MEIM | MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH MIT MANAGEMENT SLOAN SCHOOL | UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

MASTER MEIM 2021-2022

Digital AI – Unsupervised Learning Hands-on

Lesson given by prof. Alessio Ferone

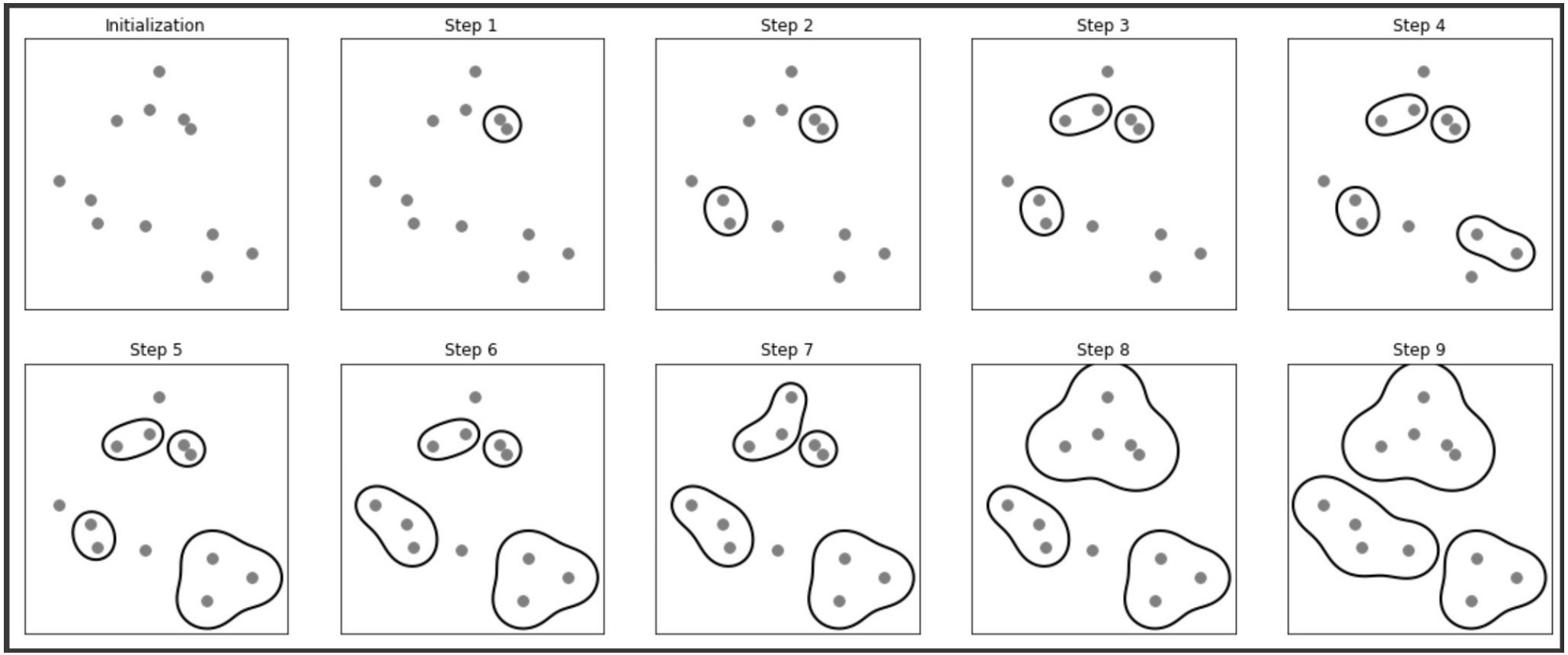
Overview

- Python tools for machine learning
  - First application
- Unsupervised learning
  - K-Means
- **Agglomerative Clustering and DBSCAN**
- Principal Component Analysis

Agglomerative Clustering

- The **algorithm** starts by declaring **each point** its own **cluster**
- The **two most similar clusters are merged** until only the **specified number of clusters** are left
- There are several **linkage criteria**
  - **ward** picks the **two clusters** to merge such that the **variance** within all clusters **increases the least**
  - **average linkage** merges the **two clusters** that have the **smallest average distance** between all **their points**
  - **complete linkage** merges the **two clusters** that have the **smallest maximum distance** between **their points**

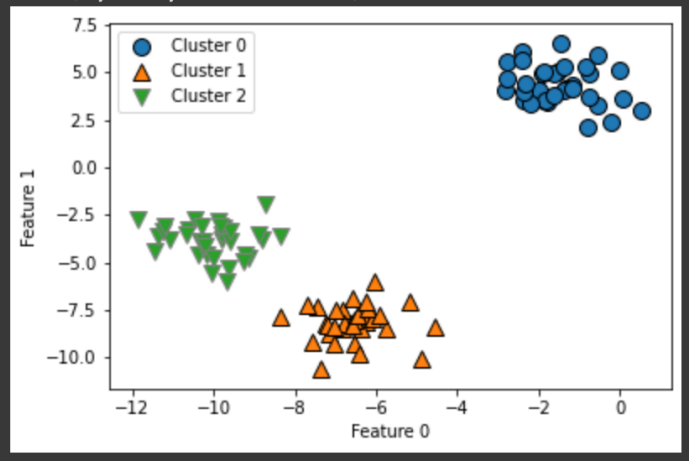
Agglomerative Clustering



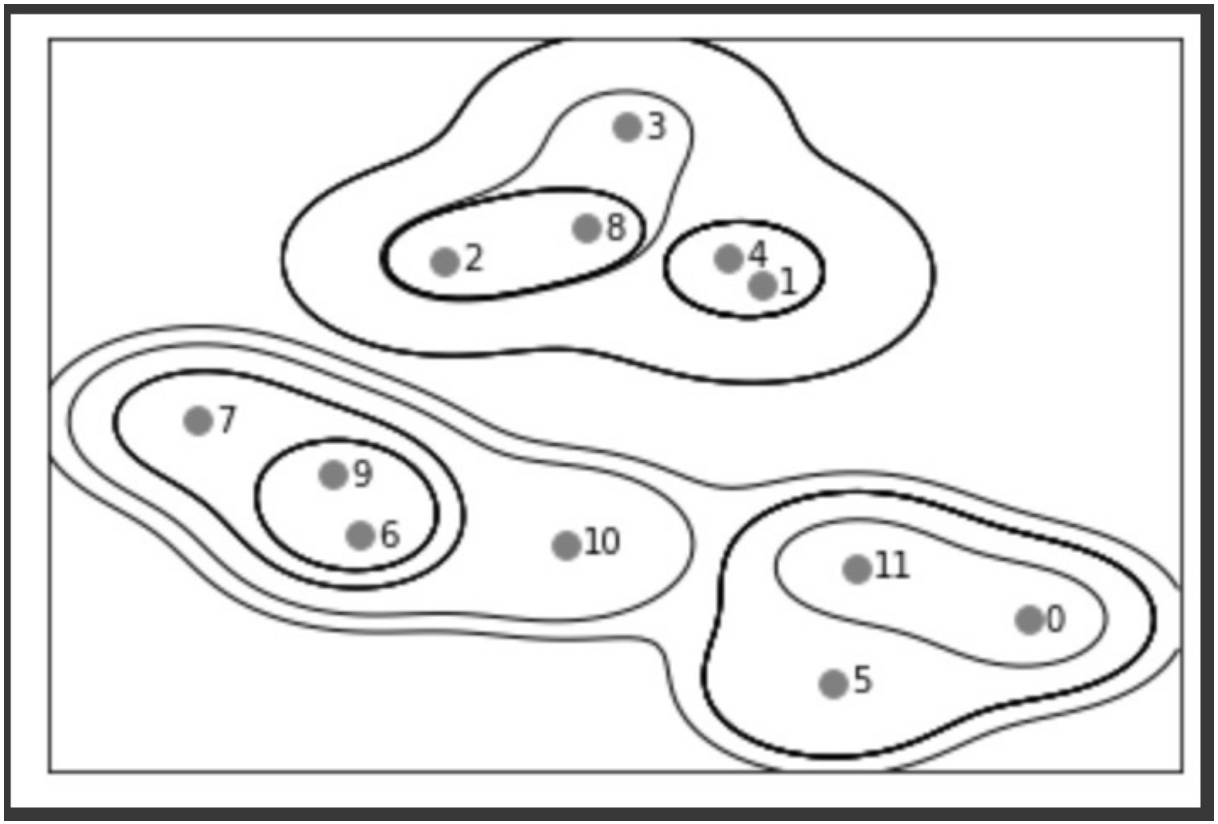
Agglomerative Clustering

```
from sklearn.cluster import AgglomerativeClustering
X, y = make_blobs(random_state=1)

agg = AgglomerativeClustering(n_clusters=3)
assignment = agg.fit_predict(X)

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignment)
plt.legend(["Cluster 0", "Cluster 1", "Cluster 2"], loc="best")
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

```
Text(0, 0.5, 'Feature 1')
```



Agglomerative Clustering



Agglomerative Clustering: dendogram

- Another **tool** to **visualize hierarchical clustering** is called a **dendrogram** (scikit-learn currently does not draw dendrograms)

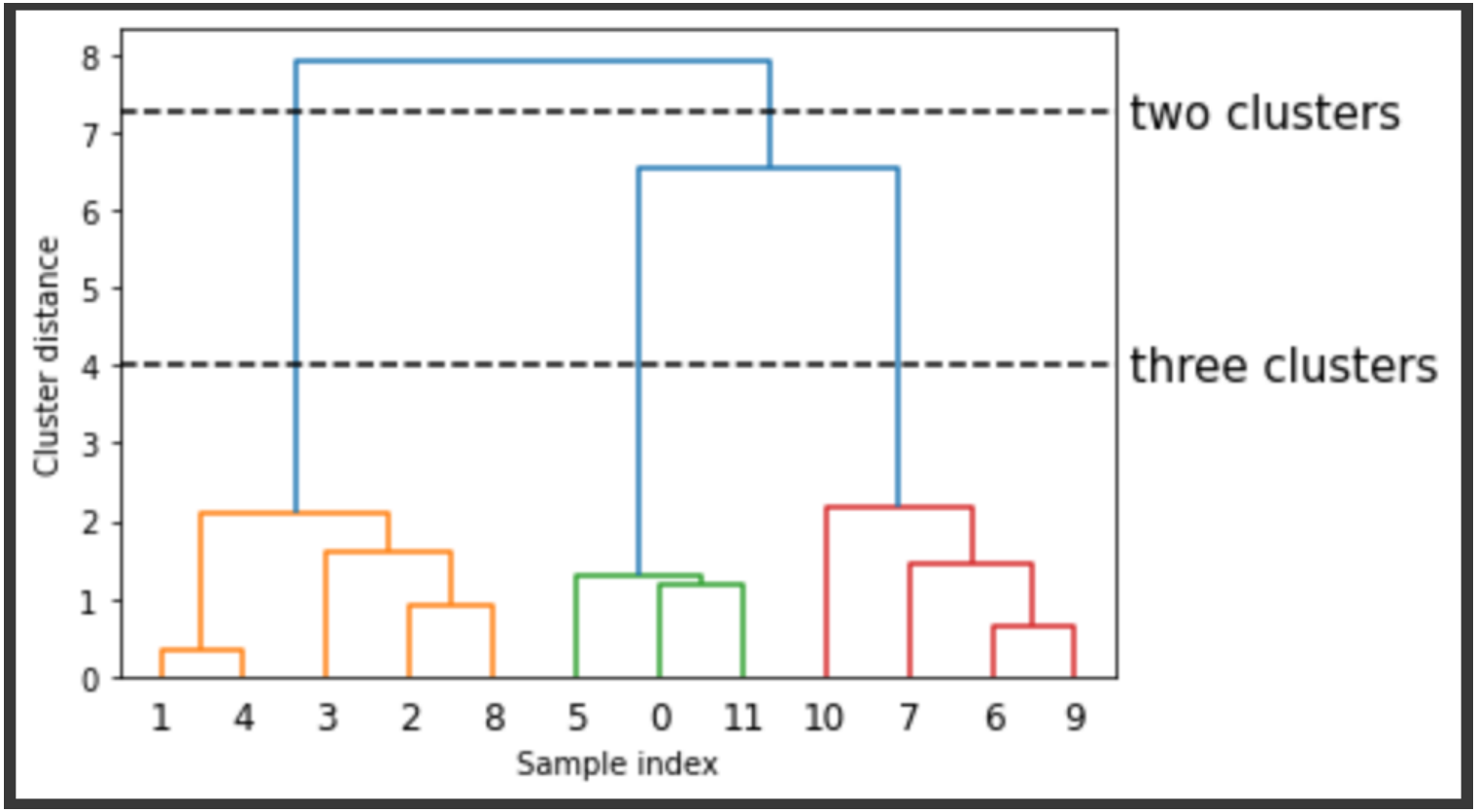
- **SciPy** provides a **function** that takes a **data array X** and **computes** a **linkage** *array*, which **encodes hierarchical cluster** similarities

- We can then **feed** this **linkage array** into the **scipy dendrogram** function to **plot** the **dendrogram**

Agglomerative Clustering: dendogram

```python
# Import the dendrogram function and the ward clustering function from SciPy
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import dendrogram, ward

X, y = make_blobs(random_state=0, n_samples=12)
# Apply the ward clustering to the data array X
# The SciPy ward function returns an array that specifies the distances
# bridged when performing agglomerative clustering
linkage_array = ward(X)
# Now we plot the dendrogram for the linkage_array containing the distances
# between clusters
dendrogram(linkage_array)

# mark the cuts in the tree that signify two or three clusters
ax = plt.gca()
bounds = ax.get_xbound()
ax.plot(bounds, [7.25, 7.25], '--', c='k')
ax.plot(bounds, [4, 4], '--', c='k')

ax.text(bounds[1], 7.25, ' two clusters', va='center', fontdict={'size': 15})
ax.text(bounds[1], 4, ' three clusters', va='center', fontdict={'size': 15})
plt.xlabel("Sample index")
plt.ylabel("Cluster distance")
```

Agglomerative Clustering: dendogram

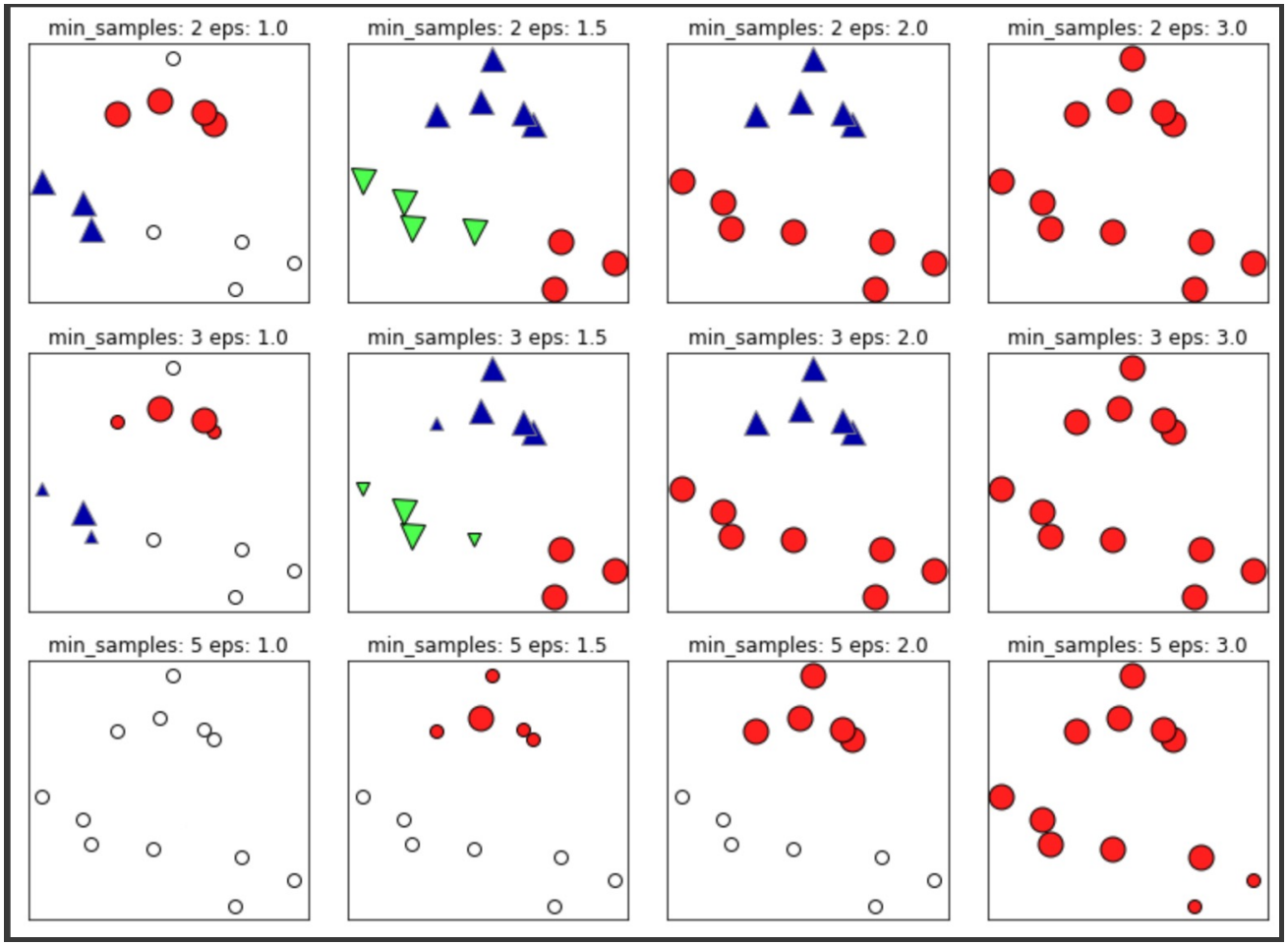


DBSCAN

- **DBSCAN** stands for "**density-based spatial clustering of applications with noise**"
- **DBSCAN** does **not require** the user to set the **number** of **clusters** *a priori*
- **DBSCAN** works by **identifying points** that are in "**crowded**" **regions** of the feature space, where many **data points** are **close** together
- If there are at least **min_samples** many data **points** within a **distance** of **eps** to a given data point, that data point is classified as a **core sample**

DBSCAN

- **Clusterings** obtained with **different parameters**
- **Points** in **clusters** are **solid**, while **noise** points are **in white**
- **Core samples** are **large markers**, while **boundary points** are **smaller markers**

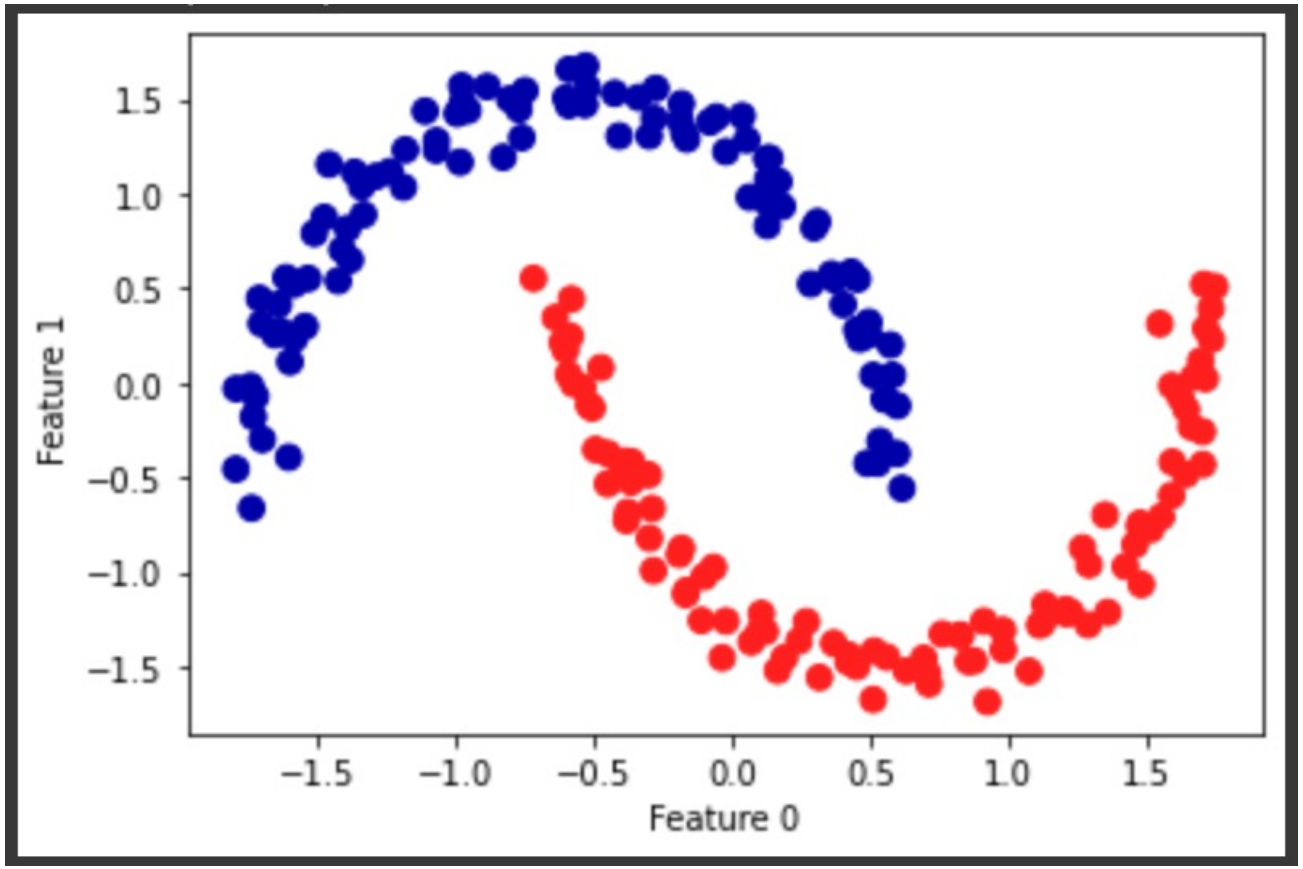


DBSCAN

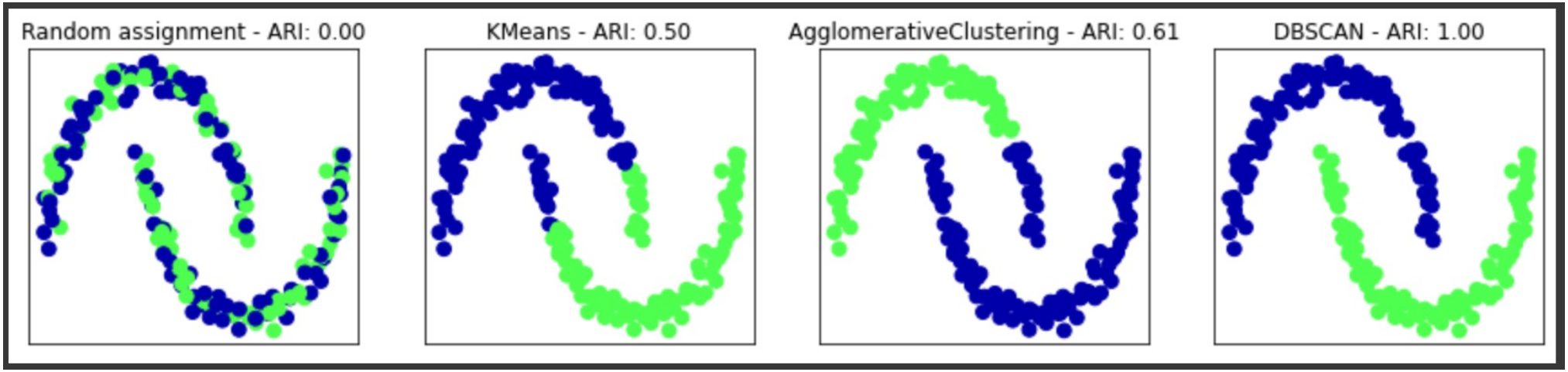
```
[22] from sklearn.preprocessing import StandardScaler
```

```
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# Rescale the data to zero mean and unit variance
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

dbscan = DBSCAN()
clusters = dbscan.fit_predict(X_scaled)
# plot the cluster assignments
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm2, s=60)
plt.xlabel("Feature 0")
plt.ylabel("Feature 1")
```

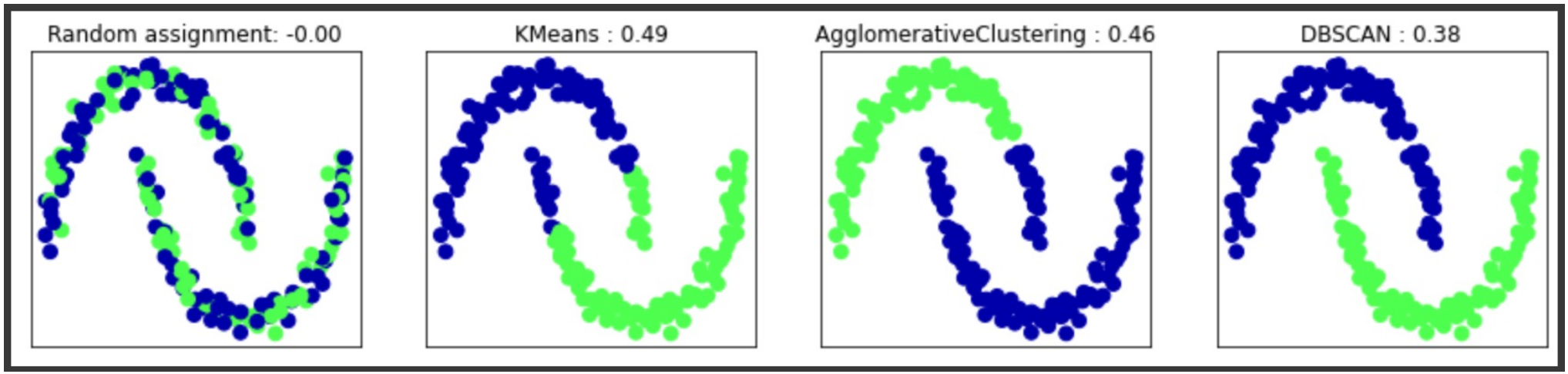
DBSCAN



Evaluating clustering with ground truth: adjusted rand index



Evaluating clustering without ground truth: Silhouette



Overview

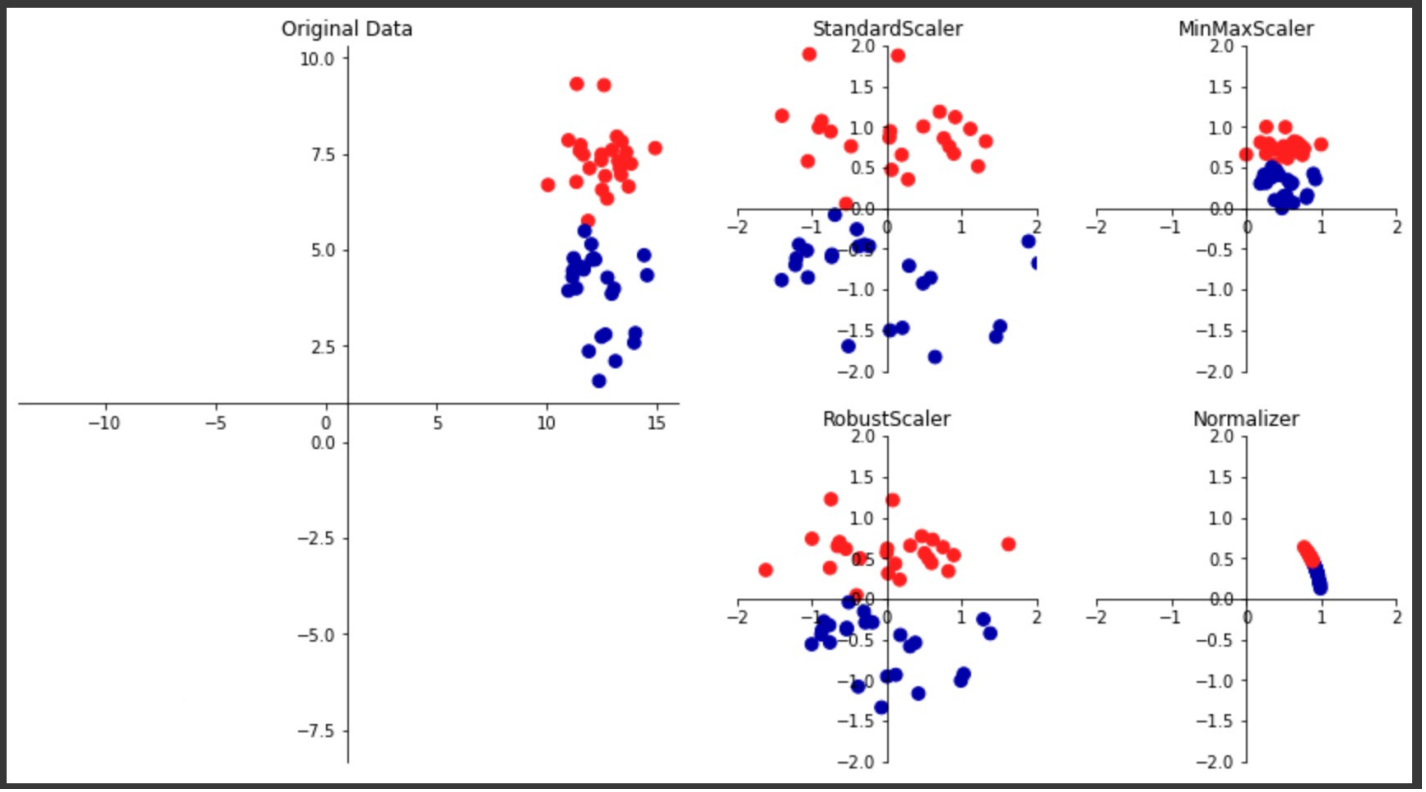
- Python tools for machine learning
  - First application
- Unsupervised learning
  - K-Means
- Agglomerative Clsutering and DBSCAN
- **Principal Component Analysis**

Preprocessing

- A common practice is to **adjust** the **features** so that the **data representation** is more suitable
- Often this is a **simple per-feature rescaling** and **shift** of the data
- A synthetic **two-class classification** dataset with **two features**
- The **first feature** (the x-axis value) is between **10 and 15** while the **second feature** (the y-axis value) is between around **1 and 9**

Preprocessing

```
mglearn.plots.plot_scaling()
```



Dimensionality reduction

- **Transforming data** using unsupervised learning can have **many motivations**

- The **most common** motivations are **visualization**, compressing the data, and finding a **representation** that is **more informative** for further processing

- One of the simplest and most widely used algorithms is **Principal Component Analysis**

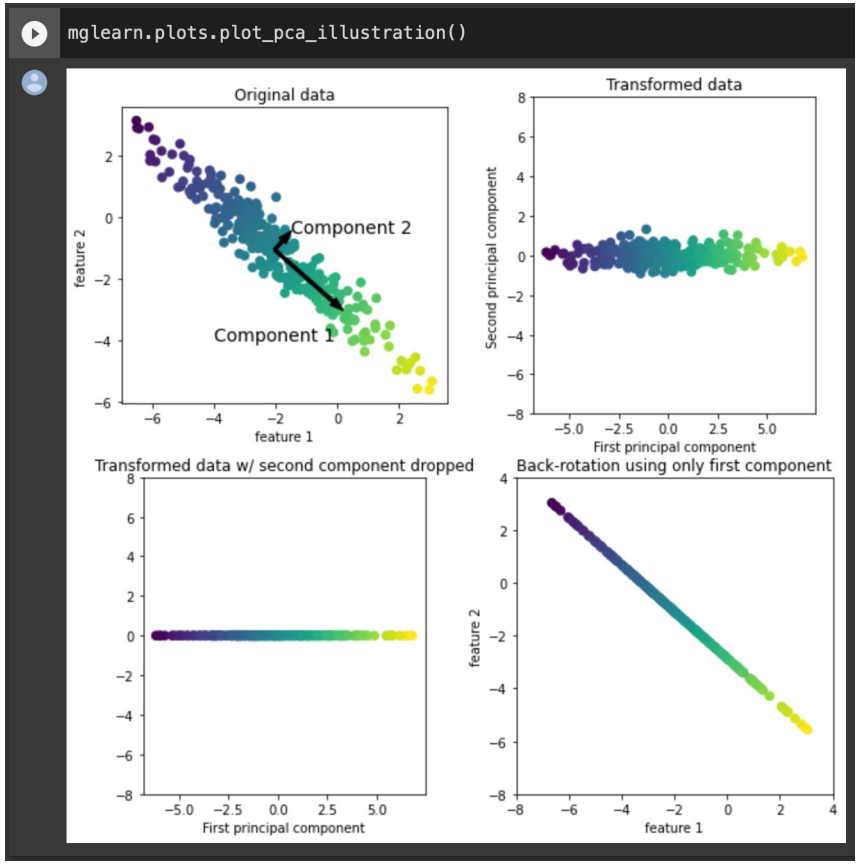
Principal Component Analysis

- **Principal component analysis** is a method that **rotates** the **dataset** in a way such that the **rotated features** are **statistically uncorrelated**

- This **rotation** is often followed by **selecting** only a **subset** of the **new features**, **according** to how **important** they are for explaining the data

Principal Component Analysis

- The first plot (top left) shows the **original data** points
- The algorithm proceeds by first finding the **direction of maximum variance**, that contains **most** of the **information**
- The second plot (top right) shows the same **data**, but now **rotated** so that the first principal component aligns with the x-axis and the second principal component aligns with the y-axis



Principal Component Analysis

- One of the most **common applications** of **PCA** is **visualizing high-dimensional data**
- It is **difficult** to create **scatter plots** of data that has **more than two features**
- There is an even **simpler visualization**, that is computing **histogram** of each feature for each class

Principal Component Analysis

- Execute cell: **Different Kinds of Preprocessing**

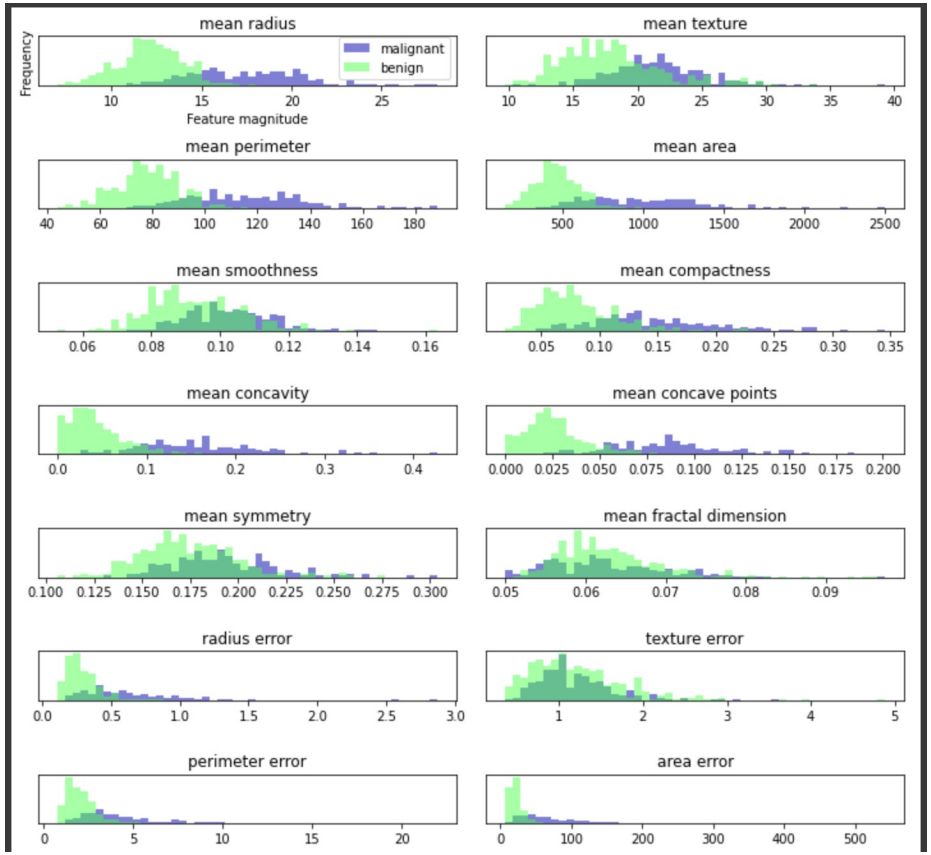
```
import matplotlib.pyplot as plt

[9] from sklearn.datasets import load_breast_cancer

[13] import numpy as np
```

- Execute cell: **Applying PCA to the cancer dataset for visualization**

Principal Component Analysis

- Histogram for each of the features, counting how often a data point appears with a feature in a certain range

- Each plot overlays two histograms, one for all of the points in the benign class and one for all the points in the malignant class



Principal Component Analysis

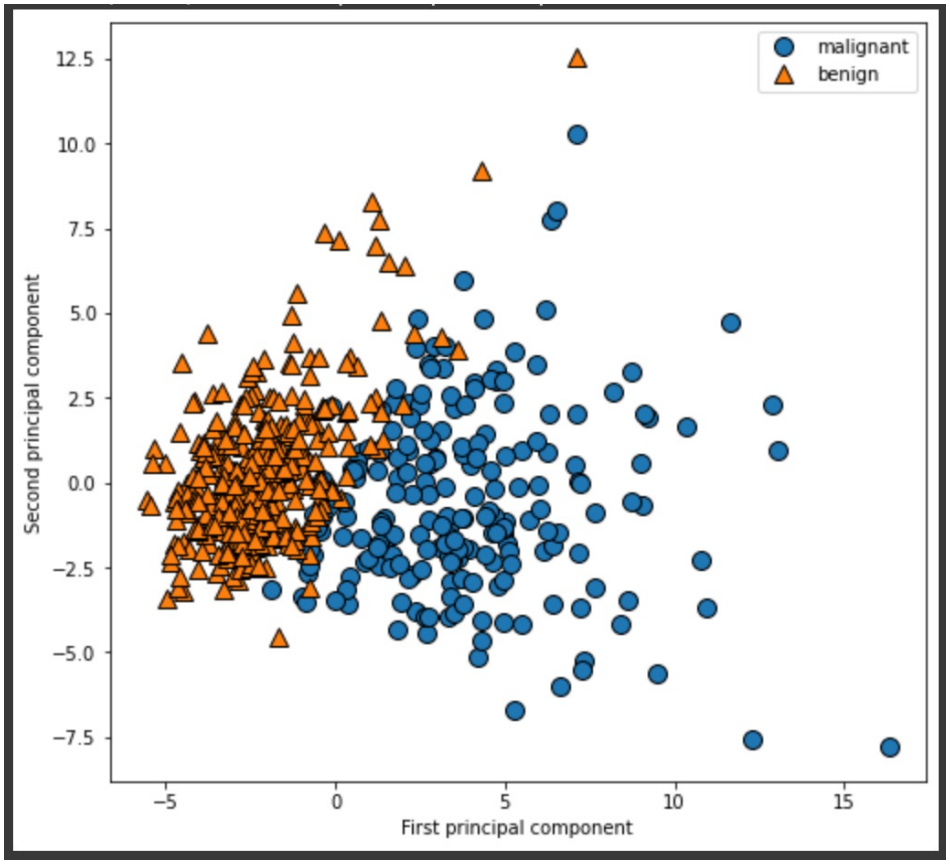
```
[16] from sklearn.preprocessing import StandardScaler
```

```
[17] from sklearn.datasets import load_breast_cancer
     cancer = load_breast_cancer()

     scaler = StandardScaler()
     scaler.fit(cancer.data)
     X_scaled = scaler.transform(cancer.data)
```

```
[18] from sklearn.decomposition import PCA
     # keep the first two principal components of the data
     pca = PCA(n_components=2)
     # fit PCA model to beast cancer data
     pca.fit(X_scaled)

     # transform data onto the first two principal components
     X_pca = pca.transform(X_scaled)
     print("Original shape: {}".format(str(X_scaled.shape)))
     print("Reduced shape: {}".format(str(X_pca.shape)))
```

```
     Original shape: (569, 30)
     Reduced shape: (569, 2)
```

Principal Component Analysis



Principal Component Analysis

- It is important to note that **PCA** is an **unsupervised method**, and does **not use** any **class information** when finding the rotation
- It simply **looks** at the **correlations** in the **data**
- A **drawback** of PCA is that the **two axes** in the plot are often **not** very **easy to interpret**
- The **principal components** correspond to **directions** in the **original data**, so they are **combinations** of the **original features**

Try different datasets...

https://scikit-learn.org/stable/modules/classes.html?highlight=dataset#module-sklearn.datasets