

Unsupervised Learning

Day 3

Sammie Omranian

June 2025



**College of
Engineering
& Applied
Science**

Unsupervised Learning

How Does Unsupervised Learning Work?

- **Self-Learning from Raw Data:** Unsupervised learning finds patterns in unlabeled data without guidance.
- **Pattern Detection:** It organizes data into groups based on inherent similarities.
- **Example:** A model analyzing weather data might group by temperature or weather patterns, which we can interpret as seasons or types of weather.

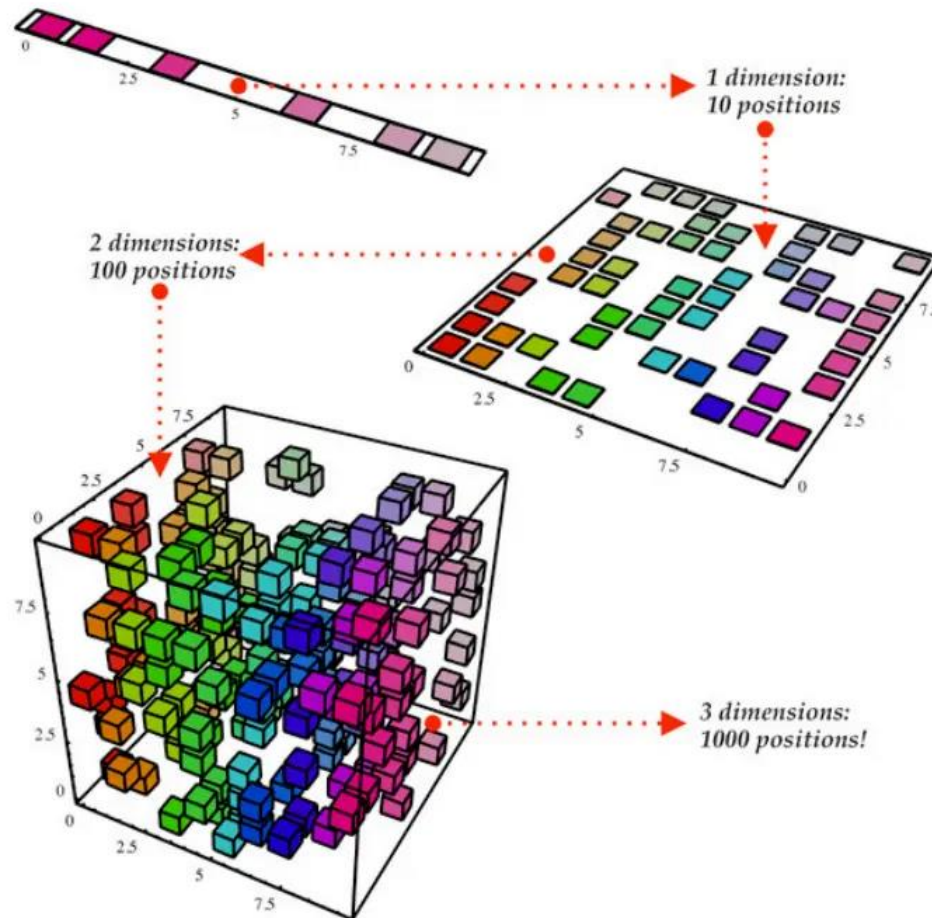
Dimensionality Reduction in Unsupervised Learning

Dimensionality Reduction

What is Dimensionality Reduction?

- Dimensionality reduction helps simplify complex data with many features
- Benefits include reducing model complexity, improving interpretability, and combating the curse of dimensionality

Dimensionality Reduction

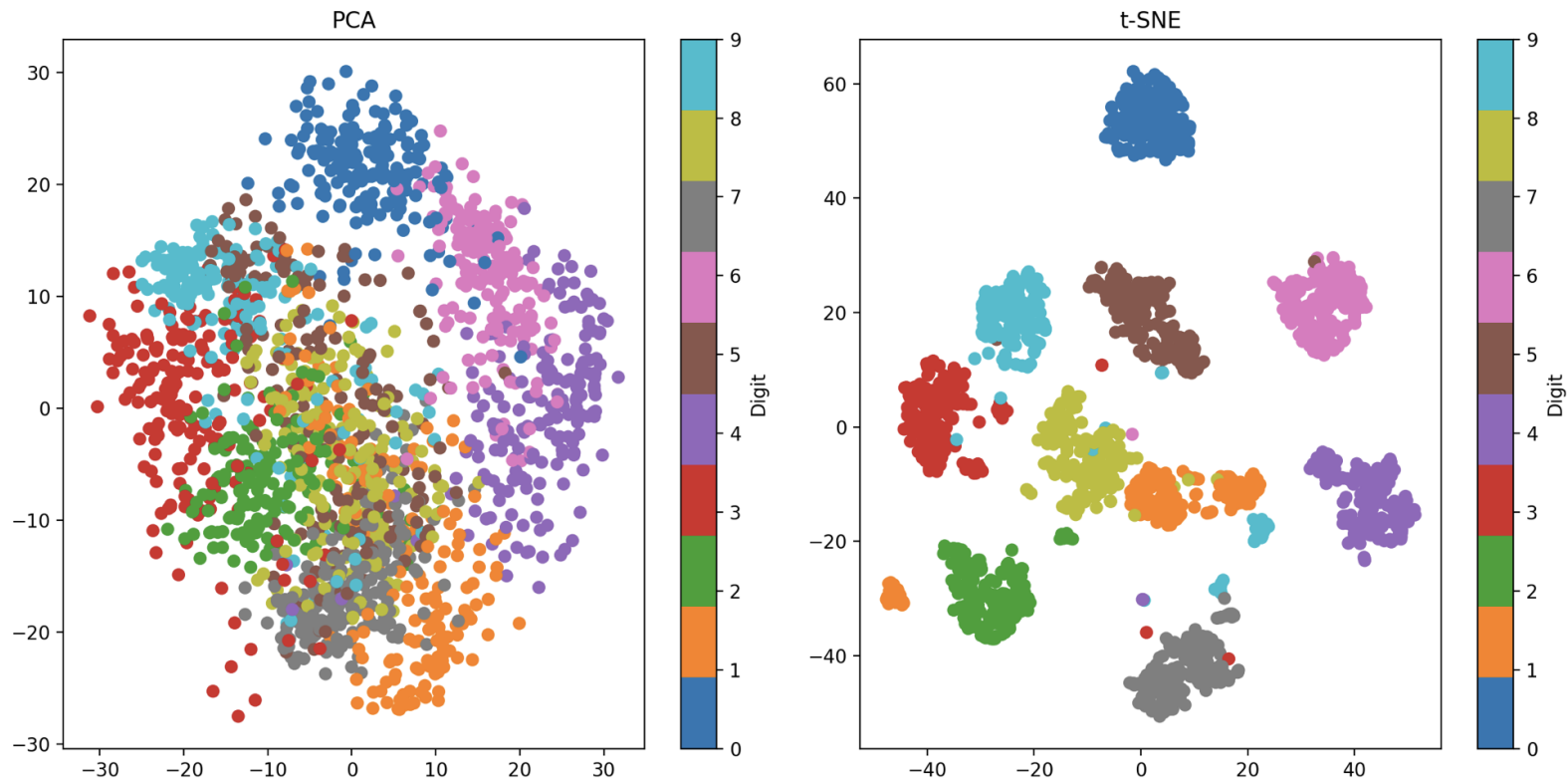


Techniques in Dimensionality Reduction

Types of Dimensionality Reduction Techniques

- **Linear Techniques:** Principal Component Analysis (PCA)
- **Non-linear Techniques:** t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Deep Learning Approach: Autoencoders

Techniques in Dimensionality Reduction

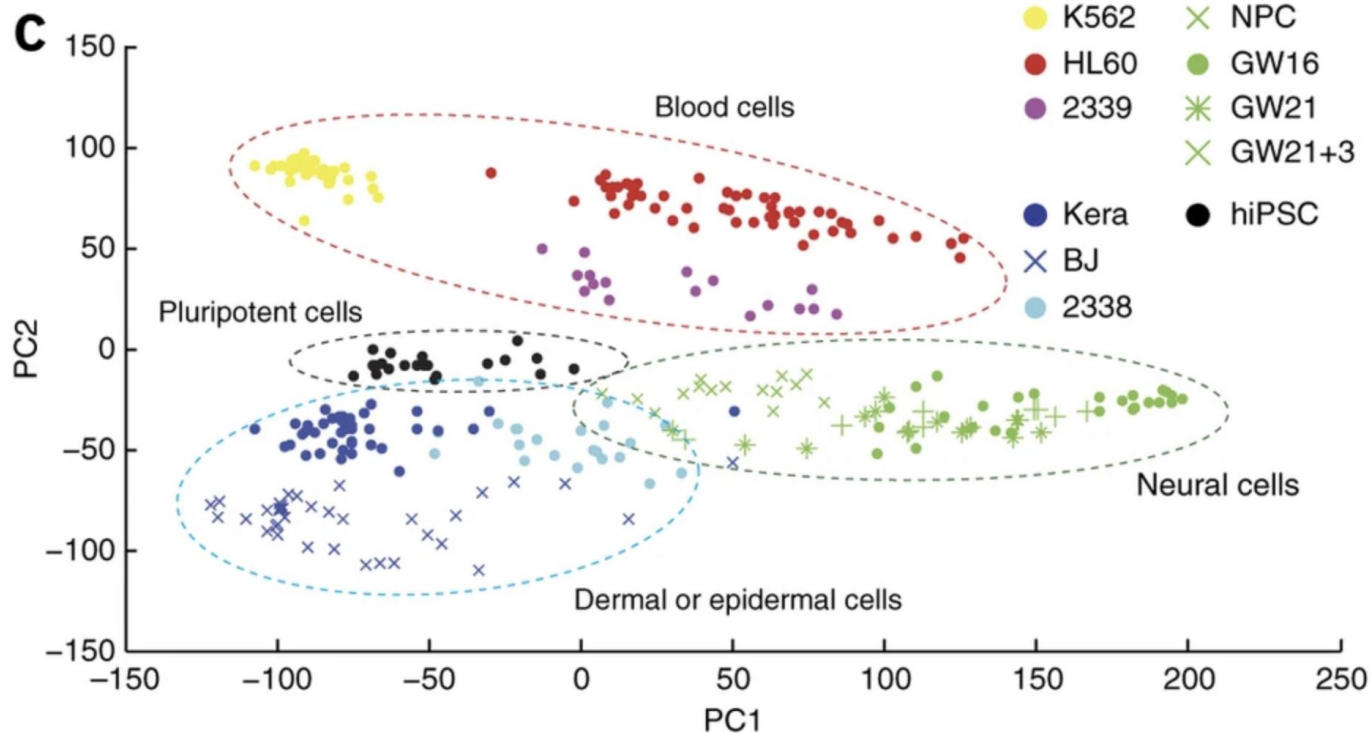


PCA (left) and t-SNE (right) applied to the handwritten digits dataset. For this dataset, t-SNE is more effective at capturing the complex structures and clusters in the data.

Principal Component Analysis (PCA) - Overview

- PCA is a linear transformation technique that identifies the most significant features
- Creates new uncorrelated variables called principal components, ordered by variance
- First few principal components retain most of the data's information
- It's often used to make data easy to explore and visualize.

PCA – How it works



Q: How does transcription from 10,000 genes (high dimensional space) get compressed to a single dot on a graph?

The answer is PCA! For compressing a lot of data into some meaningful clusters from the original data

[source](#)

PCA – How it works

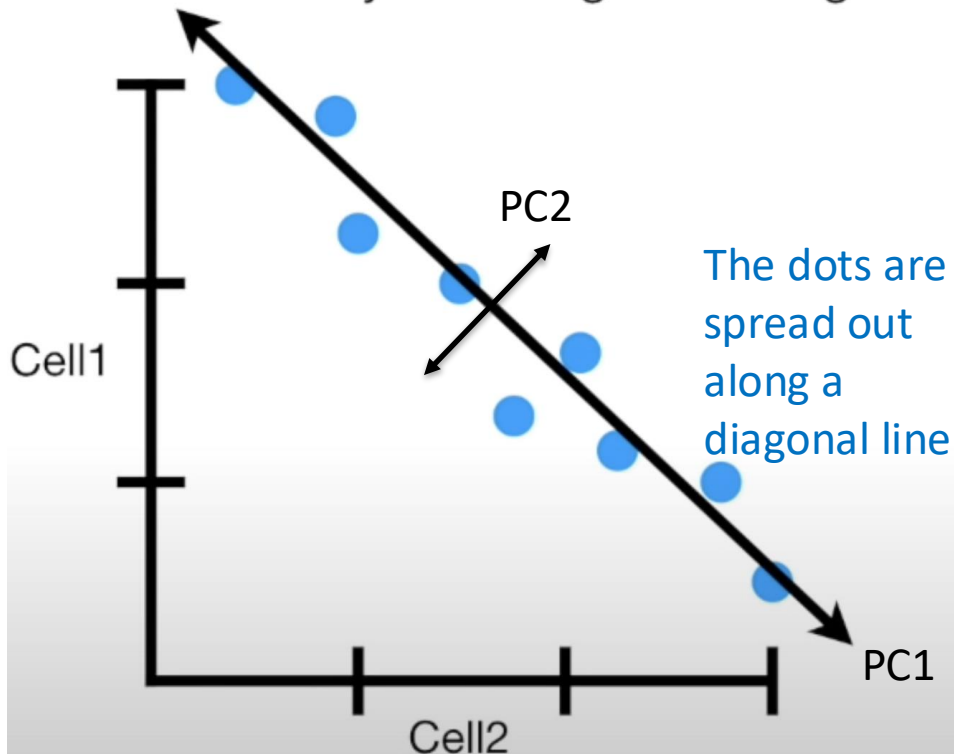
	Cell1	Cell2	Cell3	Cell4	...
Gene1	3	0.25	2.8	0.1	...
Gene2	2.9	0.8	2.2	1.8	...
Gene3	2.2	1	1.5	3.2	...
Gene4	2	1.4	2	0.3	...
Gene5	1.3	1.6	1.6	0	...
Gene6	1.5	2	2.1	3	...
Gene7	1.1	2.2	1.2	2.8	...
Gene8	1	2.7	0.9	0.3	...
Gene9	0.4	3	0.6	0.1	...

Each column shows how much each gene is transcribed in each cell.

[Example source](#)

PCA – How it works

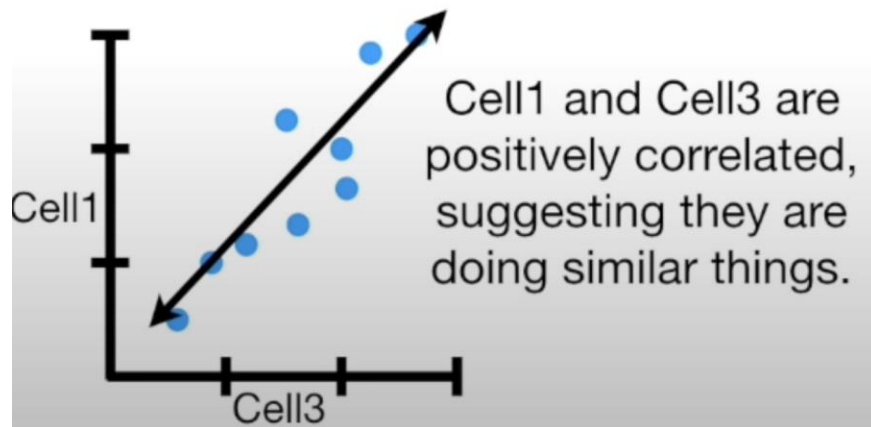
In general, Cell1 and Cell2 have an inverse correlation. This means that they are probably two different types of cells since they are using different genes.



	Cell1	Cell2
Gene1	3	0.25
Gene2	2.9	0.8
Gene3	2.2	1
Gene4	2	1.4
Gene5	1.3	1.6
Gene6	1.5	2
Gene7	1.1	2.2
Gene8	1	2.7
Gene9	0.4	3

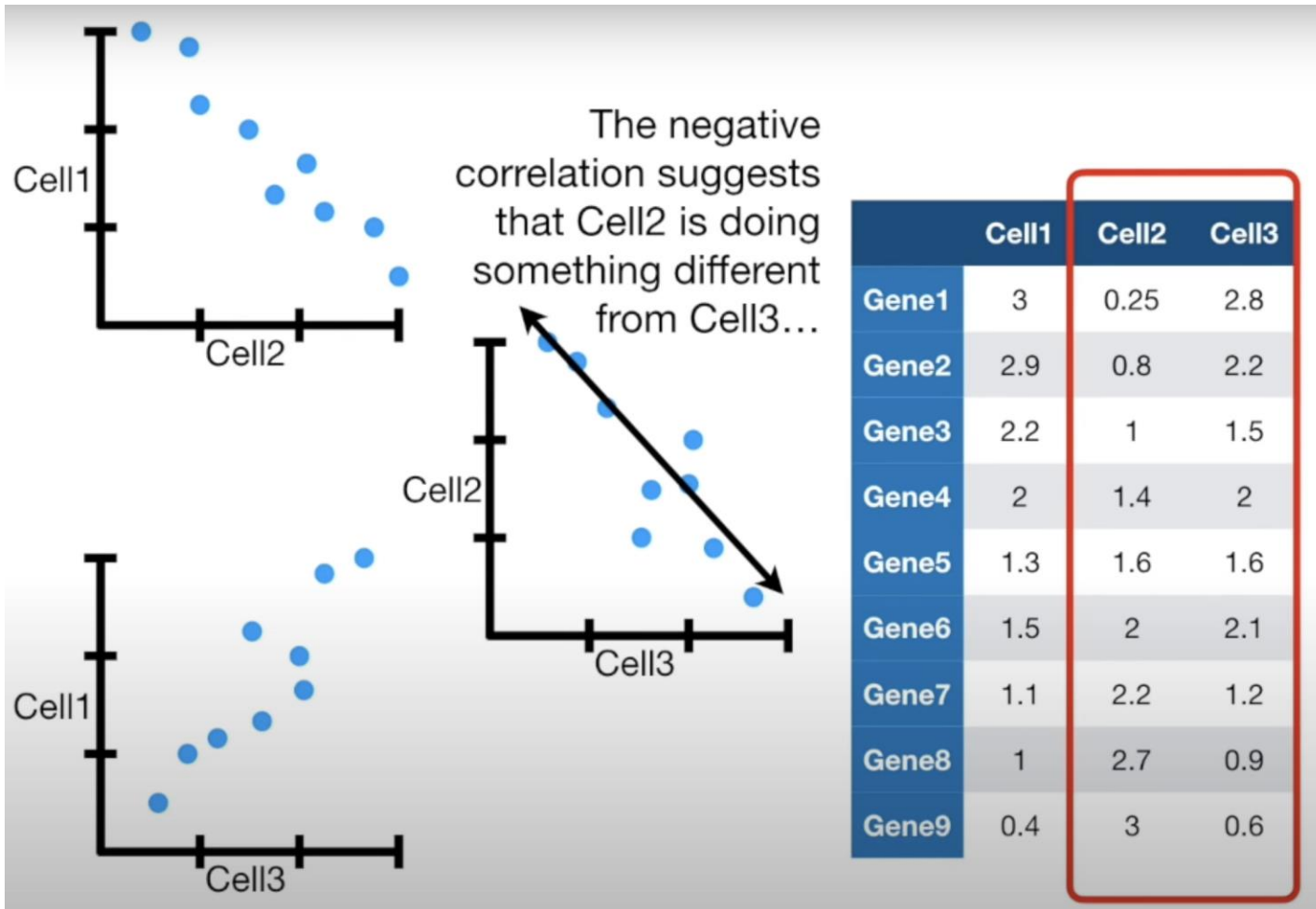
The maximum variation in the data is between the two endpoints

PCA – How it works



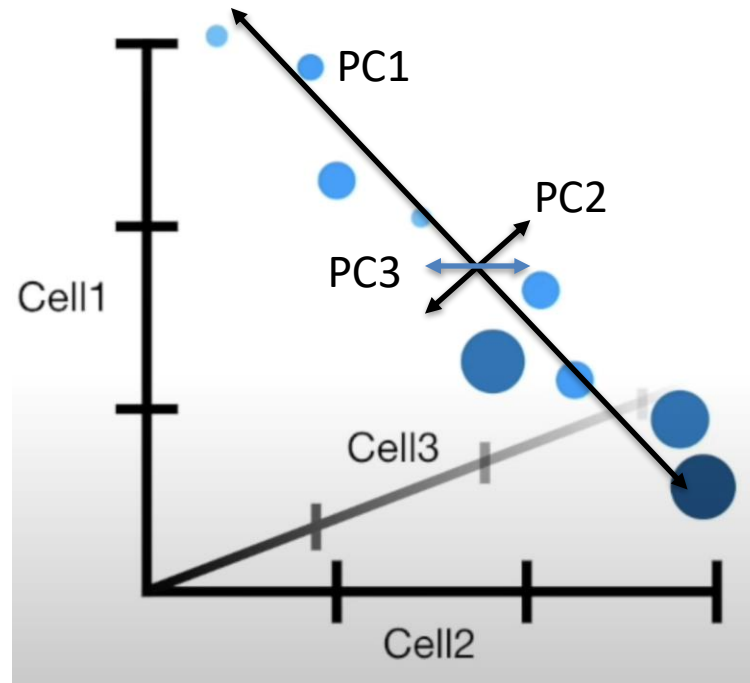
	Cell1	Cell2	Cell3
Gene1	3	0.25	2.8
Gene2	2.9	0.8	2.2
Gene3	2.2	1	1.5
Gene4	2	1.4	2
Gene5	1.3	1.6	1.6
Gene6	1.5	2	2.1
Gene7	1.1	2.2	1.2
Gene8	1	2.7	0.9
Gene9	0.4	3	0.6

PCA – How it works



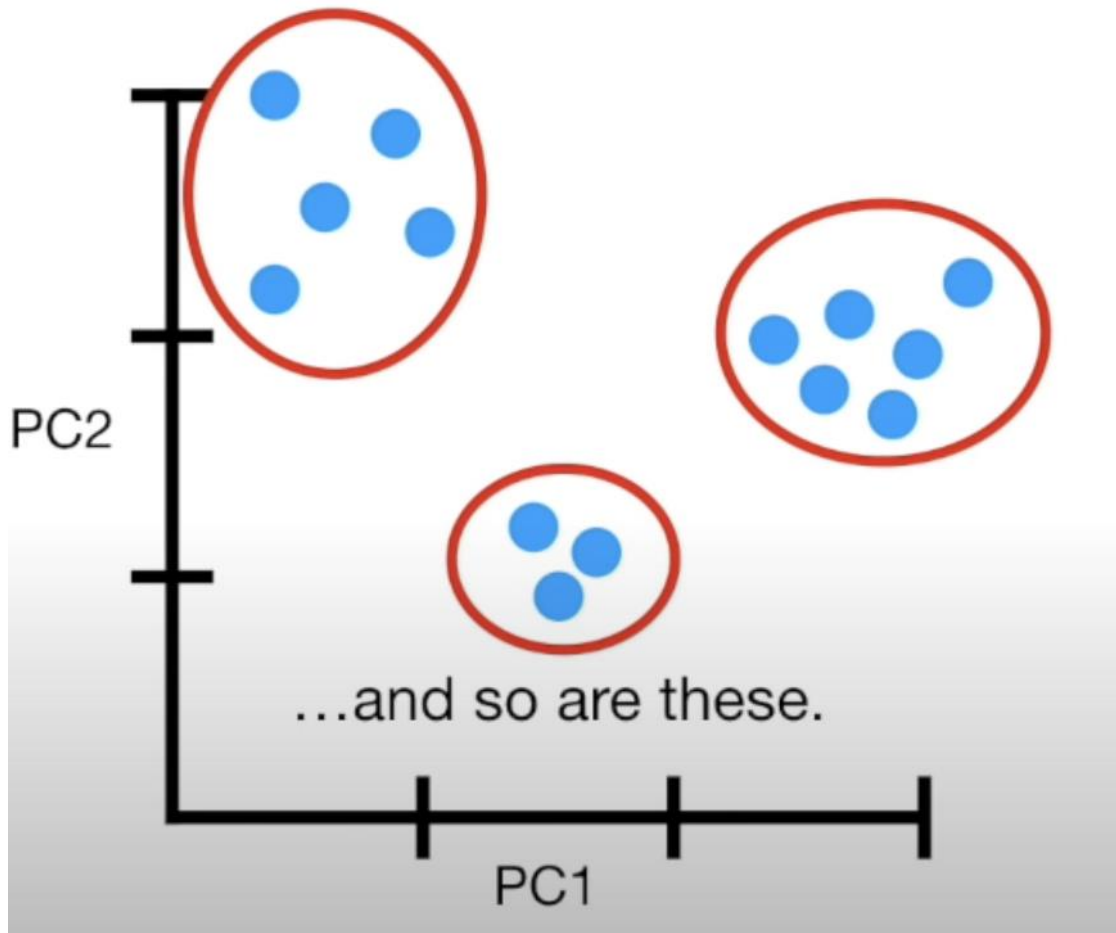
PCA – How it works

Alternatively, we could try to plot all three cells at once on a 3-dimensional graph.



But what do we do when we have 4 or more cells?

PCA – How it works



Cells that are highly correlated cluster together...

PCA In Action - Example

PCA Example: Reducing 3D Data to 2D

- **Original Data:** A 3D dataset is transformed using PCA to find the top 2 components (PC1 and PC2).
- **Result:** The data is projected onto a 2D plane defined by PC1 and PC2, retaining most of the original variance.
- **Interpretation:** Data is now easier to visualize, with minimal information loss.

PCA Example with Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()

# All four features
X = iris.data
# Class labels for coloring
y = iris.target

# Visualize the original data using a pair plot for all 4 features
plt.figure(figsize=(12, 5))

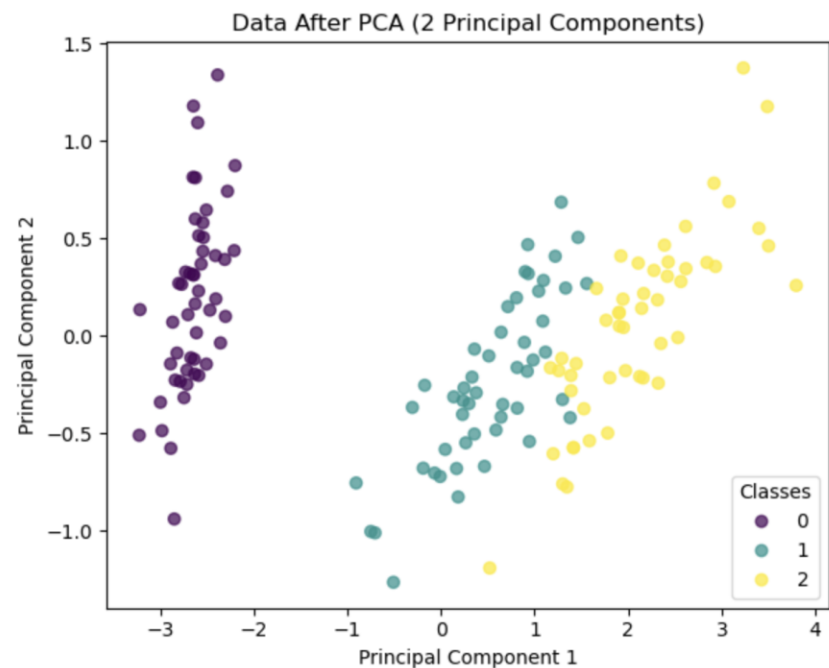
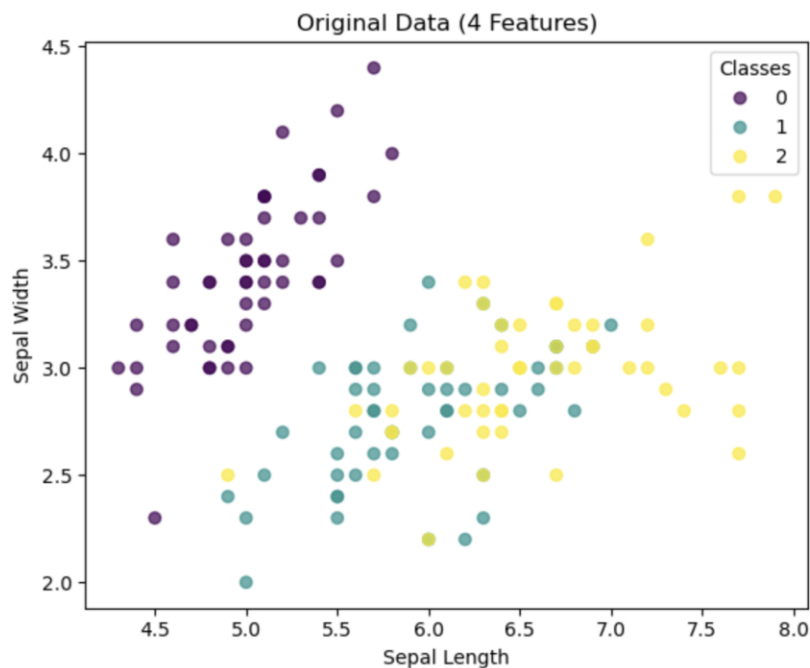
# PCA Transformation: Reducing 4D data to 2D
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Original Data Visualization - Pair Plot
plt.subplot(1, 2, 1)
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', alpha=0.7)
plt.title("Original Data (4 Features)")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend(*scatter.legend_elements(), title="Classes")

# Transformed Data Visualization (after PCA)
plt.subplot(1, 2, 2)
scatter_pca = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', alpha=0.7)
plt.title("Data After PCA (2 Principal Components)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(*scatter_pca.legend_elements(), title="Classes")

plt.tight_layout()
plt.show()
```


PCA Example with Code



t-Distributed Stochastic Neighbor Embedding (t-SNE) - Overview

- t-SNE is a non-linear technique that maps high-dimensional data into a lower-dimensional space
- Designed to preserve the local structure and similarity of data points
- Useful for visualizing complex datasets in 2D or 3D

t-SNE in Action - Example

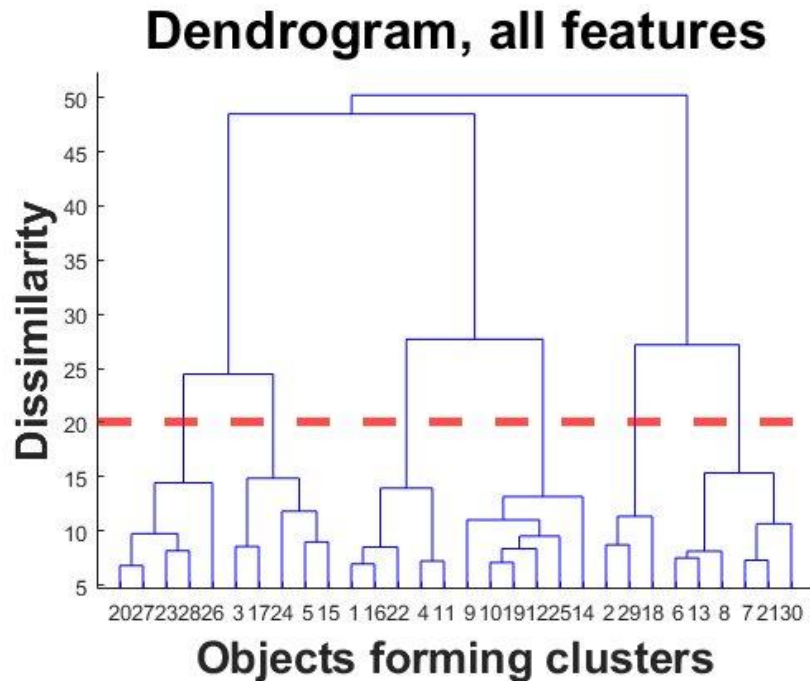


Figure 1: Dendrogram for the hierarchical clustering with Ward's linkage to determine the number of clusters in the analysis using all data. Following visual inspection we decided to opt for six clusters (highlighted with the dotted red line).

[source](#)

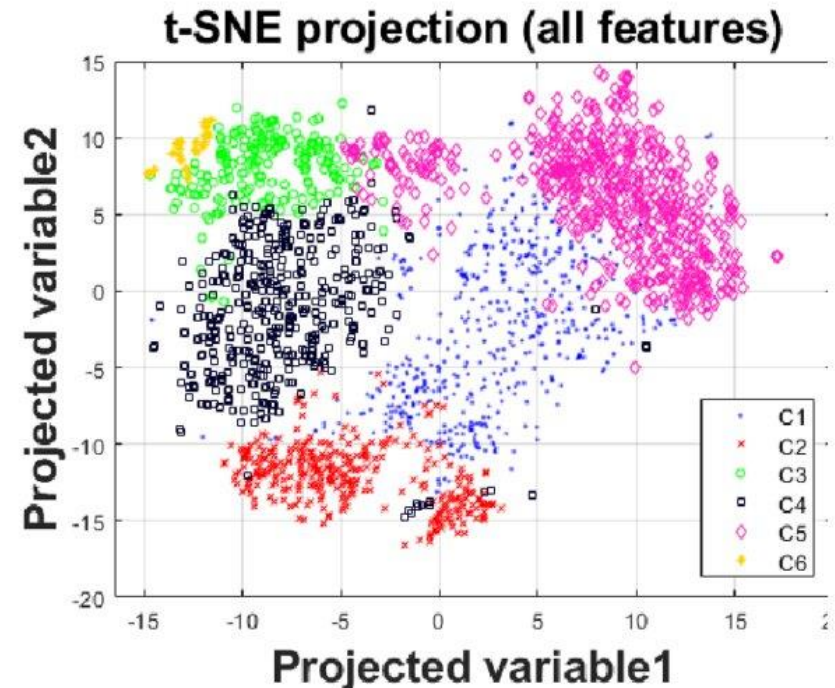


Figure 2: Two-dimensional representation of the original high-dimensional dataset using t-SNE and marking the six clusters (denoted C1...C6) computed using hierarchical clustering with the original feature set (see dendrogram in Figure 1).

How t-SNE works?

1. **Calculate Similarities in High Dimensions:** t-SNE calculates how similar (close) each pair of points is in the original high-dimensional space, treating nearby points as "neighbors" with higher similarity.
2. **Random Initialization in Low Dimensions:** It then places all points randomly in 2D or 3D space. At this stage, the neighbors from the high-dimensional space are **not yet mapped** in the low-dimensional space. The points are just randomly scattered.
3. **Adjust Positions to Match High-Dimensional Relationships:** Using optimization, t-SNE moves points in the low-dimensional space to make the low-dimensional "neighborhoods" (i.e., closeness between points) resemble those in the high-dimensional space as much as possible.

t-SNE Example with Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.manifold import TSNE

# Load the Iris dataset
iris = load_iris()
# All four features
X = iris.data
# Class labels for coloring
y = iris.target

# Initialize the figure
plt.figure(figsize=(12, 5))

# Original Data Visualization - Projected onto the first two features
plt.subplot(1, 2, 1)
scatter_orig = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', alpha=0.7)
plt.title("Original Data (First 2 Features)")
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend(*scatter_orig.legend_elements(), title="Classes")

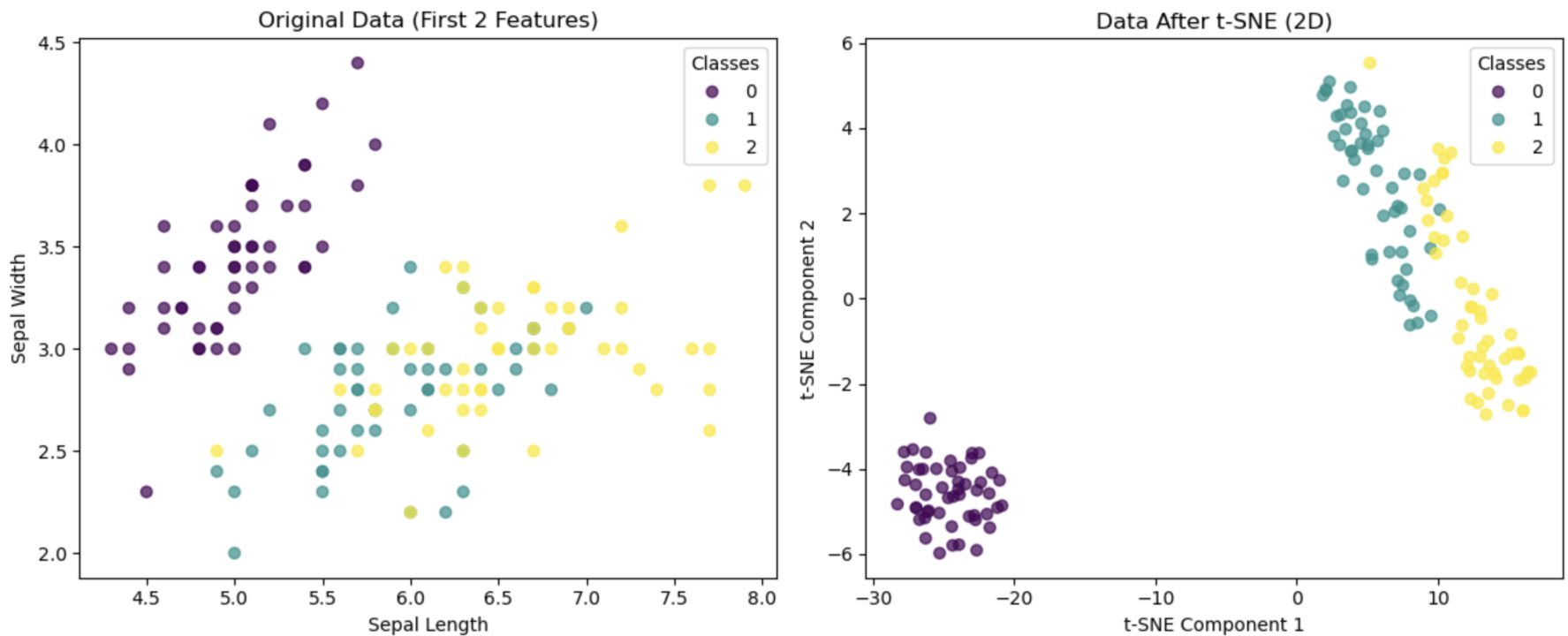
# Apply t-SNE to reduce the data from 4D to 2D
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

# t-SNE Transformed Data Visualization
plt.subplot(1, 2, 2)
scatter_tsne = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis', alpha=0.7)
plt.title("Data After t-SNE (2D)")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.legend(*scatter_tsne.legend_elements(), title="Classes")

# Display the plots
plt.tight_layout()
plt.show()
```

Number of
dimensions for
reduced data

t-SNE Example with Code



Comparison of PCA, t-SNE, and Autoencoders

- **PCA:** Linear, fast, interpretable, best for linearly structured data
- **t-SNE:** Non-linear, preserves local structure, computationally intensive, useful for visualization
- **Autoencoders:** Deep learning, can capture complex non-linear relationships, requires more data

Importance of Dimensionality Reduction in Machine Learning

- High-dimensional data complicates visualization, training, and interpretation

Why Dimensionality Reduction Matters

- Simplifies data and enhances model performance
- Improves model interpretability and reduces overfitting risks
- Allows for faster computation and visualization in lower dimensions

Summary and Applications

- Dimensionality reduction techniques help make sense of complex data
- Applications across domains: image processing, text analysis, bioinformatics, and more