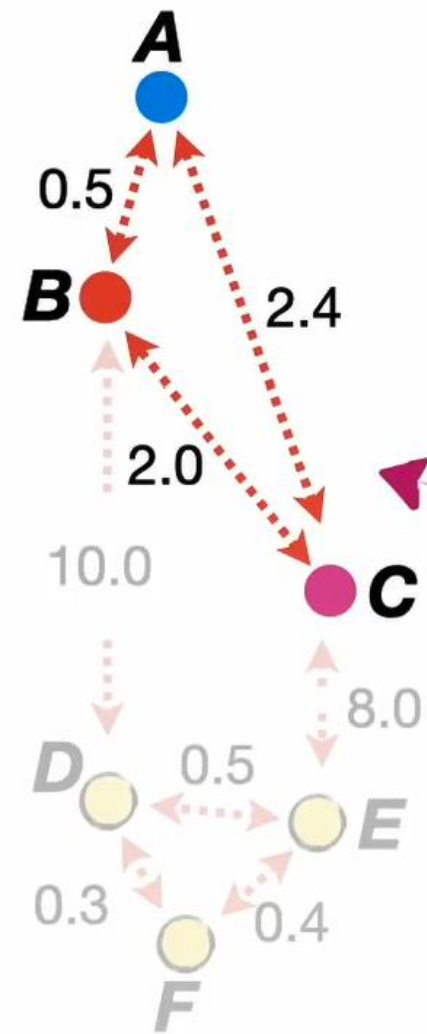
So, that said, **UMAP** takes the **$\log_2$()** of number of high-dimensional neighbors you want each point to have.

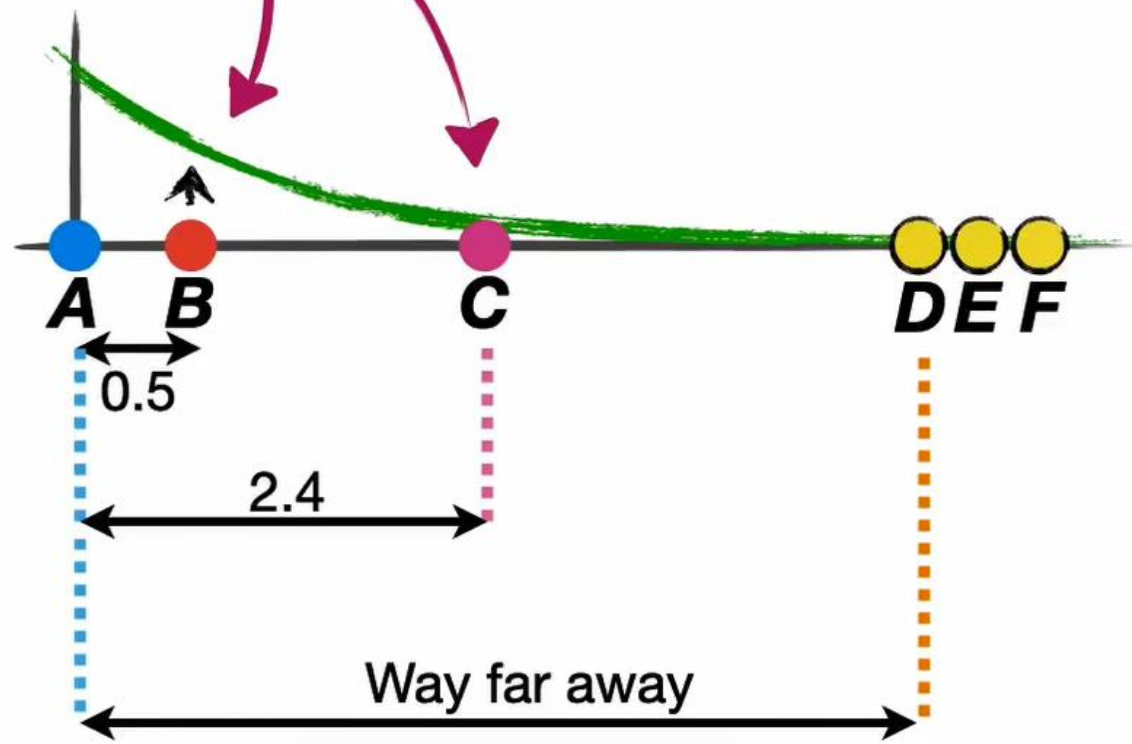
$\log_2(\text{num. neighbors}) = \log_2(3) = 1.6$

The curve is shaped in such a way that the y-axis coordinates for the nearest neighbors, which in this case are points ***B*** and ***C***...

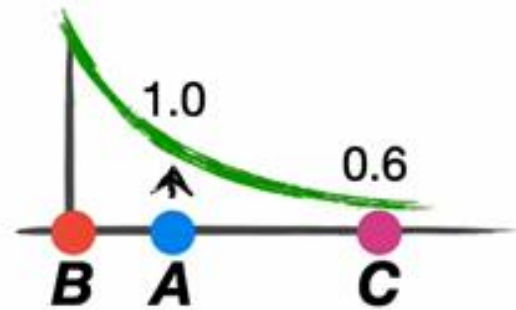
...add up to the **$\log_2$(num. neighbors)**, which, in this case, is **1.6**.

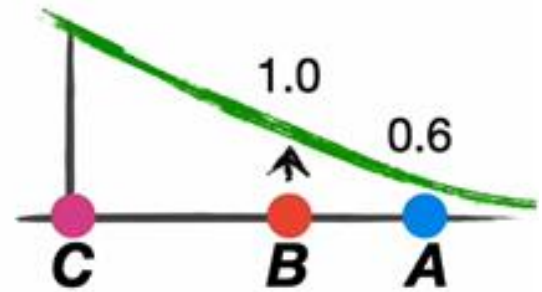
Raw Distances



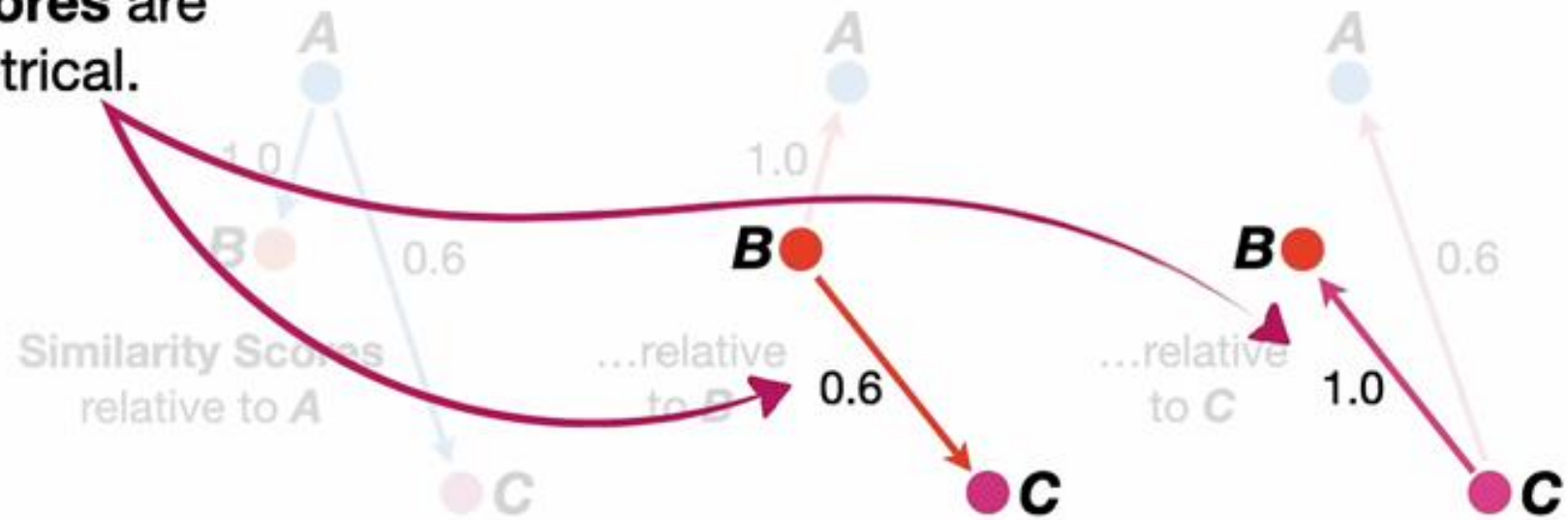


$\log_2(\text{num. neighbors}) = \log_2(3) = 1.6$



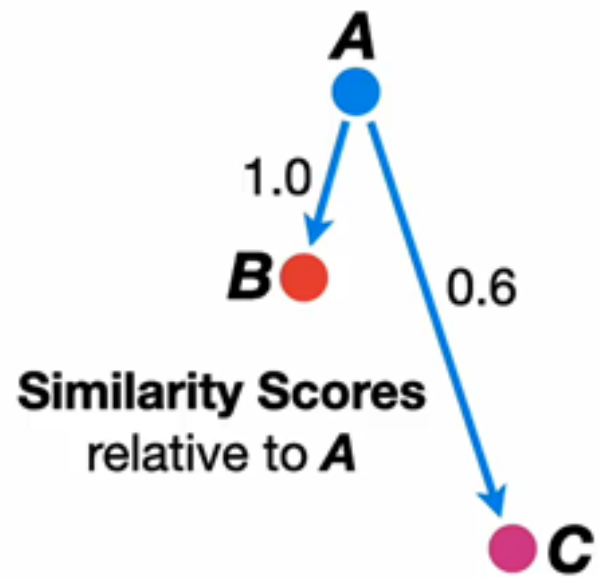


However, using different curves means the **Similarity Scores** are not symmetrical.

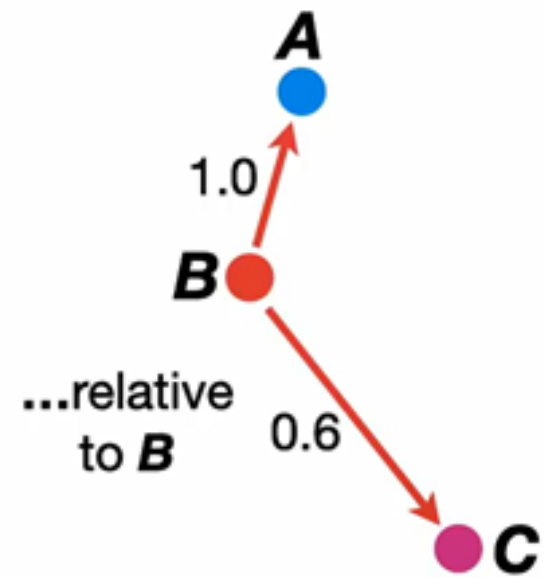


Symmetric Similarity Scores

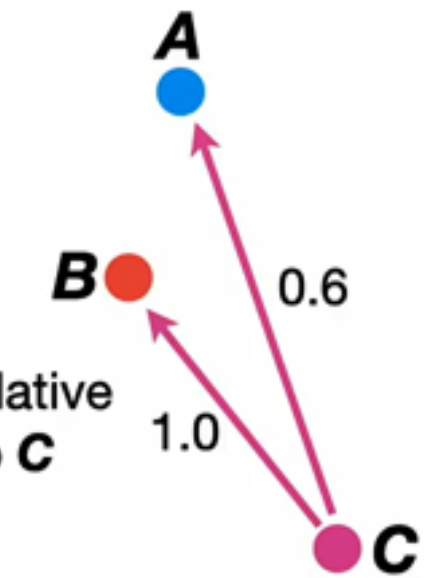
So **UMAP** makes them symmetrical using a method *similar* to taking the average,



Similarity Scores relative to ***A***

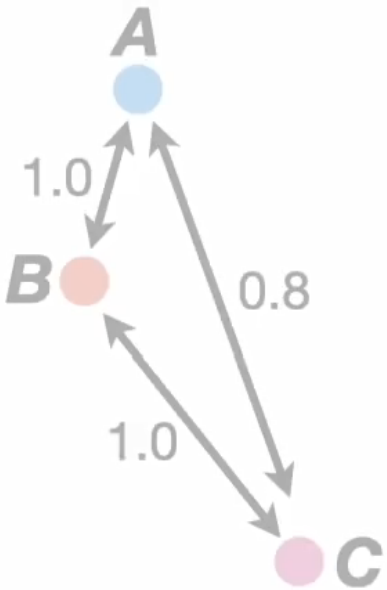


...relative to ***B***

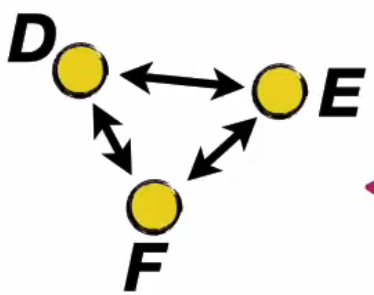


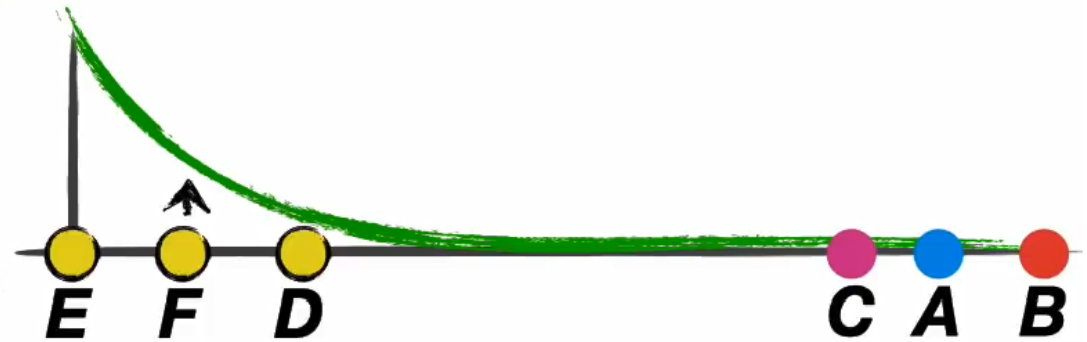
...relative to ***C***

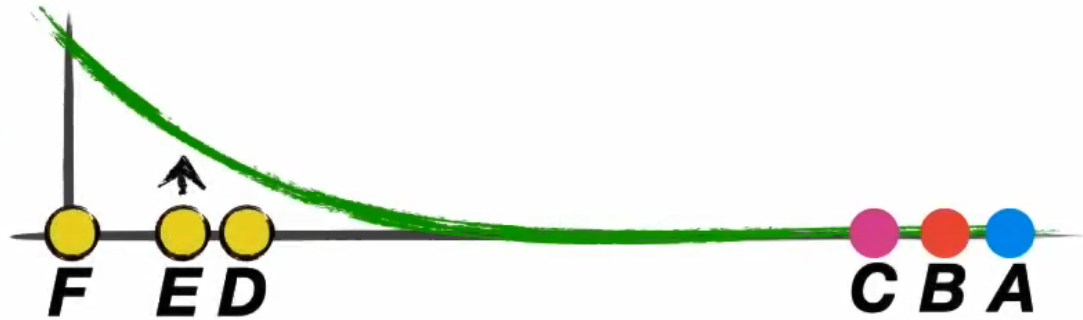
Symmetric
Similarity Scores



NOTE: In the exact same way, **UMAP** calculates the **Symmetric Similarity Scores** for points ***D***, ***E*** and ***F***.



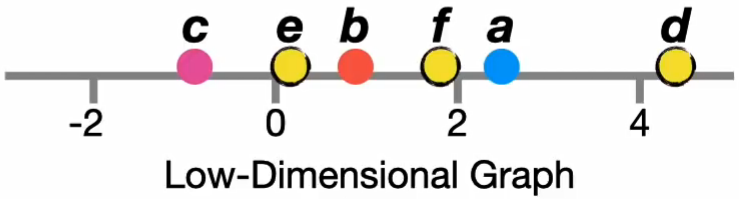




NOTE: If you are familiar with how **t-SNE** works, you may have noticed that **UMAP** is *very, very similar*.

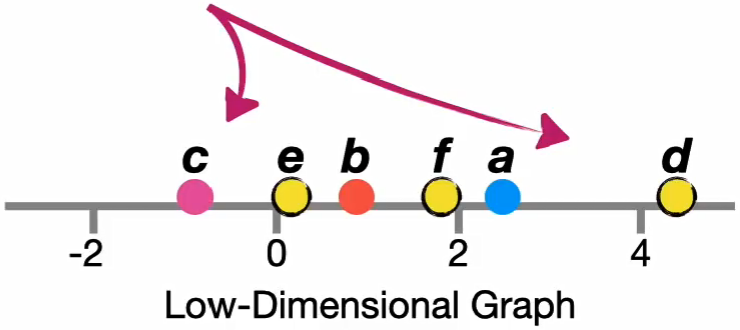
In terms of the main ideas of how **UMAP** and **t-SNE** work, they are essentially the same, and most of the differences are very subtle.

The first difference is that **t-SNE** always starts with a random initialization of the low-dimensional graph.



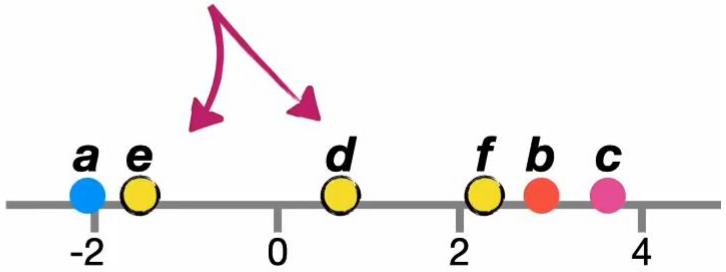
Low-Dimensional Graph

In other words, **t-SNE** might start out with the low-dimensional data looking like this…



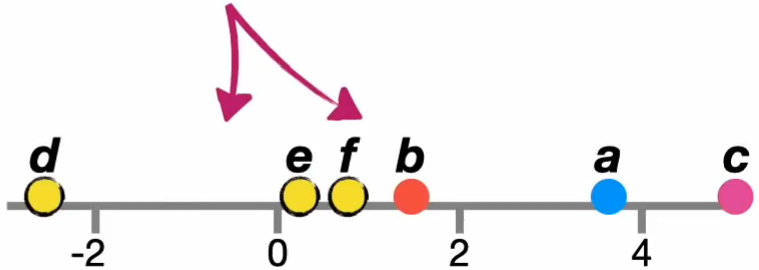
Low-Dimensional Graph

…and then, the next time you run **t-SNE** on the exact same dataset, it might start out with the low-dimensional data looking like this…



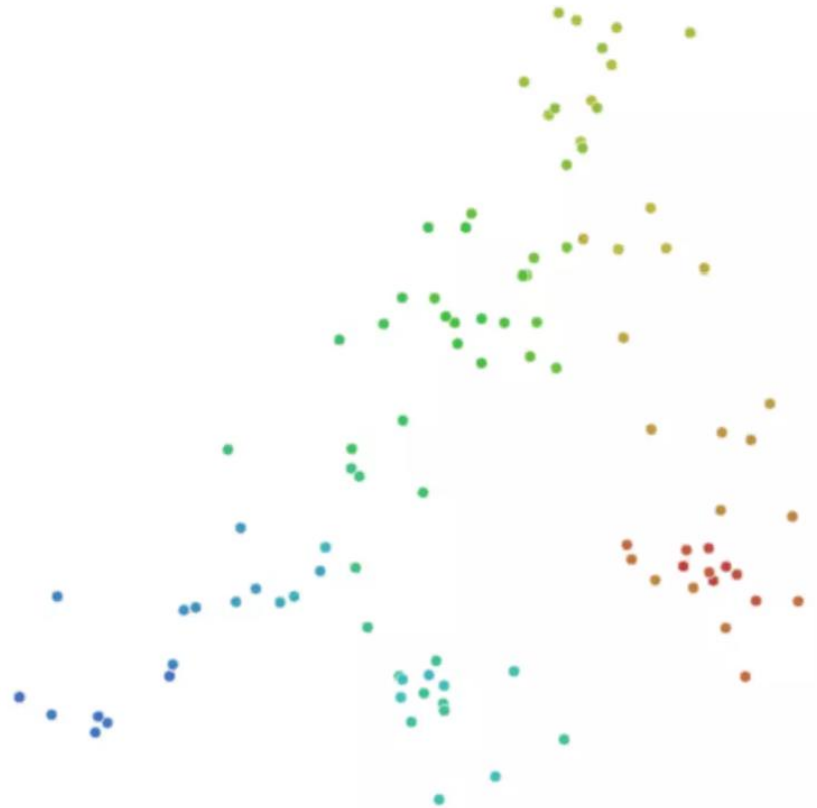
Low-Dimensional Graph

In contrast, **UMAP** uses something called **Spectral Embedding** to initialize the low-dimensional graph…



Low-Dimensional Graph

t-Distributed Stochastic Neighbour Embedding (t-SNE)

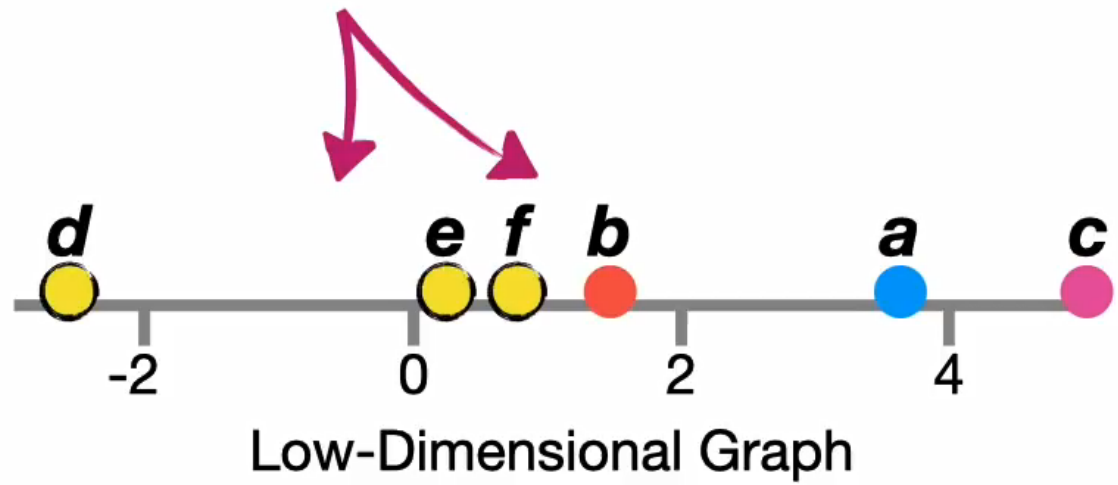


t-SNE is **non-linear** dimensionality reduction algorithm

t-SNE uses **conditional probabilities** to represent similarity between points

Has been shown to better represent underlying structure of multidimensional data

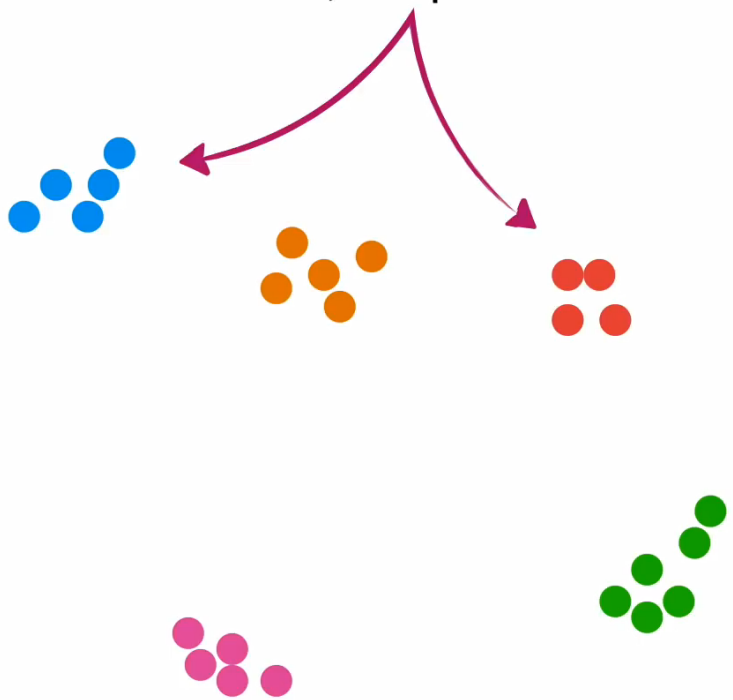
…and what that means, is that every time you use **UMAP** on a specific dataset, you always start with the exact same low-dimensional graph.



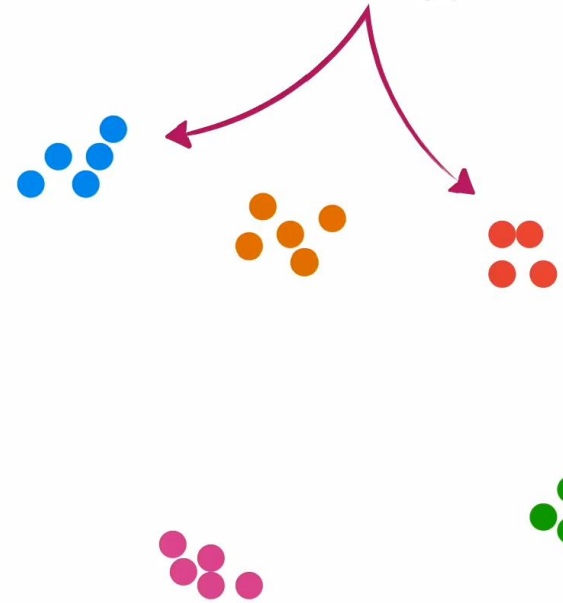
Generally speaking, when you have a lot of data…

	Weight	Height	Age	…
1	56	150	54	…
2	62	170	21	…
3	71	168	34	…
	…	…	…	…

…a relatively low value for the number of neighbors results small, independent clusters.



In some ways, this is sort of like seeing the details, but not the big picture.



In contrast, a relatively large value for the number of neighbors gives you more of the big picture and less of the details.