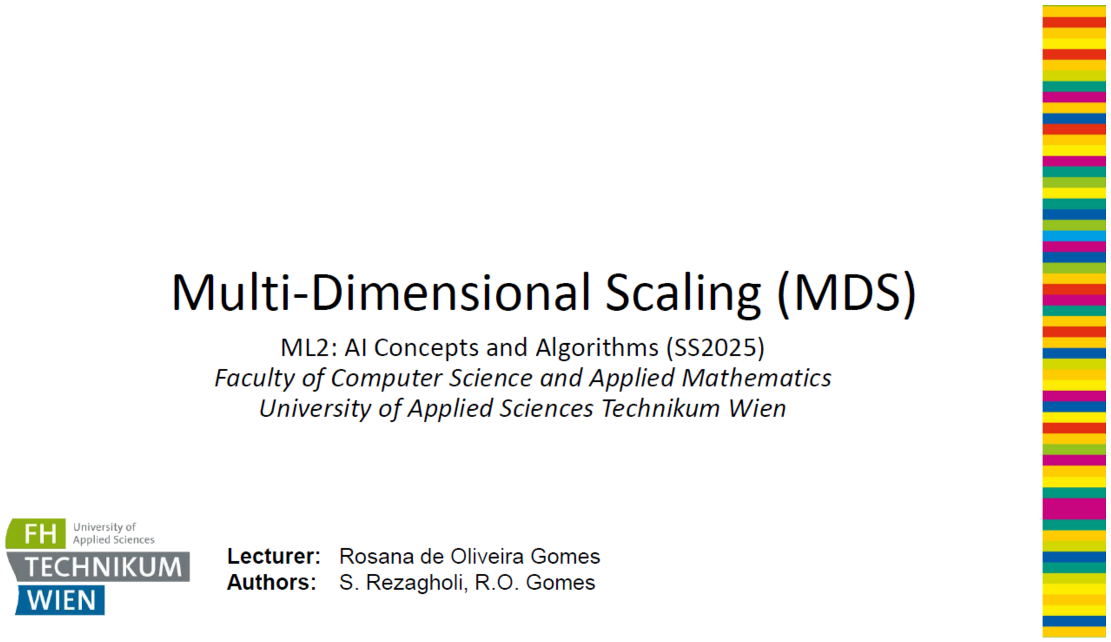
MDS

June 22, 2025



Multi-Dimensional Scaling (MDS)

ML2: AI Concepts and Algorithms (SS2025)

Faculty of Computer Science and Applied Mathematics

University of Applied Sciences Technikum Wien

Lecturer: Rosana de Oliveira Gomes

Authors: S. Rezagholi, R.O. Gomes

Presentation – Slide 1: Introduction to Multi-Dimensional Scaling (MDS)

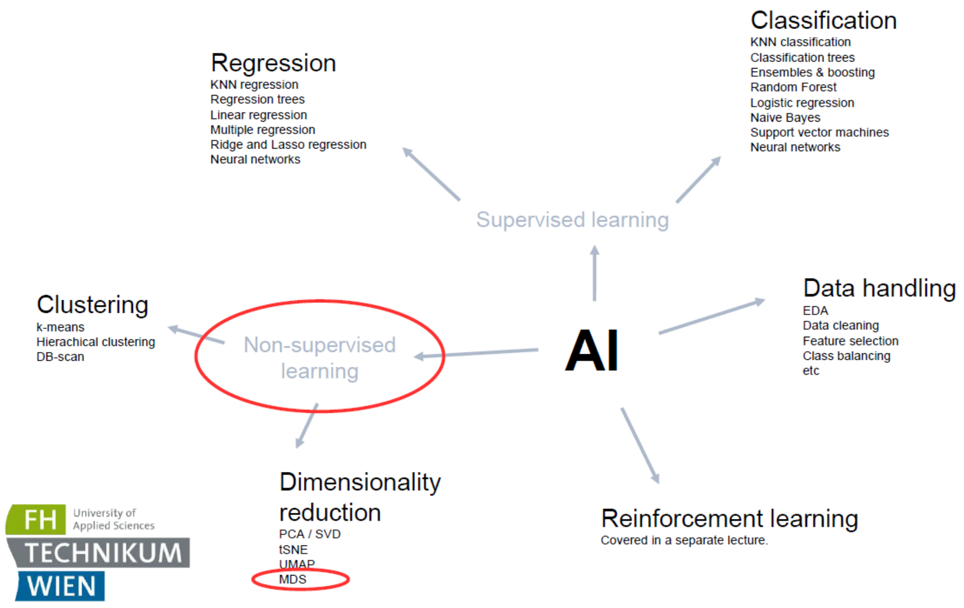
Good [morning/afternoon], everyone.

Welcome to today's presentation on **Multi-Dimensional Scaling (MDS)** — a technique used in machine learning and data analysis to visualize the similarity or dissimilarity between data points in a lower-dimensional space.

This lecture is part of the **ML2: AI Concepts and Algorithms** course for the Summer Semester 2025, presented by the **Faculty of Computer Science and Applied Mathematics** at the **University of Applied Sciences Technikum Wien**.

The material was prepared by **S. Rezagholi and Rosana de Oliveira Gomes**, who is also the lecturer for this session.

Let's now proceed to the next slide — please share it when ready.



Slide 2: Where Does MDS Fit in AI?

Let's begin by placing **Multidimensional Scaling (MDS)** in the broader context of **Artificial Intelligence**.

AI consists of various learning paradigms, and these are commonly grouped into three main categories:

- **Supervised learning** (e.g., regression and classification),
- **Unsupervised learning** (e.g., clustering and dimensionality reduction),
- **Reinforcement learning** (handled in a separate lecture).

MDS belongs to **unsupervised learning**, as highlighted by the red circle in the bottom-left quadrant.

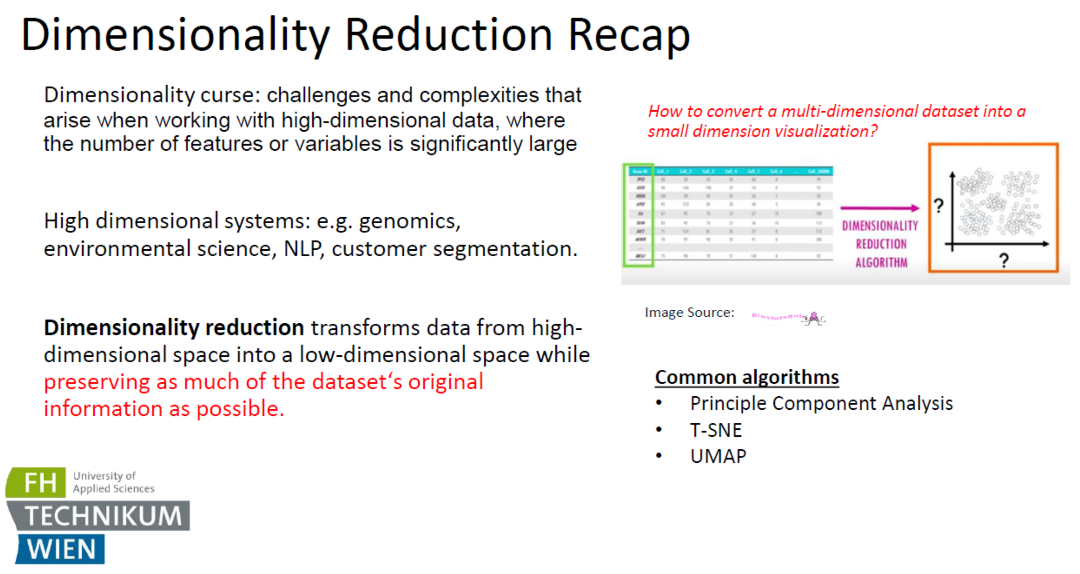
Under **Dimensionality Reduction**, MDS joins other techniques like:

- **PCA / SVD** (Principal Component Analysis, Singular Value Decomposition),
- **t-SNE**,
- **UMAP**.

The goal of these methods is to **reduce high-dimensional data** into a more **interpretable low-dimensional form**, typically 2D or 3D — while **preserving the structure or relationships** between points as best as possible.

MDS, specifically, focuses on preserving **pairwise distances or dissimilarities** in the transformation process.

When you're ready, please proceed to the next slide.



Dimensionality Reduction Recap

Dimensionality curse: challenges and complexities that arise when working with high-dimensional data, where the number of features or variables is significantly large

High dimensional systems: e.g. genomics, environmental science, NLP, customer segmentation.

Dimensionality reduction transforms data from high-dimensional space into a low-dimensional space while preserving as much of the dataset's original information as possible.

How to convert a multi-dimensional dataset into a small dimension visualization?

Image Source: [illegible]

Common algorithms

- Principle Component Analysis
- T-SNE
- UMAP

FH University of Applied Sciences TECHNIKUM WIEN

Slide 3: Dimensionality Reduction Recap

Before diving deeper into MDS, let's quickly **recap the concept of dimensionality reduction**.

The Curse of Dimensionality refers to the challenges that emerge when working with data that has **a large number of features**. As the number of dimensions increases, data becomes sparse, computations grow more expensive, and patterns become harder to detect.

This issue appears in many **high-dimensional systems**, such as:

- **Genomics**
- **Environmental science**
- **Natural Language Processing (NLP)**
- **Customer segmentation**

Dimensionality Reduction techniques aim to **project high-dimensional data into a lower-dimensional space** — typically 2D or 3D — while **preserving the structure, distance, or similarity relationships** as much as possible.

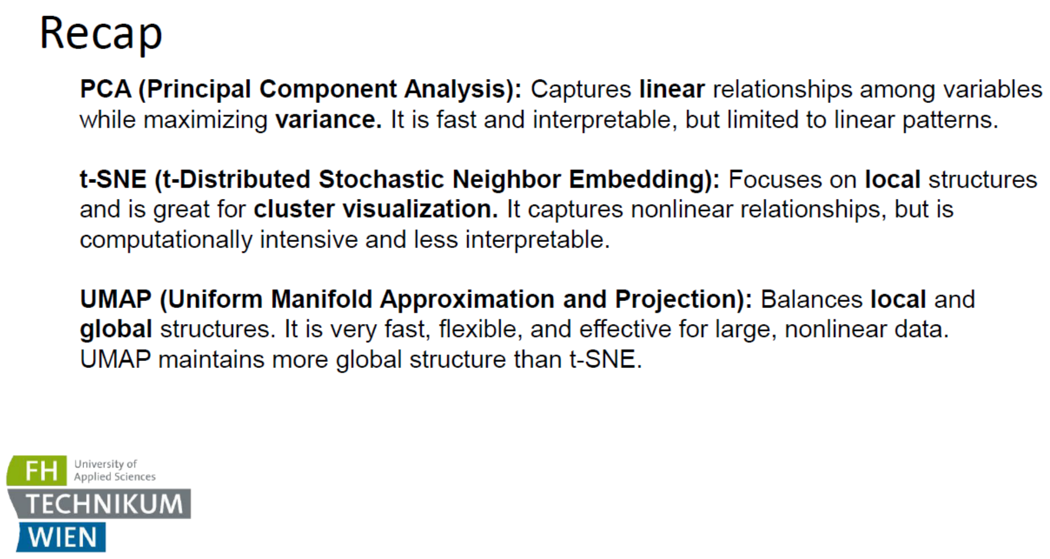
Common algorithms include:

- **PCA** (Principal Component Analysis)
- **t-SNE** (t-distributed Stochastic Neighbor Embedding)
- **UMAP** (Uniform Manifold Approximation and Projection)

The key question these algorithms answer is: **"How can we convert a multi-dimensional dataset into a compact, visual form without losing the essence of the original information?"**

MDS, the focus of this lecture, is another answer to this question — one that prioritizes **pairwise distances or dissimilarities**.

Let's move on when you're ready.


Recap

PCA (Principal Component Analysis): Captures **linear** relationships among variables while maximizing **variance.** It is fast and interpretable, but limited to linear patterns.

t-SNE (t-Distributed Stochastic Neighbor Embedding): Focuses on **local** structures and is great for **cluster visualization.** It captures nonlinear relationships, but is computationally intensive and less interpretable.

UMAP (Uniform Manifold Approximation and Projection): Balances **local** and **global** structures. It is very fast, flexible, and effective for large, nonlinear data. UMAP maintains more global structure than t-SNE.

FH University of Applied Sciences TECHNIKUM WIEN

Slide 4: Recap of Common Dimensionality Reduction Techniques

To better understand where **MDS** stands, let's review three widely used dimensionality reduction methods:

PCA (Principal Component Analysis)

- Captures **linear relationships** between variables.
- Maximizes **variance**, projecting the data into new axes (principal components) that best explain the variability.
- Fast, interpretable, but **limited to linear structures**.

---

t-SNE (t-Distributed Stochastic Neighbor Embedding)

- Emphasizes **local structure**, making it ideal for **cluster visualization**.
- Excellent at revealing **nonlinear relationships**, but…
- It is **computationally expensive** and often **less interpretable** due to stochastic behavior and lack of global structure preservation.

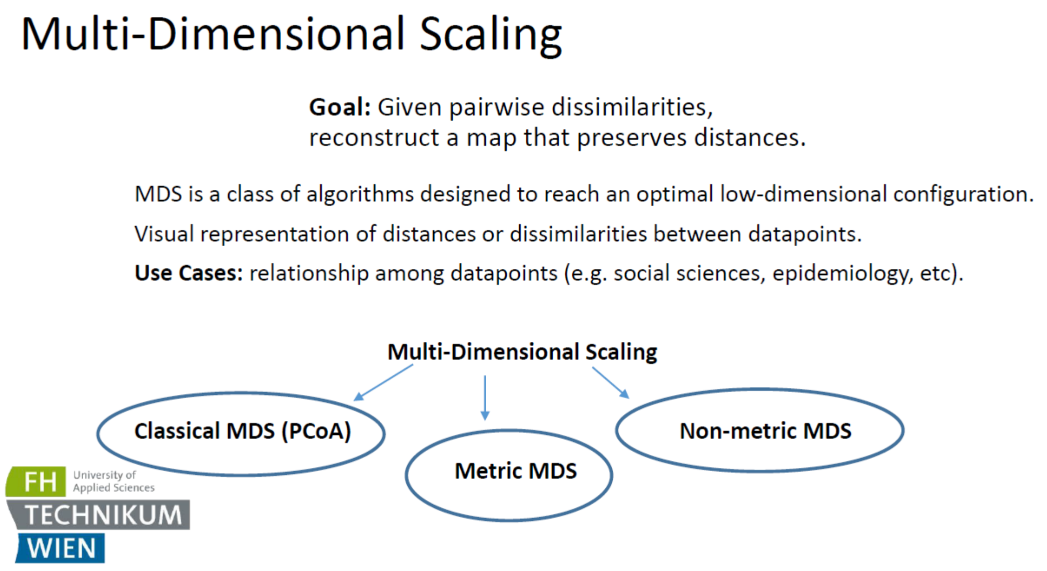
UMAP (Uniform Manifold Approximation and Projection)

- Balances **local and global** structure preservation.
- More **efficient and scalable** than t-SNE.
- Captures complex **nonlinear patterns** and tends to preserve the **overall shape** of the data better than t-SNE.

Each of these methods has a different philosophy about what should be preserved: **variance**, **local neighborhoods**, or **manifold structure**.

Now, MDS adds another perspective by preserving **pairwise distances** or **dissimilarities** directly. We'll see how it compares next.

Please go ahead with the next slide.



Multi-Dimensional Scaling

Goal: Given pairwise dissimilarities, reconstruct a map that preserves distances.

MDS is a class of algorithms designed to reach an optimal low-dimensional configuration.

Visual representation of distances or dissimilarities between datapoints.

Use Cases: relationship among datapoints (e.g. social sciences, epidemiology, etc).

Multi-Dimensional Scaling
Classical MDS (PCoA)
Metric MDS
Non-metric MDS
FH University of Applied Sciences TECHNIKUM WIEN

Slide 5: Multi-Dimensional Scaling (MDS)

Now we arrive at the core topic of today's presentation: **Multi-Dimensional Scaling**, or **MDS**.

Goal of MDS: Given **pairwise dissimilarities** between data points, MDS aims to reconstruct a **low-dimensional map** such that the distances between points in this new space reflect the **original dissimilarities** as closely as possible.

MDS is **not a single algorithm**, but rather a **family of algorithms** designed to find an optimal configuration in 2D or 3D, where the **geometry (distance relationships)** of the data is preserved.

It's often used when we don't have original high-dimensional features — only a **distance or dissimilarity matrix** is available. This makes it especially powerful for **visualizing relationships**.

Use Cases:

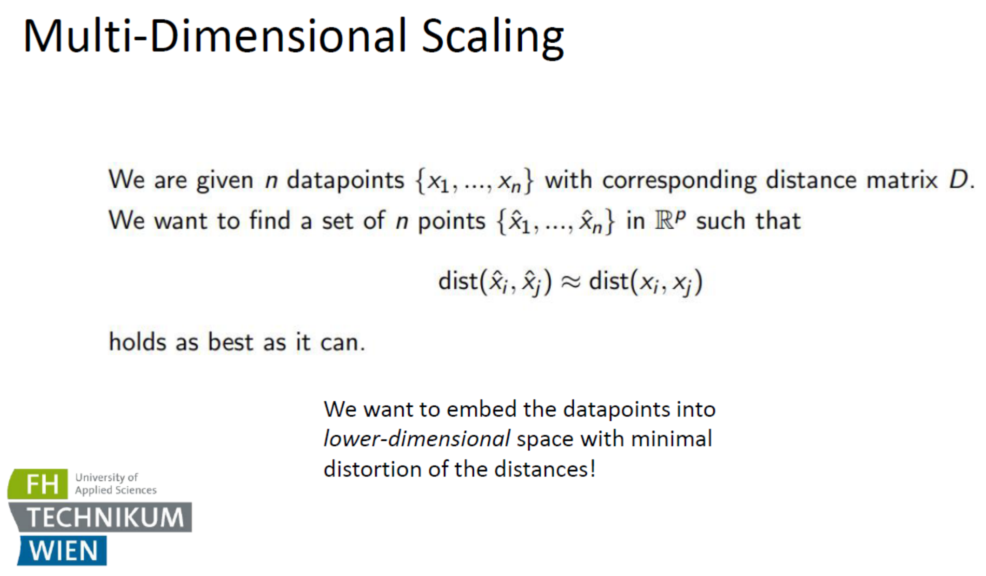
- Social sciences (e.g., opinion similarity)
- Psychology (e.g., perception studies)
- Epidemiology (e.g., genetic or phenotypic similarity)
- Marketing (e.g., brand positioning)

There are three main **types of MDS**:

1. **Classical MDS** (also called **Principal Coordinates Analysis**, or **PCoA**):
   - Based on **eigen decomposition** of the distance matrix.
2. **Metric MDS**:
   - Tries to preserve the actual **distance values**.
3. **Non-metric MDS**:
   - Focuses only on **rank order** of dissimilarities, not their exact values.

We'll break these down in upcoming slides.

Let's continue when you're ready.



Multi-Dimensional Scaling

We are given n datapoints $\{x_1, ..., x_n\}$ with corresponding distance matrix D.
We want to find a set of n points $\{\hat{x}_1, ..., \hat{x}_n\}$ in $\mathbb{R}^p$ such that

$$\text{dist}(\hat{x}_i, \hat{x}_j) \approx \text{dist}(x_i, x_j)$$

holds as best as it can.

We want to embed the datapoints into *lower-dimensional* space with minimal distortion of the distances!

FH University of Applied Sciences TECHNIKUM WIEN

Slide 6: MDS – The Mathematical Goal

Let's now formalize what **MDS** is trying to achieve.

We are given:

- A set of n data points $\{x_1, x_2, ..., x_n\}$
- Along with their **pairwise distances**, stored in a **distance matrix** D

But instead of the original space, we aim to find:

- A new set of points $\{\hat{x}_1, \hat{x}_2, ..., \hat{x}_n\}$
- Embedded in a **lower-dimensional space** $\mathbb{R}^p$, where typically $p = 2$ or 3

The objective is to ensure that:

$$\text{dist}(\hat{x}_i, \hat{x}_j) \approx \text{dist}(x_i, x_j)$$

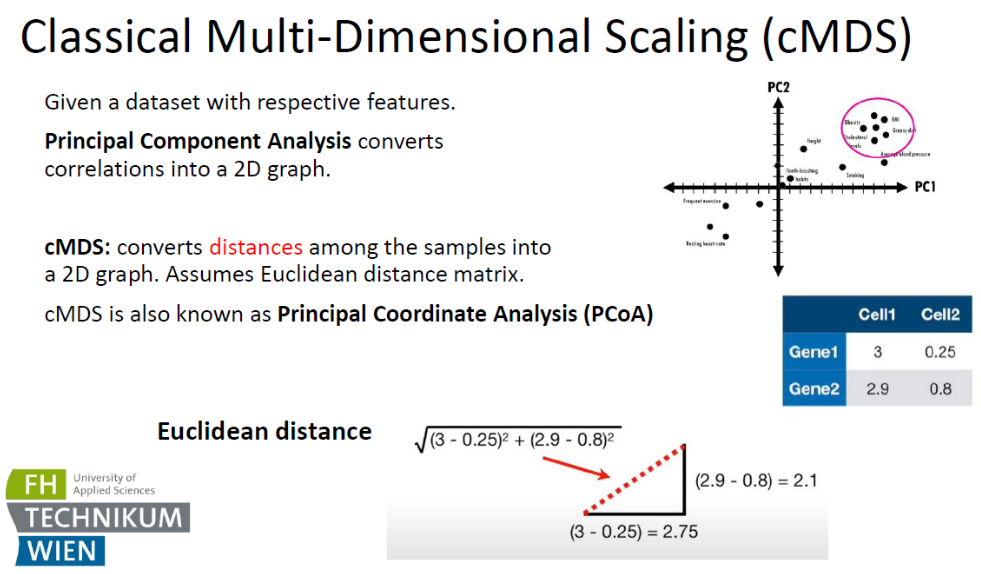
That is, the distance between the **mapped points** should approximate the **original distances** as closely as possible.

In plain terms:

> "We want to embed the datapoints into a **lower-dimensional space**, with **minimal distortion** of the original pairwise distances."

This is the heart of **MDS** — maintaining the structure of relationships, even after compression into fewer dimensions.

Ready for the next slide!



Classical Multi-Dimensional Scaling (cMDS)

Given a dataset with respective features.

Principal Component Analysis converts correlations into a 2D graph.

cMDS: converts distances among the samples into a 2D graph. Assumes Euclidean distance matrix.

cMDS is also known as **Principal Coordinate Analysis (PCoA)**

	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8

Euclidean distance

$\sqrt{(3 - 0.25)^2 + (2.9 - 0.8)^2}$

$(2.9 - 0.8) = 2.1$

$(3 - 0.25) = 2.75$

Slide 7: Classical Multi-Dimensional Scaling (cMDS)

Let's now look at the first variant of MDS — **Classical MDS**, also known as **Principal Coordinates Analysis (PCoA)**.

Contrast with PCA:

- PCA transforms **correlations or covariances** into principal components for dimensionality reduction.
- In contrast, **cMDS** starts with a **distance matrix**, not feature vectors.

What does cMDS do?

- It takes **pairwise distances** between samples (typically Euclidean)
- And converts this information into a configuration of points in a lower-dimensional space — usually **2D or 3D**
- Such that the **geometrical distances between points** approximate the original **distances among the samples**

In the bottom example:

- We have data from two cells across two genes.
- We calculate the **Euclidean distance**:

$$\sqrt{(3-0.25)^2+(2.9-0.8)^2} \approx 3.45$$

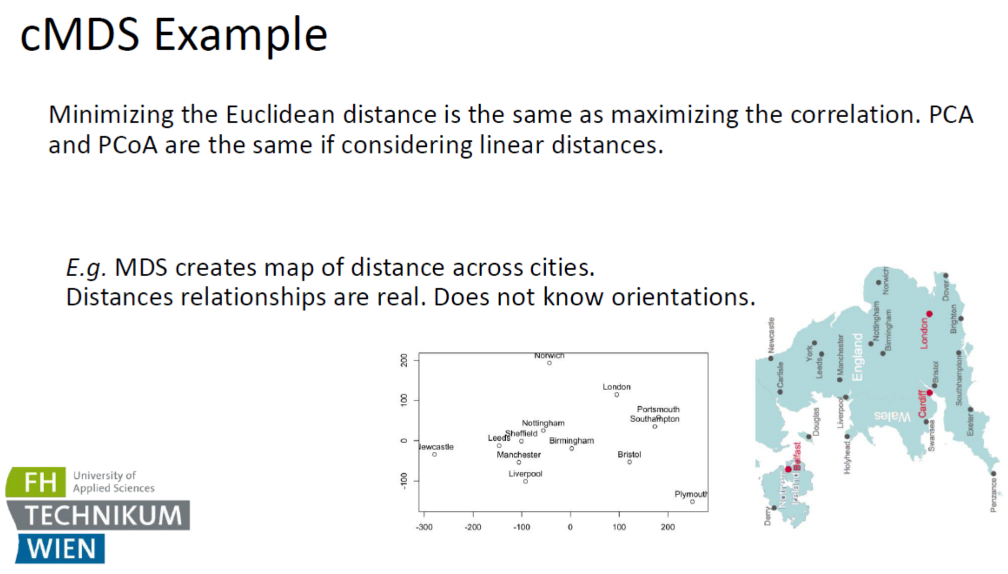
- This process is repeated for all pairwise combinations, building the full distance matrix.

So essentially:

> **Classical MDS = Eigen-decomposition of the distance matrix** Result = a spatial configuration of the points that best preserves those distances.

We'll later see how this differs from **metric** and **non-metric MDS**.

Please continue with the next slide.



cMDS Example

Minimizing the Euclidean distance is the same as maximizing the correlation. PCA and PCoA are the same if considering linear distances.

E.g. MDS creates map of distance across cities.
Distances relationships are real. Does not know orientations.

Norwich
London
Portsmouth
Southampton
Nottingham
Sheffield
Leeds
Birmingham
Newcastle
Manchester
Bristol
Liverpool
Plymouth
-300
-200
-100
0
100
200
Newcastle
Carlisle
York
Leeds
Manchester
Nottingham
Birmingham
Norwich
London
Dover
Brighton
England
Southampton
Bristol
Cardiff
Wales
Swansea
Exeter
Penzance
Liverpool
Holyhead
Douglas
Belfast
Derry
FH University of Applied Sciences TECHNIKUM WIEN

Slide 8: cMDS Example — Map of Cities

Here we have a classic and intuitive example to illustrate how **Classical MDS (cMDS)** works.

First, note this theoretical equivalence:

> **Minimizing Euclidean distance Maximizing correlation** This is why **PCA and PCoA (cMDS)** can produce similar results when we're working with **linear** distances and features.

Example: Mapping Cities

- Suppose we know only the **pairwise distances** between cities in the UK — but we don't know their coordinates.
- cMDS takes this distance matrix and produces a **2D plot** that **preserves those distances** as best as possible.
- The result is a **map-like reconstruction** of the cities' positions.

Look at the plots:

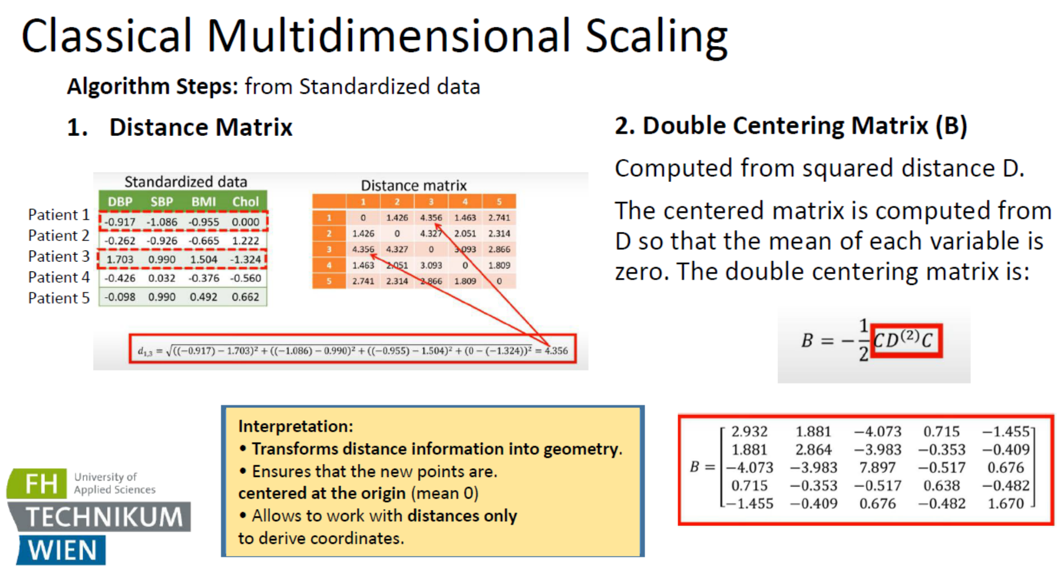
- On the **left**, cMDS output: relative positions between cities based on distances.
- On the **right**, the actual **geographical map**.

Important takeaway:

> cMDS preserves **distance relationships**, but not **absolute orientation or rotation**. You can **rotate**, **translate**, or **flip** the entire map — the relative distances remain the same.

It's a beautiful demonstration of how **geometry can be recovered from dissimilarities** alone.

Ready for the next slide.



Classical Multidimensional Scaling

Algorithm Steps: from Standardized data

1. Distance Matrix

Standardized data

	DBP	SBP	BMI	Chol
Patient 1	-0.917	-1.086	-0.955	0.000
Patient 2	-0.262	-0.926	-0.665	1.222
Patient 3	1.703	0.990	1.504	-1.324
Patient 4	-0.426	0.032	-0.376	-0.560
Patient 5	-0.098	0.990	0.492	0.662

Distance matrix

	1	2	3	4	5
1	0	1.426	4.356	1.463	2.741
2	1.426	0	4.327	2.051	2.314
3	4.356	4.327	0	3.093	2.866
4	1.463	2.051	3.093	0	1.809
5	2.741	2.314	2.866	1.809	0

$$d_{1,3} = \sqrt{((-0.917) - 1.703)^2 + ((-1.086) - 0.990)^2 + ((-0.955) - 1.504)^2 + (0 - (-1.324))^2} = 4.356$$

2. Double Centering Matrix (B)

Computed from squared distance D.

The centered matrix is computed from D so that the mean of each variable is zero. The double centering matrix is:

$$B = -\frac{1}{2}\boxed{CD^{(2)}C}$$

$$\boxed{B = \begin{bmatrix} 2.932 & 1.881 & -4.073 & 0.715 & -1.455 \\ 1.881 & 2.864 & -3.983 & -0.353 & -0.409 \\ -4.073 & -3.983 & 7.897 & -0.517 & 0.676 \\ 0.715 & -0.353 & -0.517 & 0.638 & -0.482 \\ -1.455 & -0.409 & 0.676 & -0.482 & 1.670 \end{bmatrix}}$$

Interpretation:
- **Transforms distance information into geometry.**
- Ensures that the new points are. **centered at the origin** (mean 0)
- Allows to work with **distances only** to derive coordinates.

FH University of Applied Sciences TECHNIKUM WIEN

Slide 9: Classical MDS — Step-by-Step Algorithm

Now let's break down the **algorithmic steps** behind **Classical MDS** using standardized data.

0.0.1 Step 1: Compute the Distance Matrix

- Start with a dataset of standardized features — in this case, values like **DBP, SBP, BMI, and Cholesterol** for 5 patients.
- Then compute the **Euclidean distances** between each pair of data points. Example (highlighted):

$$d_{1,3} = \sqrt{(-0.917 - 1.703)^2 + (-1.086 - 0.990)^2 + ...} = 4.356$$

- This produces a symmetric **distance matrix** D with zeros on the diagonal.

0.0.2 Step 2: Compute the Double Centering Matrix B

- First **square the distance matrix** D.
- Then apply **double centering** using the formula:

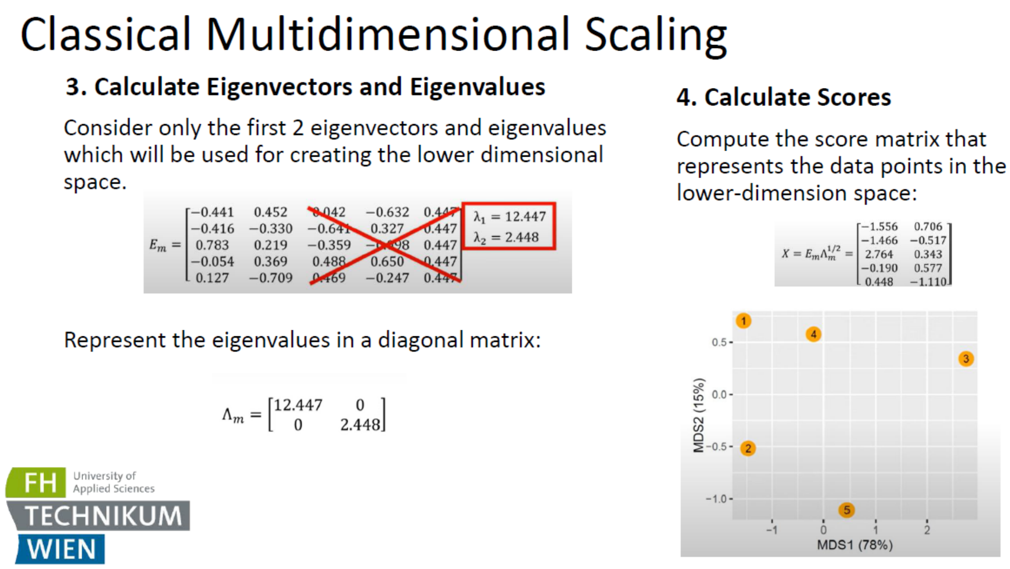
$$B = -\frac{1}{2} C D^{(2)} C$$

  where C is the **centering matrix**, which ensures that the resulting coordinates are **centered at the origin** (mean 0).

0.0.3 Why this step matters:

- It **transforms distance information into geometry**.
- It **eliminates the need for original coordinates** — distances alone are sufficient.
- The result is a matrix B that encodes **scalar products** between points in the new space — the foundation for recovering coordinates via **eigendecomposition** (in the next step).

Let's continue to the final steps of the cMDS algorithm when you're ready.



Classical Multidimensional Scaling

3. Calculate Eigenvectors and Eigenvalues

Consider only the first 2 eigenvectors and eigenvalues which will be used for creating the lower dimensional space.

$$E_m = \begin{bmatrix} -0.441 & 0.452 & 0.042 & -0.632 & 0.447 \\ -0.416 & -0.330 & -0.641 & 0.327 & 0.447 \\ 0.783 & 0.219 & -0.359 & -0.098 & 0.447 \\ -0.054 & 0.369 & 0.488 & 0.650 & 0.447 \\ 0.127 & -0.709 & 0.469 & -0.247 & 0.447 \end{bmatrix} \quad \boxed{\begin{matrix} \lambda_1 = 12.447 \\ \lambda_2 = 2.448 \end{matrix}}$$

Represent the eigenvalues in a diagonal matrix:

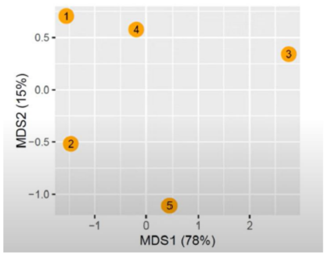
$$\Lambda_m = \begin{bmatrix} 12.447 & 0 \\ 0 & 2.448 \end{bmatrix}$$

FH University of Applied Sciences TECHNIKUM WIEN

4. Calculate Scores

Compute the score matrix that represents the data points in the lower-dimension space:

$$X = E_m \Lambda_m^{1/2} = \begin{bmatrix} -1.556 & 0.706 \\ -1.466 & -0.517 \\ 2.764 & 0.343 \\ -0.190 & 0.577 \\ 0.448 & -1.110 \end{bmatrix}$$



Slide 10: Classical MDS — Final Steps

We now conclude the **Classical MDS (cMDS)** algorithm with Steps 3 and 4.

0.0.4 Step 3: Compute Eigenvectors and Eigenvalues

- Perform **eigendecomposition** of the double-centered matrix B.
- Select the **top** m eigenvectors and eigenvalues — typically 2 — to define the new low-dimensional space.
- These represent the **principal axes** of the transformed geometry.

Notation:

- E_m: Matrix of the top m eigenvectors
- Λ_m: Diagonal matrix of the corresponding top m eigenvalues

---

0.0.5 Step 4: Compute the Coordinates (Scores)

To project the data into the low-dimensional space, we use:

$$X = E_m \Lambda_m^{1/2}$$

- This gives us the **coordinates** of the n data points in 2D.
- Each row of X is a data point.
- This is the actual **embedding** we visualize in the MDS plot.

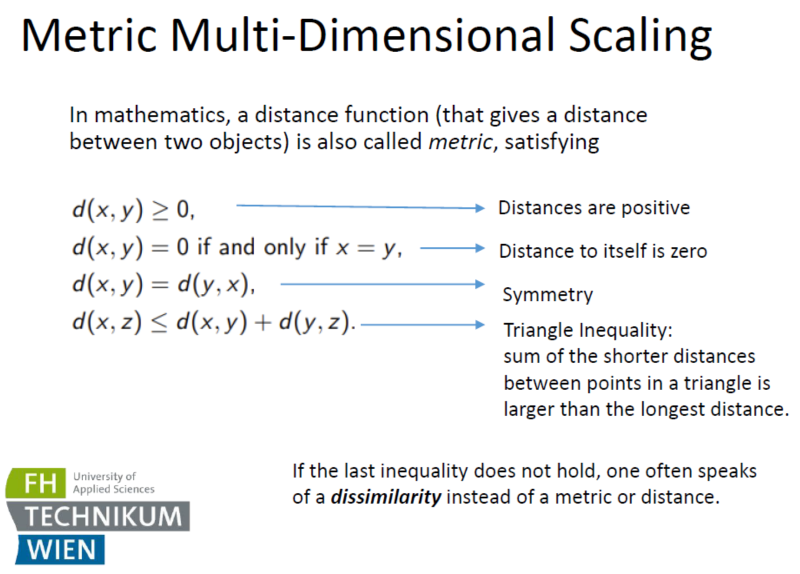
The MDS plot (bottom right):

- Plots **MDS1 vs MDS2**, showing the projected data points.
- The axes are scaled by the variance captured (e.g., 78% for MDS1, 15% for MDS2).
- The geometric relationships now reflect the **original distances** between samples.

In summary:

> Classical MDS turns pairwise distances into coordinates, purely through **distance matrix processing**, **centering**, and **eigendecomposition** — no need for the original feature vectors.

Let me know when to proceed with the next slide.



Metric Multi-Dimensional Scaling

In mathematics, a distance function (that gives a distance between two objects) is also called *metric*, satisfying

$d(x, y) \geq 0,$ → Distances are positive

$d(x, y) = 0$ if and only if $x = y$, → Distance to itself is zero

$d(x, y) = d(y, x),$ → Symmetry

$d(x, z) \leq d(x, y) + d(y, z).$ → Triangle Inequality: sum of the shorter distances between points in a triangle is larger than the longest distance.

If the last inequality does not hold, one often speaks of a ***dissimilarity*** instead of a metric or distance.

FH University of Applied Sciences TECHNIKUM WIEN

Slide 11: Metric Multi-Dimensional Scaling (MDS)

Now we shift our focus from **classical MDS** to **metric MDS**, which generalizes the idea beyond Euclidean geometry.

In mathematics, a **metric** is a function that defines a distance between any two elements in a space — and it must satisfy **four properties**:

1. **Non-negativity**

$$d(x, y) \geq 0$$

   → Distances are always positive.

2. **Identity of indiscernibles**

$$d(x, y) = 0 \text{ if and only if } x = y$$

$\rightarrow$ An object has zero distance to itself.

3. **Symmetry**

$$d(x, y) = d(y, x)$$

$\rightarrow$ The distance is the same in both directions.

4. **Triangle inequality**

$$d(x, z) \leq d(x, y) + d(y, z)$$

$\rightarrow$ Any side of a triangle can't be longer than the sum of the other two.

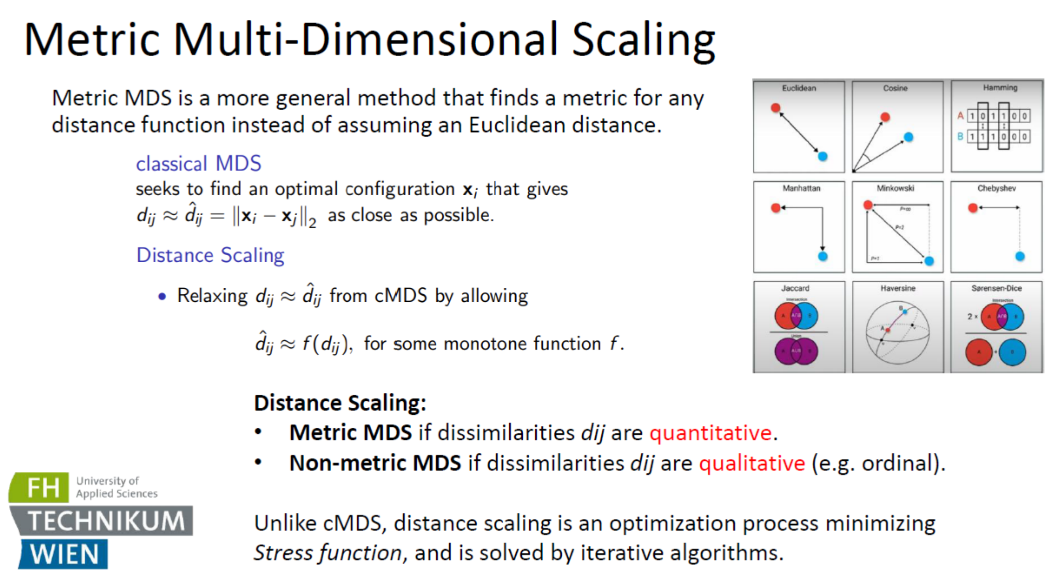
If all four of these properties are satisfied, we call it a **metric**. However, **if triangle inequality fails**, we speak of a **dissimilarity** rather than a true distance.

In **metric MDS**, we assume that the input dissimilarities behave like **true distances**, even if they're not Euclidean. The goal remains the same:

> Find a configuration of points in low-dimensional space whose **distances best match the input dissimilarities**.

This differs from **non-metric MDS**, which we'll explore next — where even the exact distance values aren't trusted, only their **rank order**.

Let me know when you're ready for the next slide.



Metric Multi-Dimensional Scaling

Metric MDS is a more general method that finds a metric for any distance function instead of assuming an Euclidean distance.

classical MDS

seeks to find an optimal configuration $\mathbf{x}_i$ that gives $d_{ij} \approx \hat{d}_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ as close as possible.

Distance Scaling

- Relaxing $d_{ij} \approx \hat{d}_{ij}$ from cMDS by allowing

$$\hat{d}_{ij} \approx f(d_{ij}), \text{ for some monotone function } f.$$

Distance Scaling:
- **Metric MDS** if dissimilarities *dij* are quantitative.
- **Non-metric MDS** if dissimilarities *dij* are qualitative (e.g. ordinal).

Unlike cMDS, distance scaling is an optimization process minimizing *Stress function*, and is solved by iterative algorithms.

Slide 12: Metric Multi-Dimensional Scaling (cont.)

Let's now deepen our understanding of **Metric MDS** by comparing it with Classical MDS and introducing the concept of **distance scaling**.

0.0.6 From Classical to Metric MDS

- In **Classical MDS (cMDS)**, we assume **Euclidean distances** and try to satisfy:

$$d_{ij} \approx \hat{d}_{ij} = \|x_i - x_j\|_2$$

- **Metric MDS** generalizes this idea:
  - Instead of matching exact Euclidean distances,
  - We allow a **monotonic transformation** of the dissimilarities:

$$\hat{d}_{ij} \approx f(d_{ij})$$

  where f is a monotonic (order-preserving) function.

This flexibility allows Metric MDS to work with **any valid metric** (Euclidean, Manhattan, Cosine, Hamming, etc.).

0.0.7 Distance Scaling & Stress

- This process of fitting the distances to the dissimilarities is called **distance scaling**.

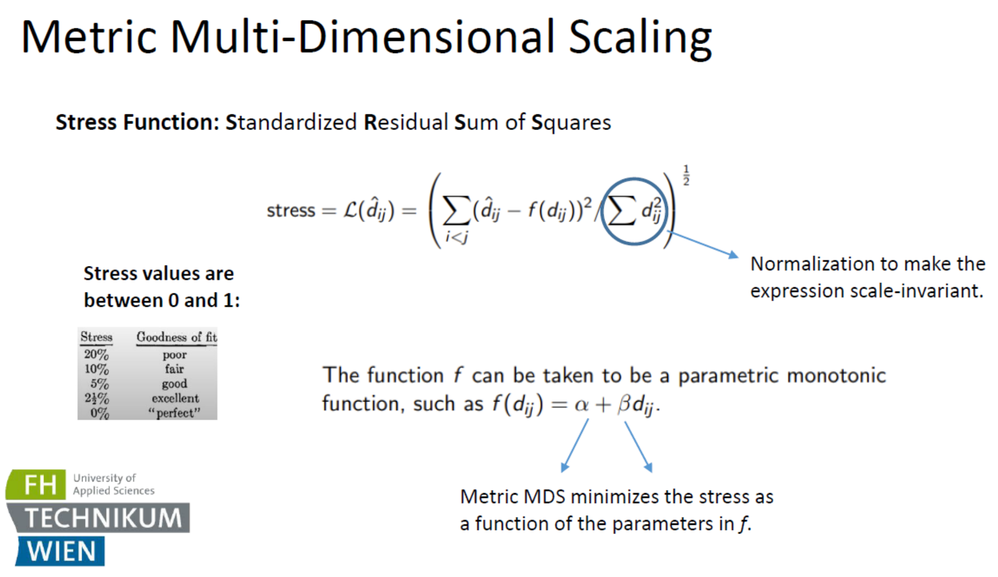
- It becomes an **optimization problem**: We try to **minimize a stress function** — a measure of how much distortion exists between input dissimilarities and resulting distances.
- Solved using iterative methods.

0.0.8 Key Distinction

- Use **Metric MDS** when dissimilarities d_{ij} are **quantitative** (e.g., real distances, scores).
- Use **Non-metric MDS** when dissimilarities are **qualitative** (e.g., ranks, preferences).

The image on the right shows various **distance metrics** (Euclidean, Cosine, Hamming, Jaccard, etc.) that can be used in Metric MDS.

Ready for the next slide: **Non-metric MDS**.



Metric Multi-Dimensional Scaling

Stress Function: Standardized **R**esidual **S**um of **S**quares

$$\text{stress} = \mathcal{L}(\hat{d}_{ij}) = \left(\sum_{i<j} (\hat{d}_{ij} - f(d_{ij}))^2 \Big/ \boxed{\sum d_{ij}^2} \right)^{\frac{1}{2}}$$

Normalization to make the expression scale-invariant.

Stress values are between 0 and 1:

Stress	Goodness of fit
20%	poor
10%	fair
5%	good
$2\frac{1}{2}$%	excellent
0%	"perfect"

The function f can be taken to be a parametric monotonic function, such as $f(d_{ij}) = \alpha + \beta d_{ij}$.

Metric MDS minimizes the stress as a function of the parameters in f.

FH University of Applied Sciences TECHNIKUM WIEN

Slide 13: Metric MDS – The Stress Function

Let's now take a closer look at the **stress function**, which is the core of how Metric MDS works.

0.0.9 What is the Stress Function?

The **stress function** quantifies the **discrepancy** between the actual distances in the low-dimensional space ($\hat{d}_{ij}$) and the transformed dissimilarities $f(d_{ij})$.

It is defined as:

$$\text{stress} = \mathcal{L}(\hat{d}_{ij}) = \left(\frac{\sum_{i<j} (\hat{d}_{ij} - f(d_{ij}))^2}{\sum d_{ij}^2} \right)^{1/2}$$

- The **numerator** captures how far the embedded distances are from the transformed dissimilarities.
- The **denominator** is a normalization term — making the expression **scale-invariant** (independent of units or magnitude).

0.0.10 What is $f(d_{ij})$?

- f is a **monotonic, parametric function** — often linear:

$$f(d_{ij}) = \alpha + \beta d_{ij}$$

- Metric MDS **optimizes** α and β (or more complex forms of f) to **minimize the stress**.

0.0.11 Interpreting Stress Values

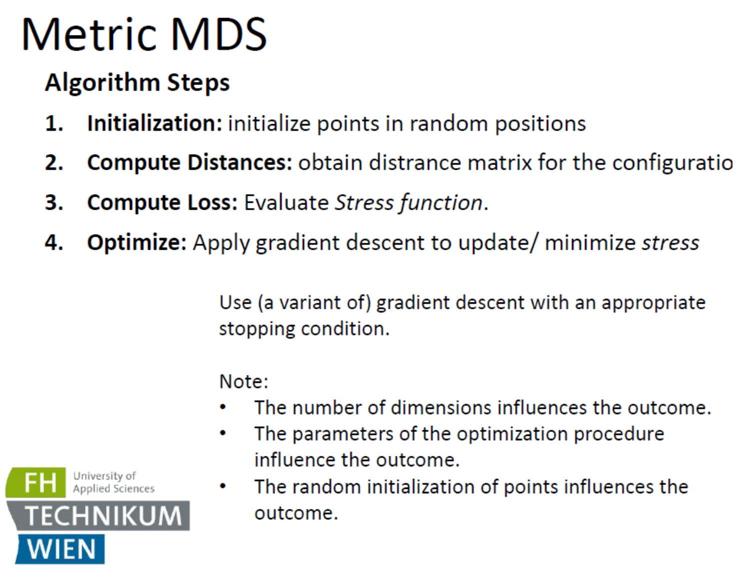
Stress	Fit Quality
$> 20\%$	Poor
~10%	Fair
~5%	Good
$< 2.5\%$	Excellent
0%	Perfect (ideal)

The **lower** the stress, the **better the embedding** preserves the original relationships.

So to recap:

> **Metric MDS** uses iterative optimization to find a low-dimensional configuration that minimizes **stress** — preserving **quantitative** dissimilarity information as faithfully as possible.

Next up: **Non-metric MDS** — even more flexible. Ready when you are.



Metric MDS

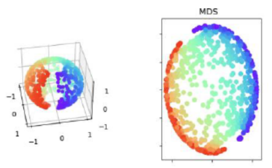
Algorithm Steps

1. **Initialization:** initialize points in random positions
2. **Compute Distances:** obtain distrance matrix for the configuratio
3. **Compute Loss:** Evaluate *Stress function*.
4. **Optimize:** Apply gradient descent to update/ minimize *stress*

Use (a variant of) gradient descent with an appropriate stopping condition.

Note:

- The number of dimensions influences the outcome.
- The parameters of the optimization procedure influence the outcome.
- The random initialization of points influences the outcome.



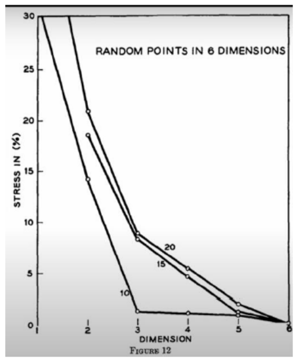


FIGURE 12

Slide 14: Metric MDS – Algorithm Steps

Let's now summarize the **iterative optimization process** that Metric MDS uses to minimize the stress.

0.0.12 Algorithm Steps

1. **Initialization** Randomly initialize the positions of the data points in the target low-dimensional space (e.g., 2D or 3D).
2. **Compute Distances** From the current configuration, compute the **Euclidean distances** $\hat{d}_{ij}$ between all pairs.
3. **Compute Loss (Stress)** Evaluate the **stress function** between the current distances $\hat{d}_{ij}$ and the transformed dissimilarities $f(d_{ij})$.
4. **Optimize** Use **gradient descent** (or a variant) to update the point positions and reduce the stress. Stop when the improvement is below a given threshold or a maximum number of iterations is reached.

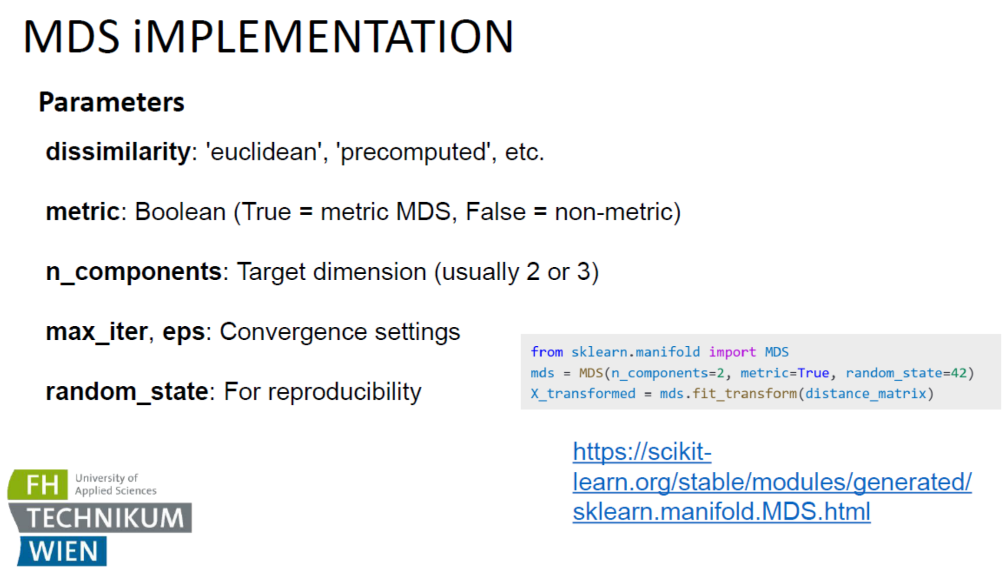
0.0.13 Important Notes:

- The **number of target dimensions** (e.g., 2, 3) affects the quality of the embedding.
- The **parameters of the optimization** (like learning rate or stopping criteria) influence the result.
- The **initial random layout** of points can impact convergence — multiple runs may yield different local minima.

The plot in the lower right corner shows how **stress decreases** as the **number of dimensions increases**, improving the fit.

The upper right visualizations show how a complex 3D structure is flattened to 2D while preserving neighborhoods — a common MDS application.

Next, we'll look at **Non-metric MDS**, which drops even more assumptions. Ready when you are.



MDS iMPLEMENTATION

Parameters

dissimilarity: 'euclidean', 'precomputed', etc.

metric: Boolean (True = metric MDS, False = non-metric)

n_components: Target dimension (usually 2 or 3)

max_iter, **eps**: Convergence settings

random_state: For reproducibility

```
from sklearn.manifold import MDS
mds = MDS(n_components=2, metric=True, random_state=42)
X_transformed = mds.fit_transform(distance_matrix)
```

https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html

FH University of Applied Sciences TECHNIKUM WIEN

Slide 15: MDS Implementation in Python (Scikit-learn)

Let's wrap up by looking at how to **implement MDS in practice** using Python and Scikit-learn.

0.0.14 Key Parameters in `sklearn.manifold.MDS`

- `dissimilarity`: Choose how distances are computed. Options:
  - `'euclidean'`: Compute Euclidean distances internally.
  - `'precomputed'`: Supply your own distance or dissimilarity matrix.
- `metric`:
  - `True` → use **Metric MDS**
  - `False` → use **Non-Metric MDS**
- `n_components`: The number of output dimensions (usually **2** or **3** for visualization).
- `max_iter`, `eps`: Control **convergence criteria** (max iterations, tolerance for stress minimization).
- `random_state`: Ensures **reproducibility** of the embedding (important for presentations or comparisons).

0.0.15 Sample Code

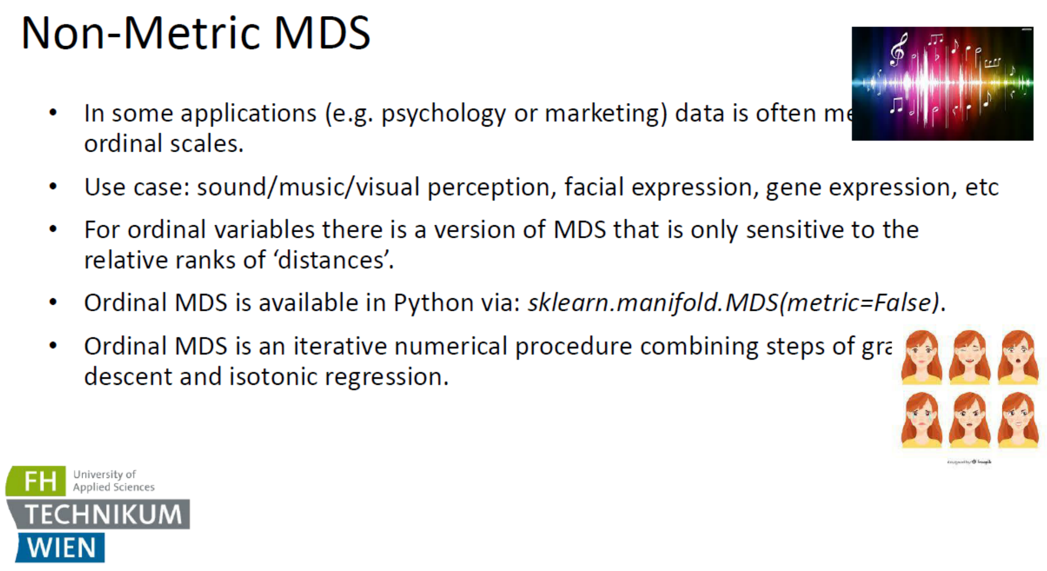
```
from sklearn.manifold import MDS

mds = MDS(n_components=2, metric=True, random_state=42)
X_transformed = mds.fit_transform(distance_matrix)
```

This runs **metric MDS** with 2 output dimensions and a fixed random seed.

Full documentation: Scikit-learn MDS Documentation

That concludes the implementation part. Please send the next slide if there's more — or let me know if we're at the end of the presentation.



Non-Metric MDS

- In some applications (e.g. psychology or marketing) data is often me ordinal scales.
- Use case: sound/music/visual perception, facial expression, gene expression, etc
- For ordinal variables there is a version of MDS that is only sensitive to the relative ranks of 'distances'.
- Ordinal MDS is available in Python via: *sklearn.manifold.MDS(metric=False)*.
- Ordinal MDS is an iterative numerical procedure combining steps of gra descent and isotonic regression.

FH University of Applied Sciences TECHNIKUM WIEN

Slide 16: Non-Metric MDS

We now come to the most flexible variant: **Non-Metric MDS**, also known as **Ordinal MDS**.

0.0.16 When is Non-Metric MDS useful?

In some fields like **psychology**, **marketing**, or **perceptual studies**, we often deal with **ordinal data** — where we know **ranking** (e.g., preferences or similarity order), but not **exact distances**.

0.0.17 Use Cases Include:

- Auditory perception (sound/music similarity)

- Visual perception (e.g., facial expressions)
- Gene expression levels
- Emotion or preference rating tasks

0.0.18 Key Idea:

- Instead of preserving exact distances, Non-Metric MDS tries to preserve the **order** of dissimilarities:
  - If item A is more similar to B than to C, this should be reflected in the plot.
- It applies a **monotonic transformation** to the dissimilarities — not necessarily linear — and focuses on **ranking**.

0.0.19 How It Works

- Available in scikit-learn as:

  ```
  MDS(metric=False)
  ```

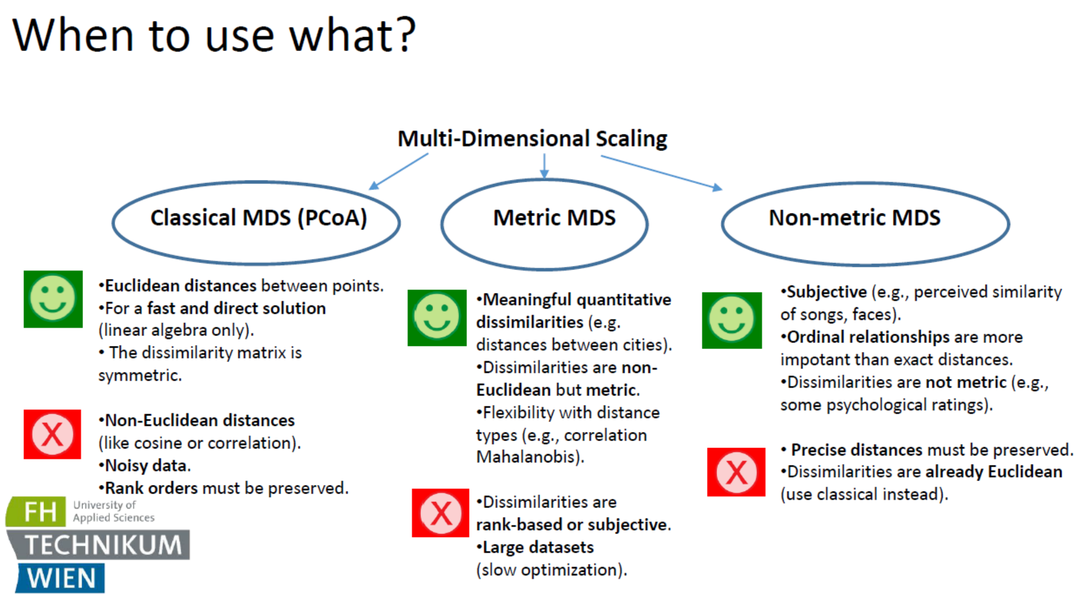
- Internally, it uses:
  - **Gradient descent** to minimize stress
  - **Isotonic regression** to find the best monotonic transformation of dissimilarities

---

In short:

> **Non-Metric MDS** is great when you have only **ordinal data** and want to visualize **relative similarity**, without assuming that your dissimilarities follow any specific metric.

Let me know if there's one final slide (e.g. summary or wrap-up) — or if we should now conclude the presentation.



When to use what?

Multi-Dimensional Scaling

Classical MDS (PCoA)

🙂
- **Euclidean distances** between points.
- For a **fast and direct solution** (linear algebra only).
- The dissimilarity matrix is symmetric.

❌
- **Non-Euclidean distances** (like cosine or correlation).
- **Noisy data.**
- **Rank orders** must be preserved.

Metric MDS

🙂
- **Meaningful quantitative dissimilarities** (e.g. distances between cities).
- Dissimilarities are **non-Euclidean but metric.**
- Flexibility with distance types (e.g., correlation Mahalanobis).

❌
- Dissimilarities are **rank-based or subjective.**
- **Large datasets** (slow optimization).

Non-metric MDS

🙂
- **Subjective** (e.g., perceived similarity of songs, faces).
- **Ordinal relationships** are more impotant than exact distances.
- Dissimilarities are **not metric** (e.g., some psychological ratings).

❌
- **Precise distances** must be preserved.
- Dissimilarities are **already Euclidean** (use classical instead).

FH University of Applied Sciences TECHNIKUM WIEN

Slide 17: When to Use What?

Let's conclude this presentation with a **comparison guide** to help you choose the right **MDS variant** based on your data and goals.

0.0.20 Classical MDS (PCoA)

Use when:

- Dissimilarities are **Euclidean**
- You want a **fast, direct solution** (uses only linear algebra)
- Your distance matrix is **symmetric**

Avoid when:

- Distances are **non-Euclidean** (like cosine)
- The data is **noisy** or contains rank-only information

0.0.21 Metric MDS

Use when:

- You have **quantitative**, meaningful dissimilarities (e.g., distances between cities, physical measurements)
- Dissimilarities follow a **metric** but are **not Euclidean**
- You need **flexibility** in distance types (e.g., correlation, Mahalanobis)

Avoid when:

- The dissimilarities are **ordinal** or **subjective**

- Dataset is **very large** (optimization becomes slow)

0.0.22 Non-Metric MDS

Use when:

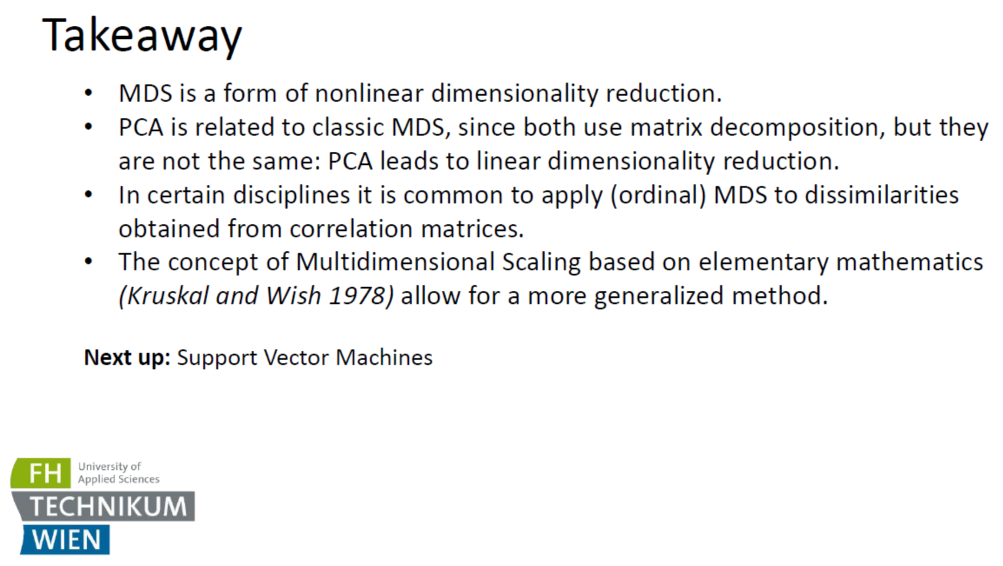
- The dissimilarities are **subjective or ordinal** (e.g., perceived similarity of songs, facial expressions)
- You care about **relative rankings**, not exact distances
- Dissimilarities may **not obey metric properties**

Avoid when:

- You need to **preserve exact distances**
- Your dissimilarities are **already Euclidean** — then prefer Classical MDS

Final tip: Choose your MDS technique based on whether your input dissimilarities are **metric**, **Euclidean**, or **ordinal** — and how important **speed**, **accuracy**, or **interpretability** is in your application.

Thank you for your attention. Let me know if you'd like a summary slide or Q&A support.



Takeaway

- MDS is a form of nonlinear dimensionality reduction.
- PCA is related to classic MDS, since both use matrix decomposition, but they are not the same: PCA leads to linear dimensionality reduction.
- In certain disciplines it is common to apply (ordinal) MDS to dissimilarities obtained from correlation matrices.
- The concept of Multidimensional Scaling based on elementary mathematics *(Kruskal and Wish 1978)* allow for a more generalized method.

Next up: Support Vector Machines

FH University of Applied Sciences TECHNIKUM WIEN

Final Slide: Takeaway

Let's close with a summary of the key points about **Multidimensional Scaling (MDS)**.

0.0.23 Key Takeaways

- **MDS** is a form of **nonlinear dimensionality reduction**, especially useful when we want to visualize or interpret **dissimilarities** between objects.
- While **PCA** and **Classical MDS** both rely on **matrix decomposition**, they are **not the same**:
  - **PCA** works on feature covariances $\rightarrow$ **linear** reduction
  - **MDS** works on distance or dissimilarity matrices $\rightarrow$ **nonlinear** reduction
- In some fields like **psychology, sociology, or marketing**, it's common to apply **ordinal (non-metric) MDS** to dissimilarities derived from **correlation matrices** or **subjective assessments**.
- The work by **Kruskal and Wish (1978)** laid the foundation for modern MDS — introducing a **generalized, flexible method** for visualizing similarities in data.

Next Topic: Support Vector Machines (SVMs)

We'll now transition to SVMs — a supervised learning technique especially powerful for classification problems in high-dimensional spaces.

Thank you for your attention! Let me know if you'd like a one-slide summary, quiz, or discussion follow-up.