Overview

In this walkthrough, we'll use **K-Means Clustering** to group data into meaningful clusters using Python. We'll work with the classic **Iris dataset**, which contains measurements of iris flowers from three species. The idea is to see if the algorithm can find natural groupings in the data without using the species labels. We'll go through loading the data, running the model, picking the right number of clusters, and visualizing the results.

Libraries

We'll use a few standard Python libraries to handle data, run the clustering algorithm, and create visuals.

- **pandas** for working with the dataset
- **numpy** for numerical operations
- **matplotlib** and **seaborn** for plotting
- **scikit-learn** for the K-Means implementation and loading the Iris dataset

Here is the python code we will use in order to import everything:

"import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

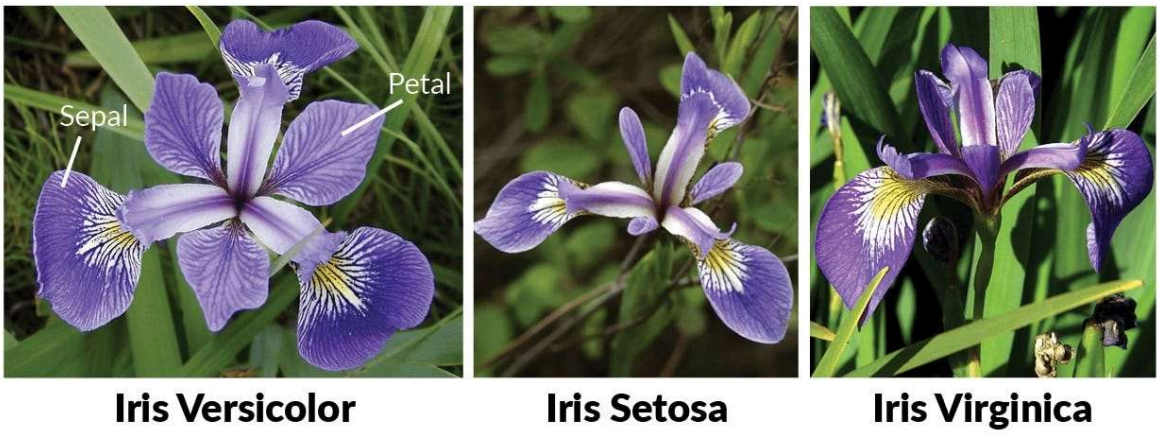
import matplotlib.pyplot as plt

import numpy as np"

Loading and Preparing the Data

We'll use the Iris dataset directly from **scikit-learn**, which makes it easy to work with. After loading it, we'll convert it into a pandas DataFrame for easier handling, then scale the features so they're on the same range. Scaling is important because K-Means relies on distance, and features with larger values can dominate the results if left unscaled.

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.



source image: https://miro.medium.com/max/2550/0*GVjzZeYrir0R_6-X.png

Here is the python code to load, convert and scale the data:

"iris = load_iris()

X = iris.data

y = iris.target

df = pd.DataFrame(X, columns=iris.feature_names)

print(df.head())

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)”

Output:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

*The Iris dataset contains **150 samples** of iris flowers, each with **four features**:*

- ***Sepal length** (cm)*
- ***Sepal width** (cm)*
- ***Petal length** (cm)*
- ***Petal width** (cm)*

*These are simple physical measurements of the flowers. We’ll use these features to group the flowers using **K-Means Clustering**, without giving the model any information about the actual species. Later, we’ll compare its clusters to the real labels (setosa, versicolor, virginica) to see how well it performed.*

*Before running the model, we need to choose the number of clusters, **k**, to tell K-Means how many groups to find. One common way to do this is the **Elbow Method**.*

Choosing k with the Elbow Method

*The Elbow Method involves running K-Means for a range of cluster numbers and tracking the **Sum of Squared Errors (SSE)** for each. SSE measures how tightly the points are grouped around their centroids. As k increases, SSE naturally decreases, but at some point the improvement slows down and forms an “elbow” shape. That point is a good choice for k.*

Here is the python code:

“sse = []

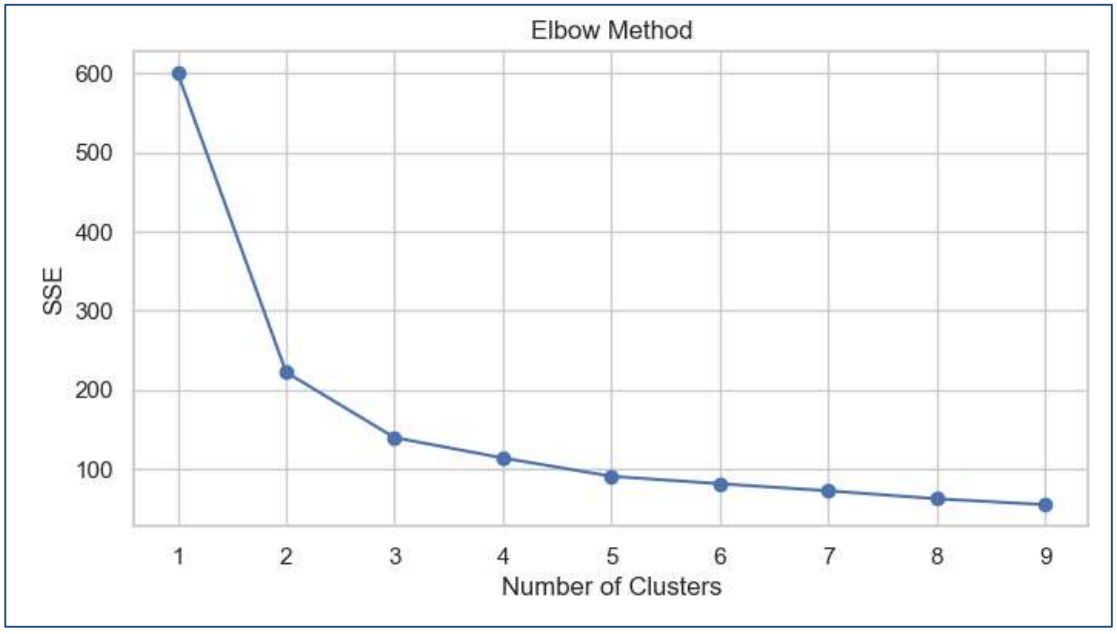
k_values = range(1, 10)

for k in k_values:

```
    kmeans = KMeans(n_clusters=k, init='k-means++', n_init=10, random_state=42)
    kmeans.fit(X_scaled)
   sse.append(kmeans.inertia_)

plt.figure(figsize=(8, 4))
plt.plot(k_values, sse, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE')
plt.xticks(k_values)
plt.show()"
```

output:



*You should see a clear **"elbow" at k = 3**, which matches the fact that there are three species in the dataset. This gives us a solid basis to proceed with K-Means using **k = 3**.*

Running K-Means with k = 3

*Now that we've chosen **k = 3**, we can run the K-Means algorithm on the scaled flower data. This will group the flowers into three clusters based on their measurements alone, without using the actual species labels.*

The python code:

"kmeans = KMeans(n_clusters=3, init='k-means++', n_init=10, random_state=42)

kmeans.fit(X_scaled)

df['cluster'] = kmeans.labels_

print(df.head())"

Output:

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2   
1                4.9               3.0                1.4               0.2   
2                4.7               3.2                1.3               0.2   
3                4.6               3.1                1.5               0.2   
4                5.0               3.6                1.4               0.2   

   cluster species  
0        1  setosa  
1        1  setosa  
2        1  setosa  
3        1  setosa  
4        1  setosa  
```

Here's what's happening:

- *We tell the model to find **3 clusters**.*
- *init='k-means++' gives it a smart way to pick starting points.*
- *n_init=10 means it will run the algorithm 10 times and pick the best result.*
- *After fitting, the model assigns each flower a **cluster label** (0, 1, or 2).*

These labels represent the group each flower belongs to according to K-Means, not the actual species.

Visualizing the Clusters

To see how well the model grouped the flowers, we'll create two scatter plots:

1. ***One colored by the K-Means clusters***
2. ***One colored by the actual flower species***

*Since the data has four features, we'll use **PCA** to reduce it to two dimensions so we can plot it on a 2D graph. We'll also make sure the **colors match** between both graphs, so it's easy to compare how close the model's clusters are to the real species.*

The python code:

"pca = PCA(n_components=2, random_state=42)

X_2d = pca.fit_transform(X_scaled)

cluster_colors = {0: 'orange', 1: 'blue', 2: 'green'}

species_colors = {0: 'blue', 1: 'orange', 2: 'green'}

plt.figure(figsize=(7, 5))

for c in np.unique(clusters):

idx = clusters == c

plt.scatter(

X_2d[idx, 0], X_2d[idx, 1],

color=cluster_colors[c],

label=f'Cluster {c}', alpha=0.7

)

plt.title('PCA view, colored by K-Means clusters')

plt.xlabel('PC1')

plt.ylabel('PC2')

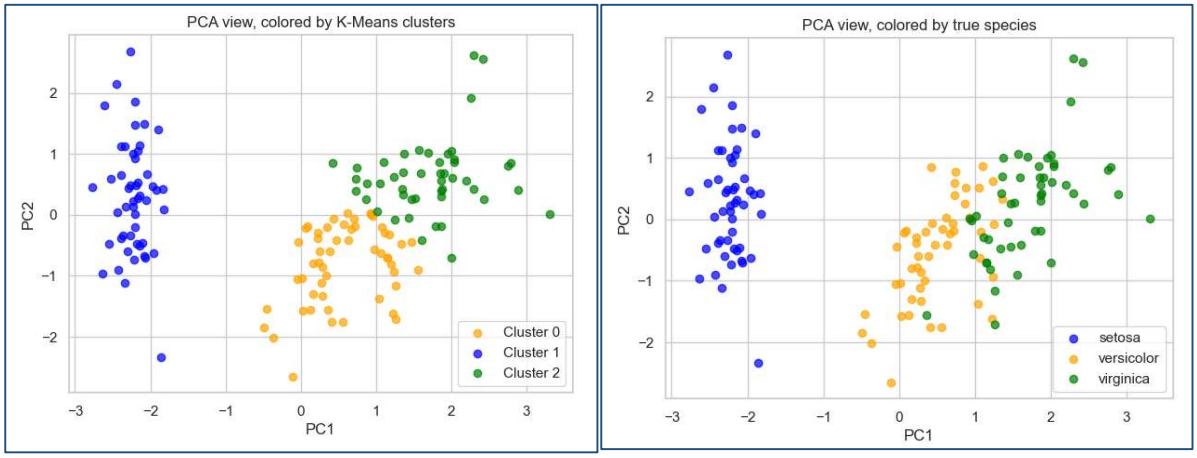
plt.legend()

plt.show()

plt.figure(figsize=(7, 5))

```
for t in np.unique(y):
    idx = y == t
    plt.scatter(
        X_2d[idx, 0], X_2d[idx, 1],
        color=species_colors[t],
        label=species_names[t], alpha=0.7
    )
plt.title('PCA view, colored by true species')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.legend()
plt.show()”
```

Output:



Understanding the Cluster Plot

The scatter plot above shows the **K-Means clusters** after reducing the data to two dimensions with PCA. Each point represents a flower, and the colors show which **cluster** the model assigned it to:

- **Cluster 0 (Orange)**
- **Cluster 1 (Blue)**
- **Cluster 2 (Green)**

You can see that the model has grouped the flowers into three distinct regions based only on their measurements, without using the actual species labels.

Comparing to the Actual Flowers

When we plot the **real species** using the same PCA projection and matching colors (Setosa = Blue, Versicolor = Orange, Virginica = Green), we can visually compare how well the model did.

- The **blue group** (Setosa) is very well separated and matches almost perfectly with Cluster 1.
- The **orange and green groups** (Versicolor and Virginica) overlap a bit, and the model's clusters reflect this.

This shows that K-Means was able to **identify the natural groups** in the data quite well, especially for Setosa, though there's some mixing between the other two species — which is expected, since their measurements are more similar.

We can measure the accuracy of the clusters by using the following code:

“ **labels = np.zeros_like(clusters)**

for i in range(3): # since k = 3

mask = (clusters == i)

labels[mask] = mode(y[mask], keepdims=True)[0]

accuracy = np.mean(labels == y)

print(f"Clustering accuracy: {accuracy * 100:.2f}%") ”

output:

```
Clustering accuracy: 83.33%
```

As you can see the clusters match the actual different flower species to about 83%.