Unsupervised learning finds patterns in data, like clusters of customers by their purchases, or compressing the data using purchasing patterns (dimension reduction)

Unsupervised learning learns without labels, nor a specific prediction task in mind

Data will be written in 2D NumPy arrays. Columns will be features, and rows will be samples.

The samples of this dataset is in 4 dimensions.

Dimension = number of features

We can reduce dimensionality with k-means clustering from sklearn

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

# new data can be clustered without starting over, they remember the centroids

new_labels = model.predict(new_samples)

# we can use scatter plots with pyplot from matplotlib to visualize

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

Also, we can use a different function to combine fit and predict

```
model.fit_predict()
```

We can put points and corresponding centroids on the same graph

```
# Import pyplot
import matplotlib.pyplot as plt

# Assign the columns of new_points: xs and ys
xs = new_points[:,0]
ys = new_points[:,1]

# Make a scatter plot of xs and ys, using labels to define the colors
plt.scatter(xs, ys, c=labels, alpha=0.5)
```

```
# Assign the cluster centers: centroids
centroids = model.cluster_centers_

# Assign the columns of centroids: centroids_x, centroids_y
centroids_x = centroids[:,0]
centroids_y = centroids[:,1]

# Make a scatter plot of centroids_x and centroids_y
plt.scatter(centroids_x, centroids_y, marker='D', s=50)
plt.show()
```

We can evaluate a customer by checking correspondence with the species

We can **cross tabulate** with pandas

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

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

We can measure clustering quality using only samples and their cluster labels
A good clustering has tight clusters, measured by the inertia, the distance from each sample to centroid of its cluster

```
# is it automatically measured
print(model.inertia_)
```

Sometimes, as number of clusters increases, inertia decreases

A good clustering has tight clusters (so low inertia), we need to choose an "elbow" point on the inertia plot for the right number of clusters

Transforming features for better clusterings

Dataset: 178 samples, from 3 distinct varieties

Feature variance requires feature transformation, KMeans won't be effective

We can use StandardScaler to standardize features

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(samples)
StandardScaler(copy=True, with_mean=True, with_std=True)
samples_scaled = scaler.transform(samples)
```

While KMeans fits and predicts, StandardScaler fits and scales

We can use a pipeline to combine these

```
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(scaler, kmeans)
pipeline.fit(samples)
labels = pipeline.predict(samples)
```