Virtual Internship Program

Author :- Mohd Farman

Beginer Level Tasks

Task-1 Iris Flowers Classification Ml Project

This particular ML project is usually referred to as the “Hello World” of Machine Learning. The iris flowers dataset contains numeric attributes, and it is perfect for beginners to learn about supervised ML algorithms, mainly how to load and handle data. Also, since this is a small dataset, it can easily fit in memory without requiring special transformations or scaling capabilities.
Datasetlink : http://archive.ics.uci.edu/ml/datasets/Iris

1. describe dataset

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

Attribute Information:

1. Sepal length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal width in cm
   Iris flower can be divided into 3 species as per the length and width of their Sepals and Petals:

   1) Iris Setosa
   2) Iris Versicolour
   3) Iris Virginica

2.Importing The Libraries

In [19]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
import os
os.environ["OMP_NUM_THREADS"] = '1'
```

3. Loading the dataset

In [33]:
```
# import csv data to a Pandas DataFrame
iris = pd.read_csv('Iris.csv')
```

In [34]:
```
iris.describe()
```

Out[34]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
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

In [35]:
```
iris.head()
```

Out[35]:

	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

In [36]:
```
iris.tail()
```

Out[36]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

In [37]:
```
iris.shape
```

Out[37]:
```
(150, 5)
```

In [38]:
```
# getting some inf.
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
```

In [39]:
```
iris['species'].value_counts()
```

Out[39]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

3.Preprocessing The Data

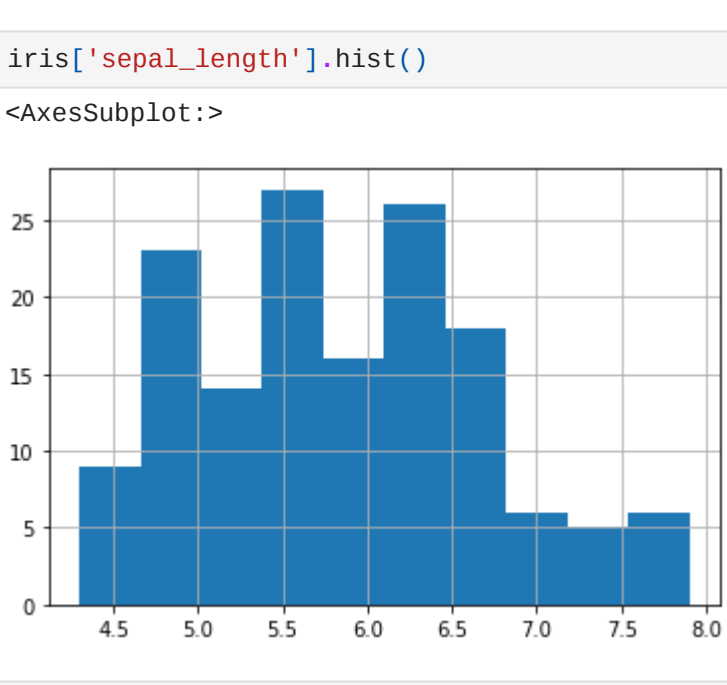
In [40]:
```
# check missing values
iris.isnull().sum()
```

Out[40]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

4. Data Analysis

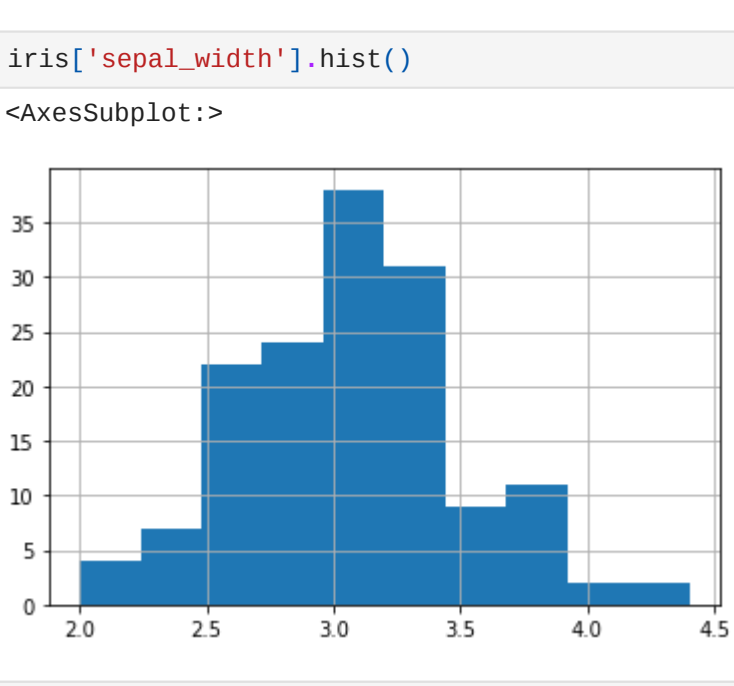
In [41]:
```
iris['sepal_length'].hist()
```

Out[41]: <AxesSubplot:>



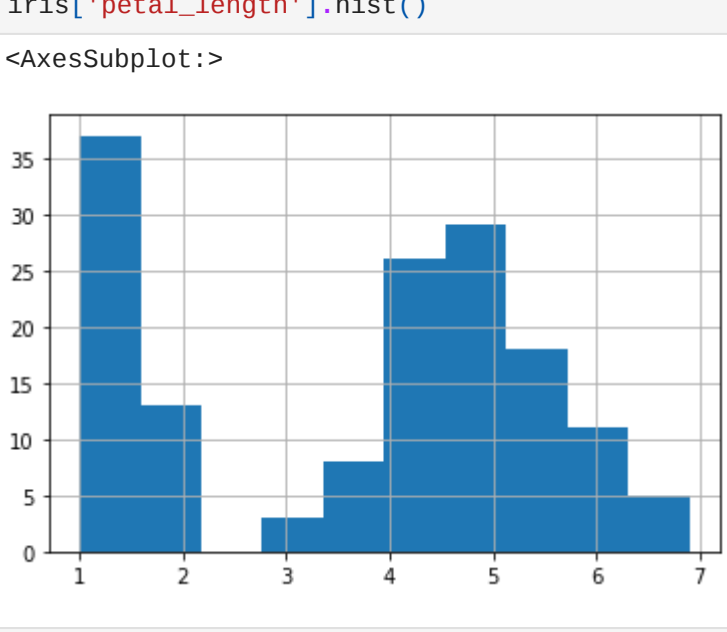
In [42]:
```
iris['sepal_width'].hist()
```

Out[42]: <AxesSubplot:>



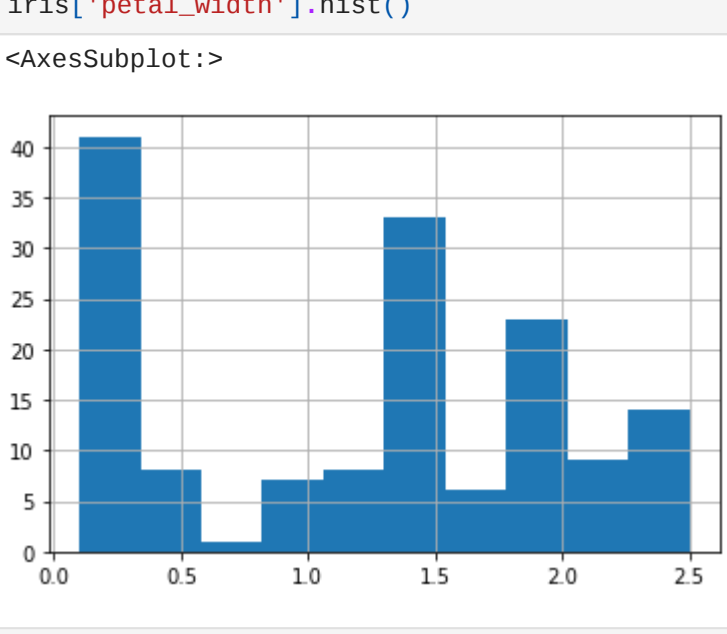
In [43]:
```
iris['petal_length'].hist()
```

Out[43]: <AxesSubplot:>



In [44]:
```
iris['petal_width'].hist()
```

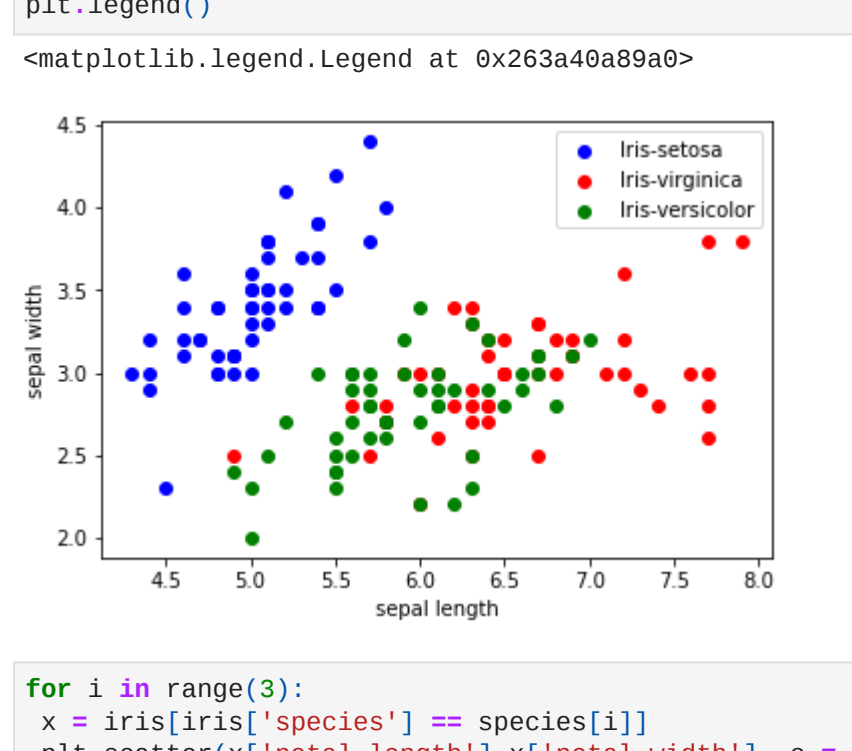
Out[44]: <AxesSubplot:>



In [47]:
```
# scatterplot
colours = ['blue','red','green']
species = ['Iris-setosa','Iris-virginica','Iris-versicolor']
```

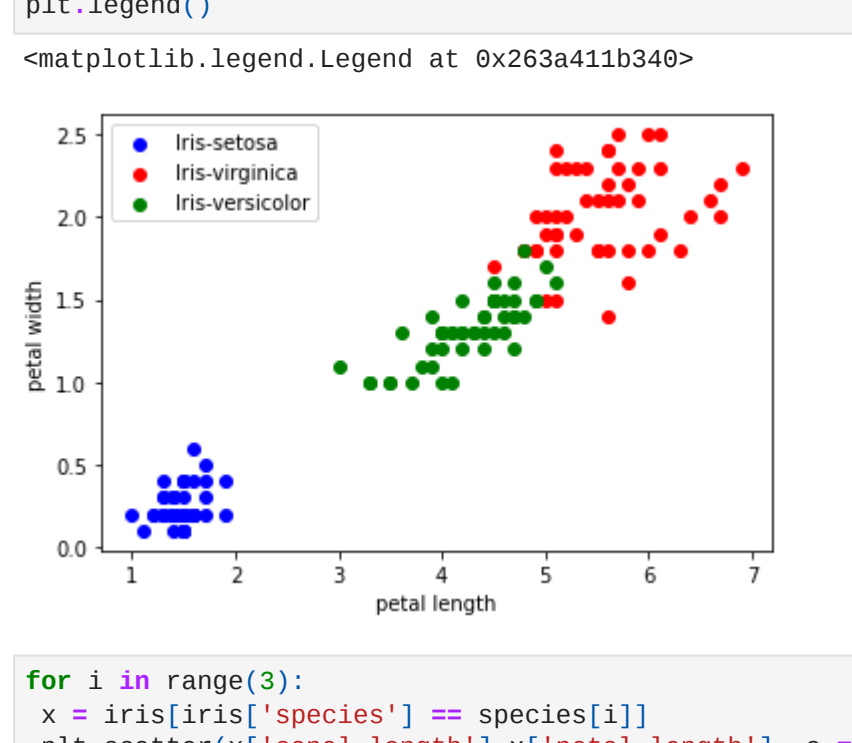
In [48]:
```
for i in range(3):
 x = iris[iris['species'] == species[i]]
 plt.scatter(x['sepal_length'],x['sepal_width'], c = colours[i], label=species[i])
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x263a40a89a0>



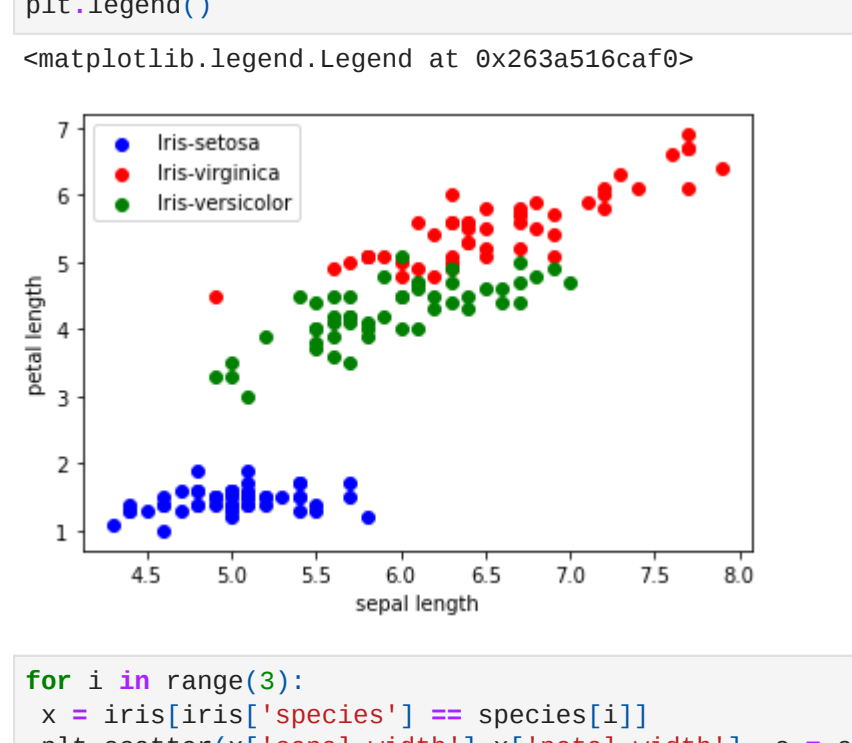
In [49]:
```
for i in range(3):
 x = iris[iris['species'] == species[i]]
 plt.scatter(x['petal_length'],x['petal_width'], c = colours[i], label=species[i])
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.legend()
```

Out[49]: <matplotlib.legend.Legend at 0x263a411b340>



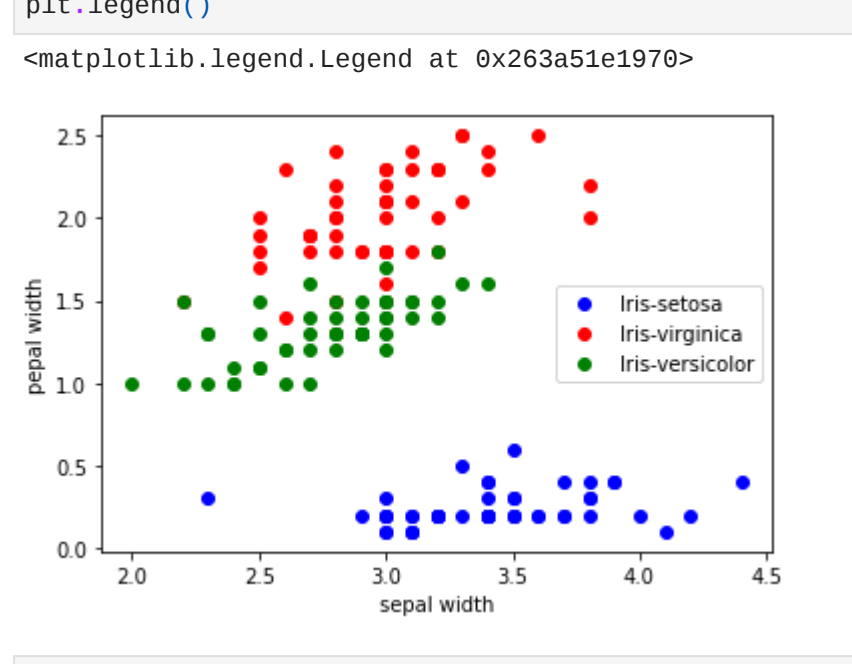
In [50]:
```
for i in range(3):
 x = iris[iris['species'] == species[i]]
 plt.scatter(x['sepal_length'],x['petal_length'], c = colours[i], label=species[i])
plt.xlabel('sepal length')
plt.ylabel('petal length')
plt.legend()
```

Out[50]: <matplotlib.legend.Legend at 0x263a516caf0>

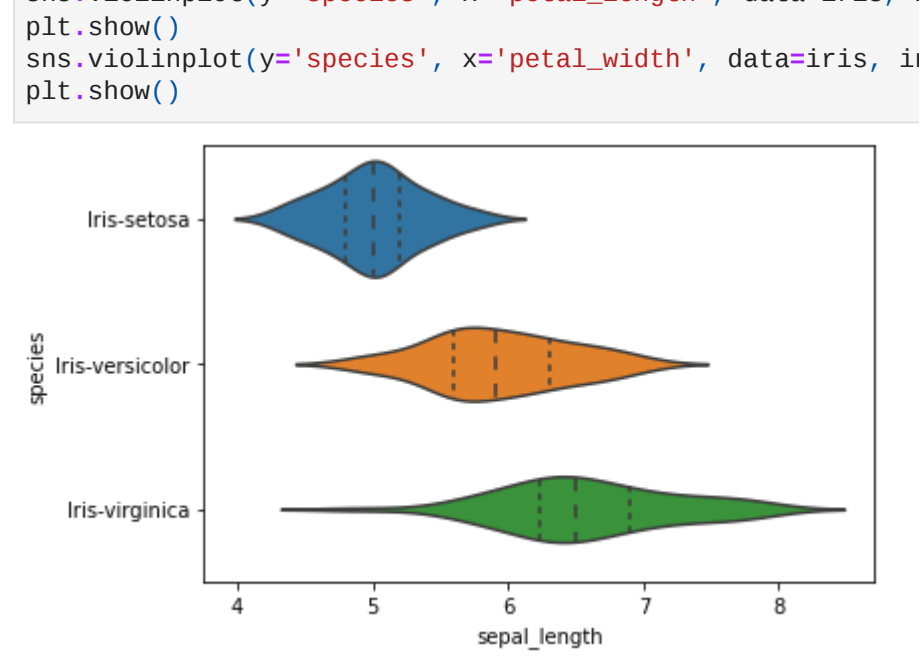


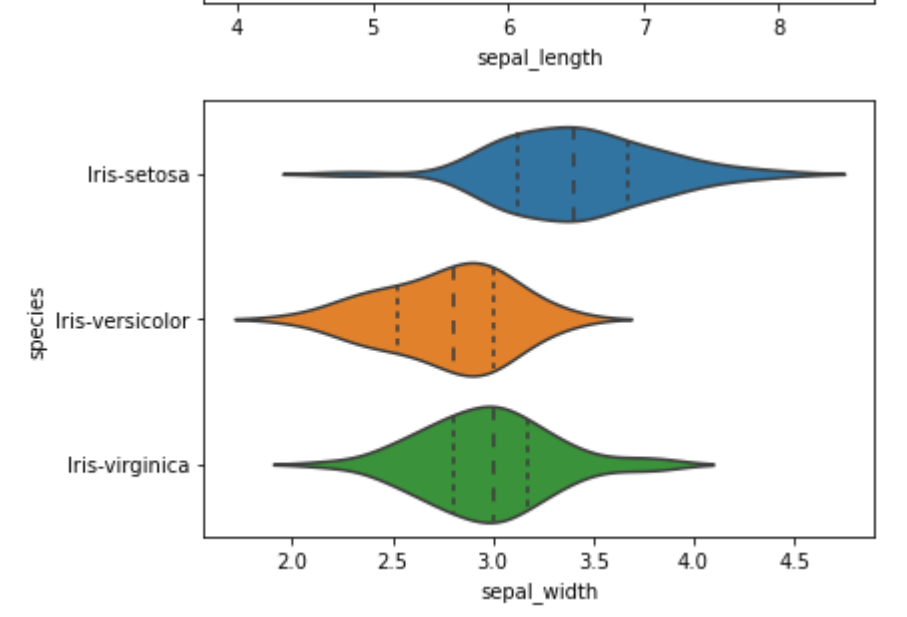
In [51]:
```
for i in range(3):
 x = iris[iris['species'] == species[i]]
 plt.scatter(x['sepal_width'],x['petal_width'], c = colours[i], label=species[i])
plt.xlabel('sepal width')
plt.ylabel('pepal width')
plt.legend()
```

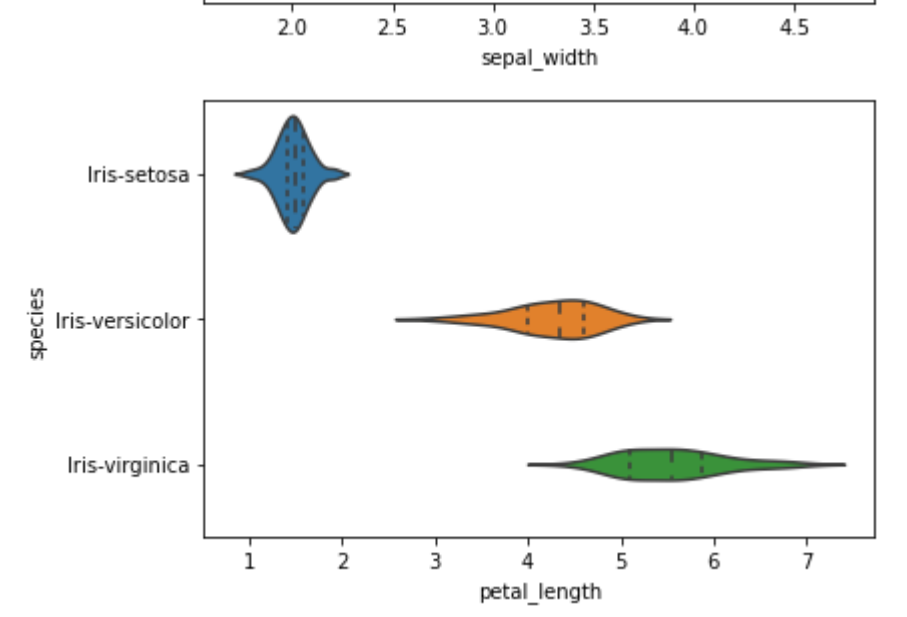
Out[51]: <matplotlib.legend.Legend at 0x263a51e1970>

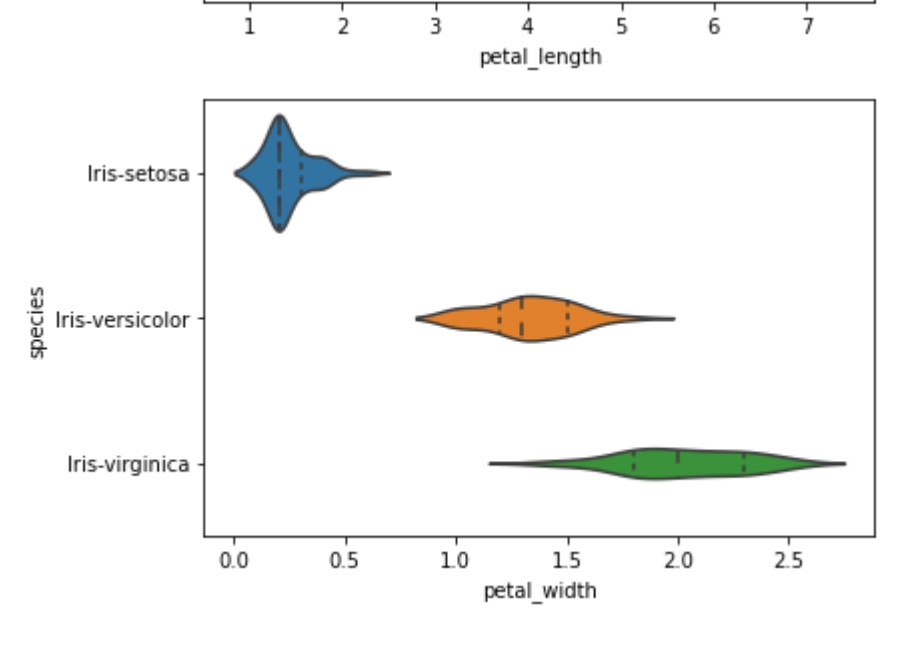


In [52]:
```
import seaborn as sns
sns.violinplot(y='species', x='sepal_length', data=iris, inner='quartile')
plt.show()
sns.violinplot(y='species', x='sepal_width', data=iris, inner='quartile')
plt.show()
sns.violinplot(y='species', x='petal_length', data=iris, inner='quartile')
plt.show()
sns.violinplot(y='species', x='petal_width', data=iris, inner='quartile')
plt.show()
```









5. Level Encoding

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering. Let's see how to implement label encoding in Python using the scikit-learn library and also understand the challenges with label encoding

In [53]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [54]:
```
iris['species'] = le.fit_transform(iris['species'])
iris.head()
```

Out[54]:

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

In [55]:
```
iris.tail()
```

Out[55]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

In [56]:
```
iris.species
```

Out[56]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: species, Length: 150, dtype: int32
```

6. Model Training

Model training is the phase in the data science development lifecycle where practitioners try to fit the best combination of weights and bias to a machine learning algorithm to minimize a loss function over the prediction range

In [57]:
```
X = iris.drop(columns='species', axis=1)
Y = iris['species']
```

In [58]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, stratify=Y, random_state=2)
```

In [59]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

In [60]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
```

Out[60]:
```
LogisticRegression()
```

In [61]:
```
# accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

In [62]:
```
print('Accuracy on Training data : ', training_data_accuracy*100)
```

```
Accuracy on Training data :  96.66666666666667
```

In [63]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

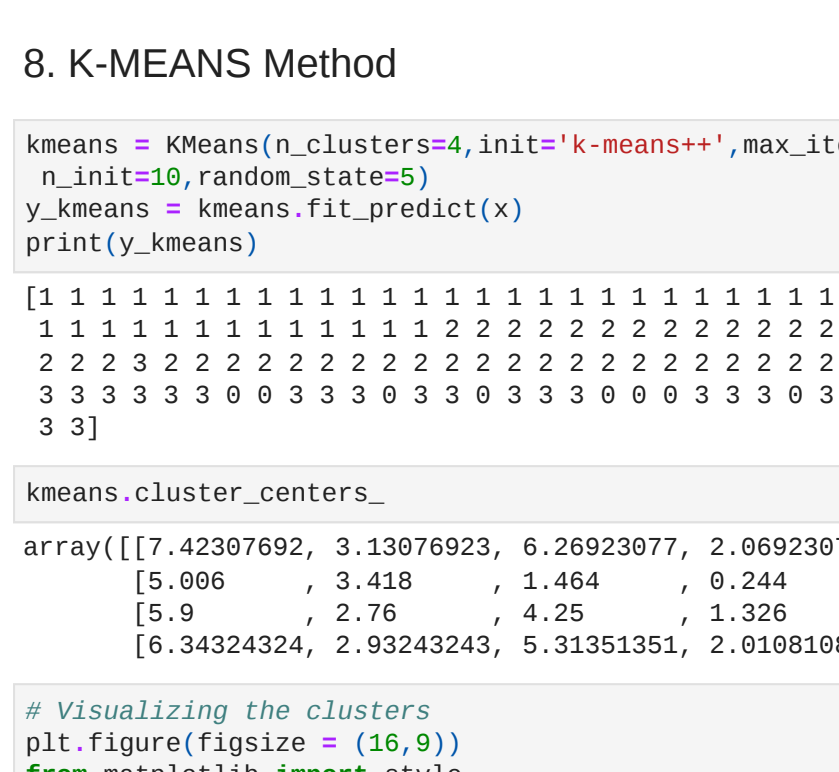
In [64]:
```
print('Accuracy on Test data : ', test_data_accuracy*100)
```

```
Accuracy on Test data :  100.0
```

7. Optimal Number Of Clustring For K-Means Classification Algoritham

In [65]:
```
# Finding the optimal number of clusters for k-means classification Algoritham
x = iris.iloc[:, [0, 1, 2, 3, 4]].values
# we import KMeans algorithm using sklearn library
from sklearn.cluster import KMeans
# we use the very first method is Elbow Method
wcss = []
# WCSS means Within Cluster Sum of Squares
for i in range(1, 11):
 kmeans = KMeans(n_clusters = i, init = 'k-means++',
 max_iter = 300, n_init = 10, random_state = 0)
 kmeans.fit(x)
 wcss.append(kmeans.inertia_)

# plotting above result in line graph format
plt.plot(range(1, 11), wcss)
plt.title('The Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters and picking the elbow of the curve as the number of clusters to use.

8. K-MEANS Method

In [66]:
```
kmeans = KMeans(n_clusters=4,init='k-means++',max_iter=300,
 n_init=10,random_state=5)
y_kmeans = kmeans.fit_predict(x)
print(y_kmeans)
```

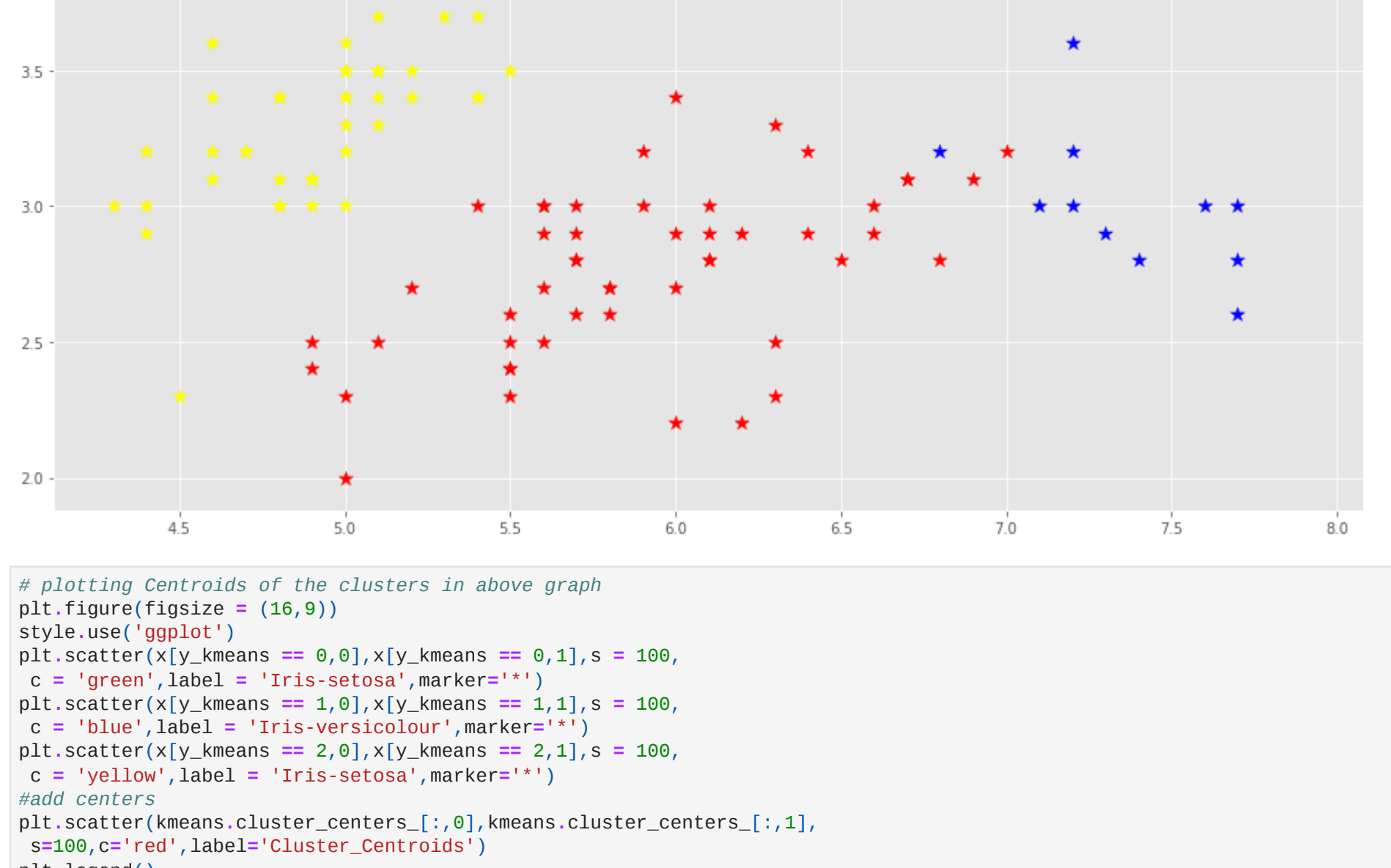
```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 0 3 3 0 2 0 3 0 3
 3 3 3 3 3 0 0 3 3 3 0 3 3 0 3 3 0 0 0 3 3 3 0 3 3 3 3 3 3 3 0 3 3 3 3
 3 3]
```

In [67]:
```
kmeans.cluster_centers_
```

Out[67]:
```
array([[7.42307692, 3.13076923, 6.26923077, 2.06923077, 2.        ],
       [5.006     , 3.418     , 1.464     , 0.244     , 0.        ],
       [5.9       , 2.76      , 4.25      , 1.326     , 1.02      ],
       [6.34324324, 2.93243243, 5.31351351, 2.01081081, 1.97297297]])
```

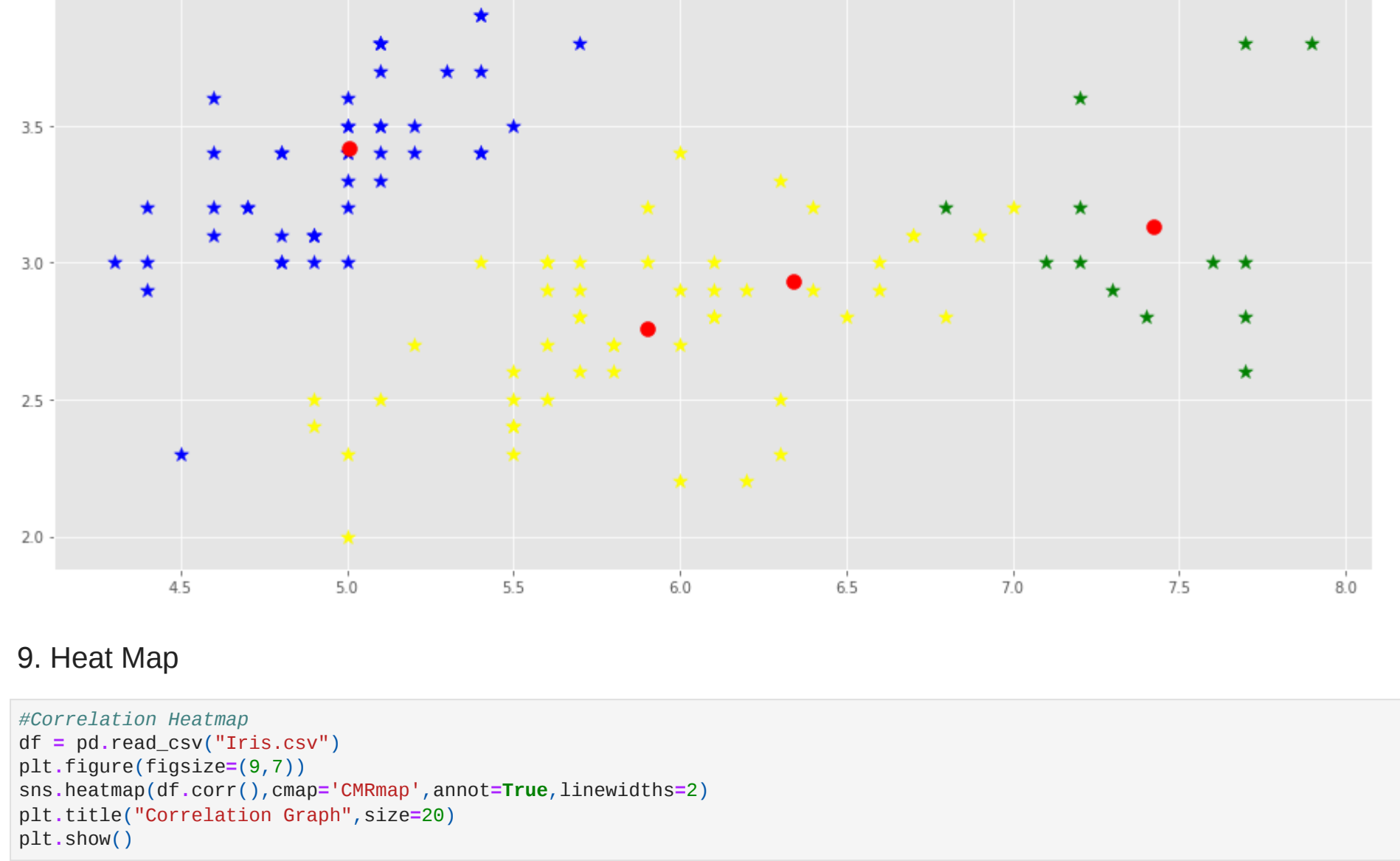
In [68]:
```
# Visualizing the clusters
plt.figure(figsize = (16,9))
from matplotlib import style
style.use('ggplot')
plt.scatter(x[y_kmeans == 0,0],x[y_kmeans == 0,1],s = 100,
 c = 'blue',label = 'Iris-setosa',marker='*')
plt.scatter(x[y_kmeans == 1,0],x[y_kmeans == 1,1],s = 100,
 c = 'yellow',label = 'Iris-versicolour',marker='*')
plt.scatter(x[y_kmeans == 2,0],x[y_kmeans == 2,1],s = 100,
 c = 'red',label = 'Iris-setosa',marker='*')
plt.legend()
```

Out[68]: <matplotlib.legend.Legend at 0x263a5346f40>



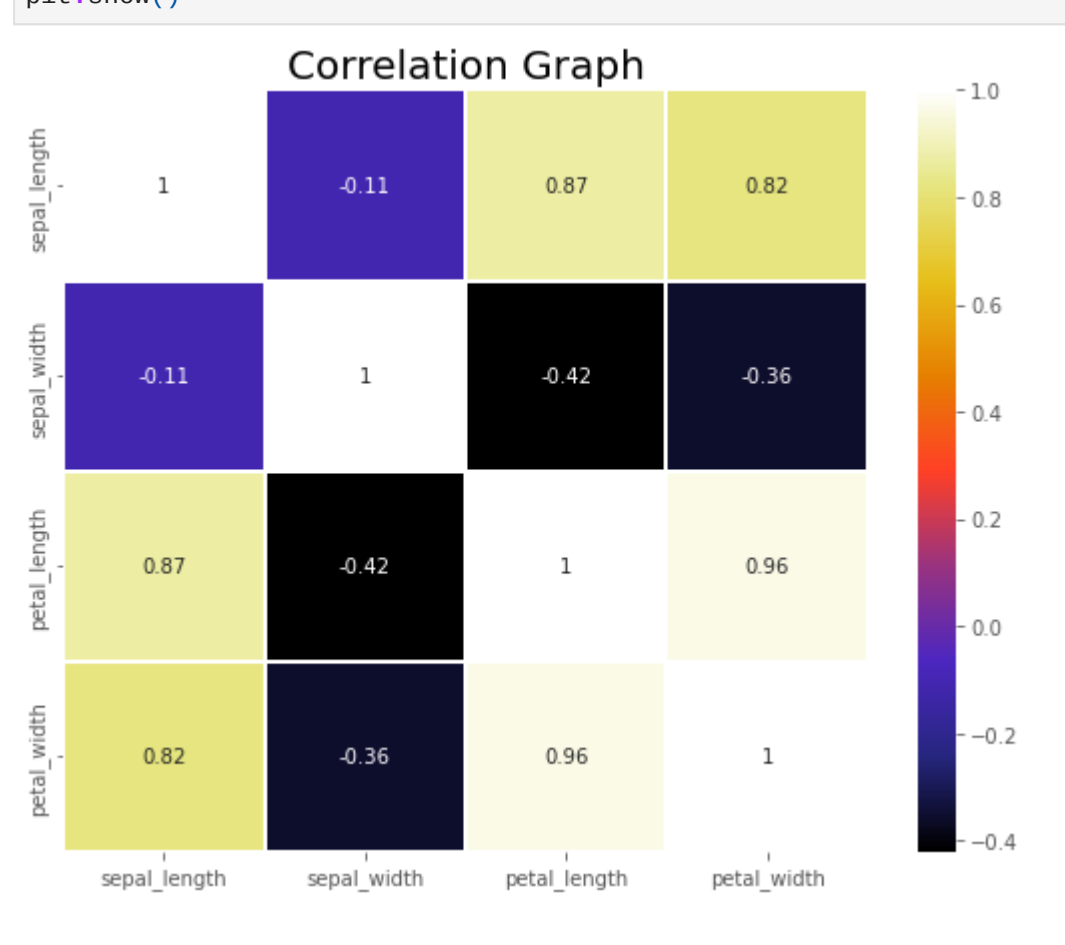
In [69]:
```
# plotting Centroids of the clusters in above graph
plt.figure(figsize = (16,9))
style.use('ggplot')
plt.scatter(x[y_kmeans == 0,0],x[y_kmeans == 0,1],s = 100,
 c = 'green',label = 'Iris-setosa',marker='*')
plt.scatter(x[y_kmeans == 1,0],x[y_kmeans == 1,1],s = 100,
 c = 'blue',label = 'Iris-versicolour',marker='*')
plt.scatter(x[y_kmeans == 2,0],x[y_kmeans == 2,1],s = 100,
 c = 'yellow',label = 'Iris-setosa',marker='*')
#add centers
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],
 s=100,c='red',label='Cluster_Centroids')
plt.legend()
```

Out[69]: <matplotlib.legend.Legend at 0x263a5ac23d0>



9. Heat Map

In [70]:
```
#Correlation Heatmap
df = pd.read_csv("Iris.csv")
plt.figure(figsize=(9,7))
sns.heatmap(df.corr(),cmap='CMRmap',annot=True,linewidths=2)
plt.title("Correlation Graph",size=20)
plt.show()
```



THANK YOU..

In [ ]: