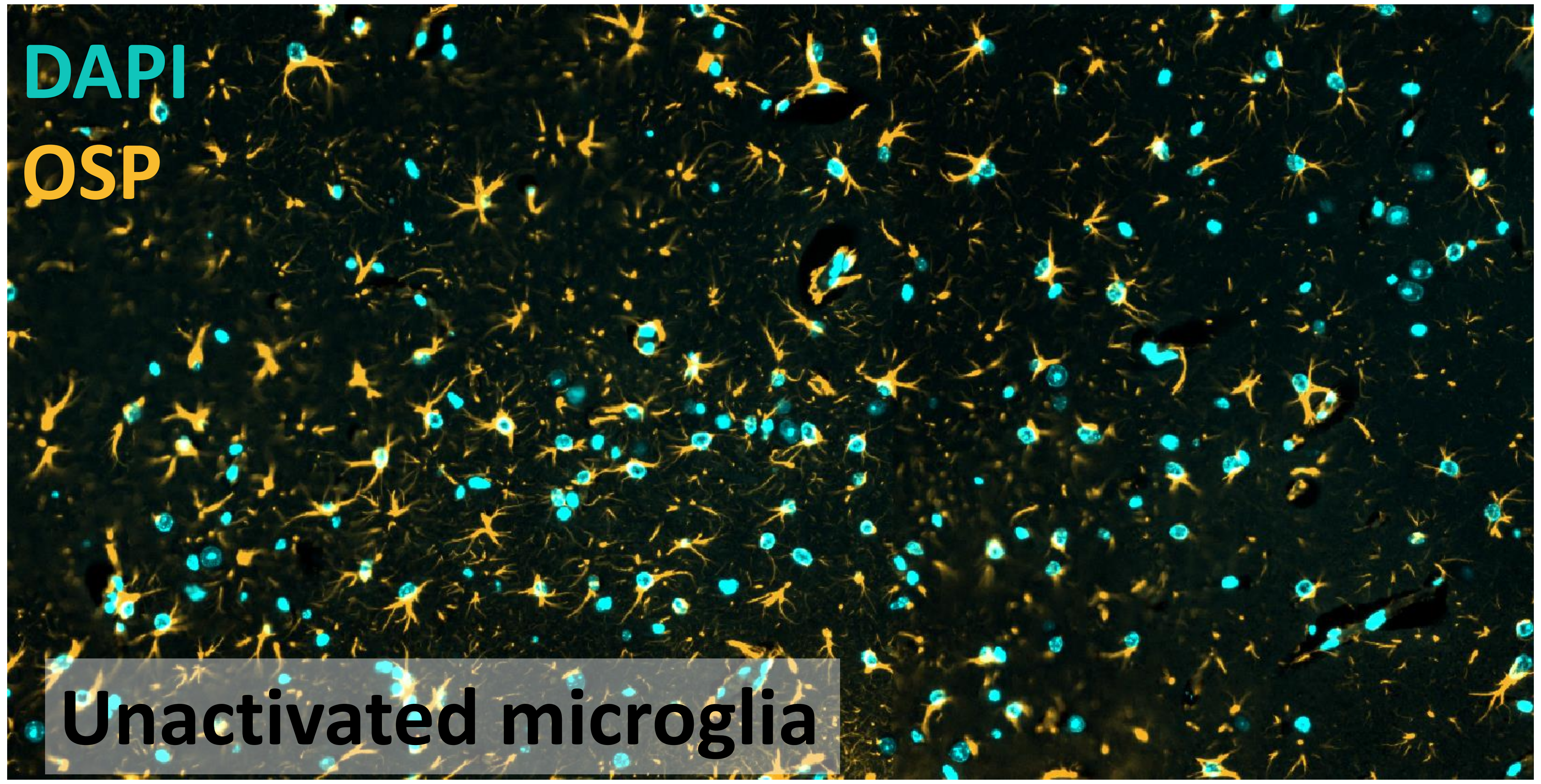
Dimensionality reduction-
PCA & UMAP

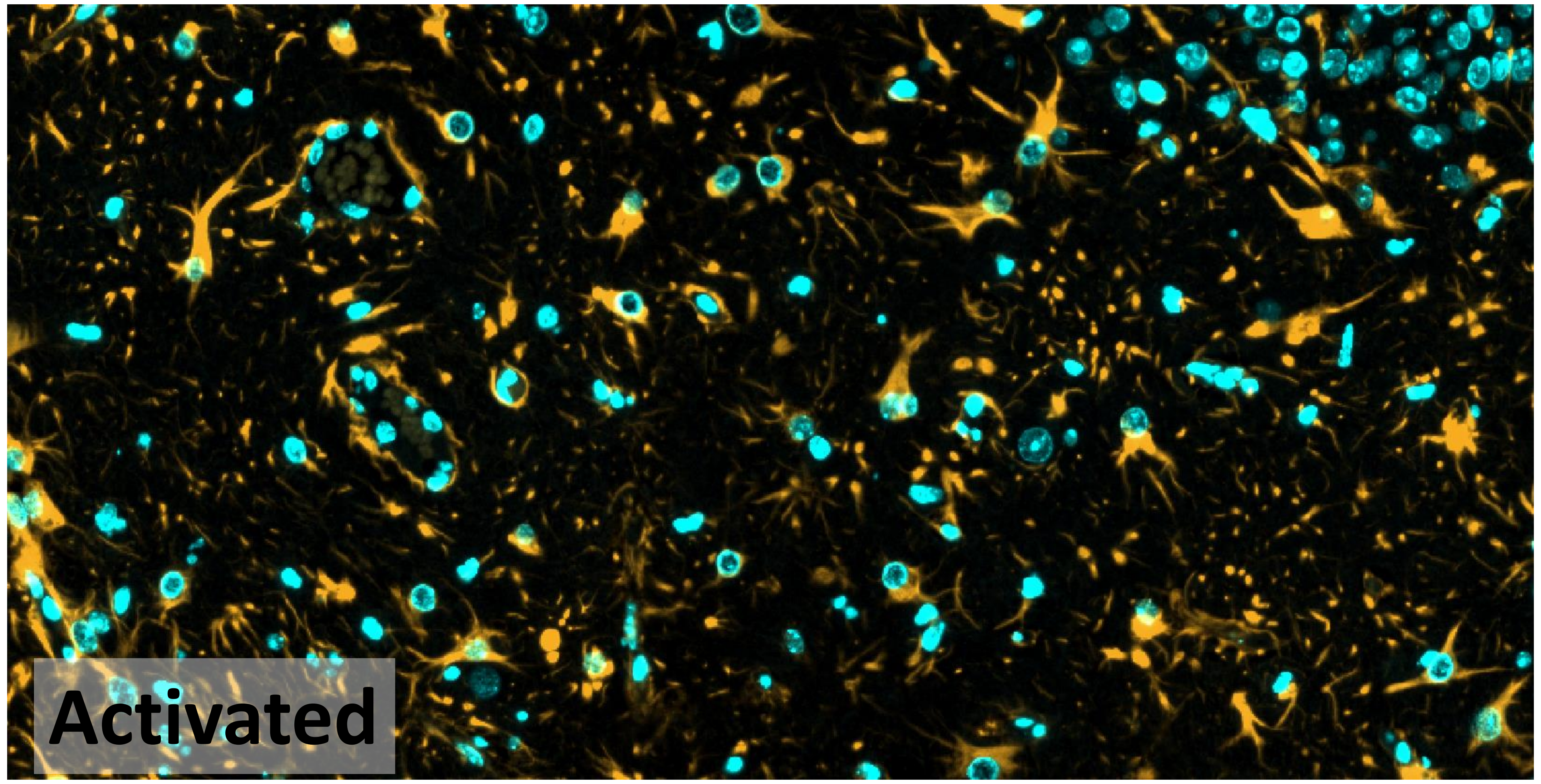
Johannes Müller

Introduction

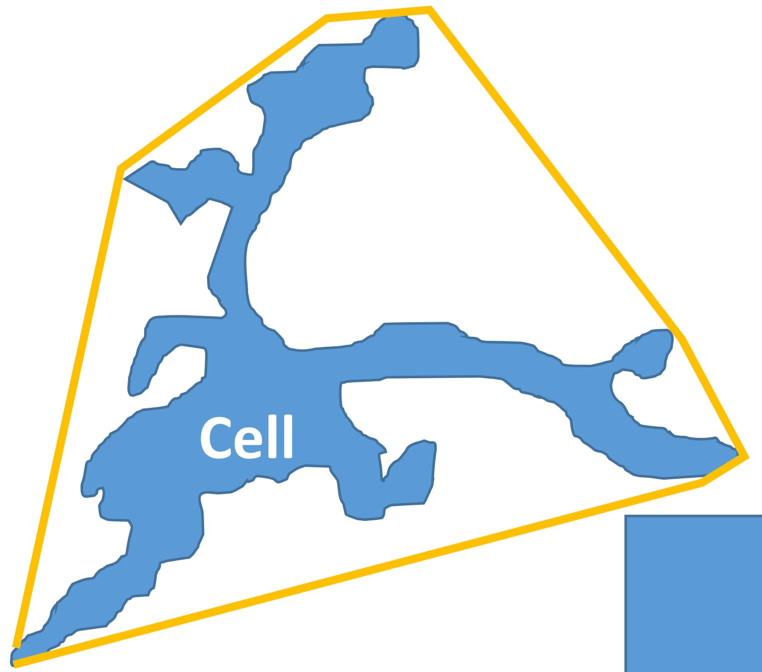
Ideal situation: Biological property is related to a known, measurable feature in image data

Example: Unactivated vs. activated microglia in mouse brain





Existing scores: “Ramification index” (Wittekind et al., 2022)



$$Rammification\ index = \frac{A_C}{A_P}$$

That’s actually just…?

Solidity | Perimeter | Circularity

Ideal workflow ⇒

Get Data
↓
Measure defined score/features
↓
Compare conditions
↓
Be done ☺

Introduction

More typical situation:

- We expect or know of a biological effect (e.g., through external cues, cell growth stages, etc.)
- We do not know how this effect can be measured or how it manifests itself

Source: Mauricio Rocha Martins, Norden lab, MPI CBG

Example:

Developing zebrafish eye

Hypothesis: Cells develop differently depending on where they are

Get Data

⬇

Measure …what exactly?

⬇

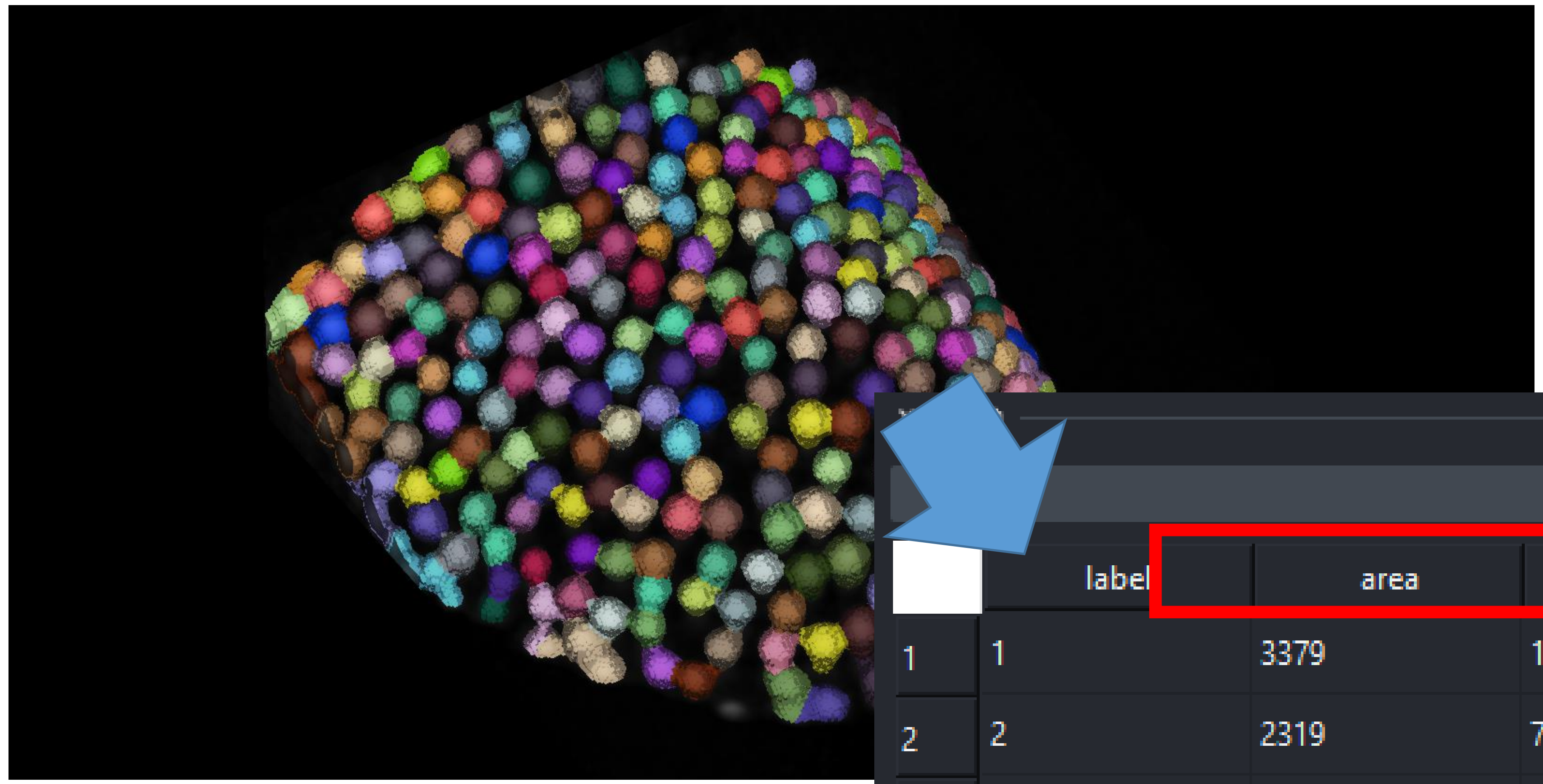
Compare …what exactly?

⬇

Be stuck ☹

Introduction

We can measure tons of features but still have no idea about what’s happening!



Which of these features reflects interesting biology?

Properties of Result of voronoi_otsu_labeling (clesperant

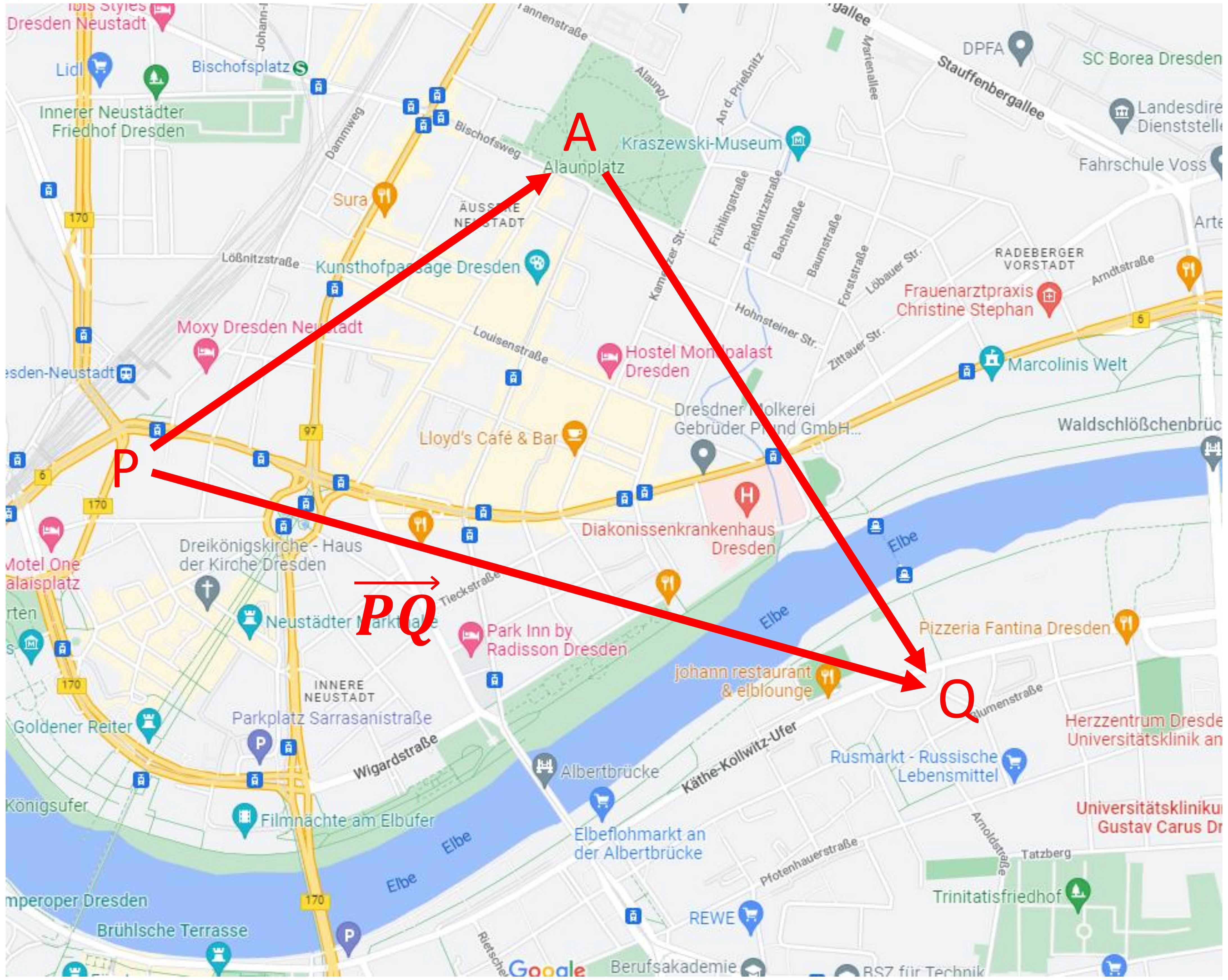
Copy to clipboard | Save as csv...

	label	area	bbox_area	convex_area	quivalent_diamete	max_intensity	mean_intensity	min_intensity	solidity	extent	eret_diameter_ma:	local_centroid-0	l
1	1	3379	13949	5120	18.61786412639...	613.0	345.6717963894...	259.0	0.6599609375	0....	37.3496987939662	15.77952056821...	18
2	2	2319	7448	3491	16.42230229224...	421.0	297.8434670116...	240.0	0....	0....	38.65229618017...	4....	17
3	3	2304	14415	4281	16.38681751812...	456.0	300.8298611111...	245.0	0....	0....	34.19064199455...	17.73828125	13
4	4	3278	13804	5139	18.43048549951...	467.0	316.1446003660...	249.0	0....	0....	34.84250278036...	15.52287980475...	10
5	5	1501	3315	1681	14.20563625190...	458.0	302.147235176549	236.0	0....	0....	17.97220075561...	6....	6.
6	6	2341	6061	2714	16.47407088948...	594.0	355.4446817599...	261.0	0....	0....	30.67572330035...	16.54250320375...	6.
7	7	1725	3584	1940	14.87979081163...	568.0	343.7866666666...	257.0	0....	0....	17.72004514666...	7.80463768115942	7.
8	8	1502	3840	1753	14.20879025650...	431.0	290.0659121171...	235.0	0....	0....	18.57417562100...	8....	6.
9	9	1602	4080	1894	14.51737058294...	475.0	297.8008739076...	241.0	0....	0....	18.70828693386...	8....	8.
10	10	1395	3600	1624	13.86304166283...	424.0	304.8494623655...	247.0	0....	0.3875	17.60681686165...	7....	7.
11	11	609	1100	697	10.51654029260...	323.0	274.2528735632...	241.0	0....	0....	13.45362404707...	3....	4.
12	12	1686	3757	1894	14.76679738567...	460.0	303.8303677342...	240.0	0....	0....	17.97220075561...	9....	7.
13	13	2157	5184	2531	16.03062694504...	576.0	339.990264255911	270.0	0....	0....	19.54482028569...	8....	8.
14	14	863	2340	1032	11.81237949737...	327.0	272.4449594438...	237.0	0....	0....	16.0312195418814	6....	5.

Basic concepts

Most common and intuitive space: **Euclidian space**

Practical example: (local) 2D space



Maps.google.de

Characteristics of Euclidian space:

- Distance between **P** and Q d(P,Q) is **symmetric:** d(P,Q) = d(Q,P)
- Distance between **P** and **Q** can be measured as the length ("norm") of the vector $\overrightarrow{PQ}$
- Distances satisfy the triangle inequality:

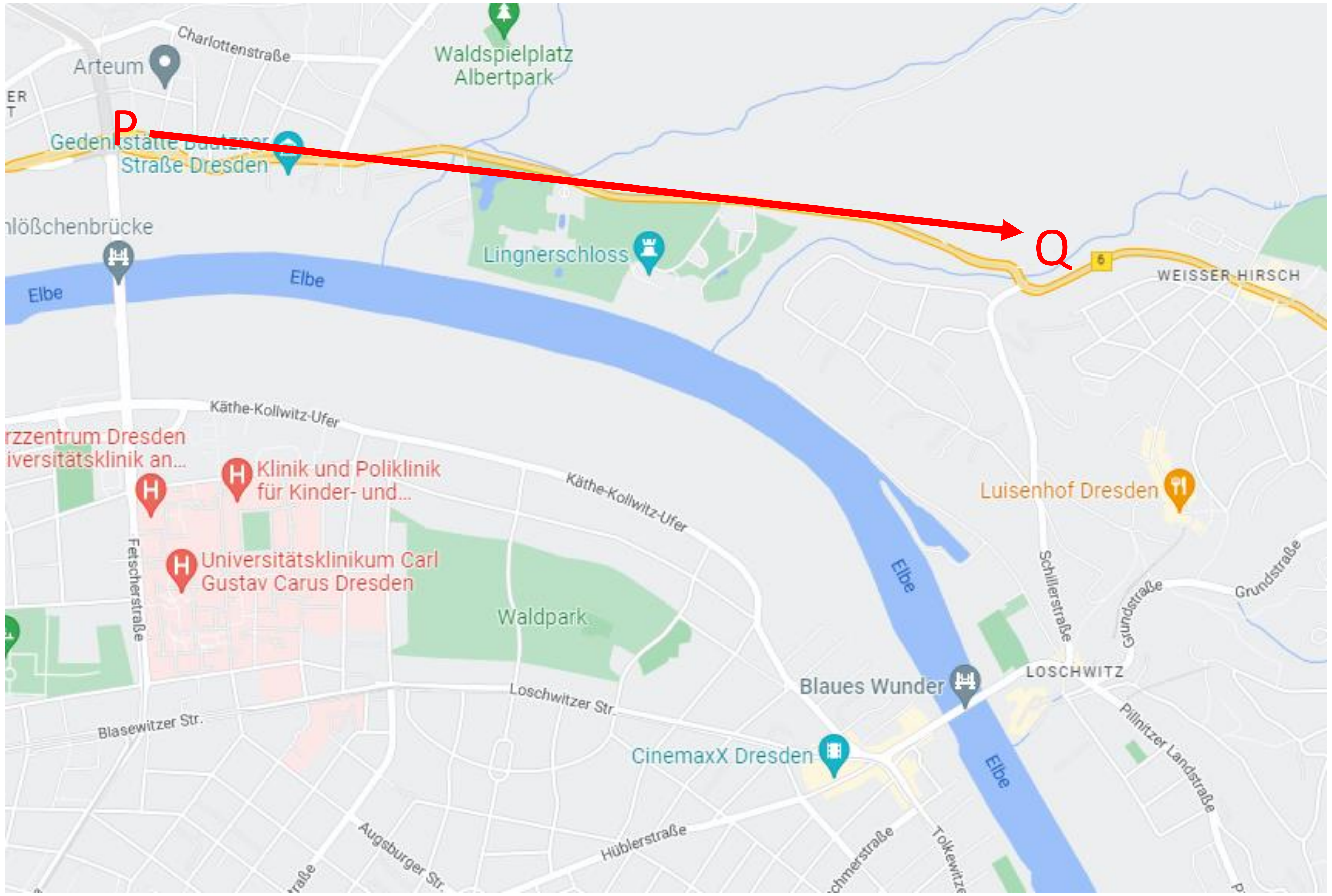
$$d(P,Q) \leq d(P,A) + d(A,Q)$$

In other words: There is no shorter paths between two points other than a straight line

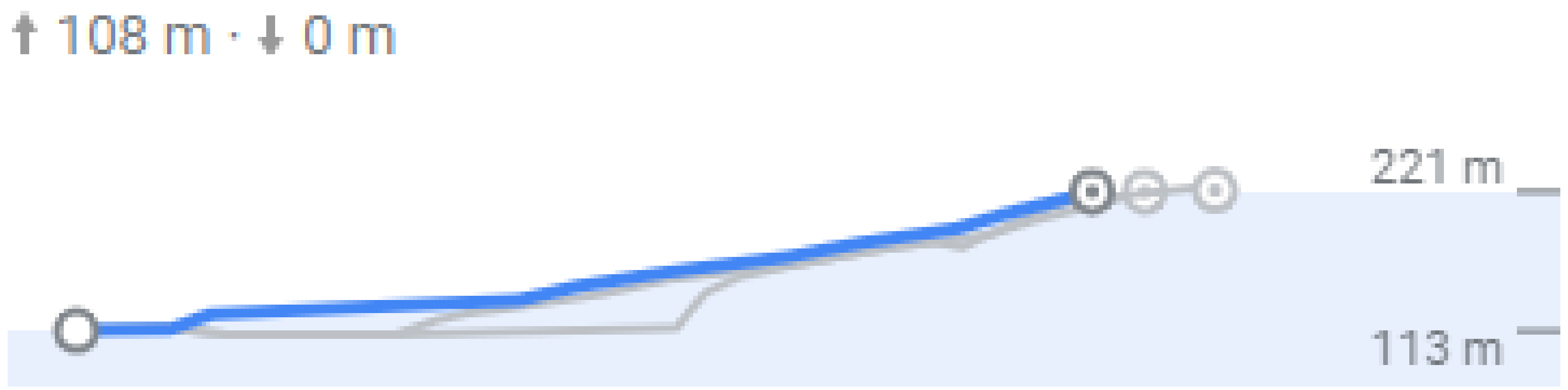
Basic concepts

How could **non-Euclidian** spaces look like? → It’s surprisingly intuitive!

- Non symmetric distances → $d(P,Q) \neq d(Q,P)$



Use travel time between P and Q as metric for distance



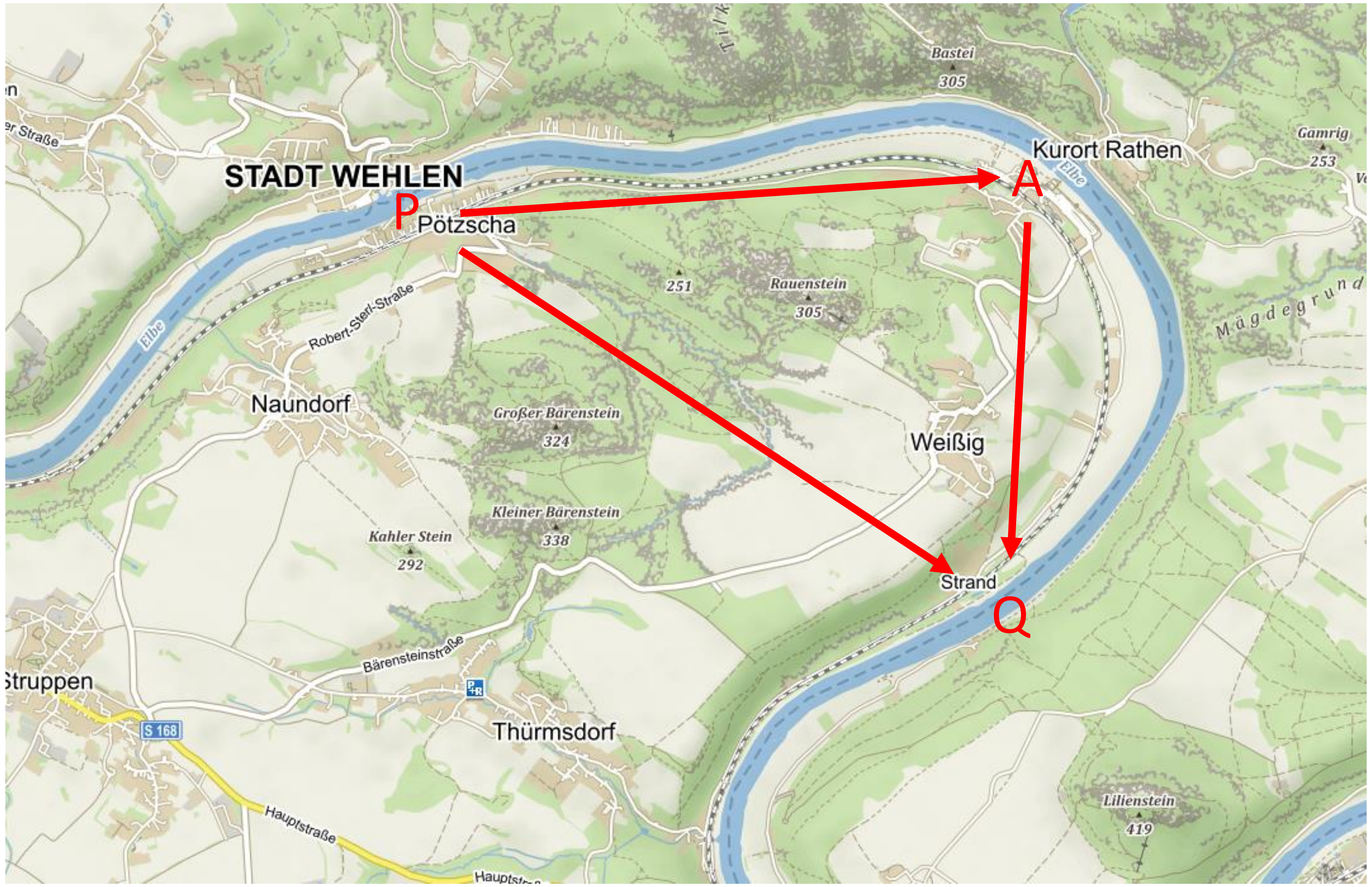
Travelling by bike/foot → it’s *much* faster to get from Q to P than vice versa
➔ d(P,Q) > d(Q, P)

Replacing distance between **P** and **Q** with time between **P** and **Q** makes Dresden non-Euclidian!

Basic concepts

How could **non-Euclidian** spaces look like? → It's surprisingly intuitive!

- Breaking the triangle inequality→$d(P,Q) > d(P,A) + d(A,Q)$

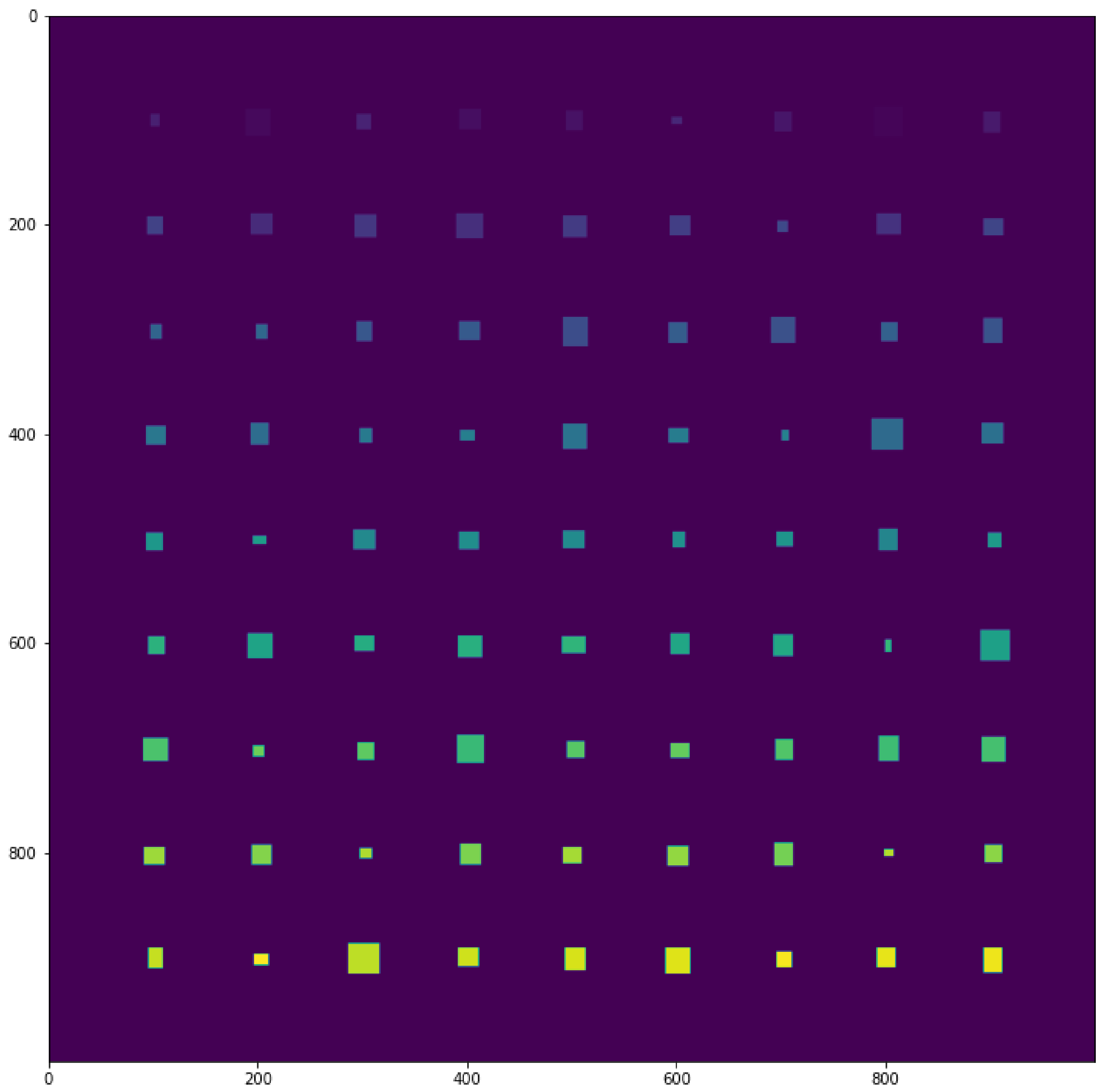


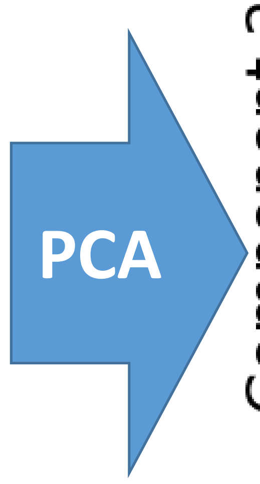
Use travel time between P and Q as metric for distance

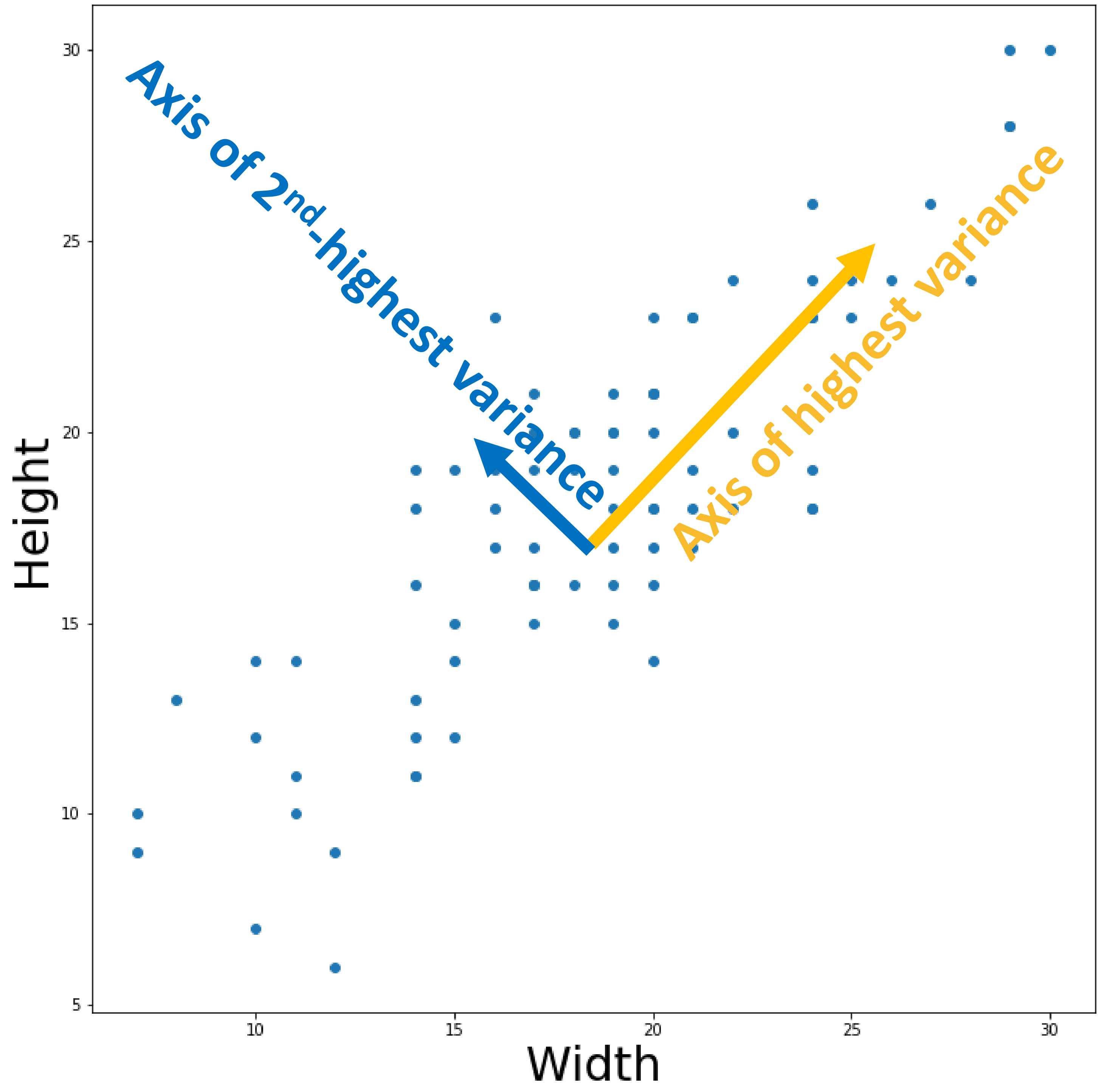
→ Travelling from Stadt Wehlen to Strand by bike is probably faster if you make a detour through Rathen

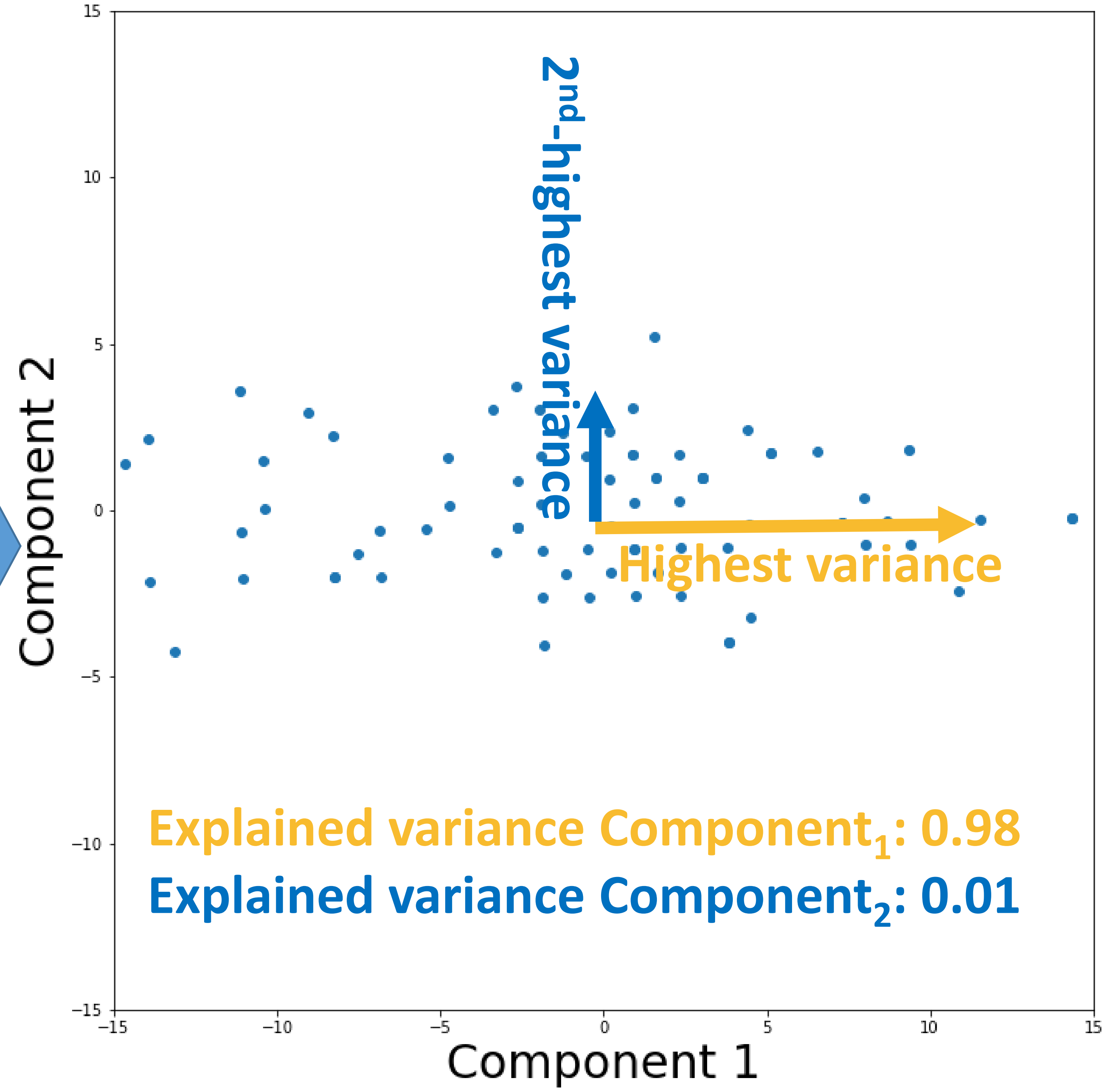
PCA: principal component analysis

Principal component analysis:
Decomposes data into linear combinations of features that explain the highest variance









Example: Squares of different size
→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

$$Component_1 = a_1 * width + b_1 * height$$
$$Component_2 = a_2 * width + b_2 * height$$

PCA: principal component analysis

- **Example:** Squares of different size

→ PCA transforms width/height measurements into a coordinate system that explains existing variance better

$$Component_1 = a_1 * width + a_2 * height$$
$$Component_2 = b_1 * width + b_2 * height$$

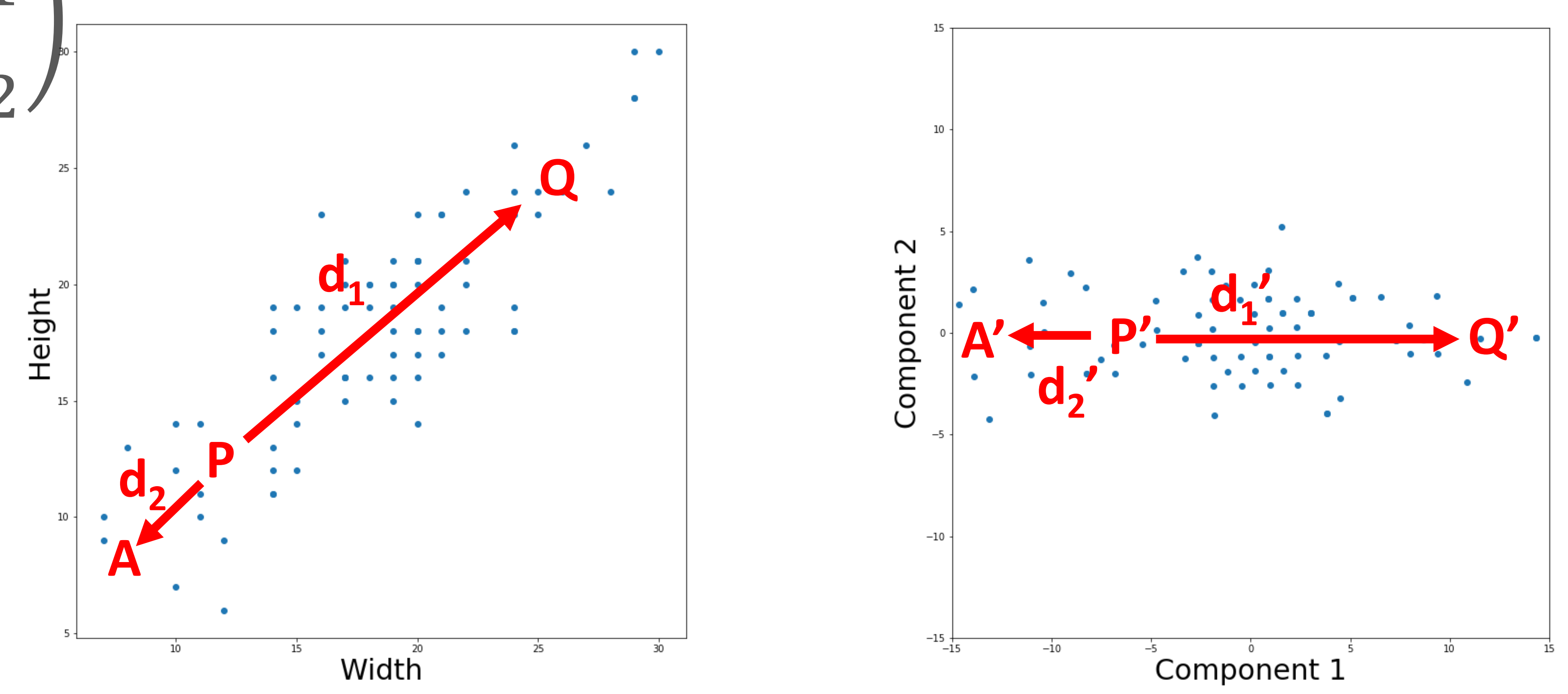
→ This is the result of a multiplication operation:

$$\underbrace{\begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}}_{T} * \begin{pmatrix} width \\ height \end{pmatrix} = \begin{pmatrix} Component_1 \\ Component_2 \end{pmatrix}$$

This is a linear operation!
Metrics remain meaningful

→ This works for any number of features!

$$T * \begin{pmatrix} feature_1 \\ \dots \\ feature_N \end{pmatrix} = \begin{pmatrix} Component_1 \\ \dots \\ Component_2 \end{pmatrix}$$

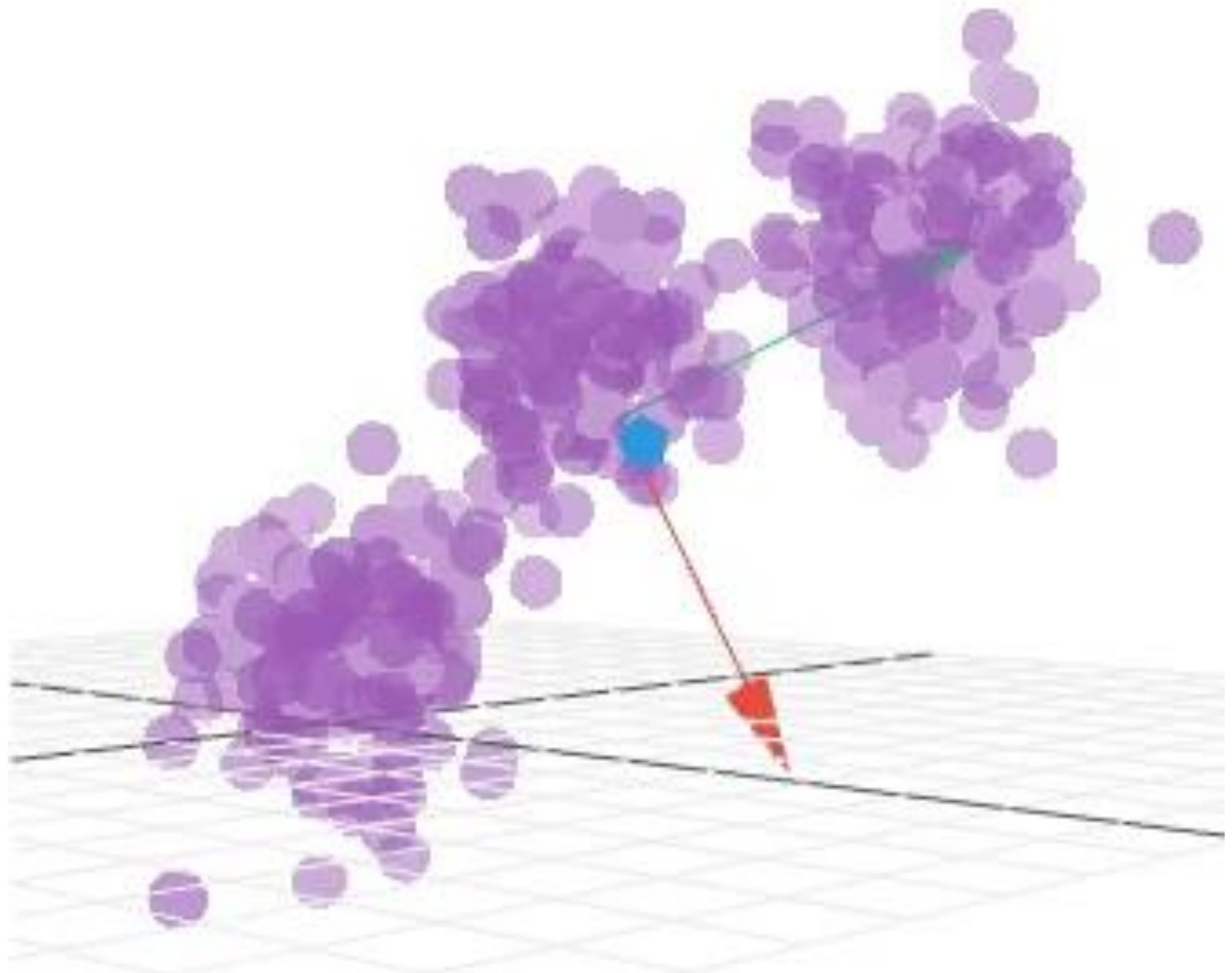


$d_1 > d_2$ ➔ $d_1' > d_2'$

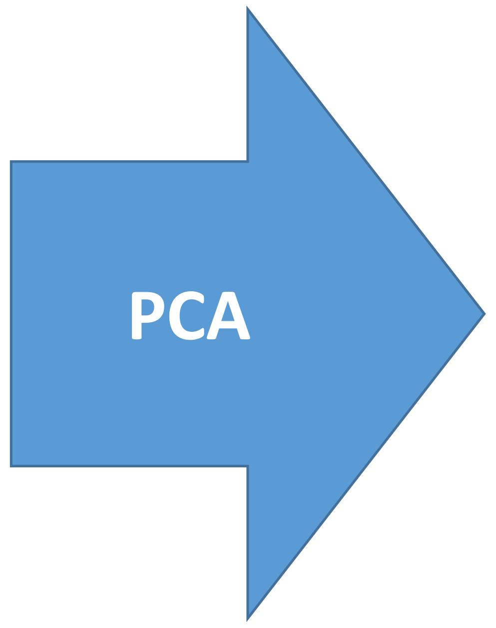
PCA: principal component analysis

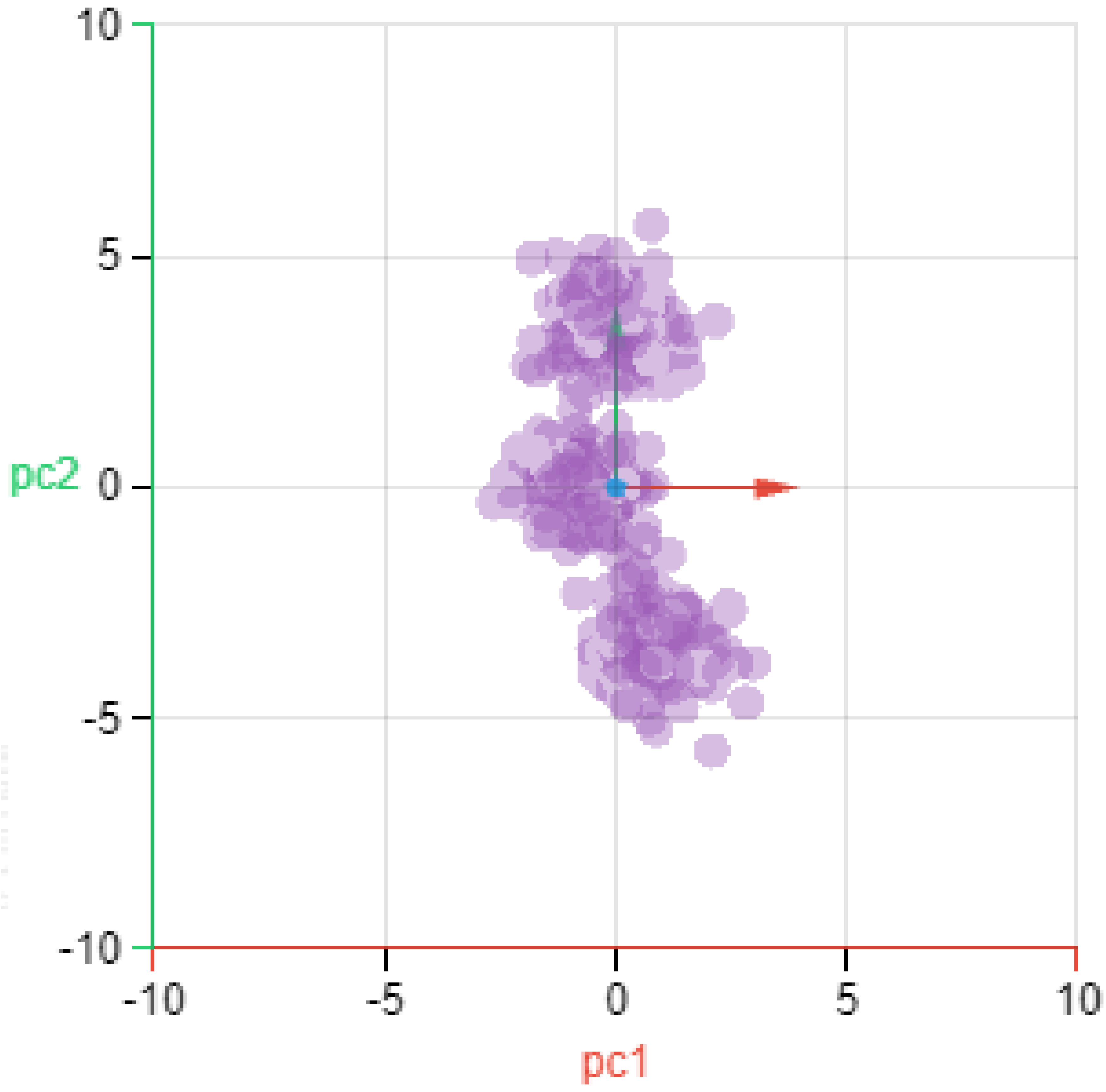
- **Example:** number of features = 3

→ This works for any number of features! $T * \begin{pmatrix} feature_1 \\ \dots \\ feature_N \end{pmatrix} = \begin{pmatrix} Component_1 \\ \dots \\ Component_2 \end{pmatrix}$



Source: https://setosa.io/ev/principal-component-analysis/





Two axes allow to get a good idea of groups in the data!

Important:
Always check the explained variance along the PCA component axes!

PCA in Python

Import package

```
from sklearn.decomposition import PCA
```

Apply PCA

```
pca = PCA(n_components=2)
pca.fit(data)
```

```
PCA
PCA(n_components=2)
```

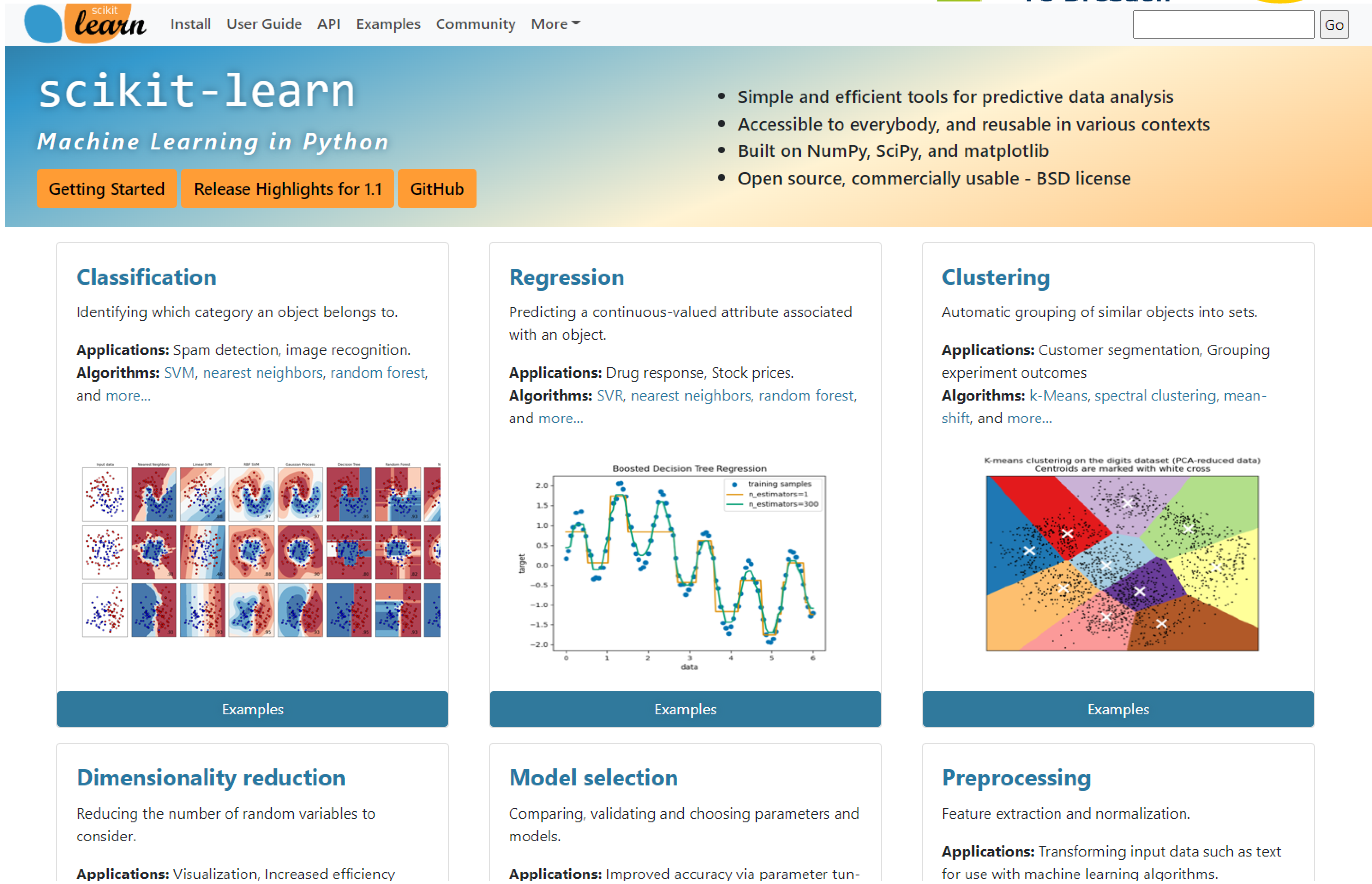
Transform data into transformed space

```
transformed_data = pca.transform(data)
```

Inspect explained variance

```
pca.explained_variance_ratio_
```

```
array([0.98773142, 0.01226858])
```



UMAP: Basic concepts

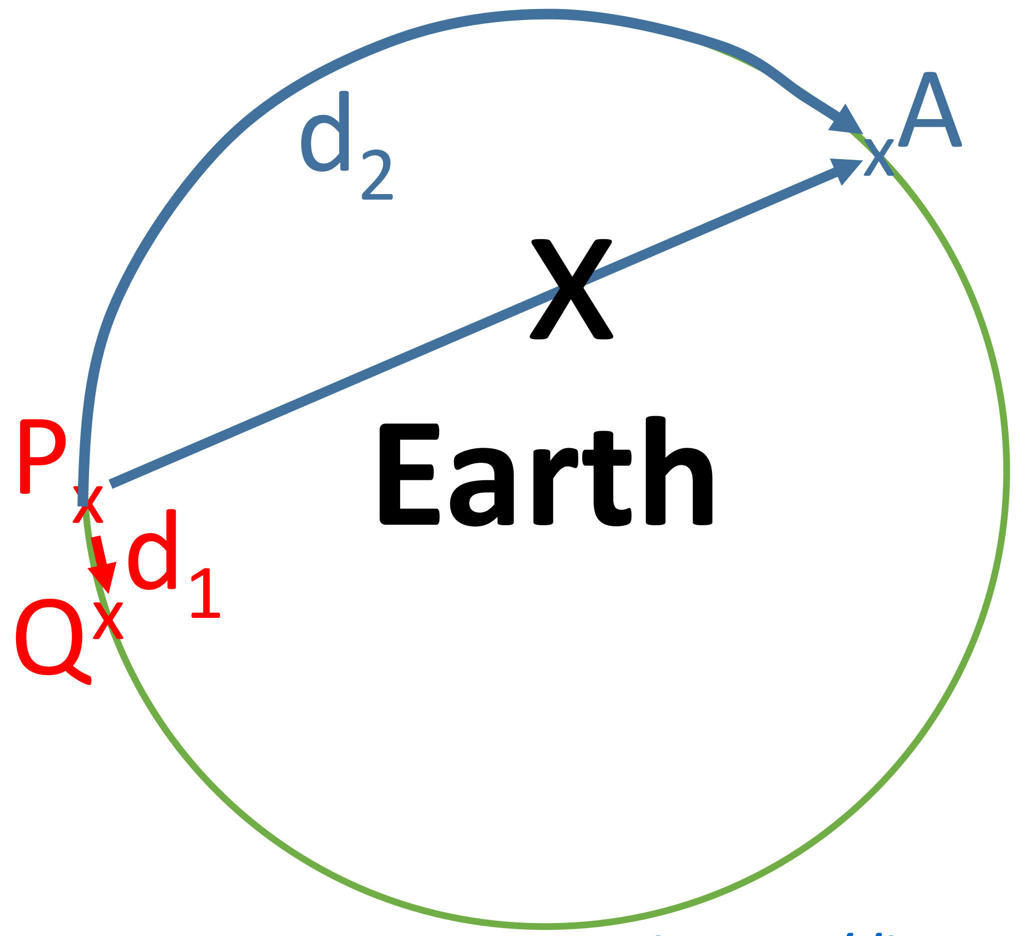
More complex concept: **Manifolds**

From Wikipedia: "*In mathematics, a manifold is a topological space that locally resembles Euclidean space near each point.*"

…Is this map Euclidian? Yes No

→ It isn't! The two vectors $\overrightarrow{PQ}$ and $\overrightarrow{AB}$ have the same length, but the real distances (the *norm*) of both are completely different!

→ Cropping a small piece from the map gives us a local Euclidian space, where the previous assumptions hold.

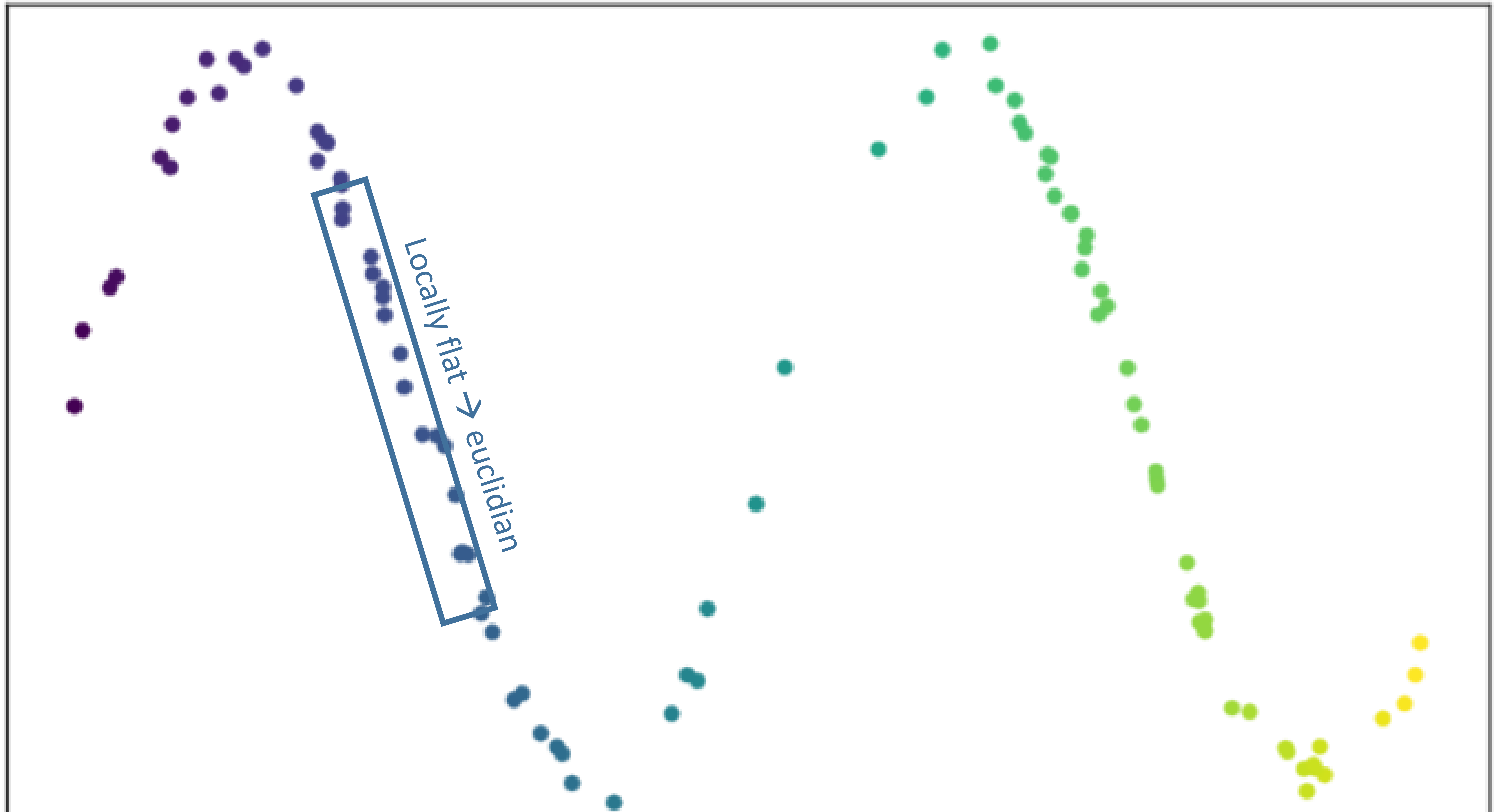


d_1: Euclidian
d_2: Non-Euclidian

Mapy.cz, https://licence.mapy.cz/?doc=mapy_pu

UMAP

Initial situation: Our data suggests an underlying structure ("topology") but we don't have a model for it



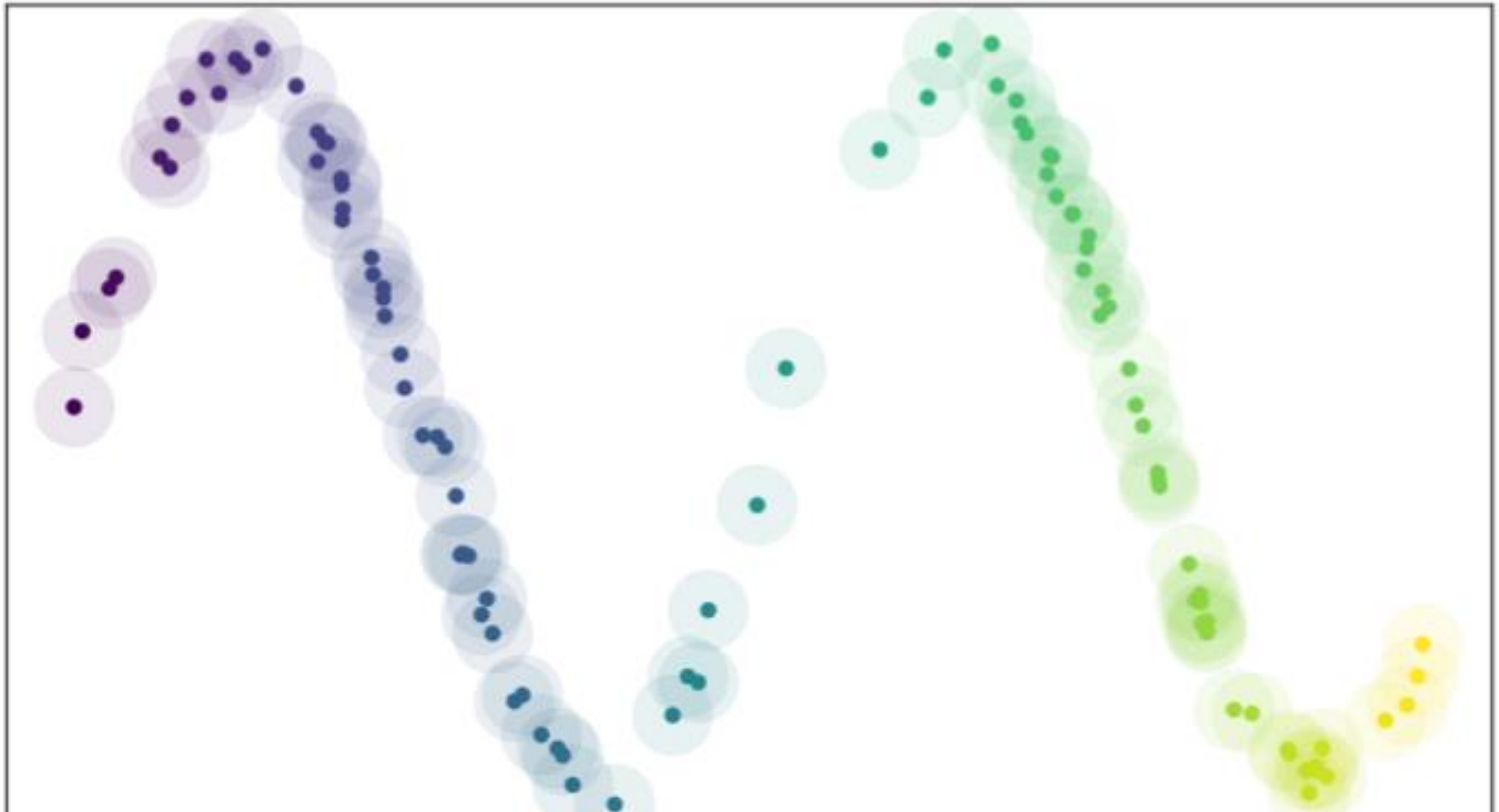
Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

Idea:
Reconstruct underlying topology to identify a space that best explains differences in our data

Approach:
Identify neighboring points in data by setting a neighborhood radius

UMAP

Initial situation: Our data suggests an underlying structure ("topology") but we don't have a model for it



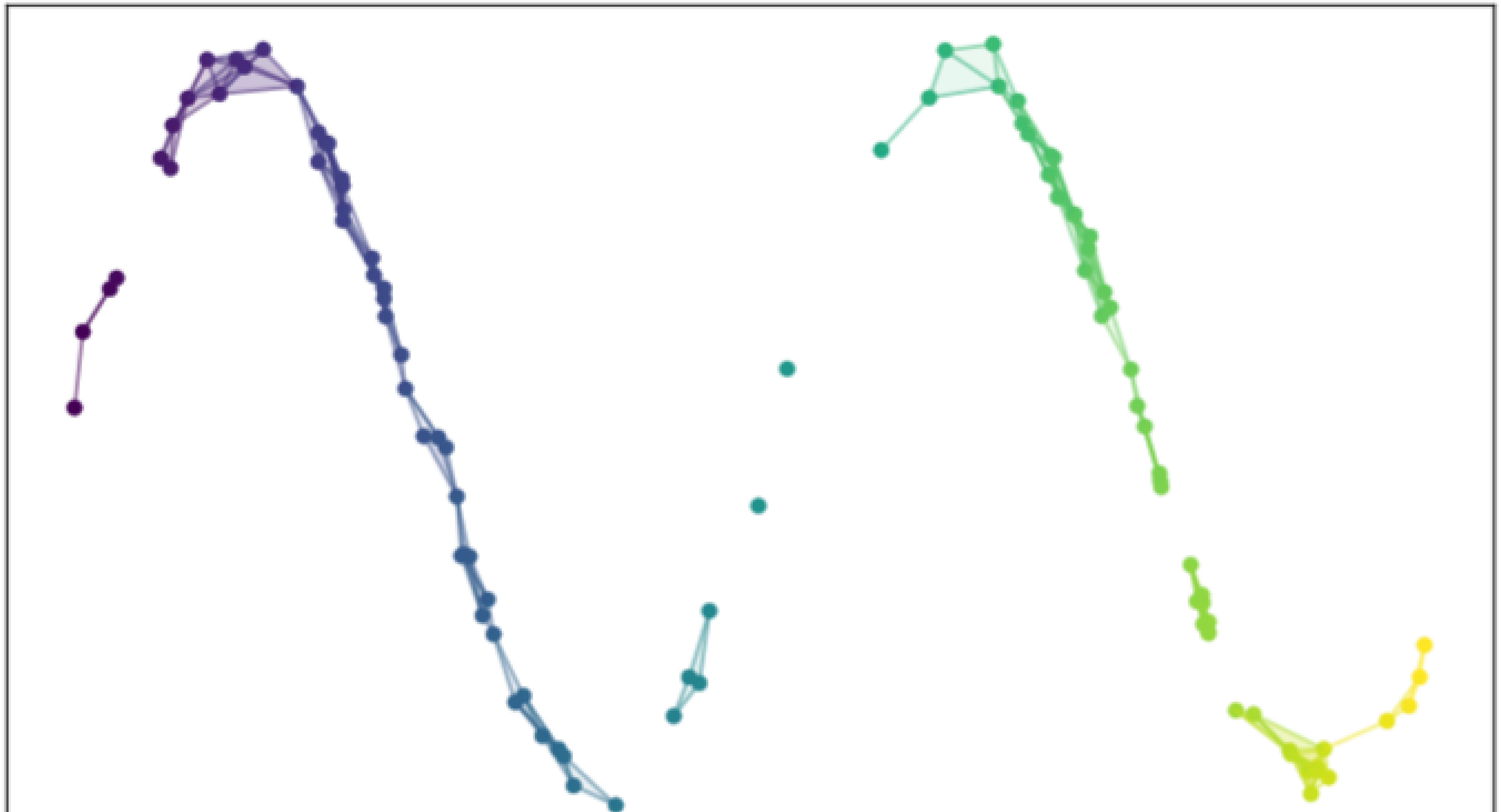
Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

Idea:
Reconstruct underlying topology to identify a space that best explains differences in our data

Approach:
Identify neighboring points in data by setting a neighborhood radius

UMAP

Initial situation: Our data suggests an underlying structure ("topology") but we don't have a model for it



Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

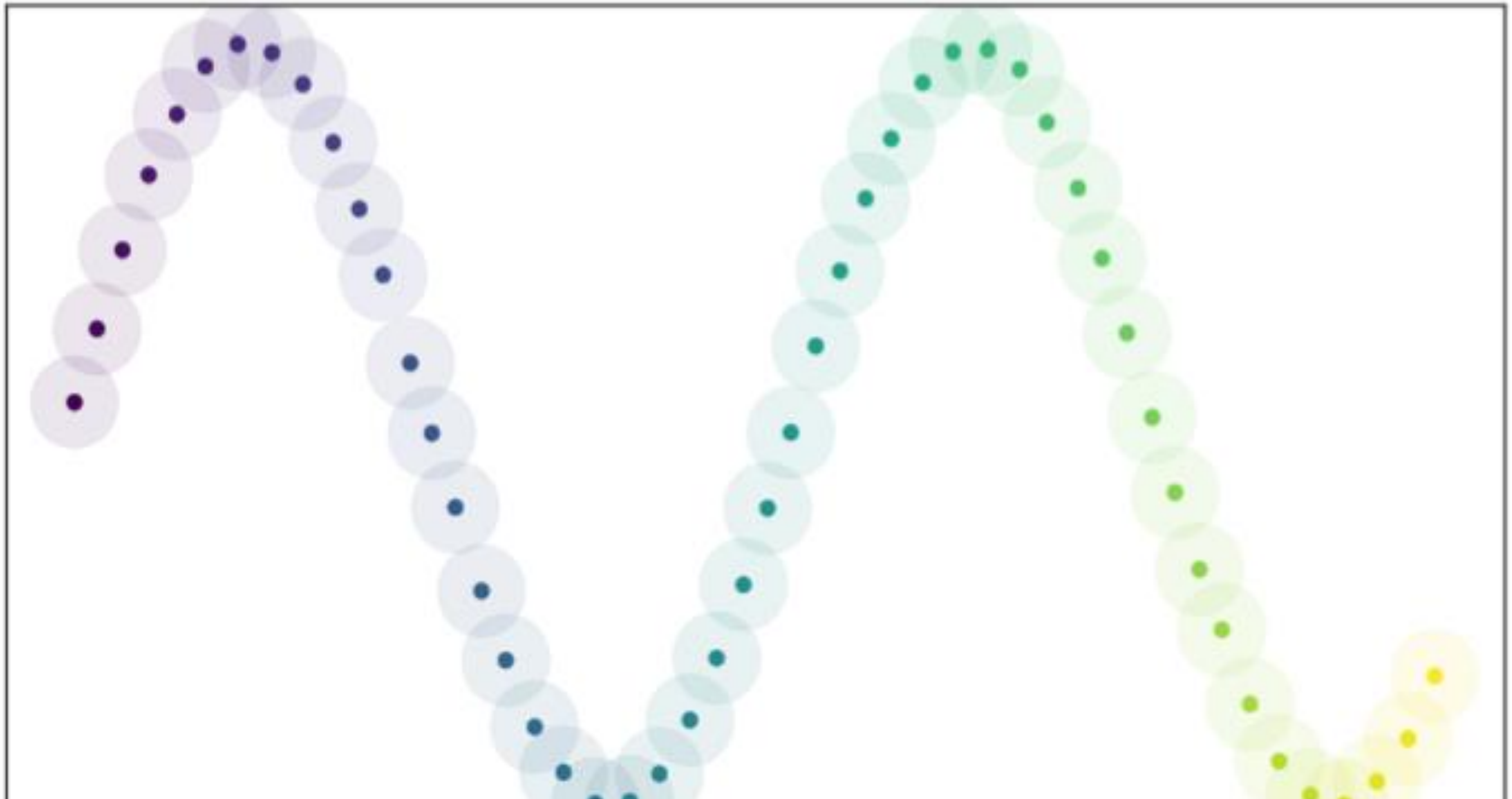
Result:
Neighborhood graph of close points

Problem:
The scarcity of the points leads to a disconnected neighborhood

→Searching neighbors by radius may not be viable

UMAP

Initial situation: Our data suggests an underlying structure (“topology”) but we don’t have a model for it



Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

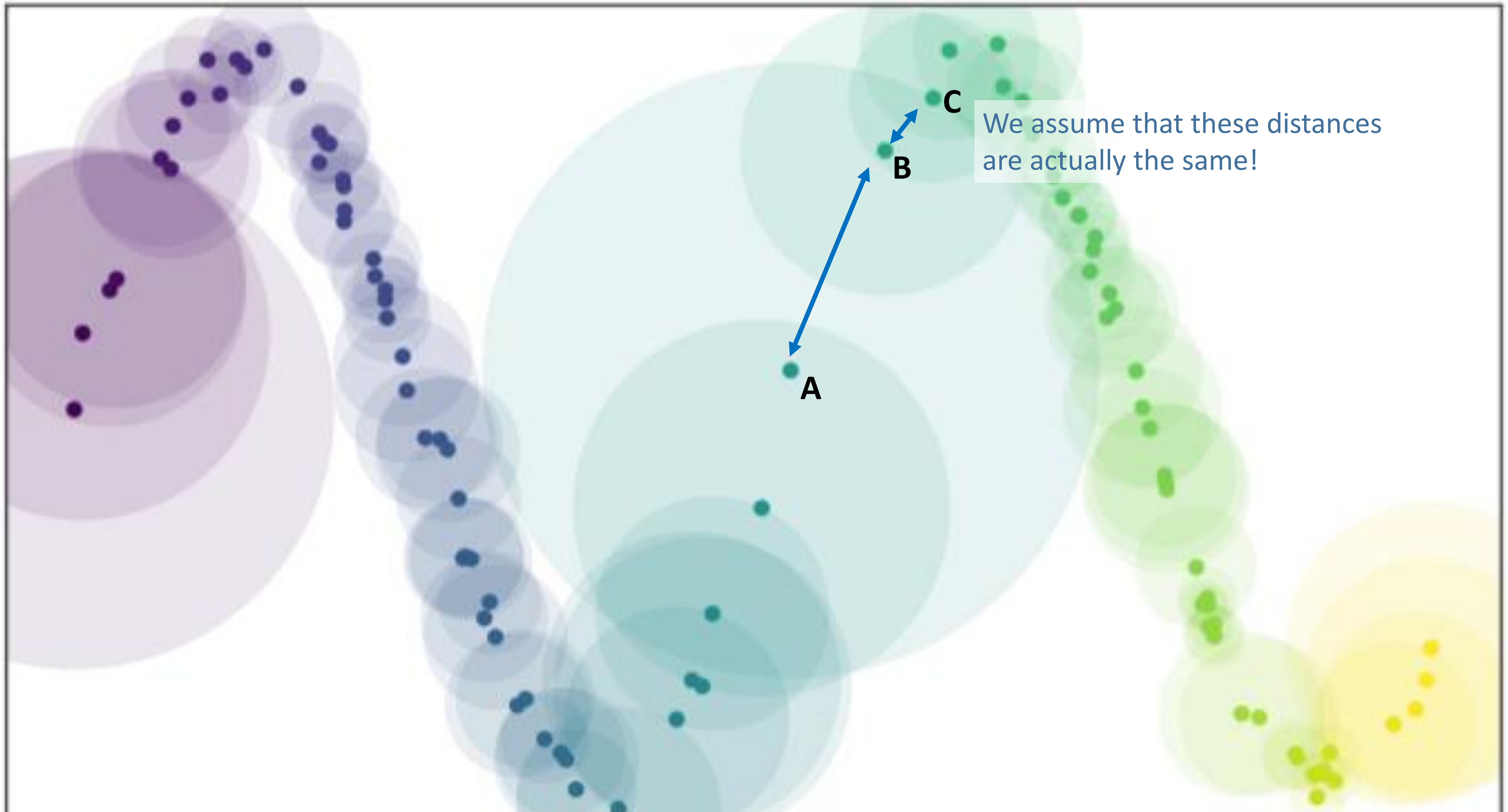
Ideally:
Data spread uniformly on the curve

→ Allows capturing the global structure through neighborhood

→ We typically don’t have this luxury

UMAP

Initial situation: Our data suggests an underlying structure (“topology”) but we don’t have a model for it



Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

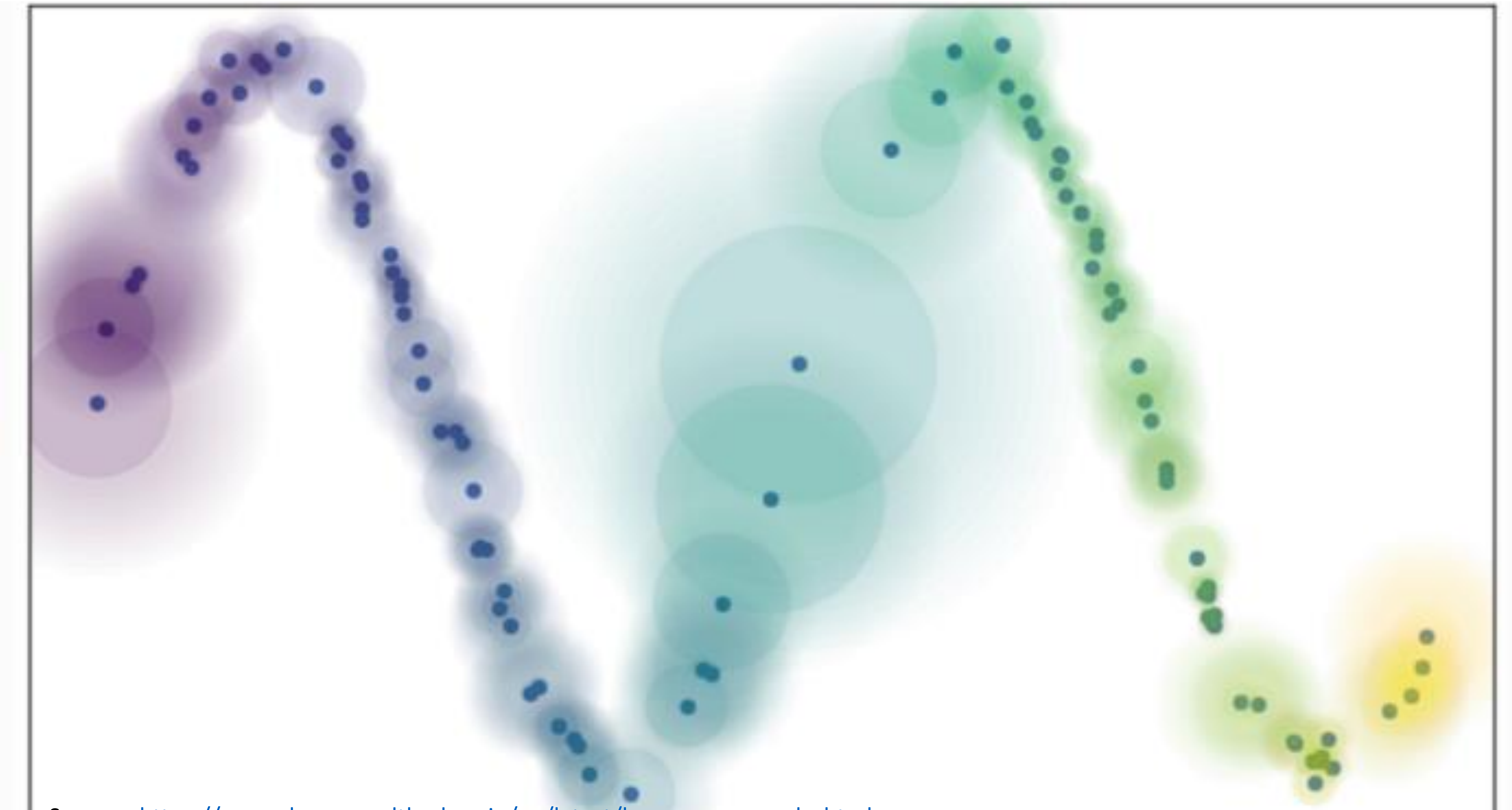
Catch:
We *assume* that the data is actually uniform!

Each point now has its own distance metric assigned to it
→ d(A, B) from A’s point of view: 1
→ d(B,C) from B’s point of view: 1

This works well globally, but doesn’t capture local structure appropriately

UMAP

Initial situation: Our data suggests an underlying structure (“topology”) but we don’t have a model for it



Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

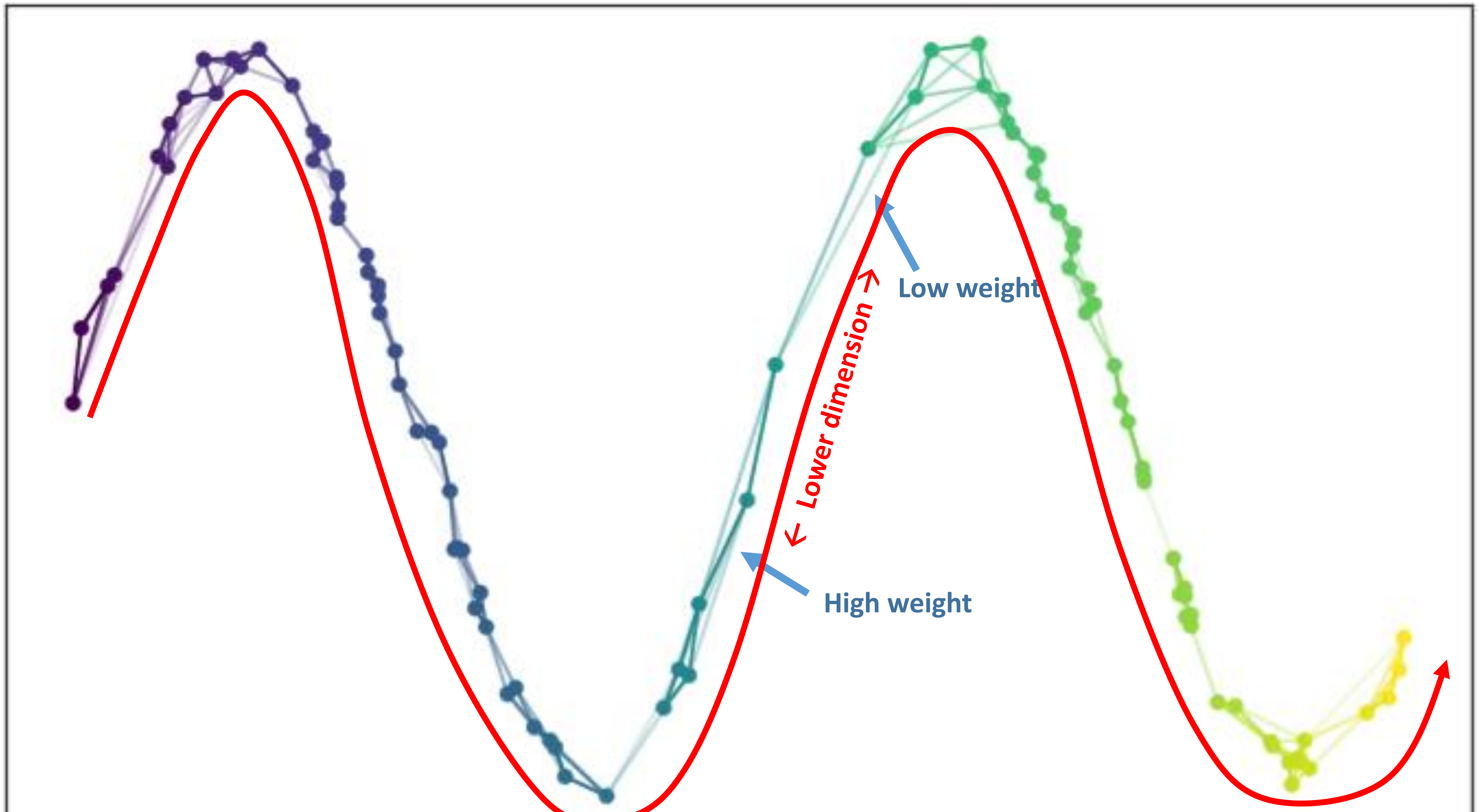
Compromise:

→ Demand that each point is connected to its closes neighbor

→ Weigh connection to further neighbors with distance beyond nearest neighbor

UMAP

Initial situation: Our data suggests an underlying structure (“topology”) but we don’t have a model for it



Source: https://umap-learn.readthedocs.io/en/latest/how_umap_works.html

Result:

→ Global neighborhood graph

→ Local scarcity is reflected through edge weights

Last step:

Project this structure into a lower dimension so that overall topology is reflected

UMAP in Python

https://umap-learn.readthedocs.io/en/latest/index.html

Import package

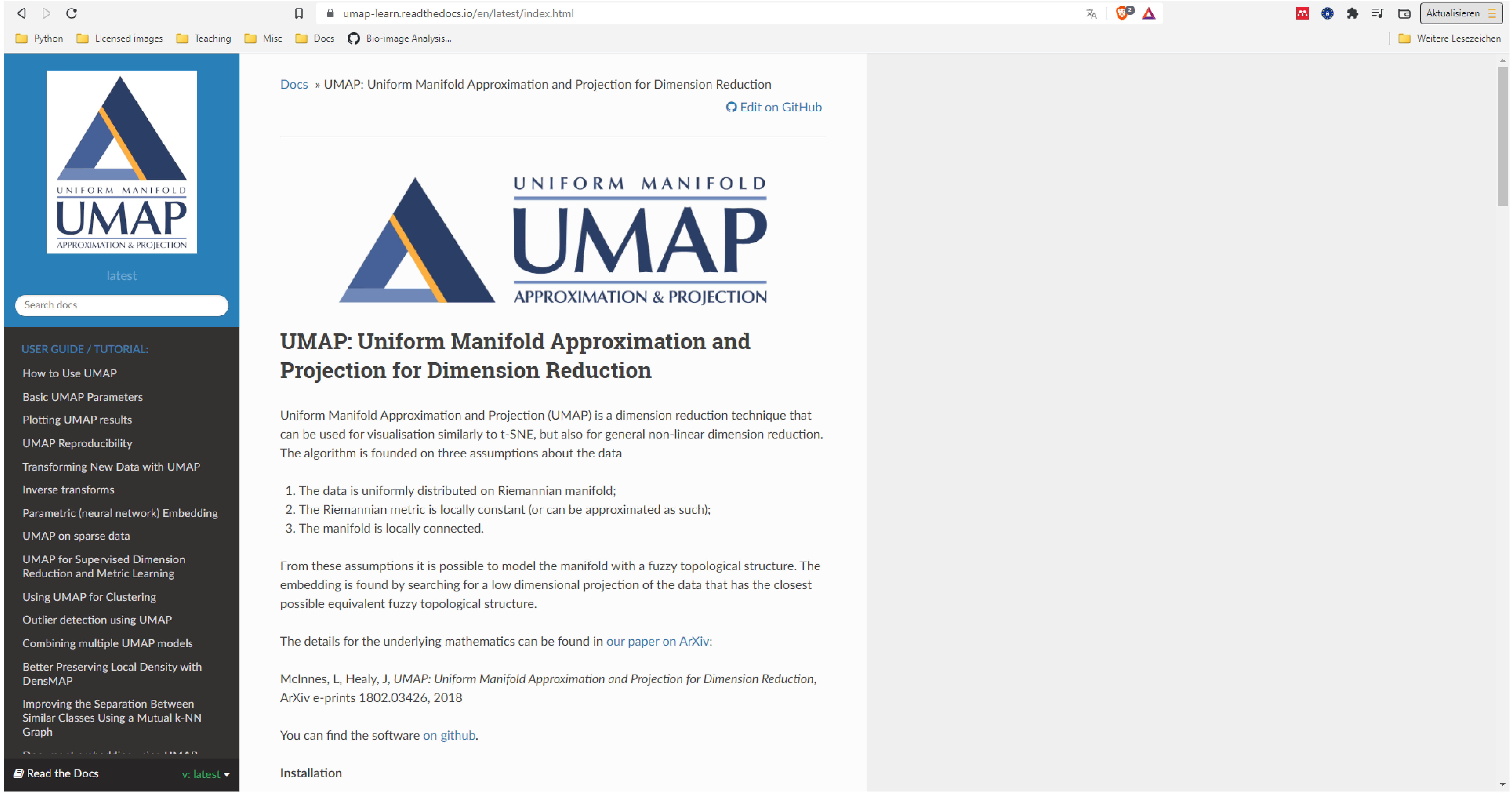
```
import umap
```

Create UMAP object

```
reducer = umap.UMAP()
```

Find projection

```
embedding = reducer.fit_transform(scaled_penguin_data)
embedding.shape
```



Conclusion

Takeaways:

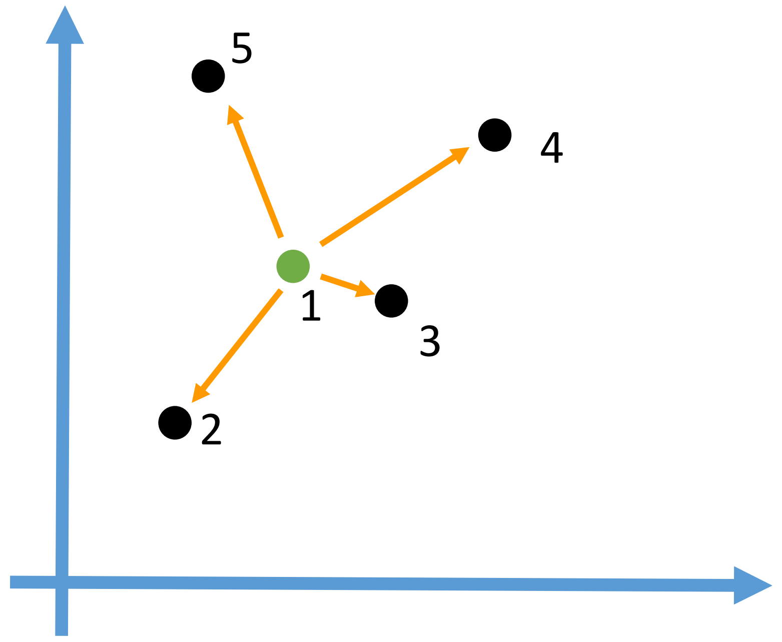
- Dimensionality reduction projects data from high-dimensional space into fewer dimensions
- Algorithms try to preserve meaning in the data

- **PCA (principal component analysis):**
  + Linear method: Metrics are preserved
  - Number of components required as input parameter

- **UMAP (uniform manifold approximation and projection):**
  + Can capture quite arbitrary topologies
  - The embedded space is Euclidian, *but the transform is not-linear!*
  → Two groups A and B being close (similar) to each other in embedded space ≠ A and B also similar in real space

Other methods

t-SNE: t-distributed Stochastic Neighbor Embedding



→ Construct neighborhood: Points 2-5 are neighbors of #1

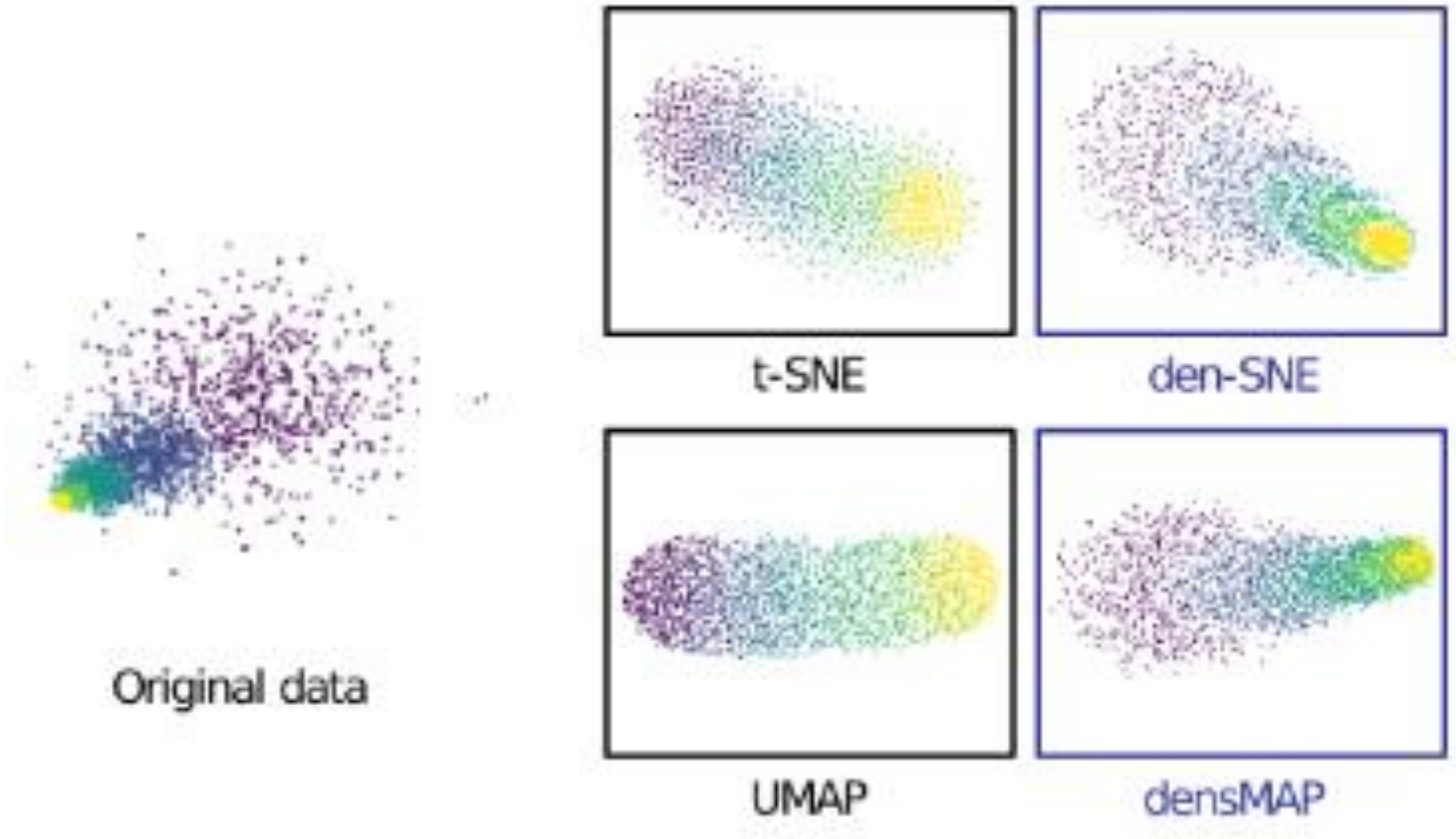
→ Introduce similarity between neighbors and scale to [0,1]: $similarity(1,2) = \frac{1}{d(1,2)}$

Neighbor	Similarity	Probability
2	0.4	0.19
3	0.8	0.39
4	0.45	0.22
5	0.38	0.19

New coordinates: (0.19, 0.39, 0.22, 0.19)
"stochastic neighborhood embedding"

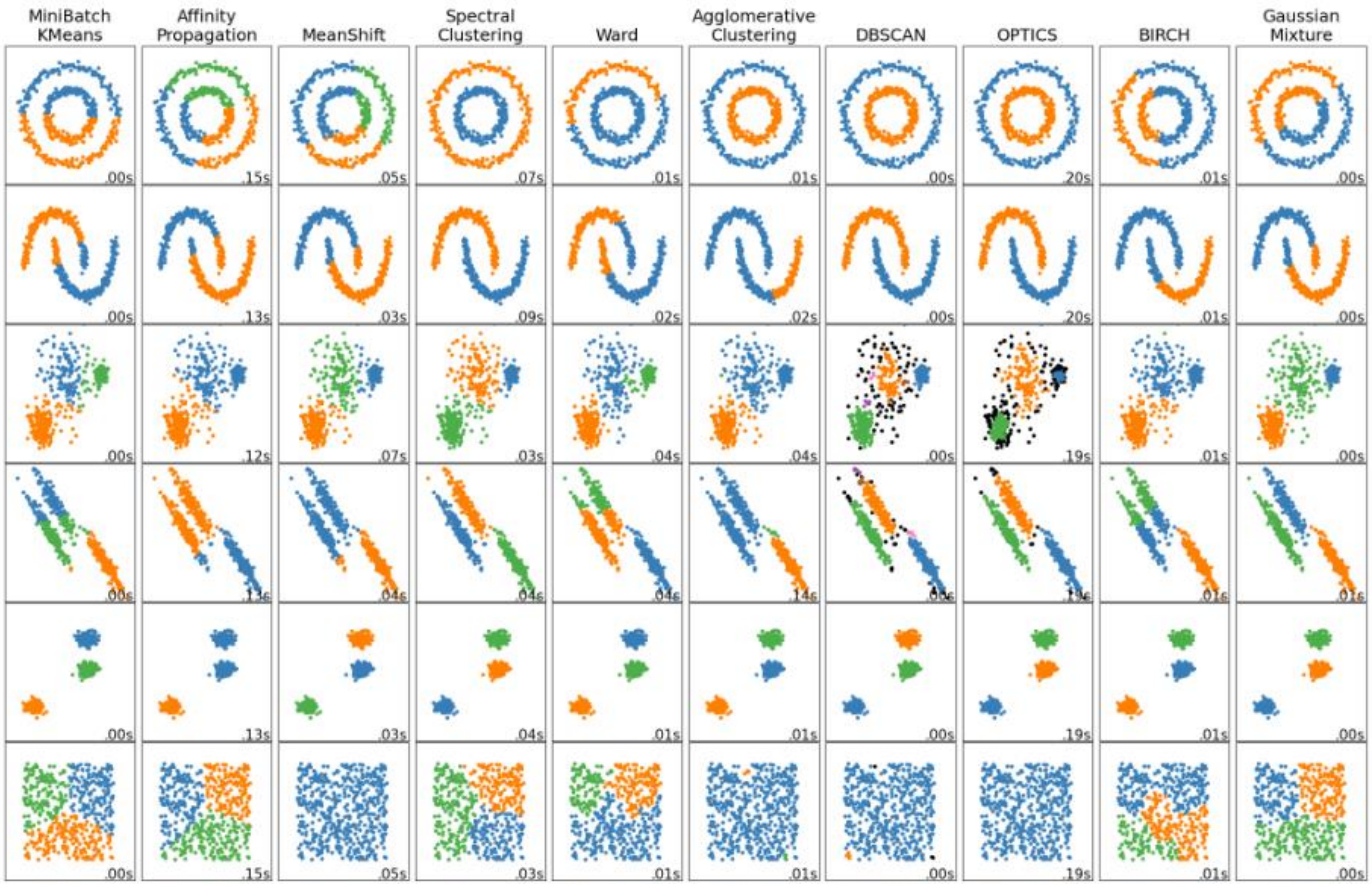
→ find similar probability distribution in lower space

densMAP: Augmented UMAP so that point density is preserved



Resources:
https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
http://cb.csail.mit.edu/cb/densvis/ (License: https://github.com/hhcho/densvis/blob/master/LICENSE)

Other methods

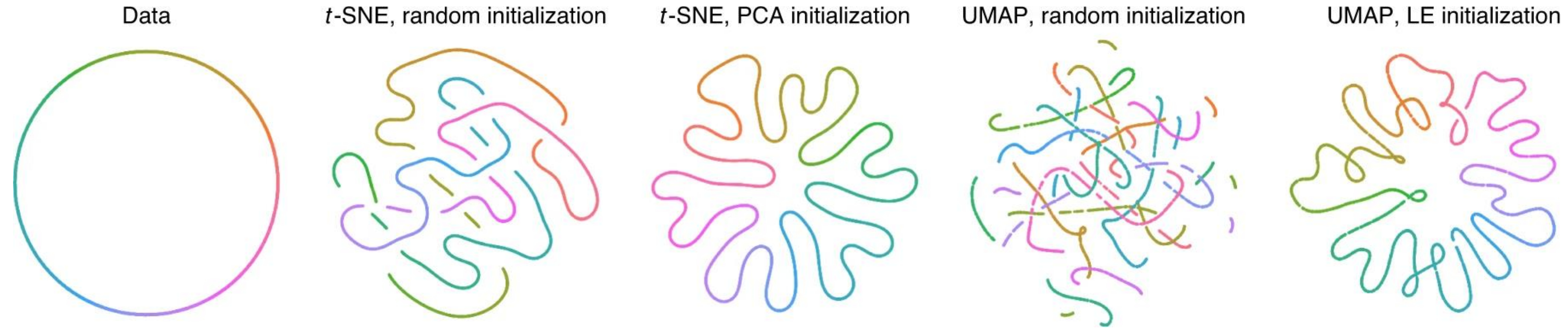


A comparison of the clustering algorithms in scikit-learn

https://scikit-learn.org/stable/modules/clustering.html

Things to consider

- Many parameters invite to “adjust” the data analysis
- Danger to over-interpret the visual “distance”
- How much data structure is preserved is still a matter of debate



Kobak & Linderman, Nature Biotechnology (2021)
https://www.nature.com/articles/s41587-020-00809-z