

Unsupervised Learning

Unsupervised learning algorithms reveal hidden patterns and structures in datasets, allowing us to analyze and model data without using labels. This is especially useful in real-world situations where getting labeled data can be hard, costly, or even impossible.

K-means clustering is a popular method used for applications such as market segmentation, anomaly detection, image compression, and more. It is one of several clustering techniques, alongside others like hierarchical clustering and DBSCAN.

Imagine you have data about the measurements of Iris flowers, and your job is to identify a flower based on its physical measurements. This is a [Data set](#)

Learning Objectives

- Learn how K-means works, when to use it, and its limitations.
- Learn how to build a K-means clustering algorithm.
- Learn how to evaluate and interpret a K-means model.

Prerequisites: Familiarity with Python, basic statistics, and machine learning concepts is helpful.

Teaching Notes:
This notebook is based on the idea that a student should be able to:

- Understand the fundamental way a model works.
- Understand when a model is applicable, and what its limitations are.
- Understand how to code a model, including the use of its parameters.
- Evaluate the quality of the model's predictions.
- Interpret any insights a model has. Very often, students fail the last step. While an accurate model is valuable, an interpretable model guides a business to make better decisions. An exercise was also included in the same notebook where students have to essentially fill in the blanks, but also interpret the output of a model. This would likely be better in a separate notebook. This notebook is a MVP and with contribution from the team and continuous feedback from students, the concepts can be refined.

Lets start by importing the libraries we'll need, and import the dataset.

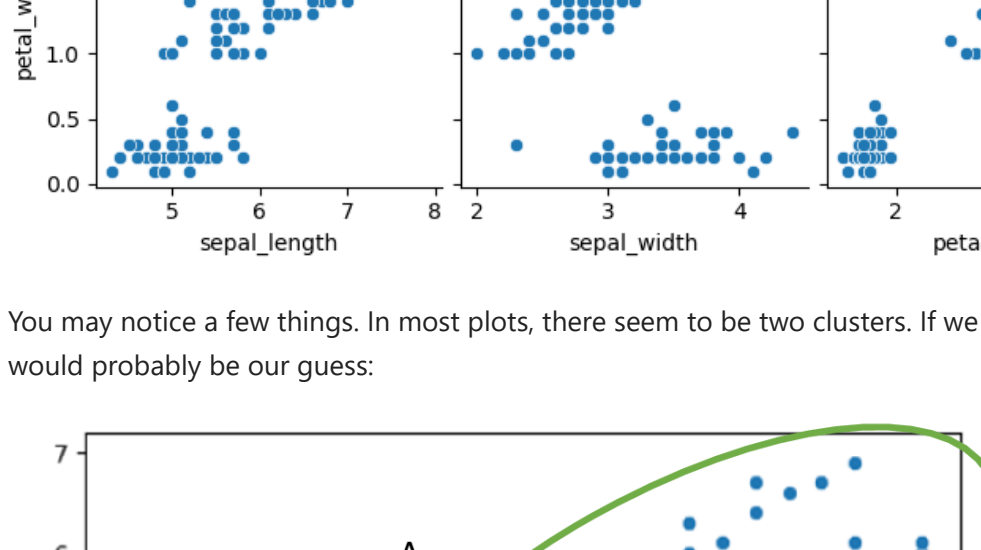
```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import MinMaxScaler

#Using the URL functionality in Pandas to import the set
data=pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data',header=None,nam
why did this work? data isn't .csv? See if you can find out.
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

The table contains the following columns:

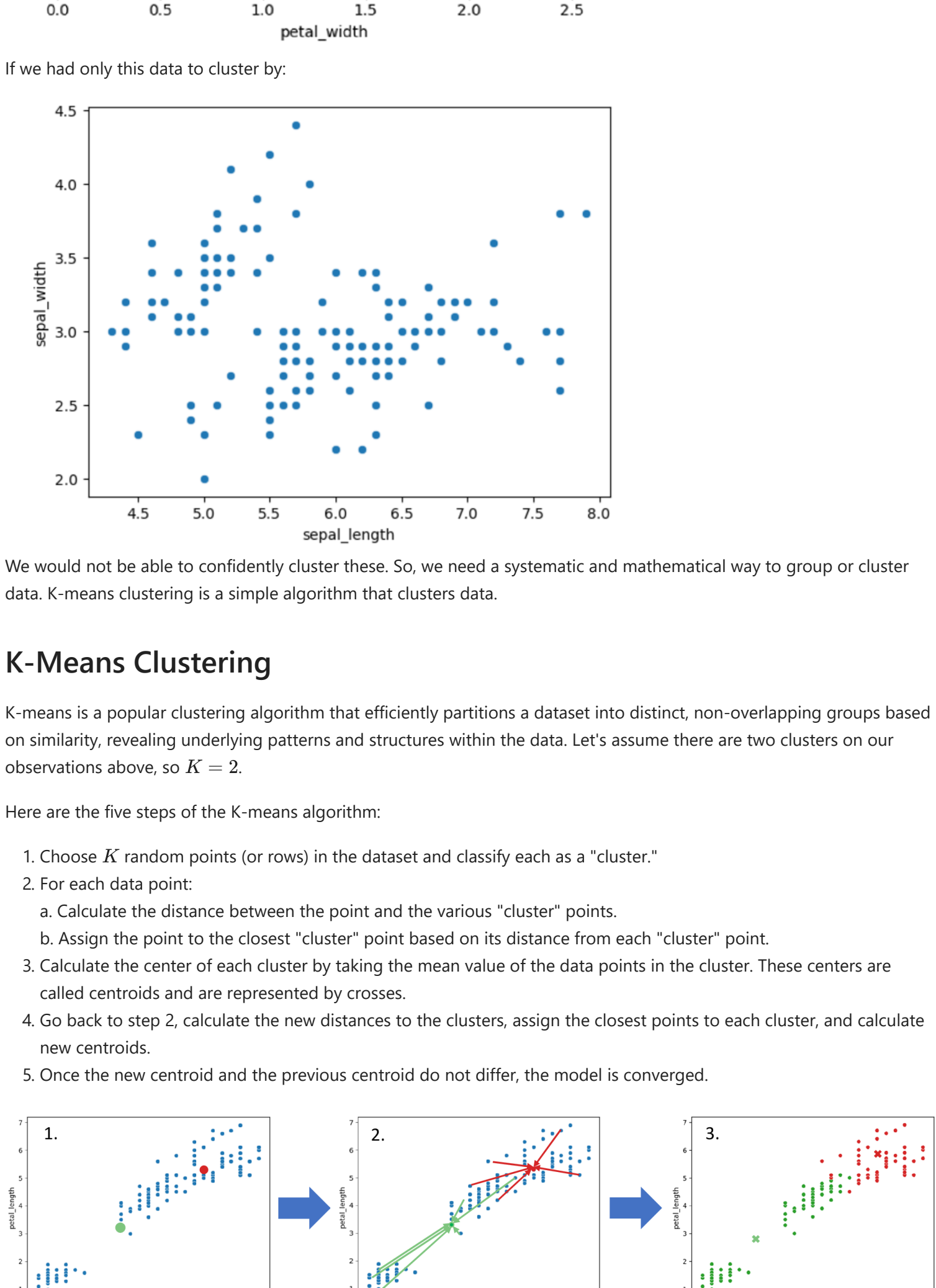
- Sepal length (cm)
- Sepal width (cm)
- Petal length (cm)
- Petal width (cm)
- Species name - Target column



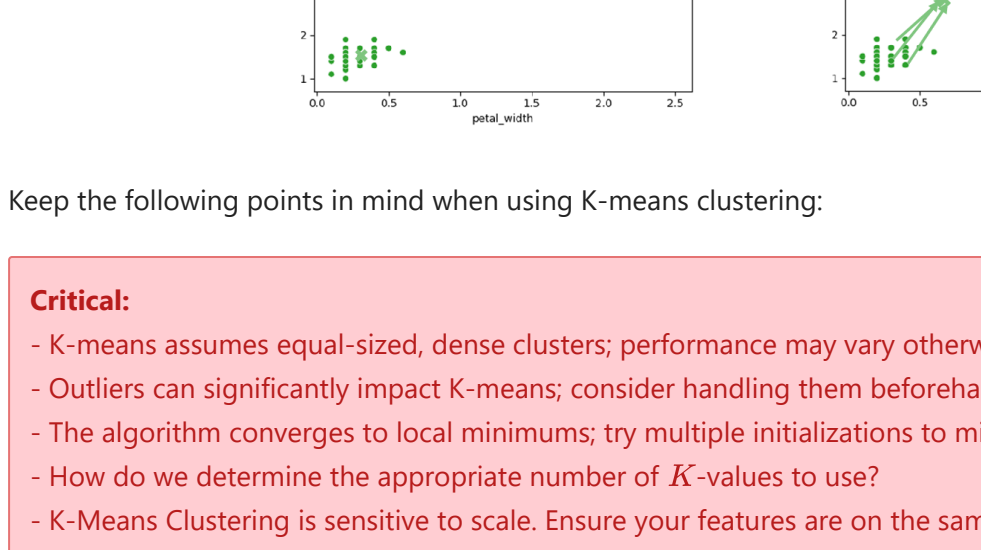
Source image: [Iris](#)

Let's ignore the "Species" column, as we won't usually have that information.

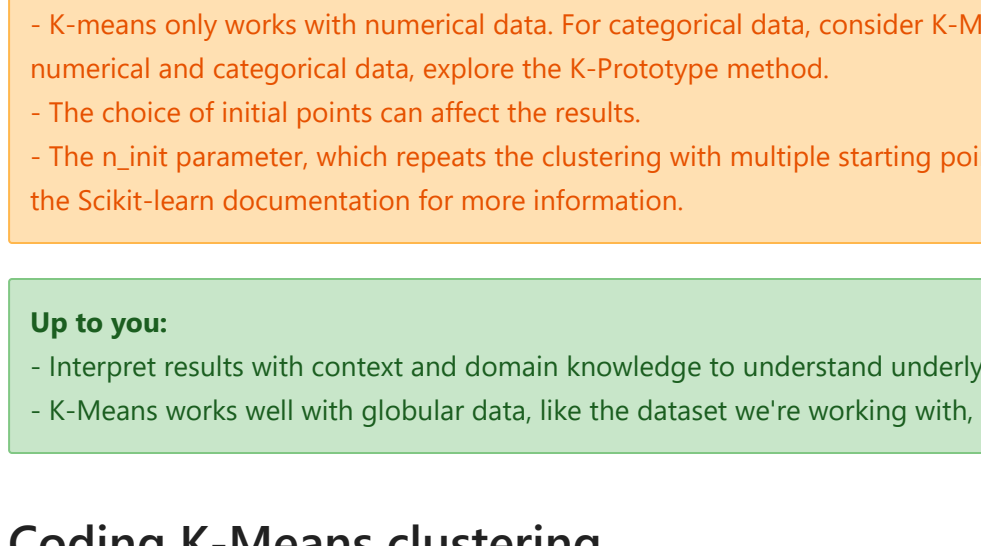
Using the seaborn library, let's plot the variables against one another to get an idea of the distributions.



You may notice a few things. In most plots, there seem to be two clusters. If we were asked to cluster these manually, this would probably be our guess:



If we had only this data to cluster by:



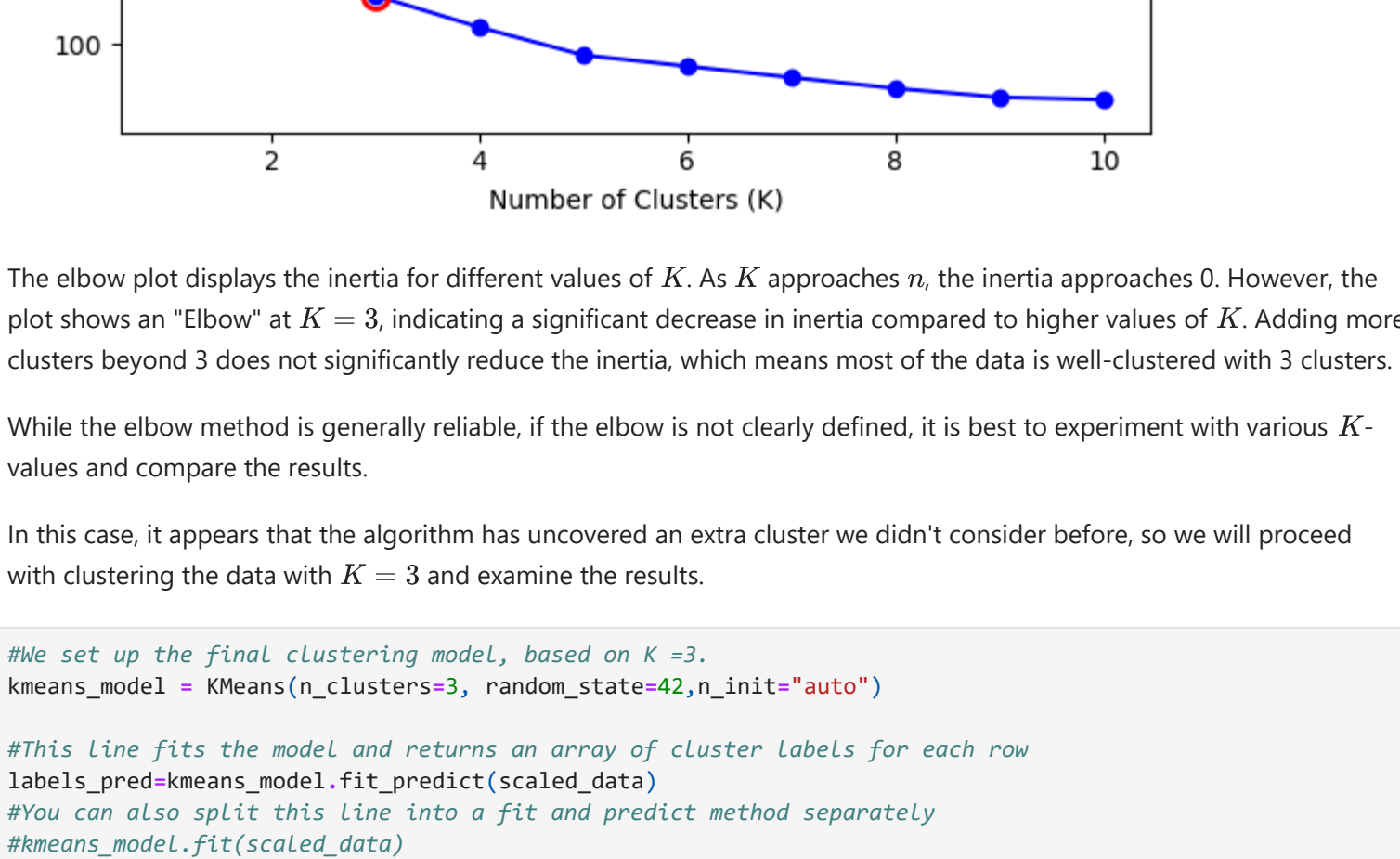
We would not be able to confidently cluster these. So, we need a systematic and mathematical way to group or cluster data. K-means clustering is a simple algorithm that clusters data.

K-Means Clustering

K-means is a popular clustering algorithm that efficiently partitions a dataset into distinct, non-overlapping groups based on similarity, revealing underlying patterns and structures within the data. Let's assume there are two clusters on our observations above, so $K = 2$.

Here are the five steps of the K-means algorithm:

- Choose K random points (or rows) in the dataset and classify each as a "cluster."
- For each data point:
 - Calculate the distance between the point and the various "cluster" points.
 - Assign the point to the closest "cluster" point based on its distance from each "cluster" point.
 - Calculate the center of each cluster by taking the mean value of the data points in the cluster. These centers are called centroids and are represented by crosses.
- Go back to step 2, calculate the new distances to the clusters, assign the closest points to each cluster, and calculate new centroids.
- Once the new centroid and the previous centroid do not differ, the model is converged.



Keep the following points in mind when using K-means clustering:

Critical:

- K-means assumes equal-sized, dense clusters; performance may vary otherwise.
- Outliers can significantly impact K-means; consider them beforehand.
- The algorithm converges to local minimums; try multiple initializations to mitigate this.
- How do we determine the appropriate number of K -values to use?
- K-Means Clustering is sensitive to scale. Ensure your features are on the same scale!

Example:

- Compare K-means with other clustering algorithms for better insights.
- K-means uses Euclidean distances; consider alternatives or transformations if needed.
- K-means only works with numerical data. For categorical data, consider K-Modes clustering, and for a mixture of numerical and categorical data, explore the K-Prototype method.
- The choice of initial points can affect the results.
- The `n_init` parameter, which repeats the clustering with multiple starting points, is now the default setting. Refer to the [Scikit-learn documentation](#) for more information.

Up to you:

- Interpret results with context and domain knowledge to understand underlying structures.
- K-Means works well with global data, like the dataset we're working with, but is not suitable for non-global data.

Coding K-Means clustering

Most of the time, you won't know beforehand how many clusters you have, so it's best to check a range of them. Usually, $1 \leq K \leq 10$, but it depends on the data. There are two methods to determine how many K-values to use.

The Elbow method calculates a term called the inertia, which is the sum of all the squared distances (SSD) in the different clusters. Imagine if $K=1$, there will be points far away from the centroid that will have large squared distances (and a large inertia). By adding another centroid, those points can be closer to a second centroid, reducing the inertia. If we choose $K=n$ where n is the number of data points, each point will be its own centroid, so the inertia will be 0. However, clustering becomes pointless. The idea is to find a value of K for which most of the data is clustered.

We will use an elbow plot soon, where the inertia is plotted against the values of K . The goal is to reduce the inertia as much as possible with a low K -value.

The silhouette metric describes the overlap between clusters and ranges between -1 and 1. See the [silhouette_score](#) for more information.

Let's put this all into practice.

```
In [3]: #First, we drop the species column, since this is actually our target column.
cluster_data = data.drop('species',axis=1)

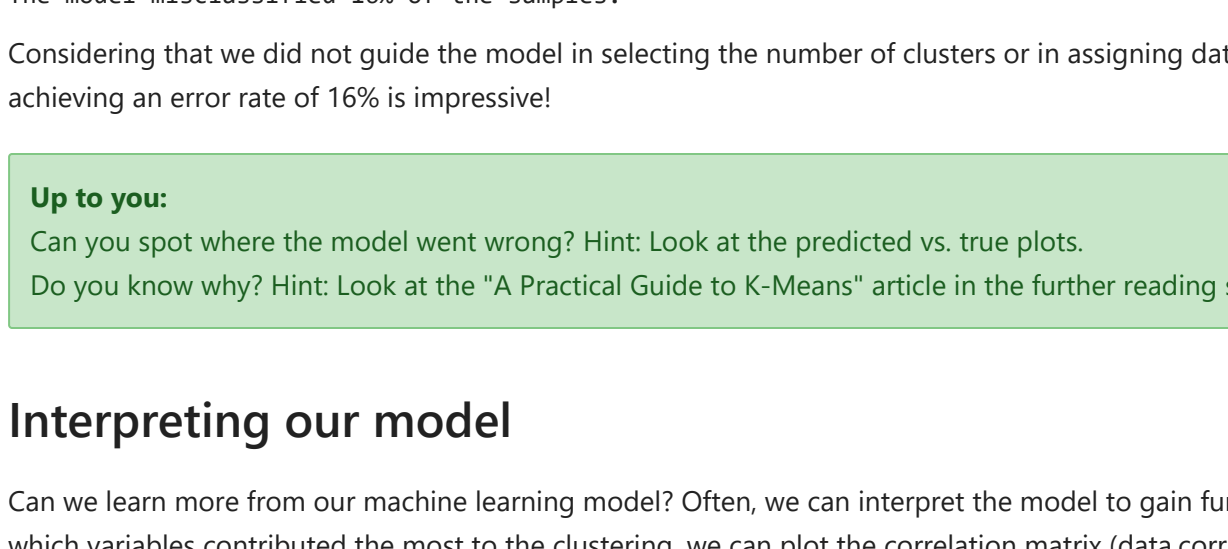
#Scale the data. In the IRIS data set, the petal lengths and widths are of the same order, so it shouldn't be
scaler = StandardScaler()
scaled_data = scaler.fit_transform(cluster_data)
```

```
In [4]: # Elbow method

#Storing the inertia values in a ??
inertia_values = []
K_range = range(1, 11)

#Set up a for loop that clusters the data with K between 1 and 10
for k in K_range:
    Random state makes sure that the same cluster is assigned to the same rows each time, and n_init="auto"
    kmeans = KMeans(n_clusters=k, random_state=42,n_init="auto")
    kmeans.fit(scaled_data)
    inertia_values.append(kmeans.inertia_)

#Plotting the elbow curve
plt.plot(K_range, inertia_values, 'bo')
axes[0, 0].set_title('Sepal Length vs Petal Length')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.tight_layout()
plt.show()
```



The elbow plot displays the inertia for different values of K . As K approaches n , the inertia approaches 0. However, the plot shows an "Elbow" at $K = 3$, indicating a significant decrease in inertia compared to higher values of K . Adding more clusters beyond 3 does not significantly reduce the inertia, which means most of the data is well-clustered with 3 clusters.

While the elbow method is generally reliable, if the elbow is not clearly defined, it is best to experiment with various K -values and compare the results.

In this case, it appears that the algorithm has uncovered an extra cluster we didn't consider before, so we will proceed with clustering the data with $K = 3$ and examine the results.

```
In [5]: #We set up the final clustering model, based on K =3.
kmeans_model = KMeans(n_clusters=3, random_state=42,n_init="auto")

#This Line fits the model and returns an array of cluster labels for each row
labels_pred=kmeans_model.fit_predict(scaled_data)
#You can also split this line into a fit and predict method separately
#kmeans_model.fit(scaled_data)
#labels_pred = kmeans_model.labels_

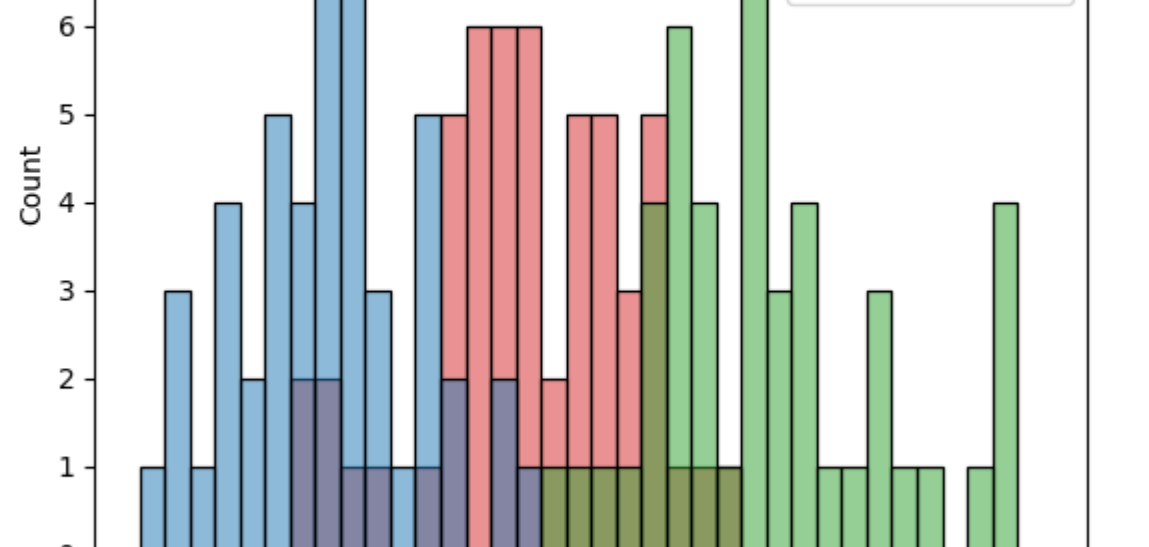
#Adding the labels back to the dataframe
data['cluster'] = labels_pred
data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	cluster
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1

```
In [6]: #Plotting to see the clustering results
fig, axes = plt.subplots(2, 2, figsize=(7, 7))

# Sepal Length vs Petal Length
sns.scatterplot(x='sepal_length', y='petal_length', hue='cluster', data=data, ax=axes[0, 0], palette='Set1')
axes[0, 0].set_title('Sepal Length vs Petal Length')
# Sepal Length vs Petal Width
sns.scatterplot(x='sepal_length', y='petal_width', hue='cluster', data=data, ax=axes[0, 1], palette='Set1')
axes[0, 1].set_title('Sepal Length vs Petal Width')
# Sepal Width vs Petal Length
sns.scatterplot(x='sepal_width', y='petal_length', hue='cluster', data=data, ax=axes[1, 0], palette='Set1')
axes[1, 0].set_title('Sepal Width vs Petal Length')
# Sepal Width vs Petal Width
sns.scatterplot(x='sepal_width', y='petal_width', hue='cluster', data=data, ax=axes[1, 1], palette='Set1')
axes[1, 1].set_title('Sepal Width vs Petal Width')

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```

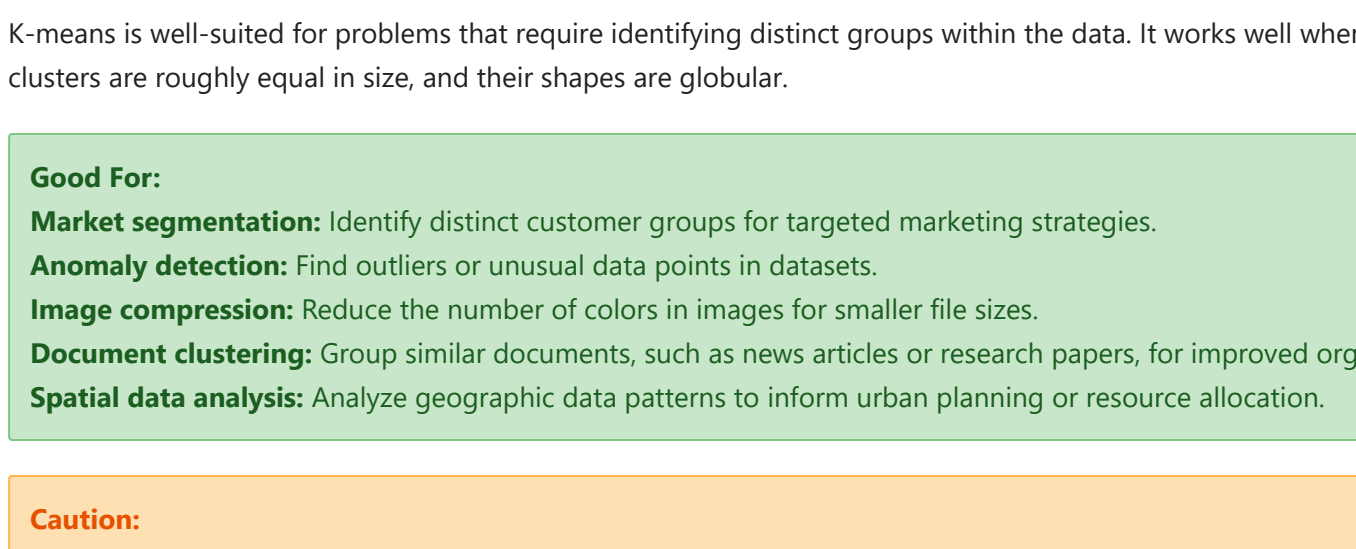


Our analysis revealed that there is one well-separated cluster and two similar clusters that we previously thought were one. To verify if the clustering worked, we will plot the same graphs but color them by 'species'.

```
In [13]: #Plotting to see the true data
fig, axes = plt.subplots(2, 2, figsize=(7, 7))
colors = ['Tab:blue', 'Tab:green', 'Tab:red']

# Sepal Length vs Petal Length
sns.scatterplot(x='sepal_length', y='petal_length', hue='species', data=data, ax=axes[0, 0], palette=colors)
axes[0, 0].set_title('Sepal Length vs Petal Length')
# Sepal Length vs Petal Width
sns.scatterplot(x='sepal_length', y='petal_width', hue='species', data=data, ax=axes[0, 1], palette=colors)
axes[0, 1].set_title('Sepal Length vs Petal Width')
# Sepal Width vs Petal Length
sns.scatterplot(x='sepal_width', y='petal_length', hue='species', data=data, ax=axes[1, 0], palette=colors)
axes[1, 0].set_title('Sepal Width vs Petal Length')
# Sepal Width vs Petal Width
sns.scatterplot(x='sepal_width', y='petal_width', hue='species', data=data, ax=axes[1, 1], palette=colors)
axes[1, 1].set_title('Sepal Width vs Petal Width')

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



From the graph, we can see that:

- Iris-setosa = 1
- Iris-versicolour = 2
- Iris-virginica = 0

However, upon closer inspection, the model does not seem to separate the larger (red/green) cluster very well. Therefore, we need to evaluate the model and determine how accurate it is in clustering.

Evaluating our K-Means model

How successful were we in predicting the species of flowers based on the 4 measurements we were given? Let's check to see how many predictions we got wrong.

```
In [8]: #This dictionary maps the value of the clusters to the species, based on the graphs above
pred_species = {'1':'Iris-setosa',
                '2':'Iris-versicolour',
                '0':'Iris-virginica'}

#Creating new column that predict the species based on the cluster it was assigned
data['pred_species'] = data['cluster'].map(pred_species)
#Checking where the predictions were True or False.
data['match'] = data['species'] == data['pred_species']
#Calculating the percentage, and printing
#is_class_perc= int(len(data[data['match'] == True]) / len(data)) * 100
print(f'The model misclassified {100 - is_class_perc}% of the samples.')

The model misclassified 16% of the samples.
```

Considering that we did not guide the model in selecting the number of clusters or in assigning data to those clusters, achieving an error rate of 16% is impressive!

Up to you:

- Can you spot where the model went wrong? Hint: Look at the predicted vs. true plots.
- Do you know why? Hint: Look at the "A Practical Guide to K-Means" article in the further reading section.

Interpreting our model

Can we learn more from our machine learning model? Often, we can interpret the model to gain further insights. To see which variables contributed the most to the clustering matrix (data.com(numeric,only=True) ['cluster']) and identify the variables that have the greatest correlation with it.

```
In [20]: #Don't edit this
plt.figure(figsize=(2,2),dpi=100)
#Plot the Pearson correlation of the variables with respect to ['cluster']
corr = data.corr(numeric_only=True)['cluster']
#Removing the labels 'cluster' and 'match', sorting and plotting
corr.iloc[: -2].sort_values().plot(kind='bar')
```

Out [20]:

The correlations between the features and the clusters are negative, and of medium/small size. This means that the model used information from all four features to cluster the data correctly. However, with more features, it is likely that some features would be more important than others. This is demonstrated in the same type of plot for another data set:



It is clear that variables like `previous` and `cellular` have a greater effect on the clustering compared to `loan_yes` / `loan_no`.

For our model, let's look at the most important variable, `sepal_length`:

```
In [10]: sns.histplot(x='sepal_length', hue='pred_species', data=data, palette=colors, binwidth=0.1)
```

Out [10]:

This plot clearly shows that the sepal length is a good predictor for separating the three clusters. Flowers with a sepal length less than 5 are likely to be **Iris-setosa**, those with a sepal length between 5 and 6 are **Iris-versicolour**, and those with a sepal length greater than 6 are likely to be **Iris-virginica**.

Try:

- To verify this insight modify the plot to color each data point based on its corresponding 'species' value.

Comments:

Summary:

In this notebook, we covered K-means clustering, a popular unsupervised learning technique used to partition data points into K distinct clusters based on their features. The goal of K-means is to minimize the sum of squared distances between data points and their respective cluster centroids, known as inertia.

Steps covered in the notebook include:

- Loading the Iris dataset and scaling the features using `StandardScaler`.
- Applying K-means clustering with a range of K -values, and choosing the correct K -value using the elbow method.
- Assigning clusters to species of flowers
- Evaluating our model
- Interpreted our model

K-means is well-suited for problems that require identifying distinct groups within the data. It works well when the clusters are roughly equal in size, and their shapes are globular.

Good For:

- **Market segmentation:** Identify distinct customer groups for targeted marketing strategies.
- **Anomaly detection:** Find outliers or unusual data points in datasets.
- **Image compression:** Reduce the number of colors in images for smaller file sizes.
- **Document clustering:** Group similar documents, such as news articles or research papers, for improved organization.
- **Spatial data analysis:** Analyze geographic data patterns to inform urban planning or resource allocation.

Caution:

- **Initial centroid placement:** K-means is sensitive to the initial placement of centroids. This can be mitigated by using techniques like K-means++ for initializing centroids or running the algorithm multiple times with different initializations and choosing the best result.
- **Cluster assumptions:** K-means assumes that clusters are globular, meaning it may not perform well when dealing with non-convex or elongated clusters.

Number of clusters: The algorithm requires the number of clusters (K) to be specified in advance. Choosing the optimal K is often not straightforward and may require the use of heuristics like the elbow method or silhouette analysis.

Convergence: K-means is not guaranteed to converge to a global optimum and may sometimes converge to a local minimum. This is another reason to run the algorithm multiple times with different initializations.

Scaling sensitivity: K-means is sensitive to the scaling of the input data. It is essential to scale the data appropriately before applying the algorithm.

Further Reading

- Scikit-learn: KMeans:** This document explains all the model parameters in the KMeans module.
- A Practical Guide to K-Means:** Read this short article for a different view on K-means in this article.
- Machine Learning with Python: K Means Clustering:** Discover more about K-Means in this article.

Exersize

Replace `<insert>` codes with your code to cluster penguins using the K-Means algorithm. Follow the steps below:

- If you haven't already, run the cell at the top to import the libraries.
- Run the cell below to get the dataset. We have removed some data to make it simpler.
- Scale the data.
- Determine the best K -value to use.
- Calculate the model for the correct K -value.
- Use the given code below to plot the outcome.

```
In [11]: #Don't edit this point
data=pd.read_csv('https://storage.googleapis.com/kaggledatasets/703056/1228604/penguins_size.csv')
#why did this work? data isn't .csv? See if you can find out.
Peng_data = Peng_data.drop(['island', 'sex', 'body_mass_g'],axis=1)
Peng_data.head()
```

	species	culmen_length_mm	culmen_depth_mm	flipper_length_mm
1	Adelie	39.5	17.4	186.0
2	Adelie	40.3	18.0	195.0
4	Adelie	36.7	19.3	193.0
6	Adelie	38.9	17.8	181.0
12	Adelie	41.1	17.6	182.0

Scale the data

```
In [ ]: #First, we drop the species column, since this is actually our target column.
<insert_code> = Peng_data.drop('species',axis=1)
```

```
<insert_code> = StandardScaler()
<insert_code> = scaler.fit_transform(<insert_code>)
```

Determine best K-value

```
In [ ]: # Elbow method
#What data do we use to model?
input_data = <insert_code>

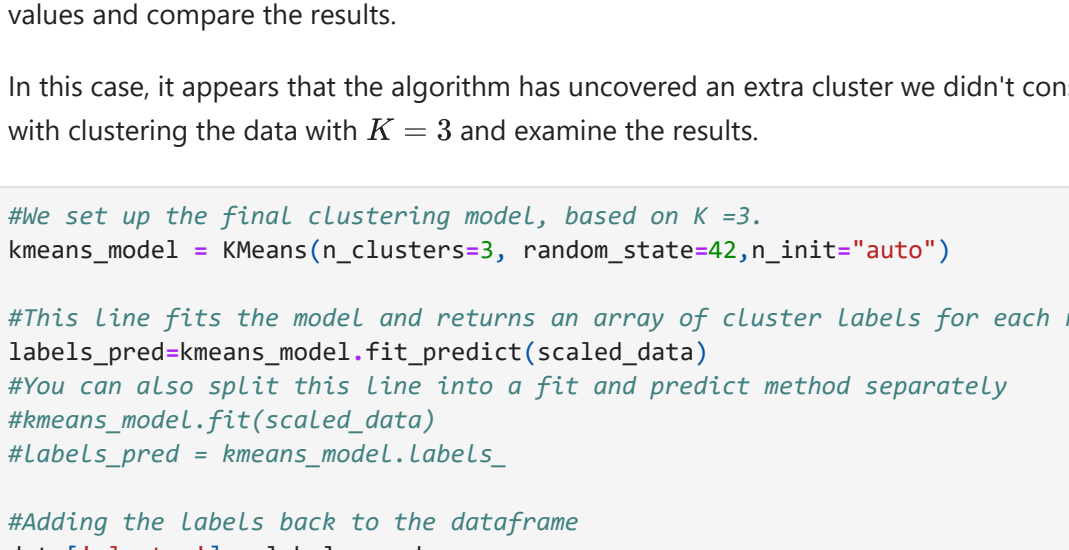
#Don't edit past this point
```

```
#We set up the final clustering model, based on K =3.
kmeans_model = KMeans(n_clusters=k, random_state=42,n_init="auto")
#This Line fits the model and returns an array of cluster labels for each row
Peng_labels_pred=kmeans_model.fit_predict(input_data)
```

```
#Adding the labels back to the dataframe
Peng_data['cluster'] = Peng_labels_pred
Peng_data.head()
```

Plot your results

```
In [10]: sns.histplot(x='sepal_length', hue='pred_species', data=data, palette=colors, binwidth=0.1)
```



This plot clearly shows that the sepal length is a good predictor for separating the three clusters. Flowers with a sepal length less than 5 are likely to be **Iris-setosa**, those with a sepal length between 5 and 6 are **Iris-versicolour**, and those with a sepal length greater than 6 are likely to be **Iris-virginica**.

Try:

- To verify this insight modify the plot to color each data point based on its corresponding 'species' value.


```
In [ ]: #Enter the strng of the column name
cluster_col_name = <insert_code>
species_col_name = <insert_code>

#Dont edit past this point
# Plotting to see the clustering results
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
colors = ['Tab:blue', 'Tab:green', 'Tab:red']

sns.scatterplot(x='culmen_length_mm', y='flipper_length_mm', hue=cluster_col_name, data=Peng_data, ax=axes[0], axes[0].set_title('Model'))
# Sepal width vs Petal length
sns.scatterplot(x='culmen_length_mm', y='flipper_length_mm', hue=species_col_name, data=Peng_data, ax=axes[1], axes[1].set_title('True'))
```

Comment on what you see: Are the predicted and true species the same?

Evaluate model

Use the plot above to map the cluster number to the species.

```
In [ ]: #This dictionary maps the value of the clusters to the species, based on the graphs above
Peng_pred_species = {0:'Chinstrap',
                     1:'Gentoo',
                     2:'Adelie'}

#creating new column that predict the species based on the cluster it was assigned
Peng_data['pred_species']=Peng_data['cluster'].map(<insert_code>)
#checking where the predictions were True or False.
Peng_data['match'] = Peng_data['species'] == Peng_data['pred_species']
#Calculating the percentage, and printing
mis_class_perc=int(len(Peng_data[Peng_data['match']==False])/len(Peng_data)*100)
print(f'The model misclassified {mis_class_perc}% of the samples.')
```

Comments:

Interpret your model

```
In [ ]: #Get the pearson correlation of the variables with respect to ['cluster']
corr = Peng_data.corr(numeric_only=True)['cluster']

plt.figure(figsize=(2,2),dpi=200)
#removing the labels 'cluster' and 'match', sorting and plotting
corr.iloc[-2:].sort_values().plot(kind='bar')
```

Which variables are most important in identifying a penguin species?

Plot Interpretation

Plot your predictions along the most important variable and try to explain the result.

```
In [ ]: sns.histplot(x=<insert_code>, hue=<insert_code>, data=Peng_data, palette=colors, binwidth=1)

In [ ]: Comments:
```