Project Info

- Creat the deciion tree classifier and visualize it graphically.
- The purpose is if we feed any new data to this classifier, it would be able to predict the right class accordingly.

Attribute Information:

1. Sepal Length in cm
2. Sepal Width in cm
3. Petal Length in cm
4. Petal Width in cm
5. Species:
   `(Iris-setosa,`
   `Iris-versicolor,`
   `Iris-virginica)`

Import necessary libraries

In [21]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier


import warnings
warnings.filterwarnings('ignore')
```

Load & Explore data

In [2]:
```
data=pd.read_csv('D:/Projects/GRIP/Iris.csv')
data.head()
```

Out[2]:

	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

In [3]:
```
data.drop('Id',axis=1,inplace=True)
```

In [4]:
```
# To show basic info about datatype
data.info()
shape = data.shape
print(f'\n Number of Rows = {shape[0]}\n Number of columns = {shape[1]} ')
```

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

 Number of Rows = 150
 Number of columns = 5
```

In [5]:
```
# To display stats about the data
data.describe()
```

Out[5]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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 [6]:
```
# To display no. of samples on each class
data['Species'].value_counts()
```

Out[6]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [7]:
```
# Check for null values
data.isnull().sum()
```

Out[7]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [8]:
```
# drop ID and Species columns
iris = data.drop(['Species'],axis=1)
iris.head()
```

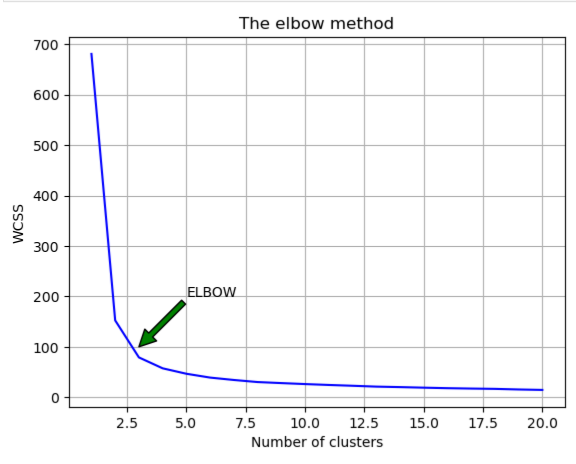
Out[8]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

Find the optimum number of clusters for K Means

In [9]:
```
x = iris.iloc[:, :].values
wcss = []      # WCSS means >> Within cluster sum of squares

for i in range(1, 21):
    kmean = KMeans(n_clusters = i, init = 'k-means++',
                   max_iter = 200, n_init = 15, random_state = 0)
    kmean.fit(x)
    wcss.append(kmean.inertia_)
```

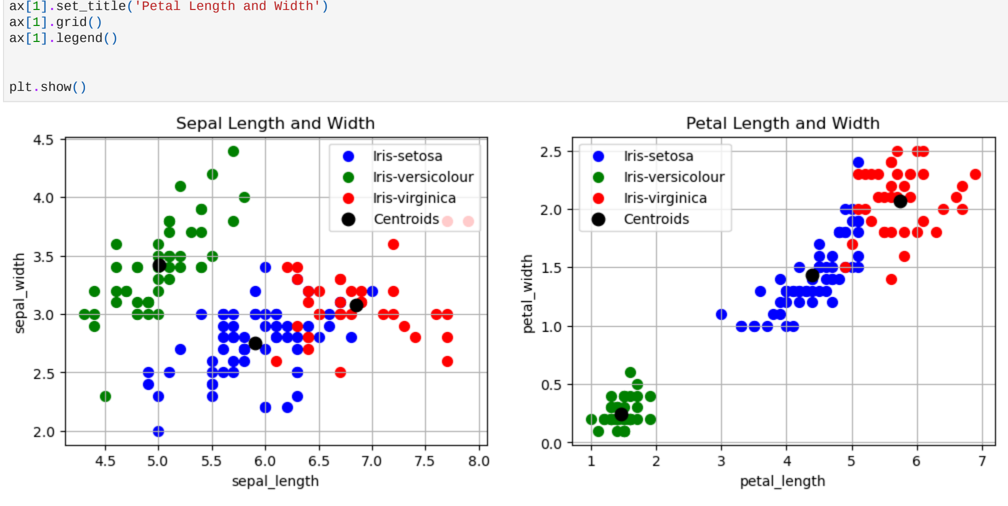
In [10]:
```
# Plotting the results onto a line graph,
# `allowing us to observe 'The elbow'
plt.plot(range(1, 21), wcss, color = "blue")
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.annotate('ELBOW',xytext=(5,200),xy=(3,100),arrowprops={'facecolor':'green'})
plt.grid()
plt.show()
```



- You can clearly see why it is called 'The elbow method' from the above graph, the optimum clusters is where the elbow occurs.
- This is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration.
- From this we choose the number of clusters as 3 Clusters

In [11]:
```
# Applying kmeans to the dataset
kmean = KMeans(n_clusters = 3, init = 'k-means++',
               max_iter = 200, n_init = 15, random_state = 0)
y = kmean.fit_predict(x)
```

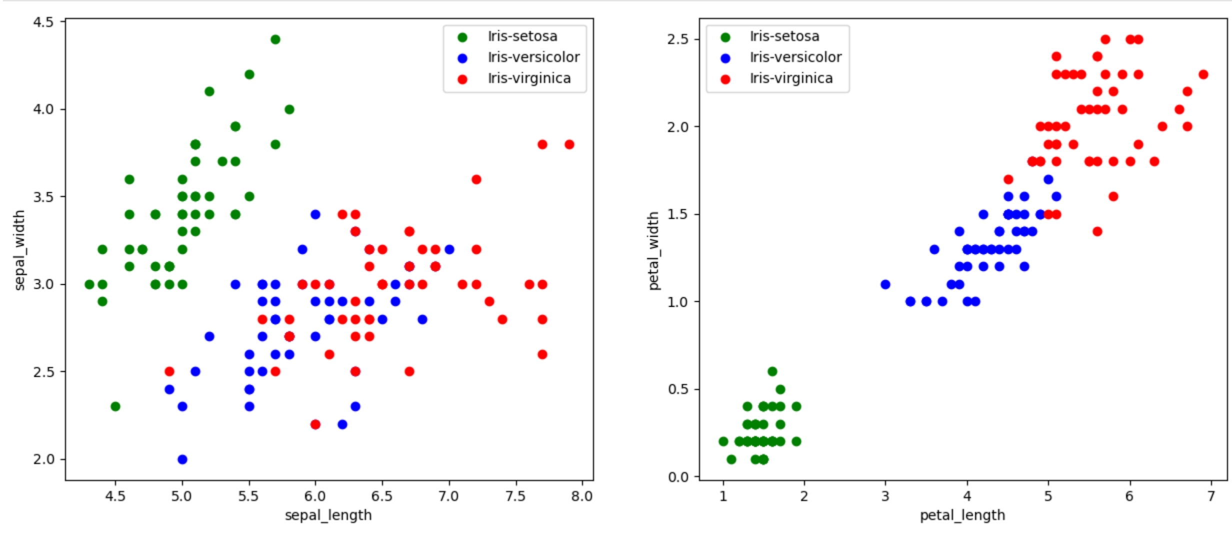
In [12]:
```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))

# Visualising the clusters - On the first two columns (sepal length, sepal width)
ax[0].scatter(x[y == 0, 0], x[y == 0, 1],
              s=50, c='blue', label='Iris-setosa')
ax[0].scatter(x[y == 1, 0], x[y == 1, 1],
              s=50, c='green', label='Iris-versicolour')
ax[0].scatter(x[y == 2, 0], x[y == 2, 1],
              s=50, c='red', label='Iris-virginica')
ax[0].scatter(kmean.cluster_centers_[:, 0], kmean.cluster_centers_[:, 1],
              s=80, c='black', label='Centroids')
ax[0].set_xlabel('sepal_length')
ax[0].set_ylabel('sepal_width')
ax[0].set_title('Sepal Length and Width')
ax[0].grid()
ax[0].legend()

# Visualising the clusters - On the second two columns (petal length, petal width)
ax[1].scatter(x[y == 0, 2], x[y == 0, 3],
              s=50, c='blue', label='Iris-setosa')
ax[1].scatter(x[y == 1, 2], x[y == 1, 3],
              s=50, c='green', label='Iris-versicolour')
ax[1].scatter(x[y == 2, 2], x[y == 2, 3],
              s=50, c='red', label='Iris-virginica')
ax[1].scatter(kmean.cluster_centers_[:, 2], kmean.cluster_centers_[:, 3],
              s=80, c='black', label='Centroids')
ax[1].set_xlabel('petal_length')
ax[1].set_ylabel('petal_width')
ax[1].set_title('Petal Length and Width')
ax[1].grid()
ax[1].legend()


plt.show()
```



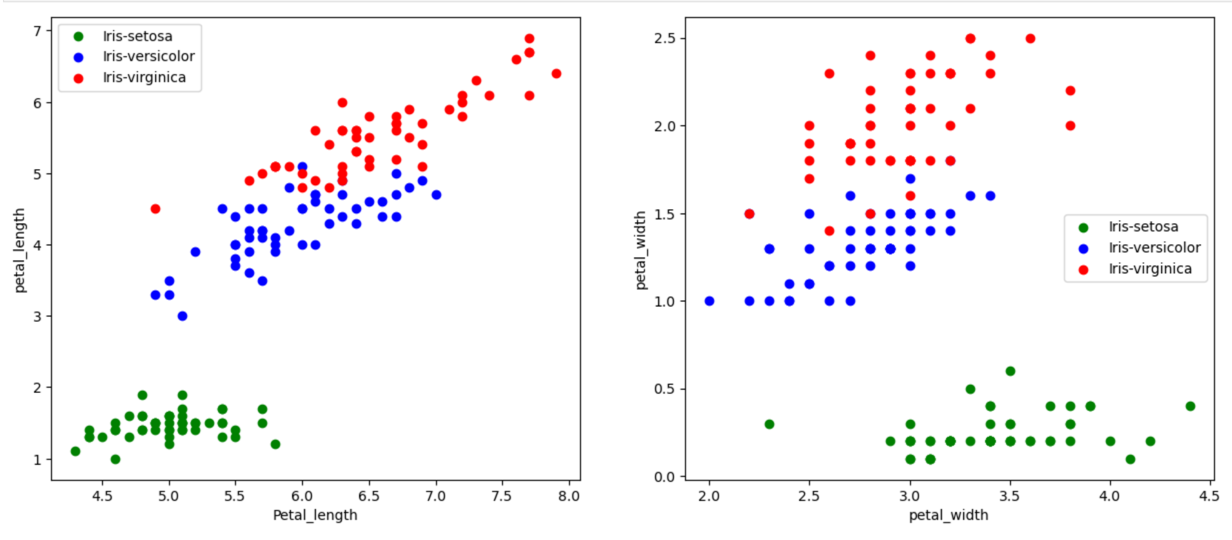
The Original data

In [13]:
```
colors = ['green', 'blue', 'red']
species = ['Iris-setosa','Iris-versicolor','Iris-virginica']
```

In [14]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,6))

for i in range(3):
    x_sepal = data[data['Species']==species[i]]
    ax1.scatter(x_sepal['SepalLengthCm'], x_sepal['SepalWidthCm'], c = colors[i], label = species[i])

ax1.set_xlabel('sepal_length')
ax1.set_ylabel('sepal_width')
ax1.legend()


for i in range(3):
    x_petal = data[data['Species']==species[i]]
    ax2.scatter(x_petal['PetalLengthCm'], x_petal['PetalWidthCm'], c = colors[i], label = species[i])

ax2.set_xlabel('petal_length')
ax2.set_ylabel('petal_width')
ax2.legend()
plt.show()
```



In [15]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(15,6))

for i in range(3):
    x = data[data['Species']==species[i]]
    ax1.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], label = species[i])

ax1.set_xlabel('Petal_length')
ax1.set_ylabel('petal_length')
ax1.legend()

for i in range(3):
    x = data[data['Species']==species[i]]
    ax2.scatter(x['SepalWidthCm'], x['PetalWidthCm'], c = colors[i], label = species[i])

ax2.set_xlabel('petal_width')
ax2.set_ylabel('petal_width')
ax2.legend()
plt.show()
```



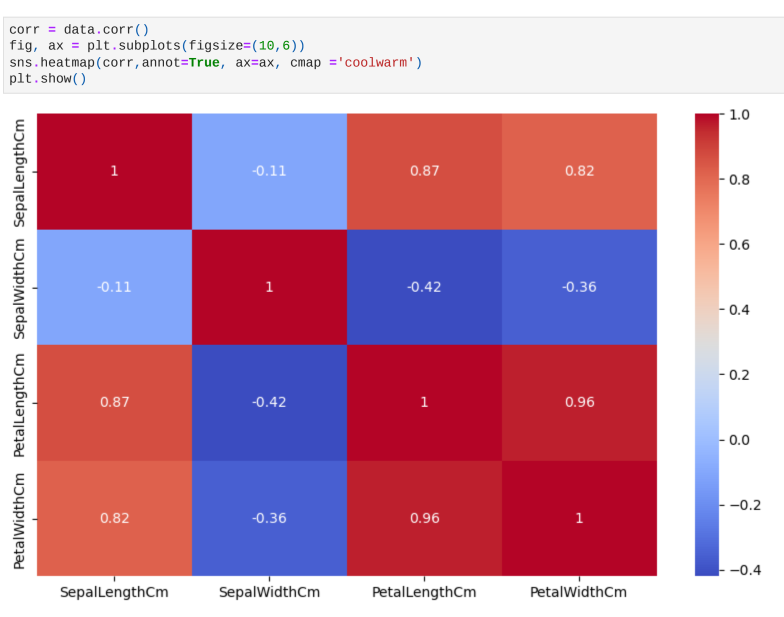
Coorelation Matrix

In [16]:
```
data.corr()
```

Out[16]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

In [17]:
```
corr = data.corr()
fig, ax = plt.subplots(figsize=(10,6))
sns.heatmap(corr,annot=True, ax=ax, cmap ='coolwarm')
plt.show()
```



Model Training

In [18]:
```
x = data.drop('Species', axis =1)
y = data['Species']
```

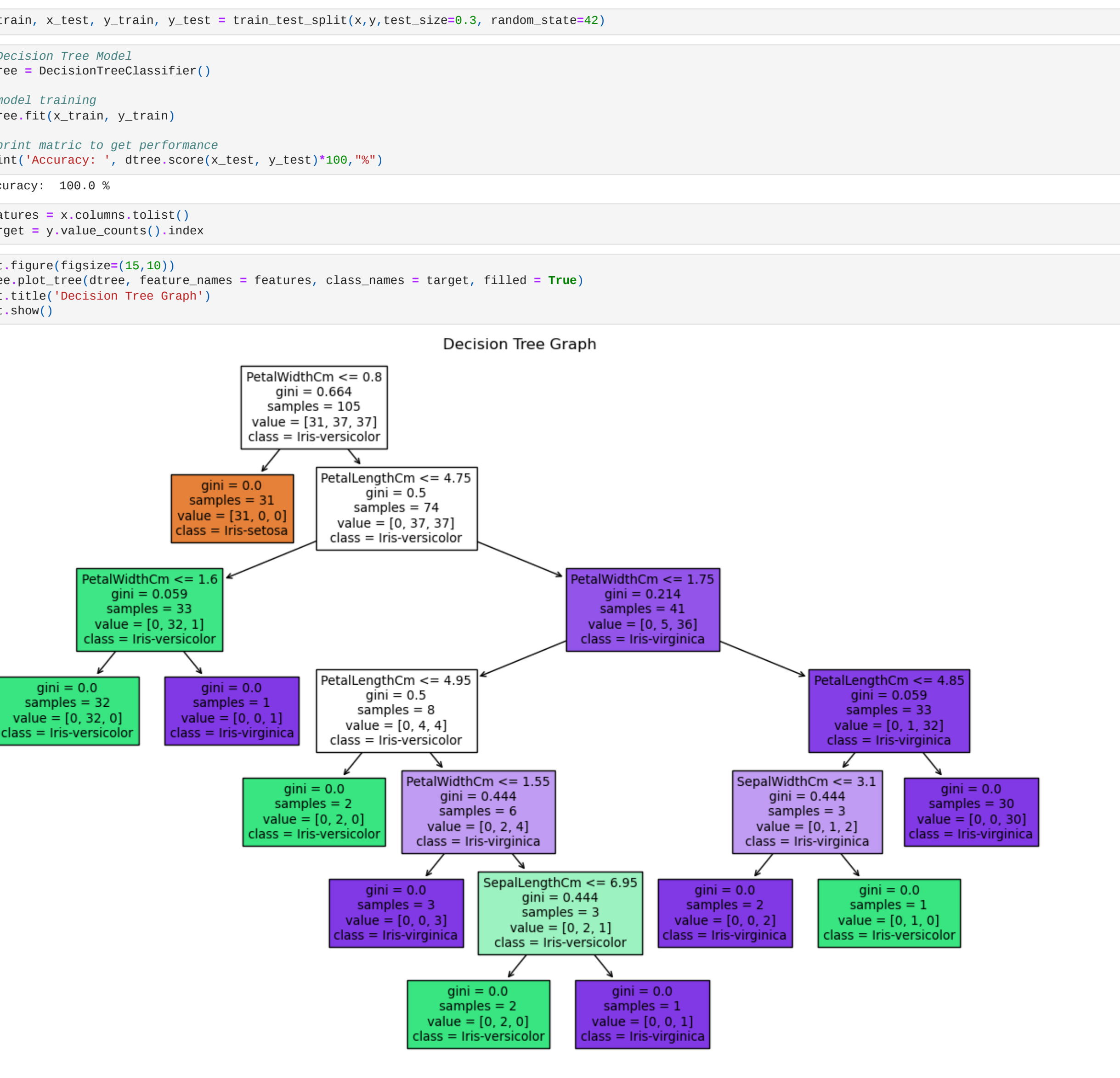
In [19]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state=42)
```

In [22]:
```
# Decision Tree Model
dtree = DecisionTreeClassifier()

# model training
dtree.fit(x_train, y_train)

# print matric to get performance
print('Accuracy: ', dtree.score(x_test, y_test)*100,"%")
```

```
Accuracy:  100.0 %
```

In [23]:
```
features = x.columns.tolist()
target = y.value_counts().index
```

In [24]:
```
plt.figure(figsize=(15,10))
tree.plot_tree(dtree, feature_names = features, class_names = target, filled = True)
plt.title('Decision Tree Graph')
plt.show()
```



In [ ]:

In [ ]: