Unsupervised Learning

UNSUPERVISED LEARNING IN PYTHON

Unsupervised learning

- Unsupervised learning finds patterns in data
- E.g., *clustering* customers by their purchases
- Compressing the data using purchase patterns (*dimension reduction*)

Purchase Patterns: In the context of purchase patterns, the data typically involves customer behaviors, such as products bought, quantities, and frequencies. Dimension reduction could help:

- Identify key factors or patterns that explain the majority of purchase behaviors (e.g., grouping similar products or customers based on purchasing habits).
- Simplify data for visualization or downstream analysis (e.g., clustering).

Why It's Unsupervised:

No Explicit Labels: The data consists of raw purchase information without predefined labels (e.g., categories of customers or product groups).
Discovering Structure: The algorithm identifies patterns or underlying factors (e.g., principal components or clusters) without supervision or external guidance.

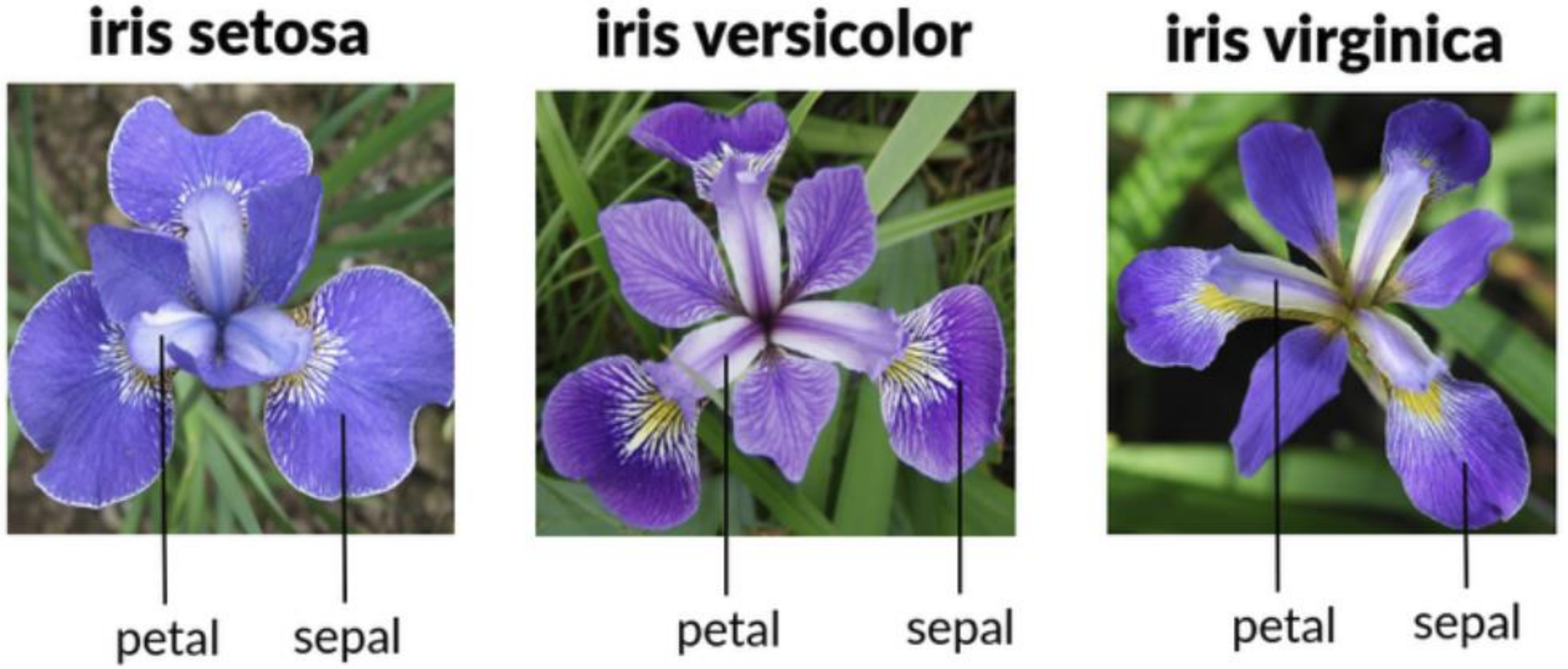
Examples of Unsupervised Learning for Purchase Patterns:
PCA: Reducing dimensions by identifying combinations of products frequently purchased together.
Clustering: Grouping customers or products based on similarity in purchasing behavior (e.g., k-means clustering).

Supervised vs unsupervised learning

- *Supervised* learning finds patterns for a prediction task
- E.g., classify tumors as benign or cancerous (*labels*)
- Unsupervised learning finds patterns in data
- ... but *without* a specific prediction task in mind

Iris (붓꽃) dataset

- Measurements of many iris plants
- Three species of iris:
  - *setosa*
  - *versicolor*
  - *virginica*
- Petal length, petal width, sepal length, sepal width (the *features* of the dataset)



[Petal: 꽃잎, Sepal: 꽃받침]

[1] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

Arrays, features & samples

- 2D NumPy array
- Columns are measurements (the *features*)
- Rows represent iris plants (the *samples*)

Iris data is 4-dimensional

- Iris samples are points in 4 dimensional space
- Dimension = number of features
- Dimension too high to visualize!
- ... but unsupervised learning gives insight

k-means clustering

- Finds clusters of samples
- Number of clusters must be specified
- Implemented in `sklearn` ("scikit-learn")

K-Means clustering is a **popular unsupervised machine learning algorithm** used to partition a dataset into a predefined number of groups (clusters).
Each cluster consists of data points that are more similar to each other than to those in other clusters, based on a chosen distance metric (e.g., Euclidean distance).

How K-Means Clustering Works
1. Initialization:
Choose the number of clusters, k
Randomly initialize k cluster centroids (points representing the center of each cluster).
2. Assignment Step:
Assign each data point to the nearest cluster centroid, typically using the Euclidean distance as the similarity measure.
3. Update Step:
Recalculate the centroid of each cluster by averaging the data points assigned to it.
4. Iteration:
Repeat the assignment and update steps until the cluster assignments no longer change significantly or a maximum number of iterations is reached.

```
print(samples)
```

```
[[ 5.    3.3   1.4   0.2]
  [ 5.    3.5   1.3   0.3]
  ...
  [ 7.2   3.2   6.    1.8]]
```

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3) model.fit(samples)
```

```
KMeans(n_clusters=3)
```

```
labels = model.predict(samples) print(labels)
```

```
[0 0 1 1 0 1 2 1 0 1 ...]
```

Cluster labels for new samples

- New samples can be assigned to existing clusters
- k-means remembers the mean of each cluster (the "centroids")
- Finds the nearest centroid to each new sample

Cluster labels for new samples

```
print(new_samples)
```

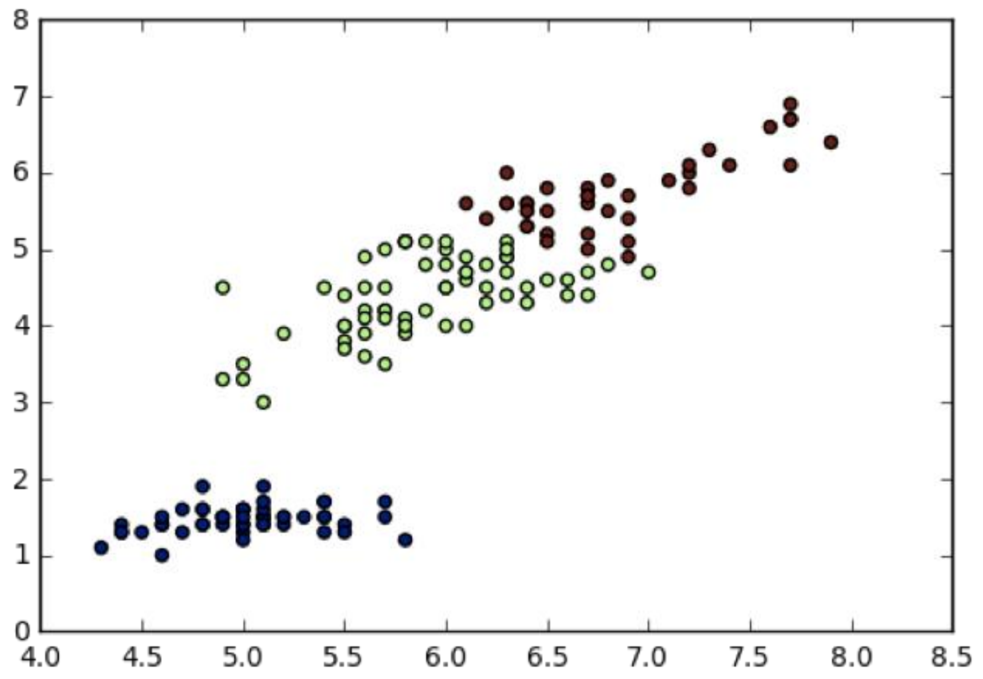
```
[[  5.7     4.4     1.5     0.4]
 [  6.5     3.      5.5     1.8]
 [  5.8     2.7     5.1     1.9]]
```

```
new_labels = model.predict(new_samples)
print(new_labels)
```

```
[0 2 1]
```

Scatter plots

- Scatter plot of sepal length vs. petal length
- Each point represents an iris sample
- Color points by cluster labels
- PyPlot (`matplotlib.pyplot`)



Scatter plots

```
import matplotlib.pyplot xs = as plt
samples[:,0]
ys = samples[:,2]
plt.scatter(xs, ys, c=labels) plt.show()
```

Evaluating a clustering

UNSUPERVISED LEARNING IN PYTHON

Evaluating a clustering

- Can check correspondence with e.g. iris species
- ... but what if there are no species to check against?
- Measure quality of a clustering
- Informs choice of how many clusters to look for

Iris: clusters vs species

- k-means found 3 clusters amongst the iris samples
- Do the clusters correspond to the species?

```
species     setosa    versicolor    virginica
labels
0                0             2           36
1               50             0            0
2                0            48           14
```

Cross tabulation with pandas

- Clusters vs species is a "cross-tabulation"
- Use the `pandas` library
- Given the species of each sample as a list `species`

```
print(species)
```

```
['setosa', 'setosa', 'versicolor', 'virginica', ... ]
```

Aligning labels and species

```
import pandas  as  pd
df = pd.DataFrame({'labels': labels, 'species': species}) print(df)
```

```
         labels        species
0             1         setosa
1             1         setosa
2             2     versicolor
3             2      virginica
4             1         setosa
...
```

Crosstab of labels and species

```
ct = pd.crosstab(df['labels'], df['species']) print(ct)
```

```
species      setosa      versicolor      virginica
labels
0                 0               2             36
1                50               0              0
2                 0              48             14
```

How to evaluate a clustering, if there were no species information?

Measuring clustering quality

- Using only samples and their cluster labels
- A good clustering has tight clusters
- Samples in each cluster bunched together

- In clustering, **inertia** measures the quality of clusters by calculating the sum of squared distances (SSD) between each data point and its assigned cluster's centroid.
- It reflects how tightly the data points in a cluster are grouped around the centroid.

Lower Inertia Indicates Better Clustering:

1. A lower inertia value suggests that the data points within each cluster are closer to their centroids, indicating tighter and more cohesive clusters.

2. Minimizing Inertia: The goal of algorithms like K-Means is to minimize inertia. It adjusts the centroids and reassigns points to clusters iteratively to achieve this.

Inertia measures clustering quality

- Measures how spread out the clusters are (*lower* is better)
- Distance from each sample to centroid of its cluster
- After `fit()`, available as attribute `inertia_`
- k-means attempts to minimize the inertia when choosing clusters

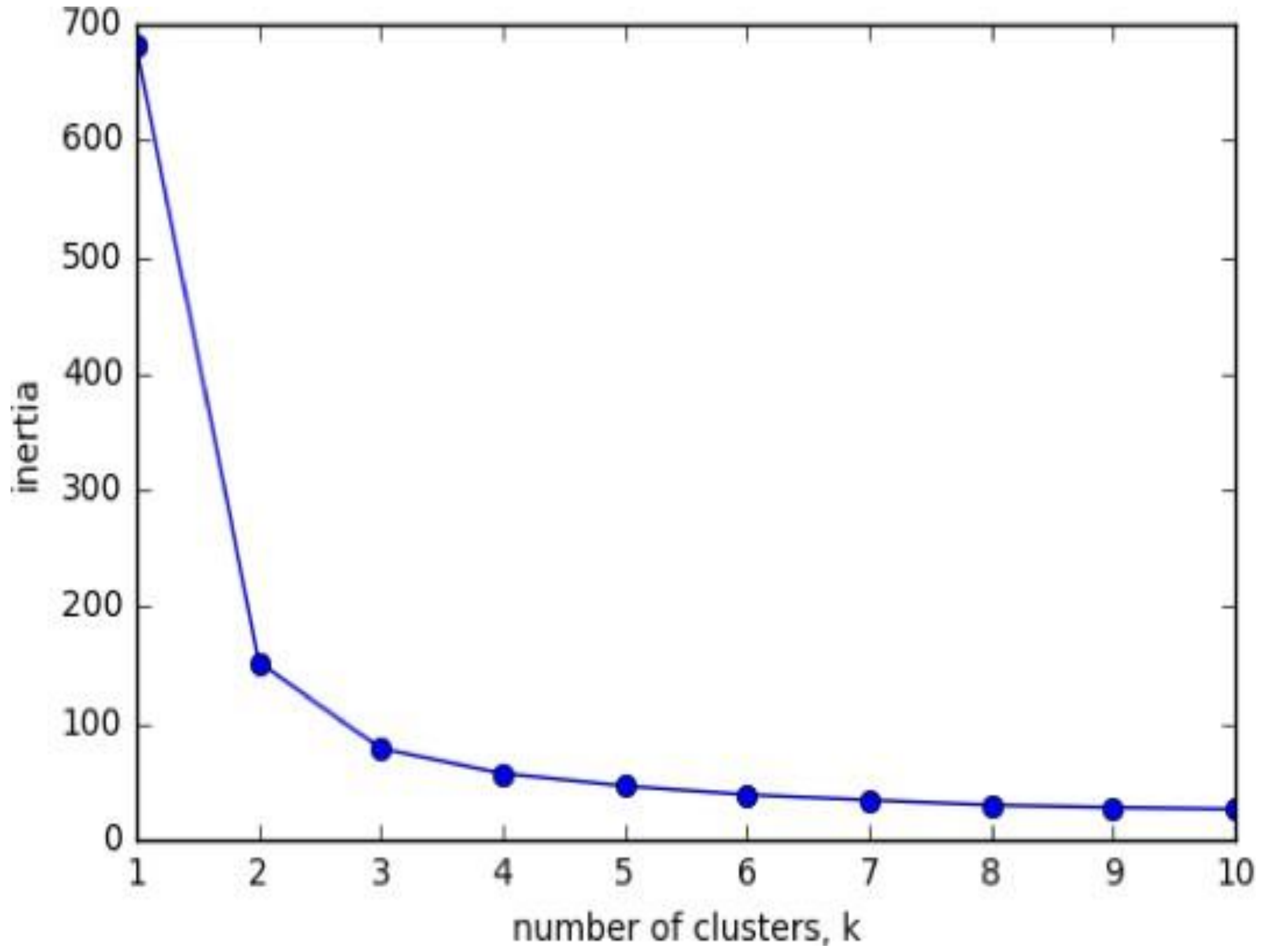
```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)
model.fit(samples) print(model.inertia_)
```

```
78.9408414261
```

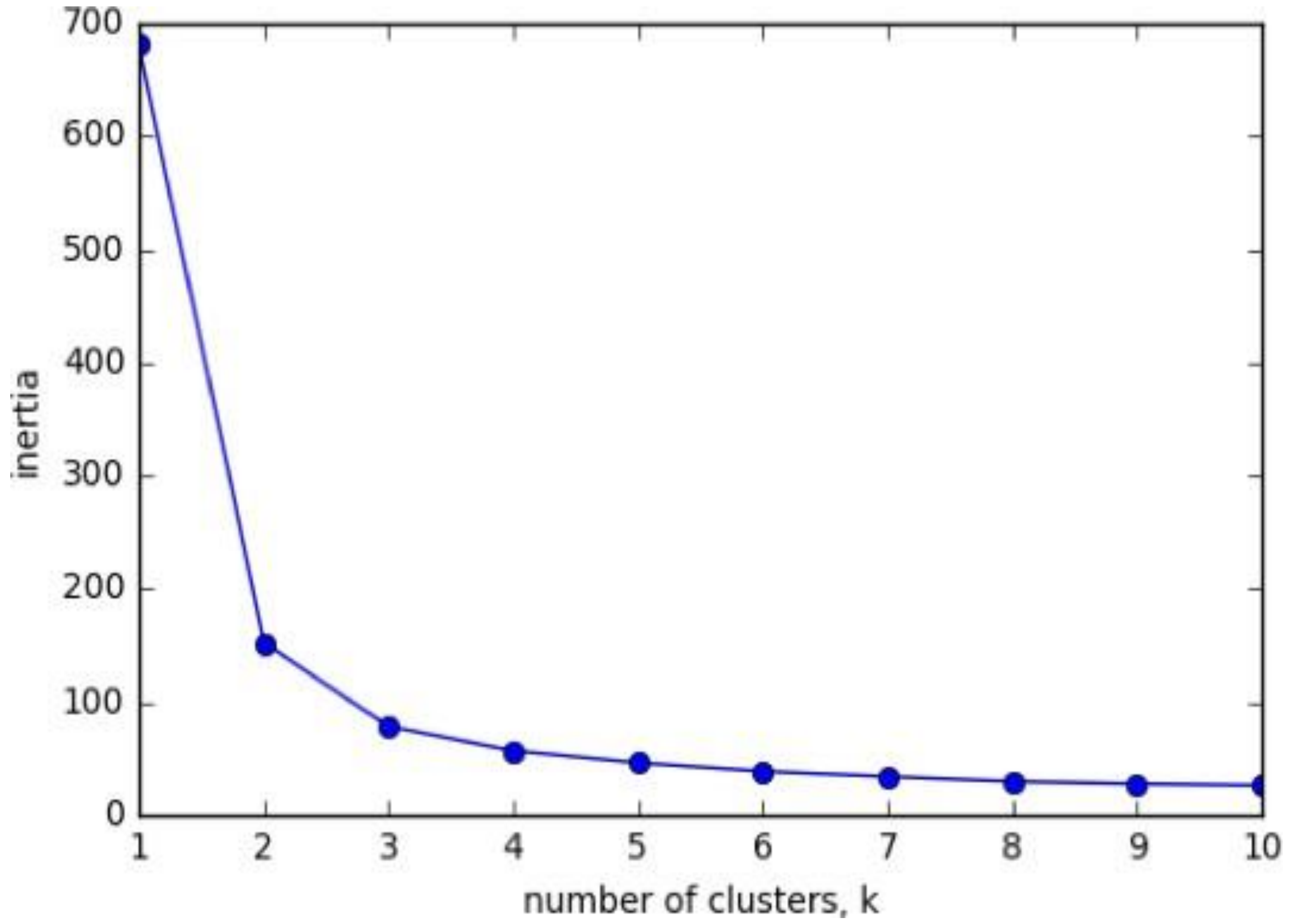
The number of clusters

- Clusterings of the iris dataset with different numbers of clusters
- More clusters means lower inertia
- What is the best number of clusters?



How many clusters to choose?

- A good clustering has tight clusters (so low inertia)
- ... but not too many clusters!
- Choose an "elbow" in the inertia plot
- Where inertia begins to decrease more slowly
- E.g., for iris dataset, 3 is a good choice



Transforming features for better clusterings

UNSUPERVISED LEARNING IN PYTHON

Piedmont wines dataset

- 178 samples from 3 distinct varieties of red wine: Barolo, Grignolino and Barbera
- Features measure chemical composition e.g. alcohol content
- Visual properties like "color intensity"

[1] Source: https://archive.ics.uci.edu/ml/datasets/Wine

Clustering the wines

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
labels = model.fit_predict(samples)
```

Clusters vs. varieties

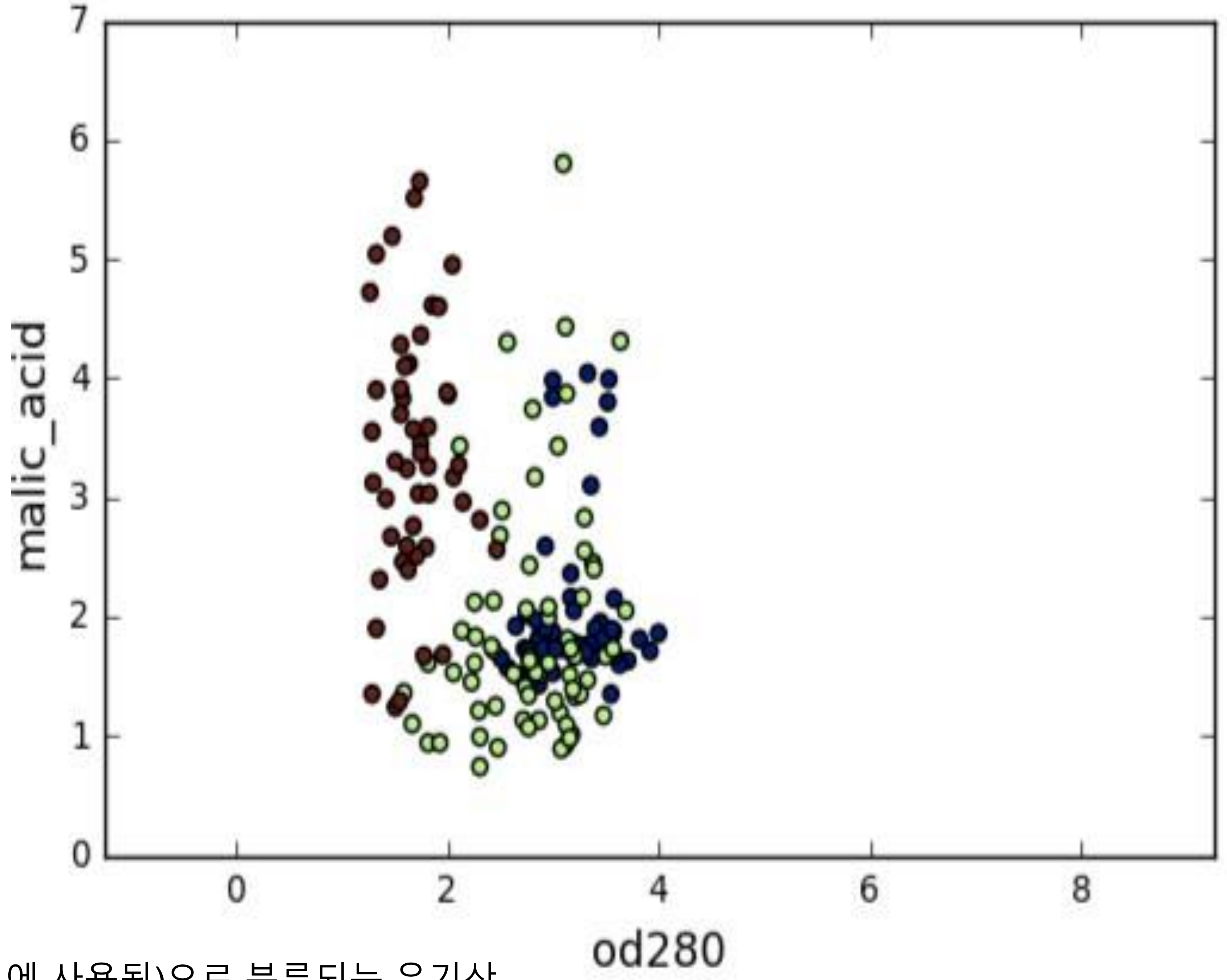
```
df = pd.DataFrame({'labels': labels,
                                    'varieties':    varieties})    ct    =
pd.crosstab(df['labels'], df['varieties']) print(ct)
```

```
varieties      Barbera    Barolo     Grignolino
labels
0                   29        13             20
1                    0        46              1
2                   19         0             50
```

Feature variances

- The wine features have very different variances!
- Variance of a feature measures spread of its values

feature	variance
alcohol	0.65
malic_acid	1.24
...	
od280	0.50
proline	99166.71

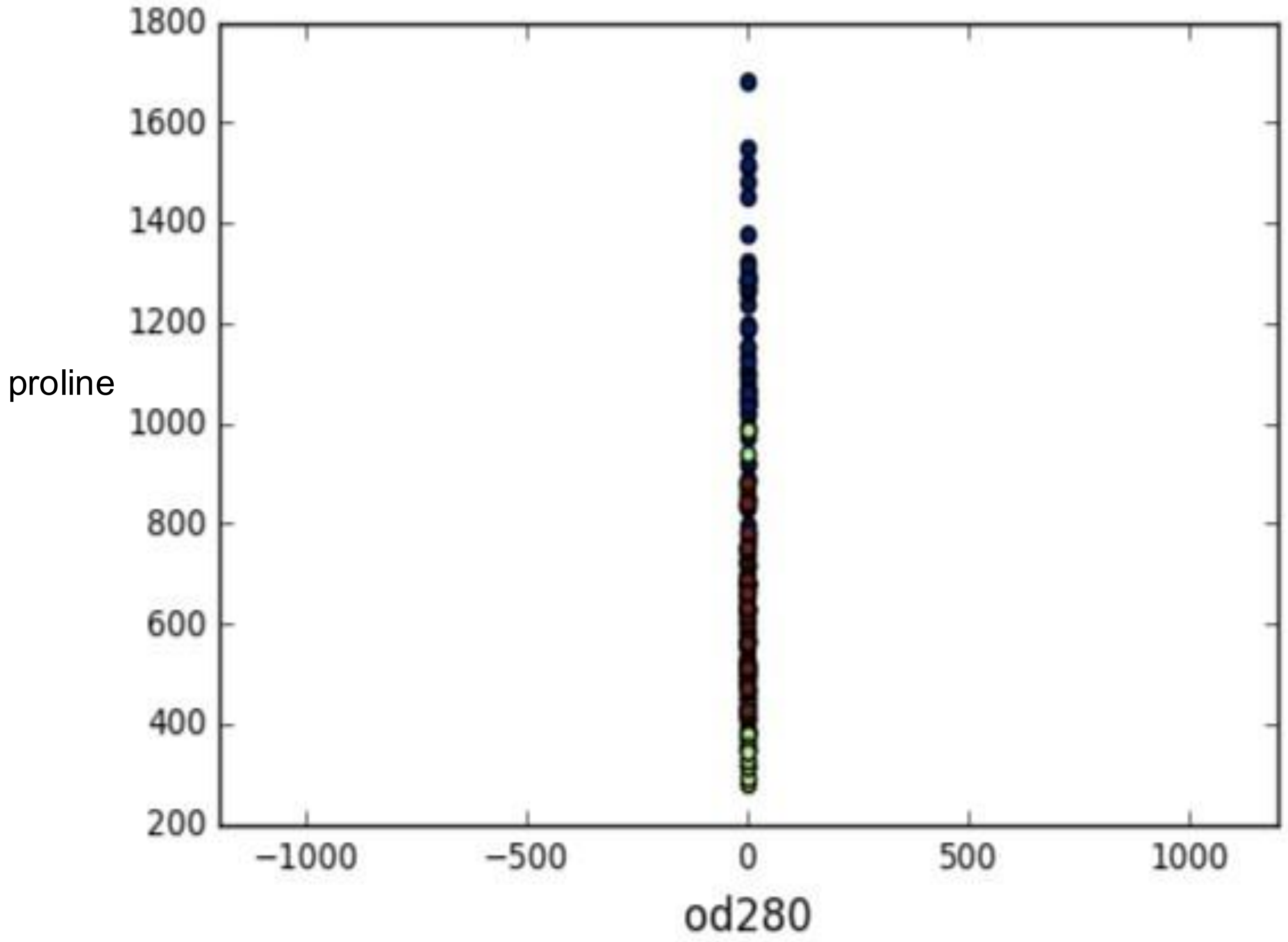


proline: 단백질생성성 아미노산(단백질의 생합성에 사용됨)으로 분류되는 유기산
od280: method for calculating protein concentration
malic acid: 말산은 모든 생물에서 만들어지고, 과일의 유쾌한 신맛에 기여하며, 식품 첨가물로 사용되는 다이카복실산

Feature variances

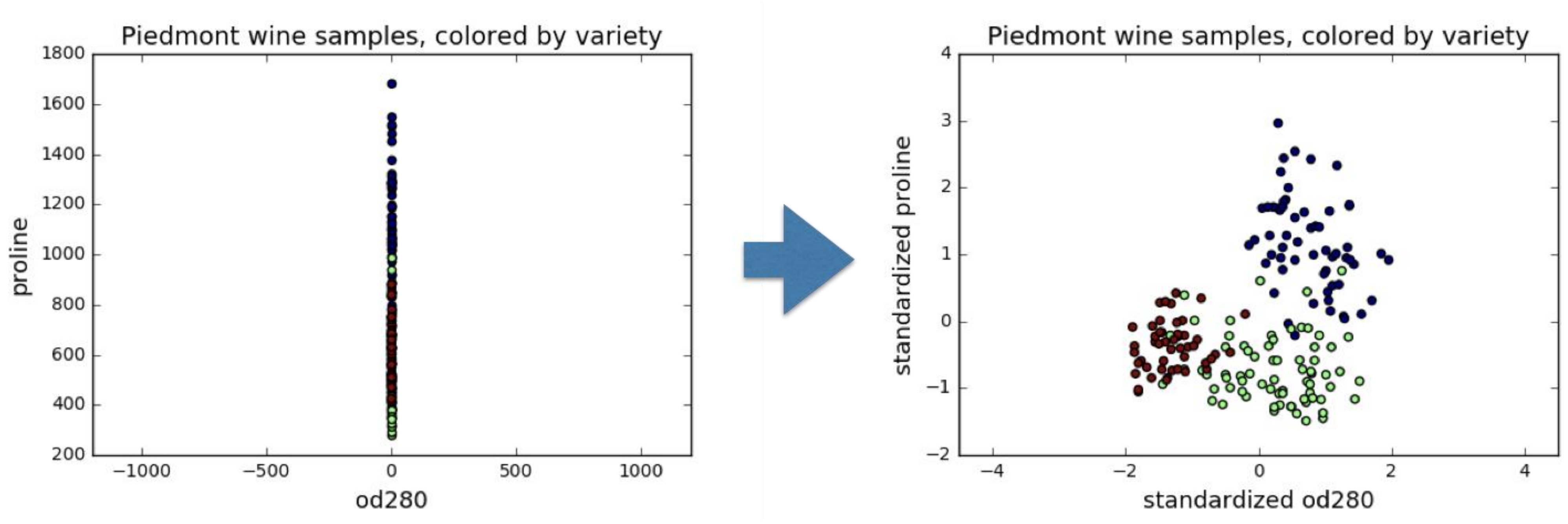
- The wine features have very different variances!
- Variance of a feature measures spread of its values

feature	variance
alcohol	0.65
malic_acid	1.24
...	
od280	0.50
proline	99166.71



StandardScaler

- In kmeans: feature variance = feature influence
- `StandardScaler` transforms each feature to have mean 0 and variance 1
- Features are said to be "standardized"



sklearn StandardScaler

```
from sklearn.preprocessing scaler =   import StandardScaler
StandardScaler() scaler.fit(samples)


StandardScaler(copy=True, with_mean=True, with_std=True) samples_scaled =
scaler.transform(samples)
```

Similar methods

- `StandardScaler` and `KMeans` have similar methods
- Use `fit()` / `transform()` with `StandardScaler`
- Use `fit()` / `predict()` with `KMeans`

StandardScaler, then KMeans

- Need to perform two steps: `StandardScaler`, then `KMeans`
- Use `sklearn` pipeline to combine multiple steps
- Data flows from one step into the next

Pipelines combine multiple steps

```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
scaler = StandardScaler()
kmeans = KMeans(n_clusters=3)
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(scaler, kmeans) pipeline.fit(samples)
```

```
Pipeline(steps=...)
```

```
labels = pipeline.predict(samples)
```

Feature standardization improves clustering

With feature standardization:

```
varieties      Barbera   Barolo   Grignolino
labels
0                    0       59            3
1                   48        0            3
2                    0        0           65
```

Without feature standardization was very bad:

```
varieties      Barbera   Barolo   Grignolino
labels
0                   29       13           20
1                    0       46            1
2                   19        0           50
```

sklearn preprocessing steps

- `StandardScaler` is a "preprocessing" step
- `MaxAbsScaler` and `Normalizer` are other examples

MaxAbsScaler is a feature scaling technique provided by the sklearn.preprocessing module in scikit-learn.
It scales each feature individually such that the **maximum absolute value of each feature becomes 1**, while preserving the sparsity of the data.
It is particularly useful for data that contains both positive and negative values and does not center the data at zero.

The **Normalizer** in sklearn.preprocessing is a preprocessing technique that **scales each data sample individually** to have a unit norm.
This is done by dividing each element in a sample by the norm of the sample vector.
It is commonly used when the direction of the data is more important than its magnitude, such as in text classification, clustering, or machine learning models sensitive to cosine similarity.

The norm $\|x\|$ can be calculated using one of the following:
L1 Norm
Scales the sample so that the sum of absolute values of its components equals 1.

L2 Norm:
Scales the sample so that the Euclidean norm (vector magnitude) equals 1. This is the default.

L∞ Norm:
Scales the sample so that the maximum absolute value equals 1.