Clustering on Iris Dataset- Prediction using Unsupervised ML

It is a project based on Iris_dataset, here I will show you visualisation and kMN Clustering, also predict the optimum number of clusters with the help of Elbow Method and represent it visually.

In [2]:
```
#importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
#loading Iris data
iris=sns.load_dataset('iris')
iris
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Let's check the data

To check if there is null or duplicate value.

In [4]:
```
iris.columns
```

Out[4]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [5]:
```
iris.duplicated().sum()
```

```
Out[5]: 1
```

```
In [6]: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

So, there is one duplicate, no null value and 5 columns with dtype Object.

Exploring Data

To Explore the data, we apply certain statistical measures to check the distribution of the data.

```
In [7]: iris.describe()
```

Out[7]:

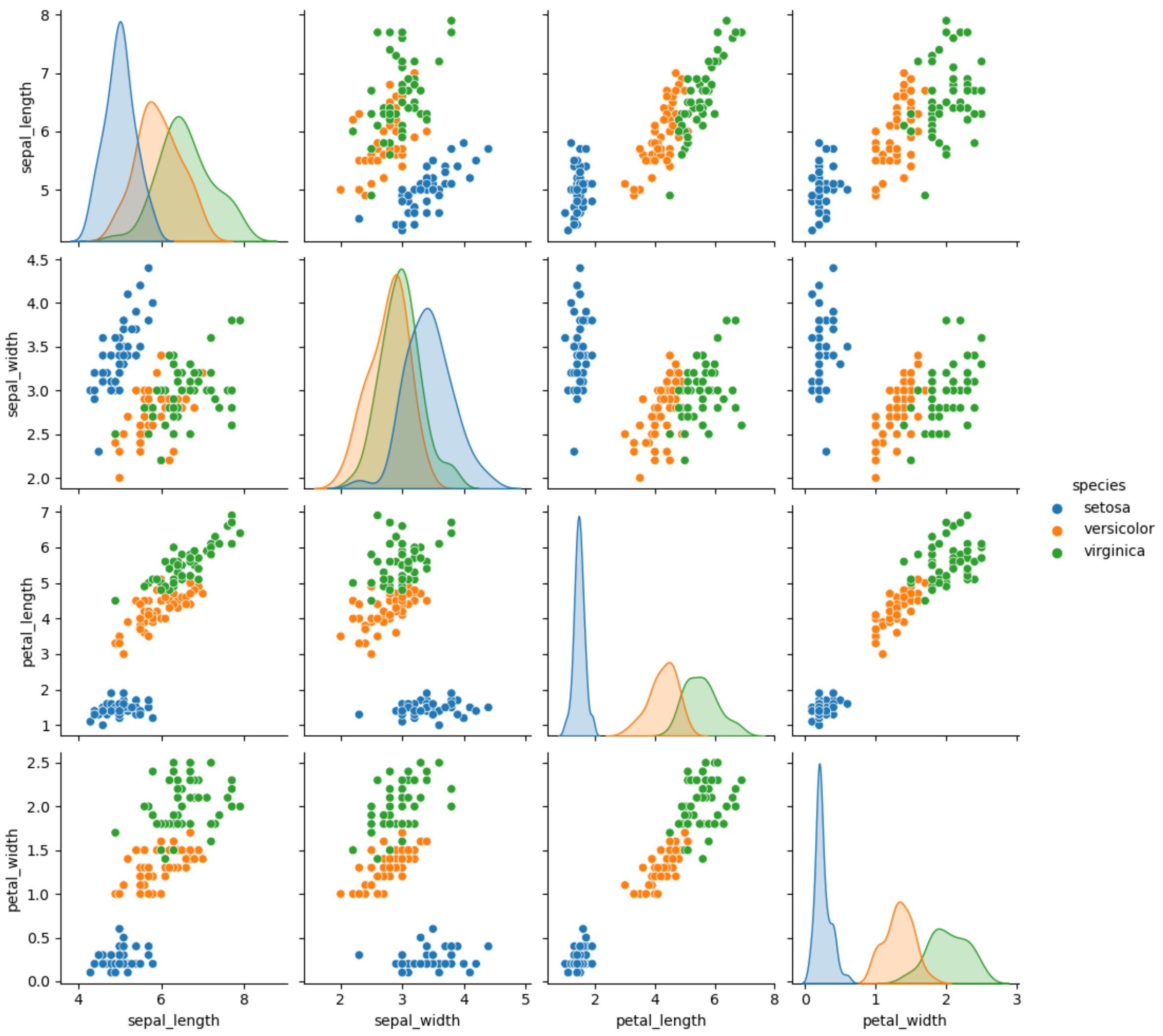
	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Data Visualisation

We will see various plots to get a clear idea of the relationship of the data.

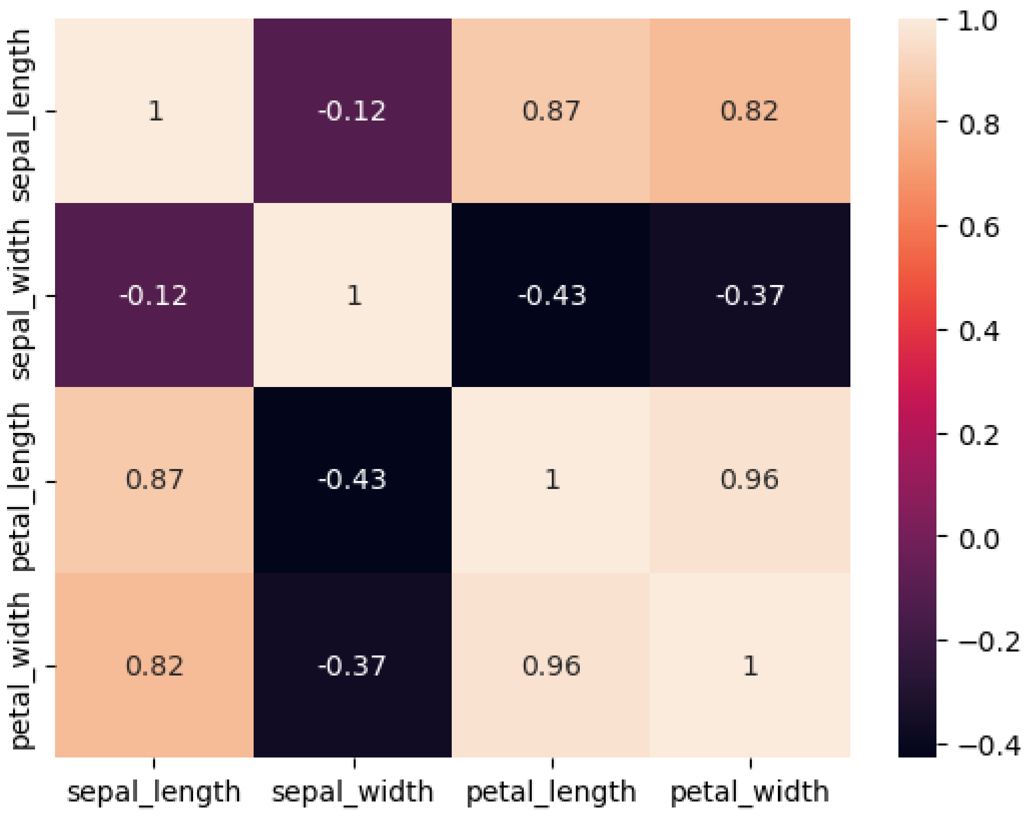
A pair plot in Python is a visualization technique that creates a grid of scatter plots or histograms to showcase the pairwise relationships between multiple variables in a dataset. It helps to identify patterns, correlations, and distributions, making it useful for exploratory data analysis and understanding the interactions between different variables.

```
In [8]: sns.pairplot(iris,hue="species")
        plt.show()
```



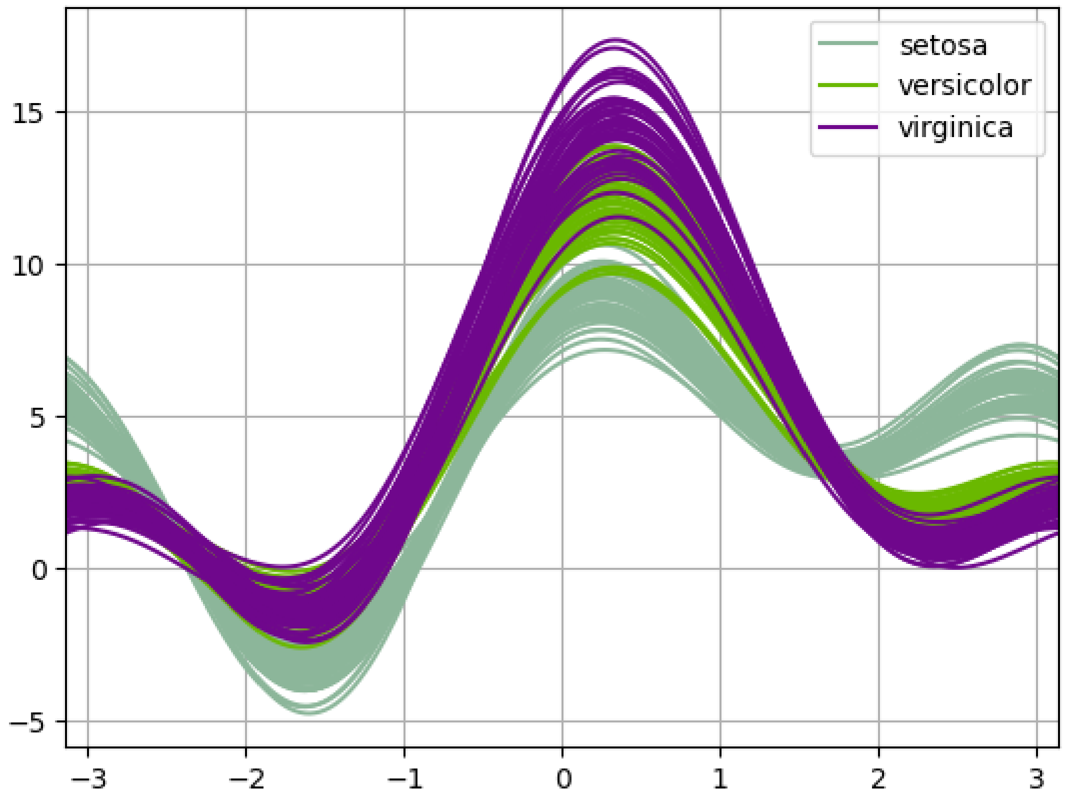
Heatmaps in Python are used to visually represent data using colors. They are particularly useful for displaying the intensity or distribution of values across two dimensions, helping to identify patterns, correlations, or hotspots in the data quickly.

In [9]:
```
annot=True
hm=sns.heatmap(data=iris.corr(), annot=annot)
plt.show()
```



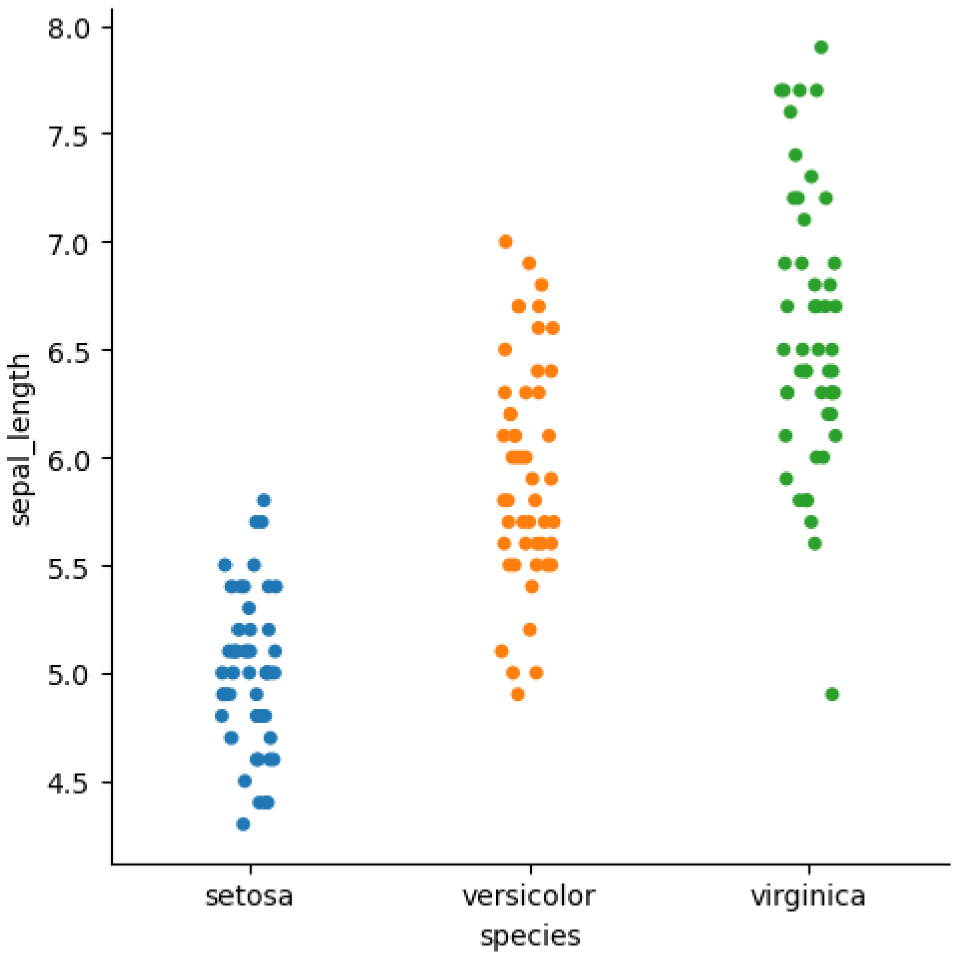
Andrews curves in Python are used to visualize high-dimensional data by mapping each instance to a unique curve. They provide a simplified representation of complex datasets, enabling the identification of patterns and differences between classes or groups of data points.

```
In [10]: p=pd.plotting.andrews_curves(iris,"species")
         p.plot()
         plt.show()
```

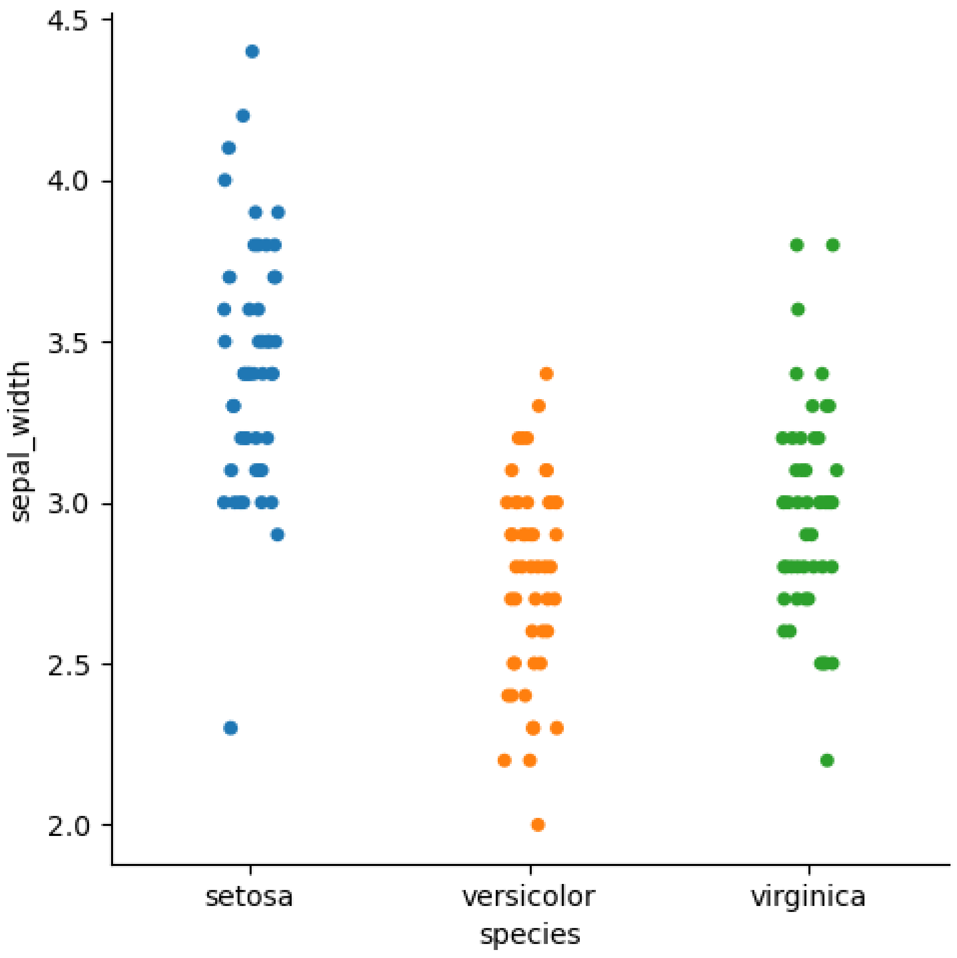


Catplot is used to visualize the relationship between categorical variables by creating various types of plots (e.g., bar plots, point plots) and supports additional parameters like hue and col for further comparisons.

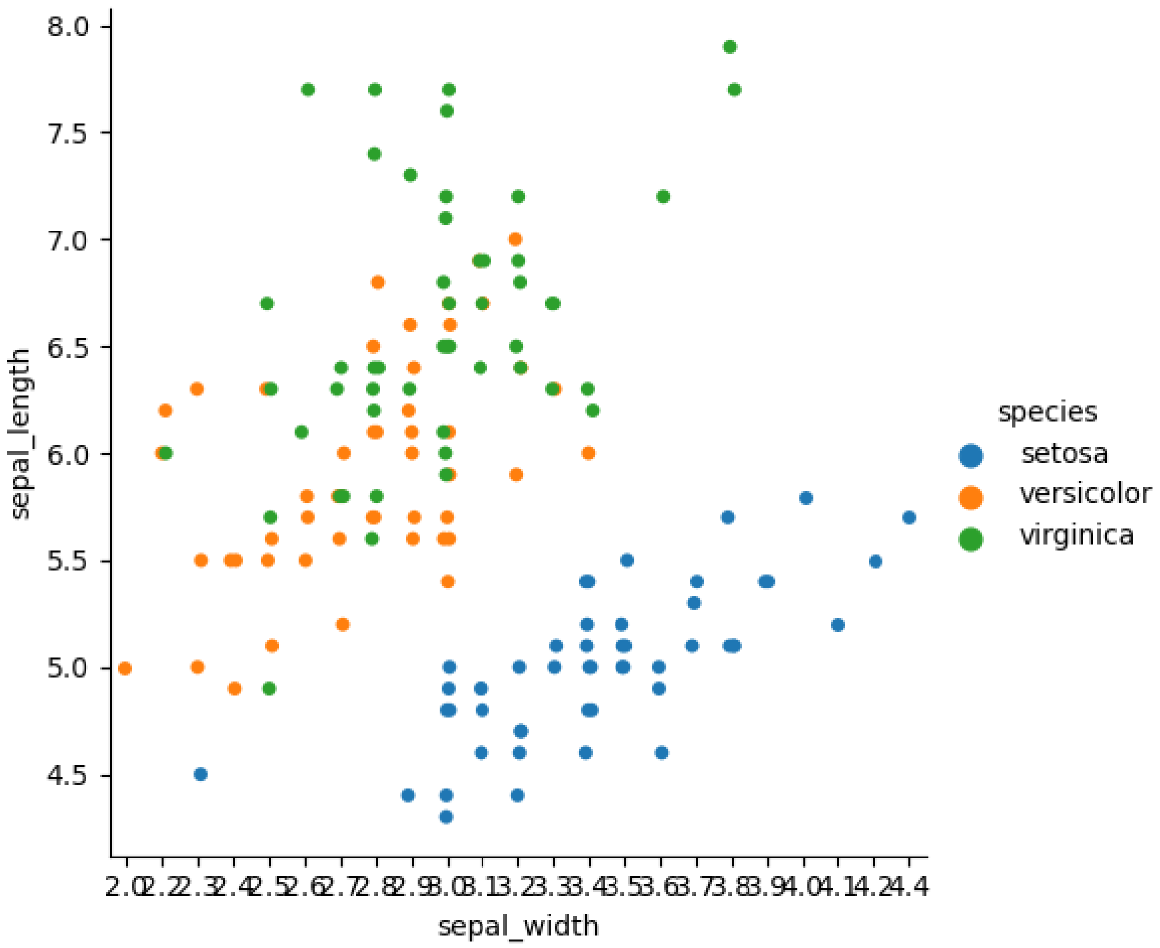
In [11]:
```
sns.catplot('species','sepal_length',data=iris)
plt.show()
```



In [12]:
```
sns.catplot('species','sepal_width',data=iris)
plt.show()
```

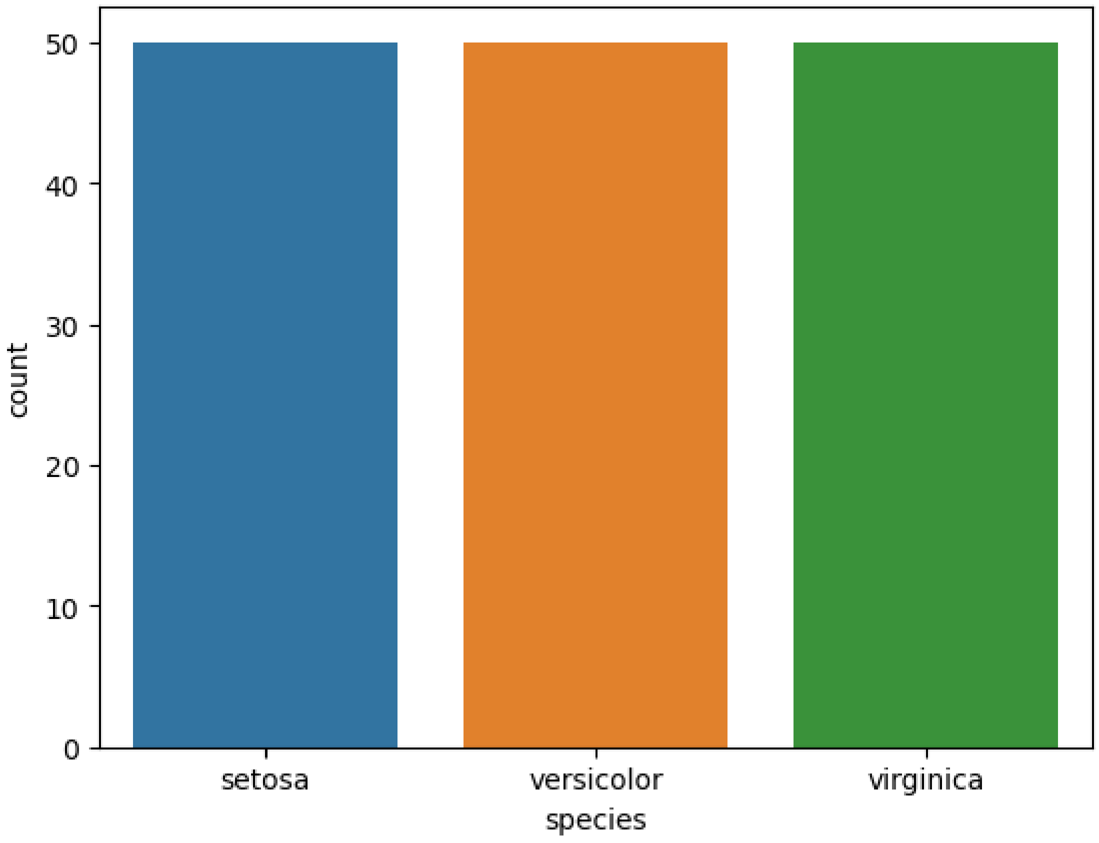


```
In [13]: sns.catplot('sepal_width','sepal_length',data=iris,hue='species')
         plt.show()
```



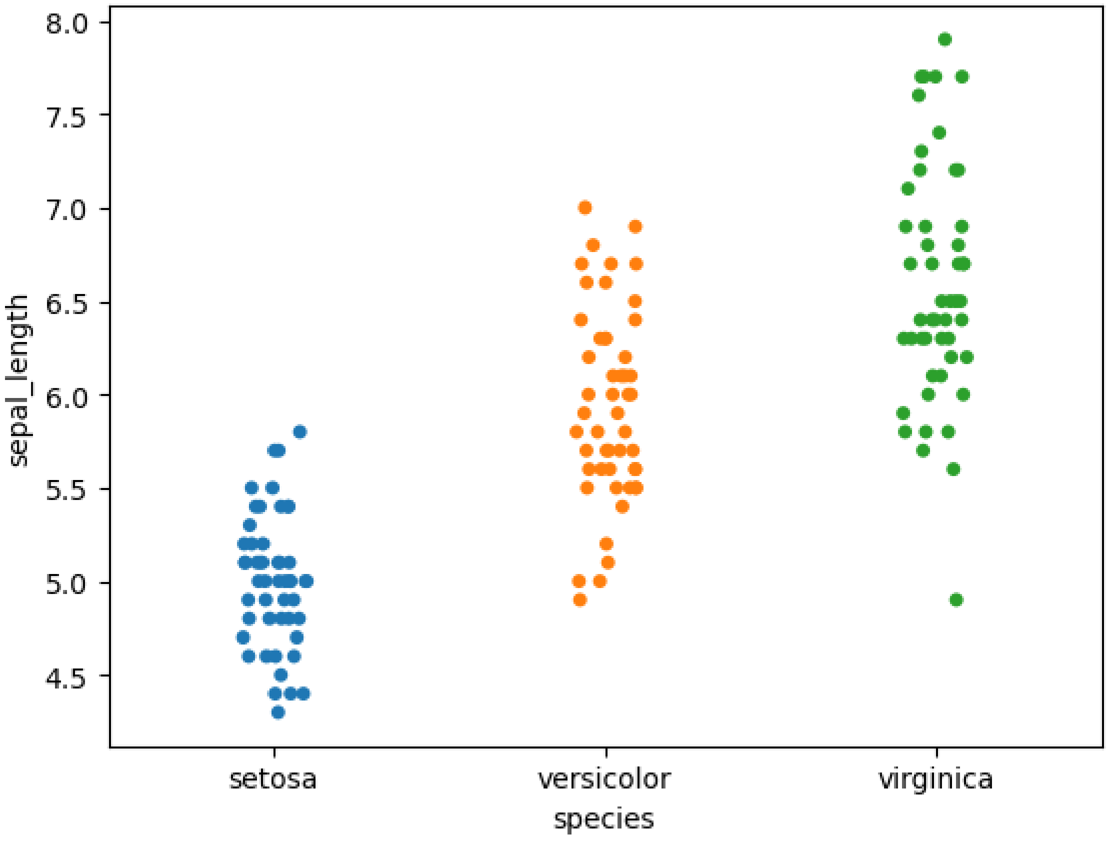
Countplot is used to visualize the count or frequency of categorical variables by creating a bar plot that represents the number of occurrences of each category in the dataset.

```
In [14]: sns.countplot('species',data=iris)
         plt.show()
```



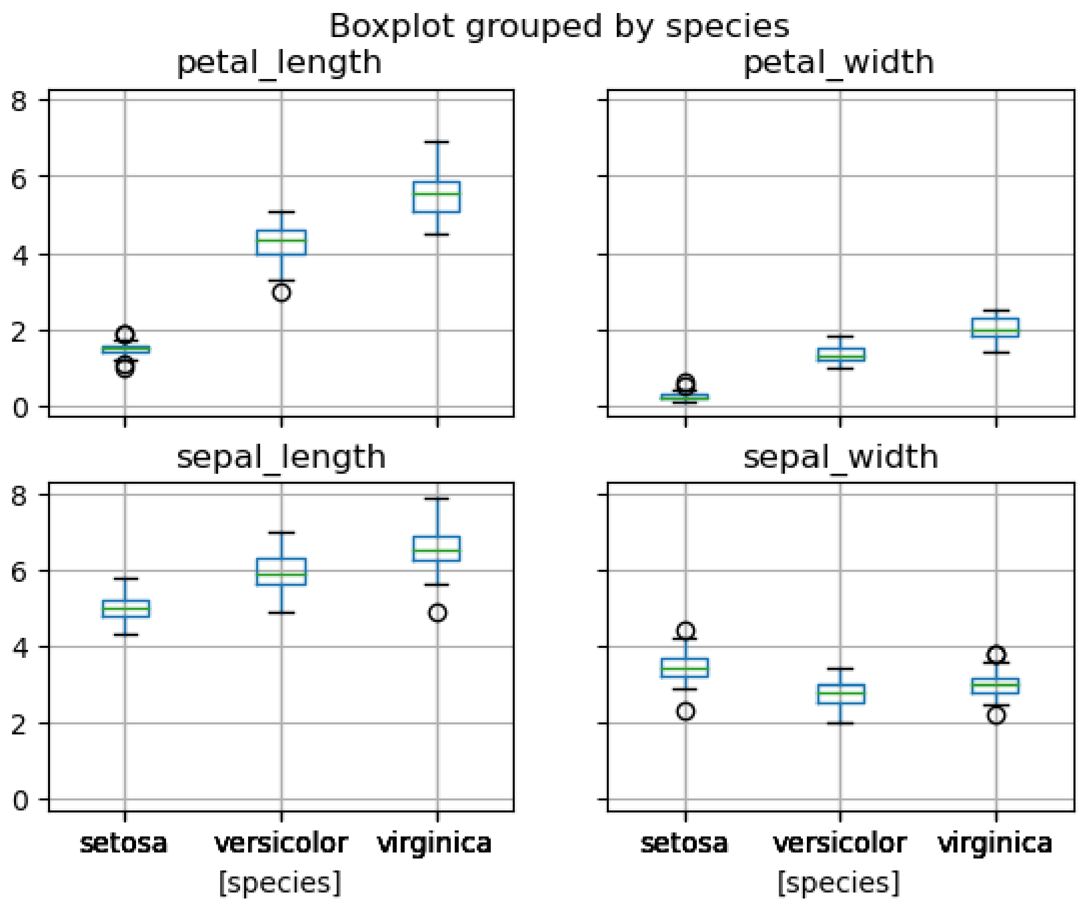
Stripplot is used to display the distribution of categorical data points along a continuous axis, allowing easy comparison of individual data points.

```
In [15]: sns.stripplot(x='species',y='sepal_length',data=iris)
         plt.show()
```



A box-and-whisker plot, is a graphical representation of the distribution of a dataset. It displays the minimum, maximum, median, and quartiles (25th and 75th percentiles) of the data, providing insights into the central tendency, spread, and skewness of the values in a concise manner.

In [16]:
```
iris.boxplot(by='species')
plt.show()
```



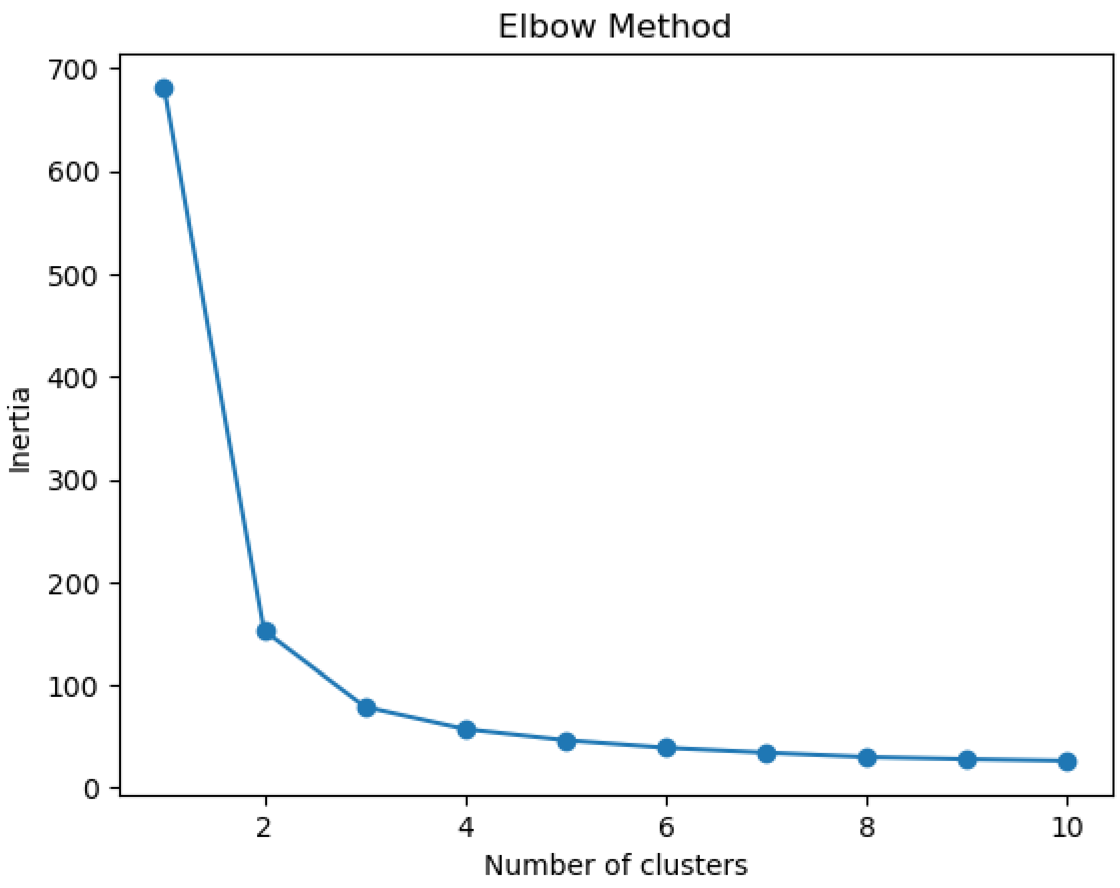
sns.jointplot(x="petal_length",y="petal_width", data=iris, kind='reg') plt.show()

Predicting the optimal number of Clusters using the Elbow method

In [17]:
```
x = iris.iloc[:, [0, 1, 2, 3]].values
```

The elbow method in Python is used to determine the optimal number of clusters in a dataset. It helps identify a point on a plot where the decrease in within-cluster sum of squares (WCSS) slows down significantly, indicating a suitable number of clusters to use in clustering algorithms.

In [18]:
```
inertias=[]
for i in range(1,11):
    kmeans=KMeans(n_clusters=i)
    kmeans.fit(x)
    inertias.append(kmeans.inertia_)
plt.plot(range(1,11),inertias, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



We see that the point of the X-axis where there is a sharp drop in the value of the inertia is (3,0). Hence, we choose k=3.

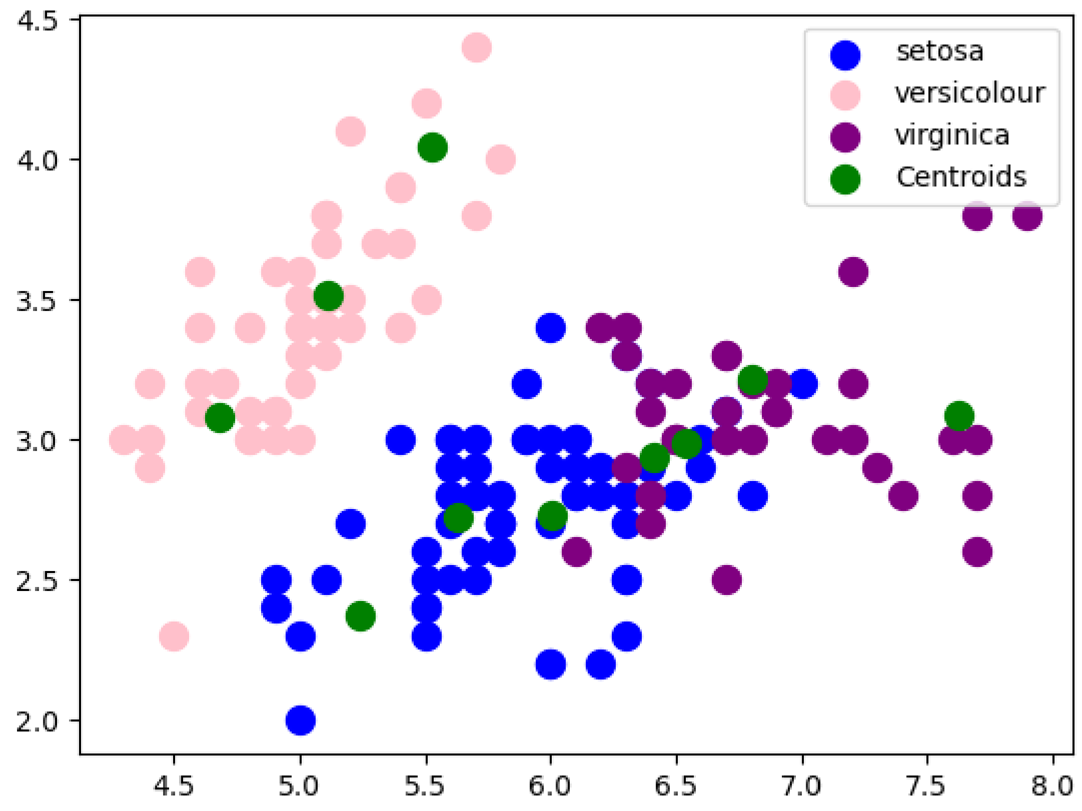
In [19]:
```
k_means=KMeans(n_clusters=3,init="k-means++",max_iter=300,n_init=10,random_state=0
y_kmeans=k_means.fit_predict(x)
```

K-means clustering in Python is used to partition data points into distinct groups based on similarity. It helps identify natural clusters in the data by minimizing the within-cluster sum of squared distances and is commonly used for tasks such as customer segmentation, image compression, or anomaly detection.

In [20]:
```
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
            s = 100, c = 'blue', label = 'setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
            s = 100, c = 'pink', label = 'versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],
            s = 100, c = 'purple', label = 'virginica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],
            s = 100, c = 'green', label = 'Centroids')

plt.legend()
plt.show()
```



By Sneha Bhattacharjee Date:- 22nd March 2023