Prediction using UnSupervised ML

In this task, we are going to predict the optimum number of clusters from the given iris dataset and represent it visually. This includes unsupervised learning.

Importing Libraries needed to perform task

In [2]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
```

Reading the data-set

In [3]:
```
# Loading and Reading the iris dataset
# Data available at the link - 'https://bit.ly/3kXTdox'

data = pd.read_csv('Iris.csv')
print('Data imported successfully')
```

```
Data imported successfully
```

In [4]:
```
data.head(10) # loads the first five rows
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [5]:
```
data.tail(10) # loads the last five rows
```

Out[5]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
140	141	6.7	3.1	5.6	2.4	Iris-virginica
141	142	6.9	3.1	5.1	2.3	Iris-virginica
142	143	5.8	2.7	5.1	1.9	Iris-virginica
143	144	6.8	3.2	5.9	2.3	Iris-virginica
144	145	6.7	3.3	5.7	2.5	Iris-virginica
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [6]:
```
# Checking for NaN values
data.isna().sum()
```

Out[6]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

NaN standing for not a number, is a numeric data type used to represent any value that is undefined or unpresentable. For example, 0/0 is undefined as a real number and is, therefore, represented by NaN. So, in this dataset, we don't have such values.

In [7]:
```
# Checking statistical description
data.describe()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

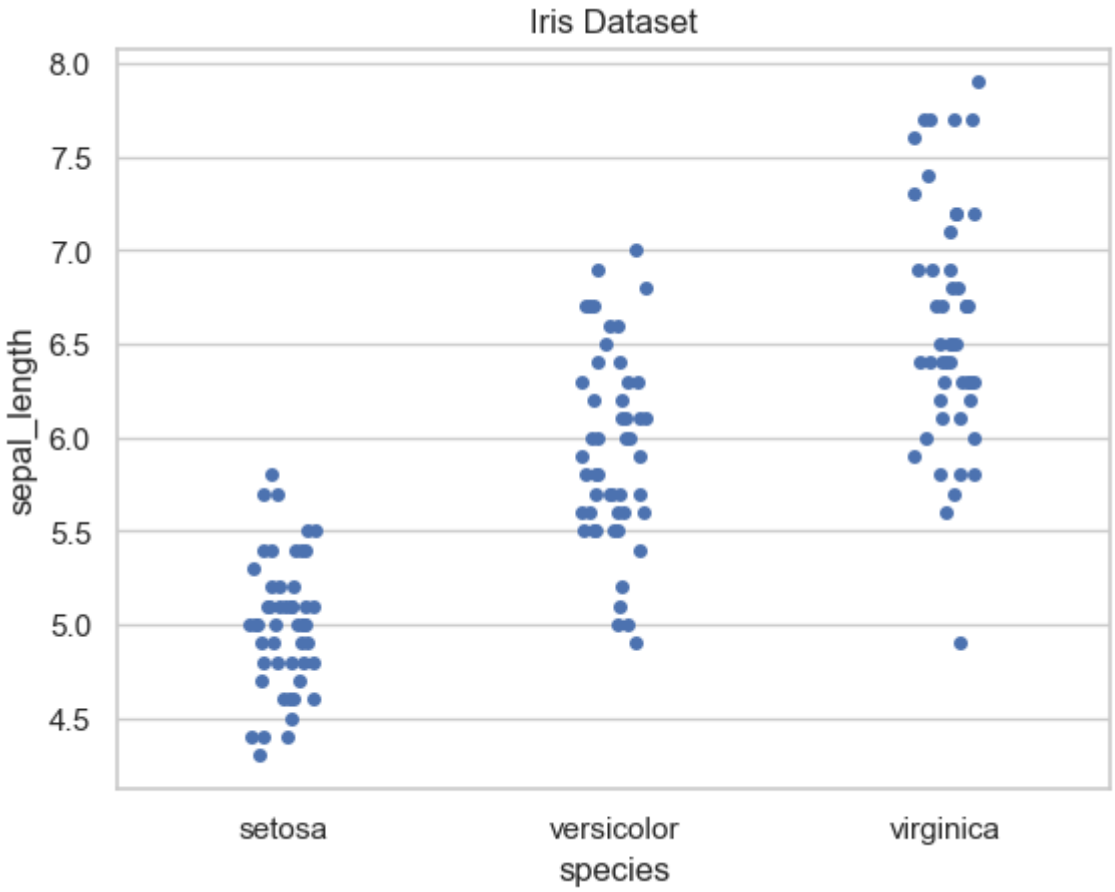
Now, let's check for unique classes in the dataset.

In [8]:
```
print(data.Species.nunique())
print(data.Species.value_counts())
```

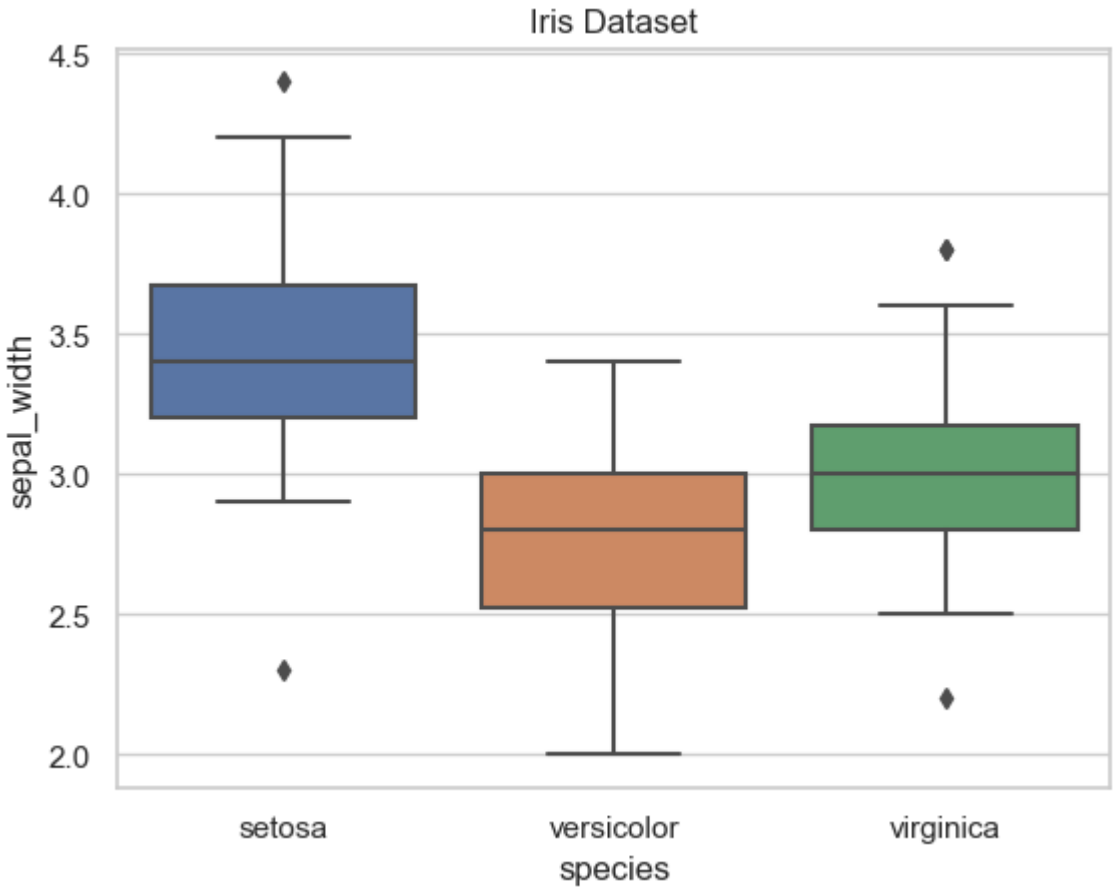
```
3
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

Data Visualization

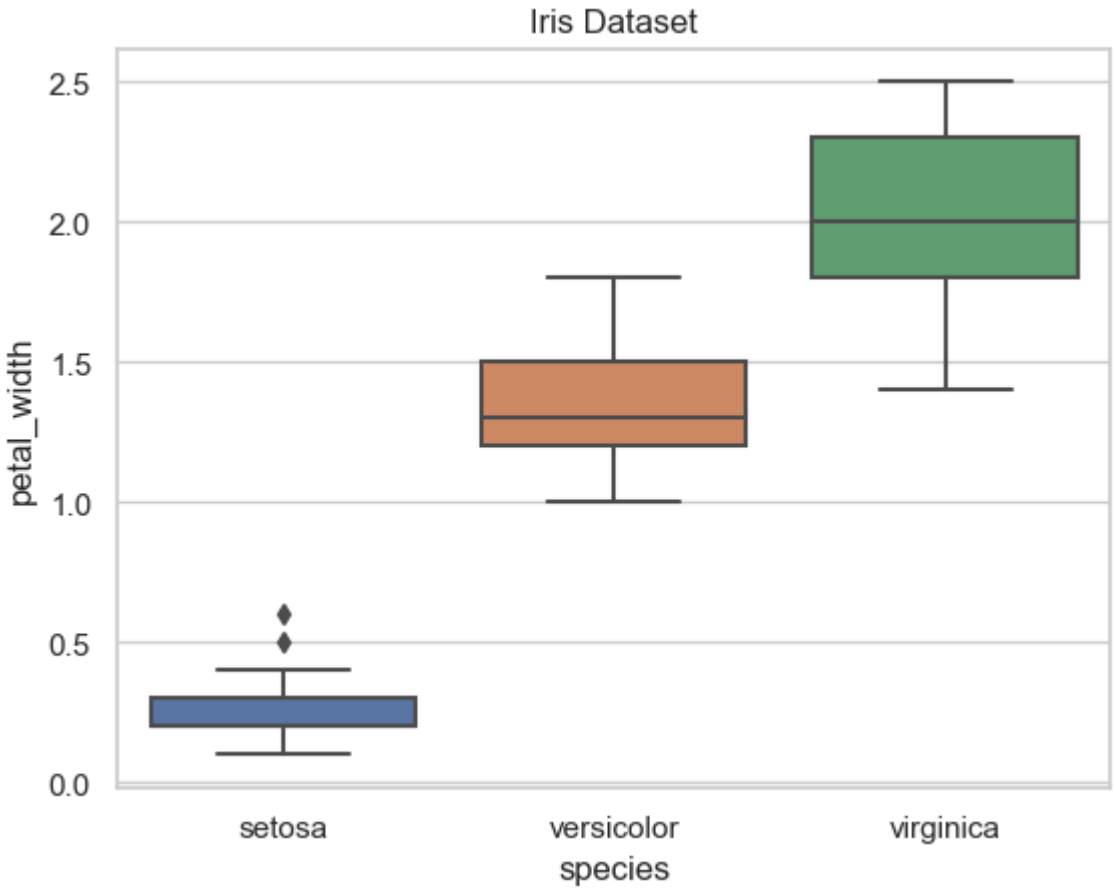
In [9]:
```
sns.set(style = 'whitegrid')
iris = sns.load_dataset('iris');
ax = sns.stripplot(x = 'species', y = 'sepal_length', data = iris);
plt.title('Iris Dataset')
plt.show()
```



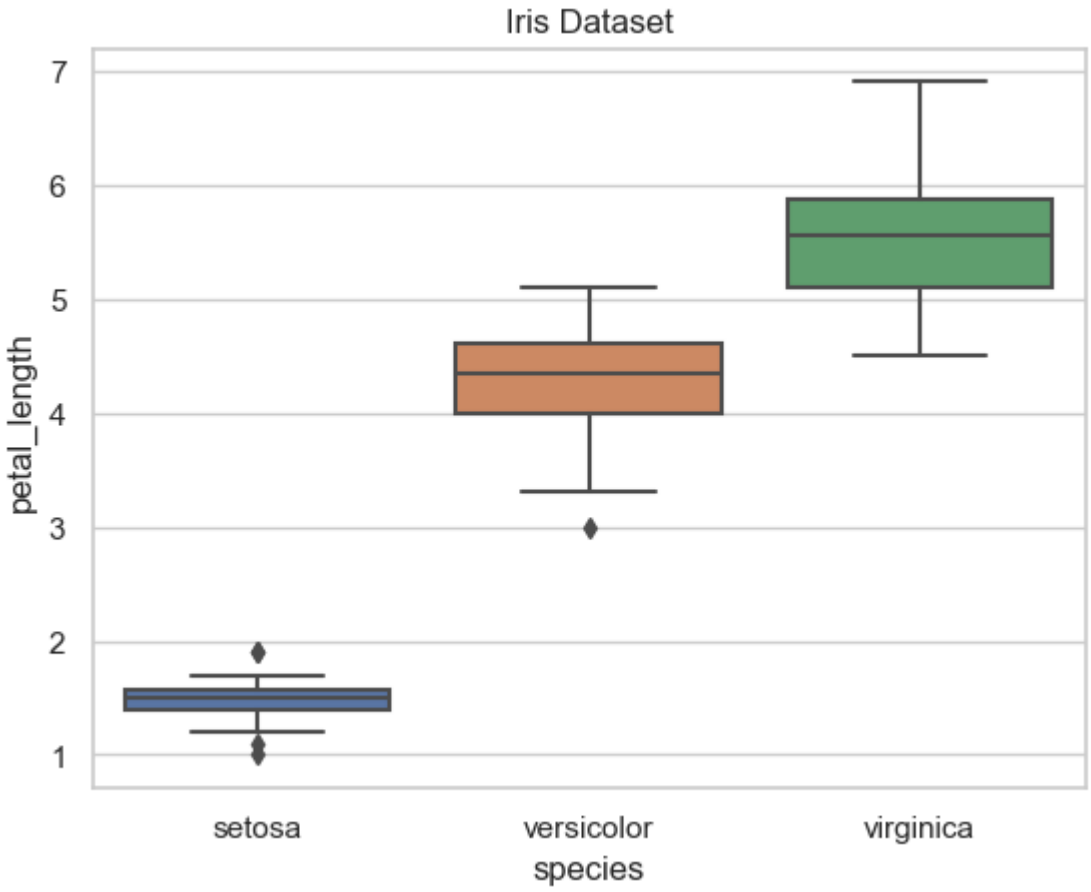
In [10]:
```
sns.boxplot(x = 'species', y = 'sepal_width', data = iris)
plt.title("Iris Dataset")
plt.show()
```



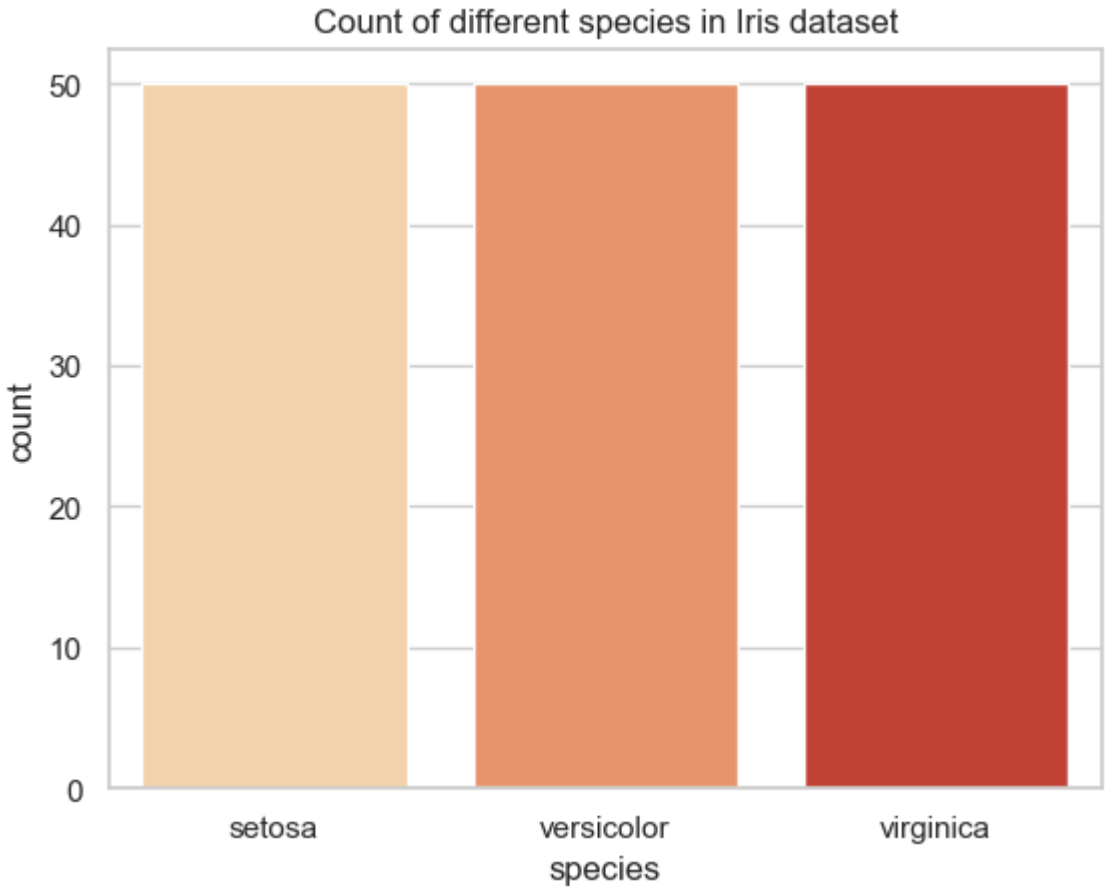
In [11]:
```
sns.boxplot(x = 'species', y = 'petal_width', data = iris)
plt.title("Iris Dataset")
plt.show()
```



In [12]:
```
sns.boxplot(x='species',y='petal_length',data=iris)
plt.title("Iris Dataset")
plt.show()
```

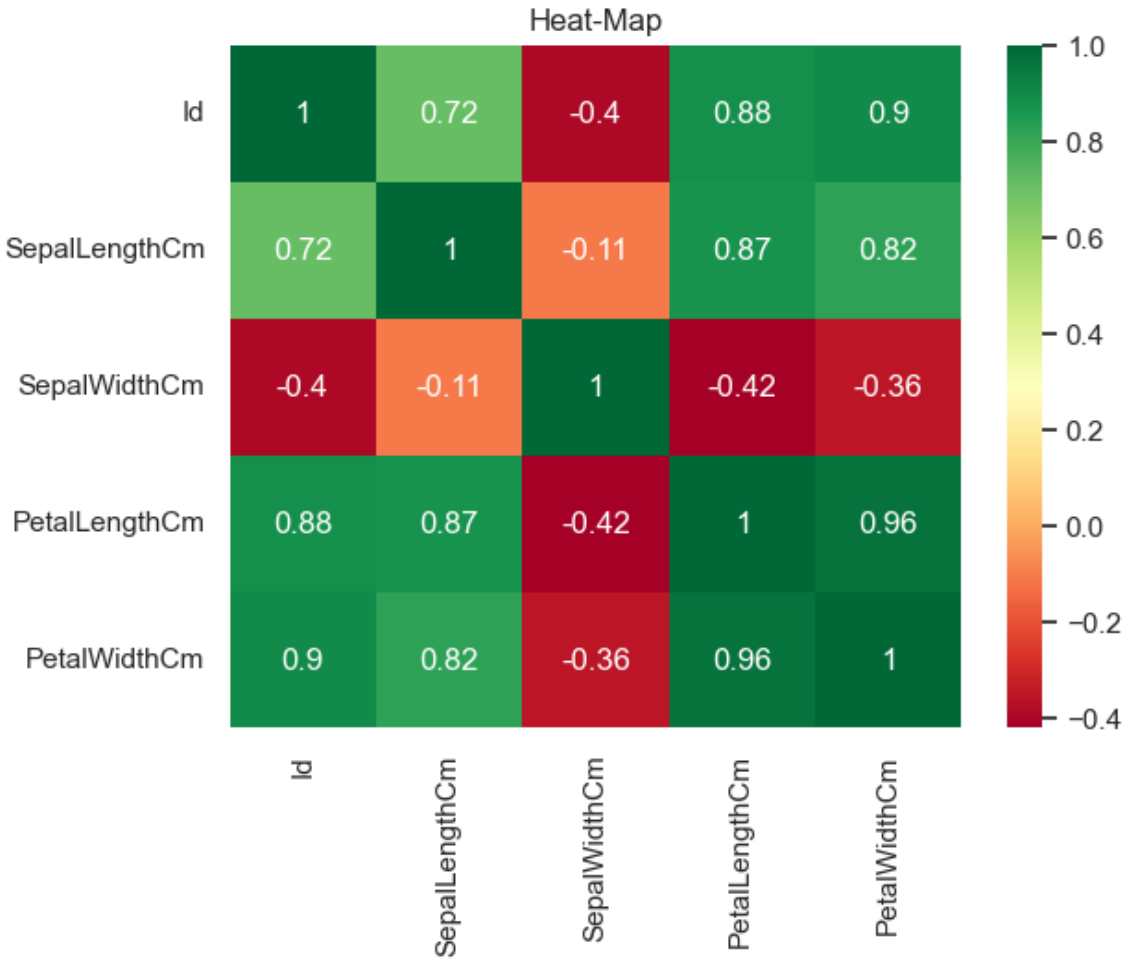


In [13]:
```
# Count plot
sns.countplot(x='species', data=iris, palette="OrRd")
plt.title("Count of different species in Iris dataset")
plt.show()
```



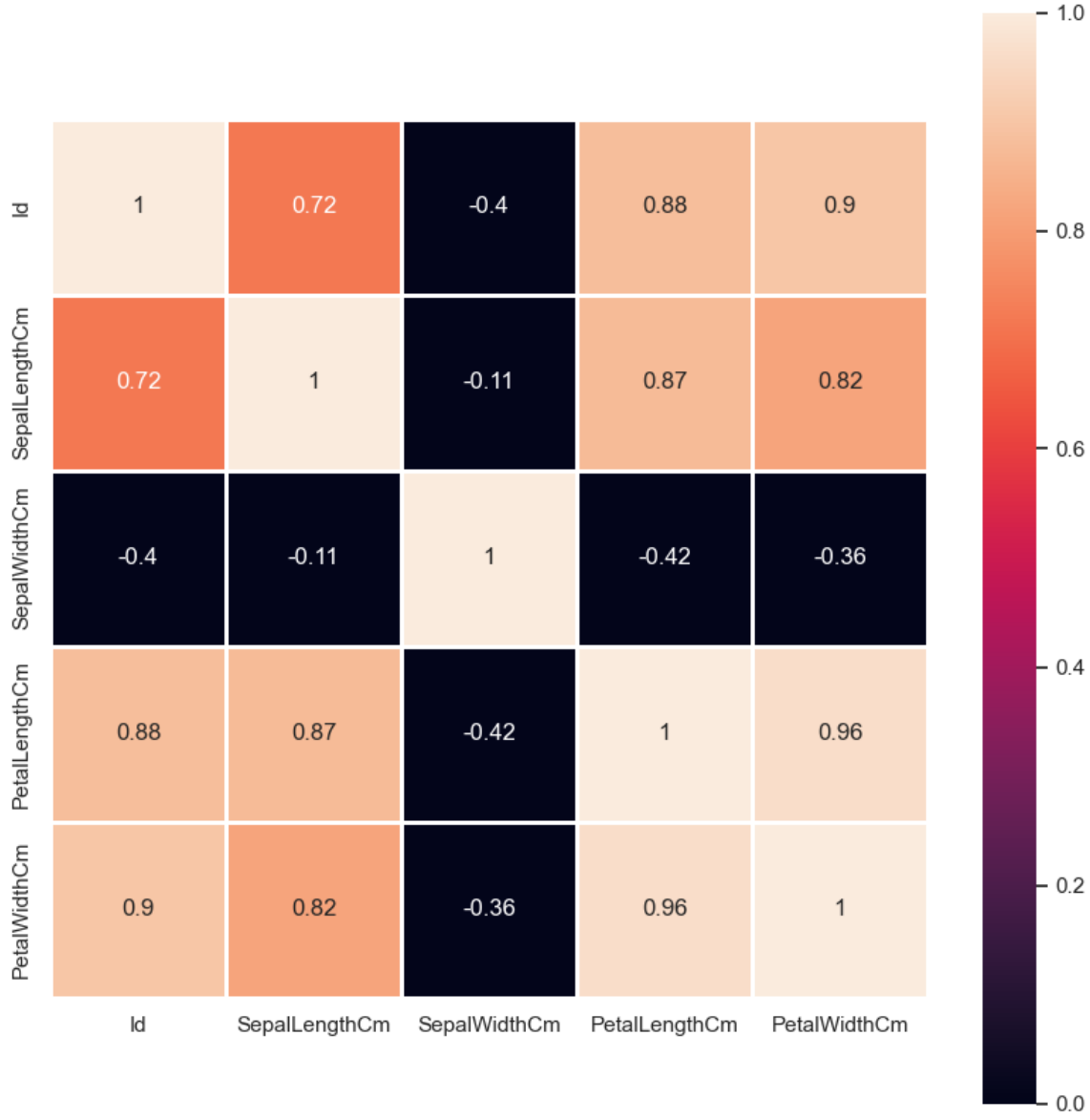
In [14]:
```
#This is needed for the analysis of two variables, for determining the empir
sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')
plt.title("Heat-Map")
plt.show()
```

```
C:\Users\baps\AppData\Local\Temp\ipykernel_13508\1166310983.py:2: FutureWa
rning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(), annot=True, cmap='RdYlGn')
```



In [15]:
```
iris1 = data.corr() #finding correlation between variables of iris dataset
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(iris1, vmin=0, vmax=1, square=True, annot=True, linewidths=1)
plt.show()
```

```
C:\Users\baps\AppData\Local\Temp\ipykernel_13508\1515835543.py:1: FutureWa
rning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  iris1 = data.corr() #finding correlation between variables of iris datas
et
```

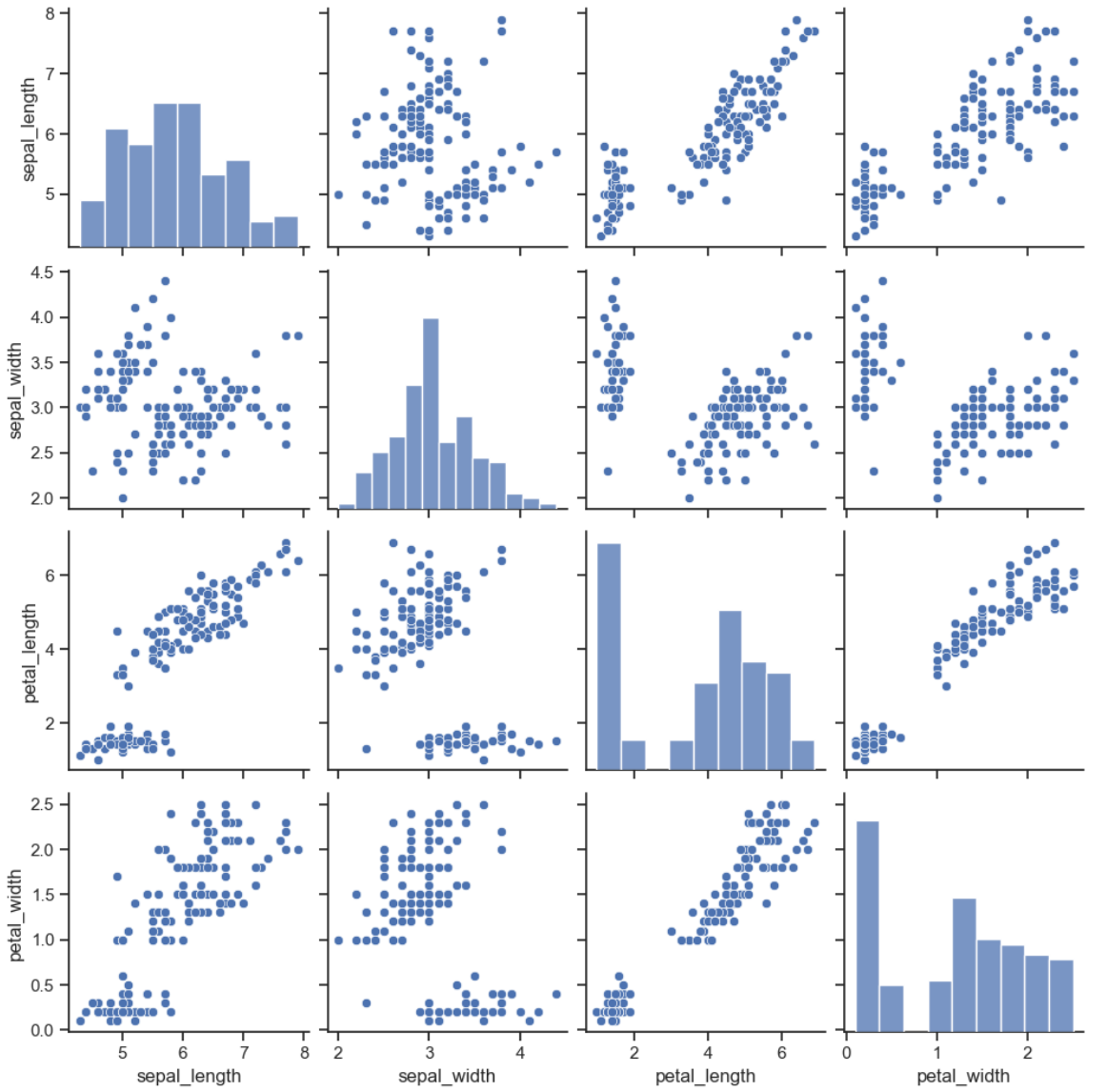


Heatmap is a two-dimensional graphical representation of data where the individual values that are contained in a matrix are represented as colors. Or we can also say that these Heat maps display numeric tabular data where the cells are colored depending upon the contained value.

Heat maps are great for making trends in this kind of data more readily apparent, particularly when the data is ordered and there is clustering.

The columns with the correlation 1 are the best correlated and vice versa.

In [16]:
```
import seaborn as sns

sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris)


import matplotlib.pyplot as plt
plt.show()
```



Pairplots are a really simple way to visualize relationships between each variable. It produces a matrix of relationships between each variable in the data for an instant examination of our data.

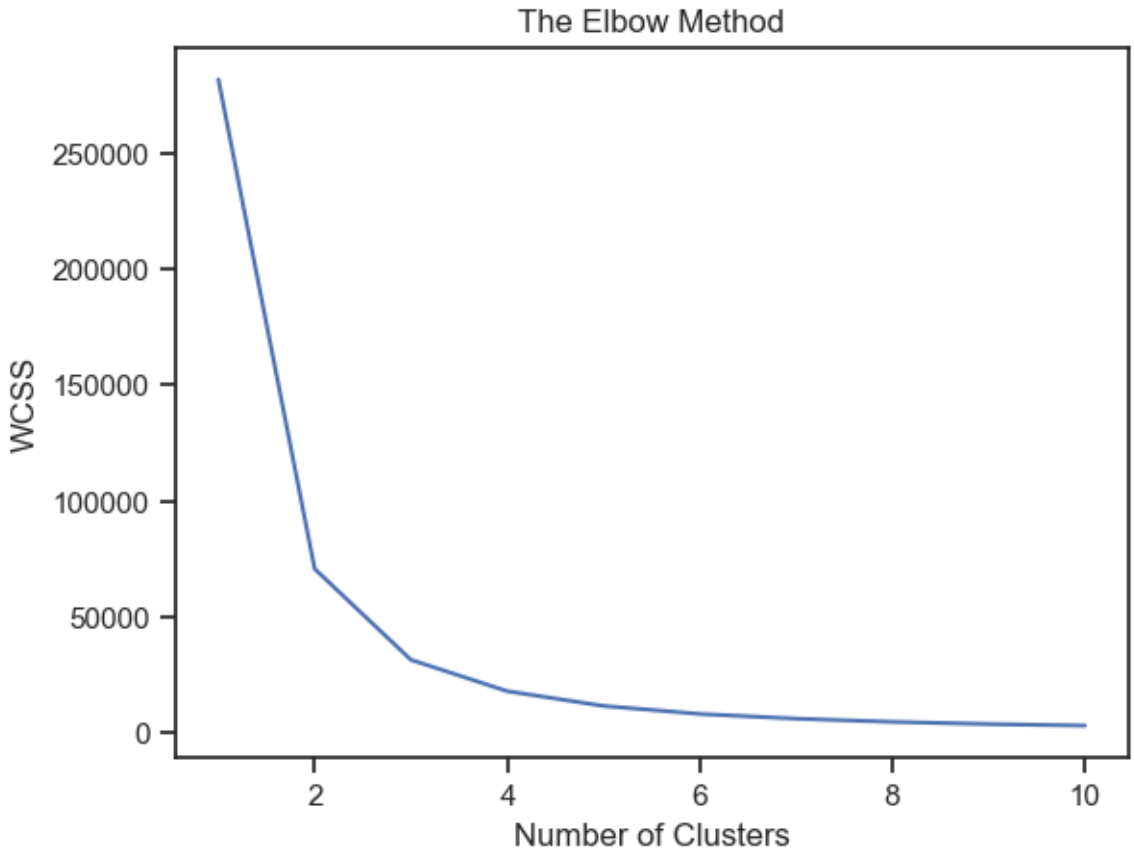
Finding the optimum number of clusters using k-means clustering

In [17]:
```
# Finding the optimum number of clusters using k-means

x = data.iloc[:,[0,1,2,3]].values

from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
    kmeans.fit(x)
    ## appending the WCSS to the list (kmeans.inertia_ returns the WCSS valu
    wcss.append(kmeans.inertia_)
    print('k:',i ,"wcss:",kmeans.inertia_)
```

```
C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 1 wcss: 281831.54466666665

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 2 wcss: 70581.3808

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 3 wcss: 31320.711199999998

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 4 wcss: 17793.59050704551

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 5 wcss: 11424.765205784202
```

```
C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 6 wcss: 7961.65012687353

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 7 wcss: 5922.7418560606075

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 8 wcss: 4542.214812865496

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 9 wcss: 3588.4878586601303

C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(

k: 10 wcss: 2933.402648809524
```

In [18]:
```
# Plotting the results onto a line graph, allowing us to observe 'The elbow'

plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```



We can see that after 3 the drop in WCSS is minimal. So we choose 3 as the optimal number of clusters.

Initializing K-Means With Optimum Number Of Clusters

In [19]:
```
# Fitting K-Means to the Dataset
kmeans = KMeans(n_clusters = 3, init = 'k-means++',max_iter = 300, n_init =

# Returns a label for each data point based on the number of clusters
y_kmeans = kmeans.fit_predict(x)
```

```
C:\Users\baps\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1436:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, wh
en there are less chunks than available threads. You can avoid it by setti
ng the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

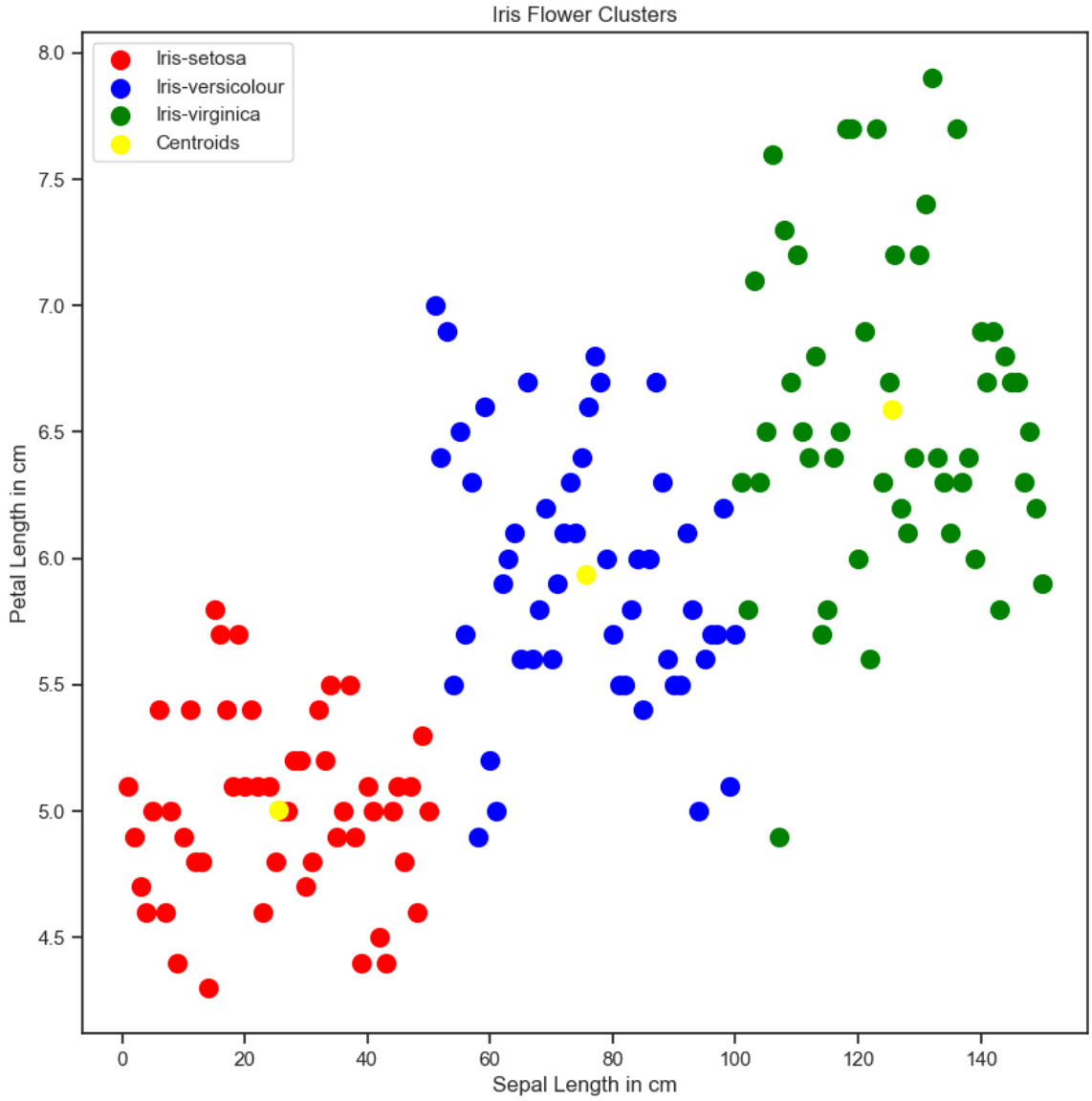
Predicting Values

In [20]:
```
y_kmeans
```

Out[20]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Visualizing the Clusters

In [21]:
```
# Visualising the clusters
plt.figure(figsize=(10,10))
plt.scatter(x[y_kmeans==0,0],x[y_kmeans==0,1],s=100,c='red',label='Iris-setc
plt.scatter(x[y_kmeans==1,0],x[y_kmeans==1,1],s=100,c='blue',label='Iris-ver
plt.scatter(x[y_kmeans==2,0],x[y_kmeans==2,1],s=100,c='green',label='Iris-vi

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=100,
plt.title('Iris Flower Clusters')
plt.xlabel('Sepal Length in cm')
plt.ylabel('Petal Length in cm')
plt.legend()
plt.show()
```



THANK YOU!

GitHub: https://github.com/anujtiwari21?tab=repositories (https://github.com/anujtiwari21?tab=repositories)