# Dataset Information

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
5. class: -- Iris Setosa -- Iris Versicolour -- Iris Virginica

# Import modules

In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
pwd
```

Out[4]:
```
'C:\\Users\\AHSAN'
```

# Loading the dataset

In [15]:
```
df = pd.read_csv('Iris.csv')
df.head()
```

Out[15]:

| | 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]:
```
df.shape
```

Out[3]:
```
(150, 6)
```

In [16]:
```
# delete a column
#df = df.drop(columns = ['Id'])
#df.head()
```

In [17]:
```
# to display stats about data
df.describe()
```

Out[17]:

| | 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 |

In [18]:
```
# to basic info about datatype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 6.5+ KB
```

In [19]:
```
# to display no. of samples on each class
df['Species'].value_counts()
```

Out[19]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

# Preprocessing the dataset

In [20]:

```
# check for null values
df.isnull().sum()
```

Out[20]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

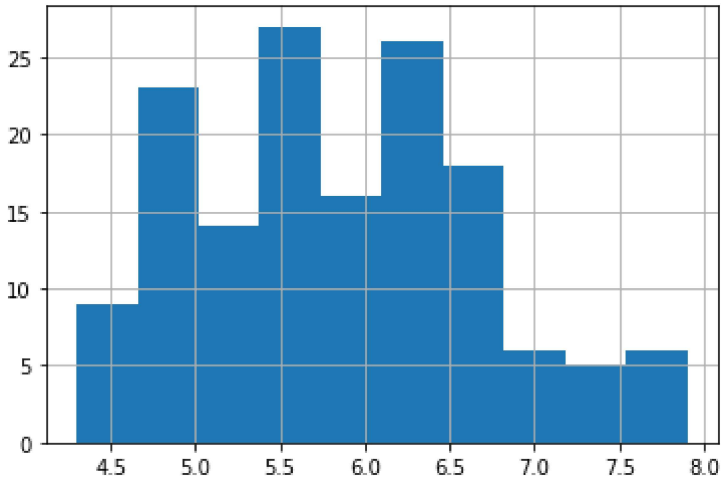
In [ ]:
```

```

# Exploratory Data Analysis

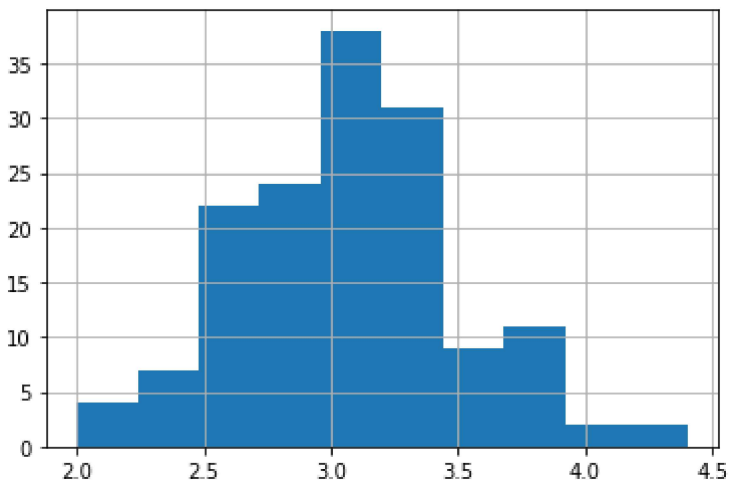
In [11]:
```
# histograms
df['SepalLengthCm'].hist()
```

Out[11]:
```
<AxesSubplot:>
```



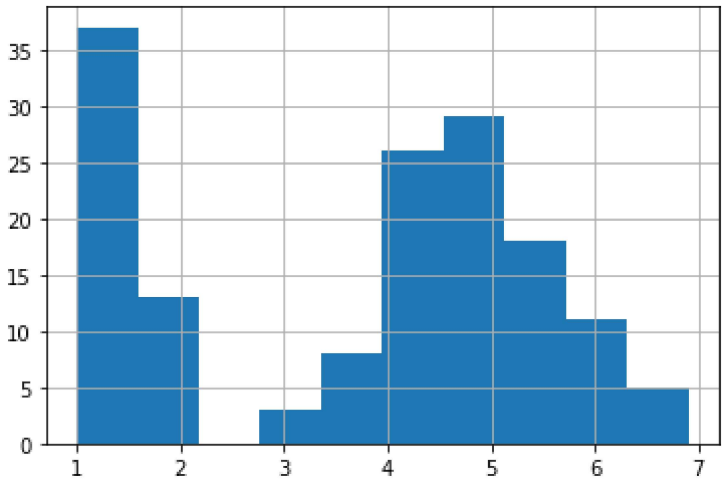

In [21]:
```
df['SepalWidthCm'].hist()
```

Out[21]:
```
<AxesSubplot:>
```



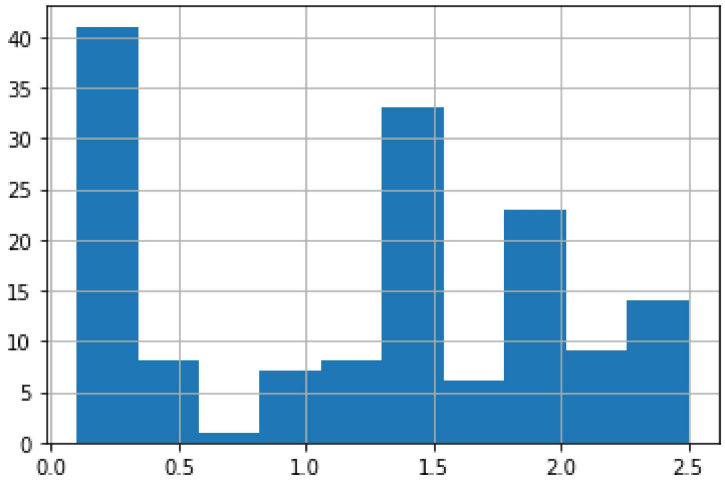
In [22]:
```
df['PetalLengthCm'].hist()
```

Out[22]:
```
<AxesSubplot:>
```



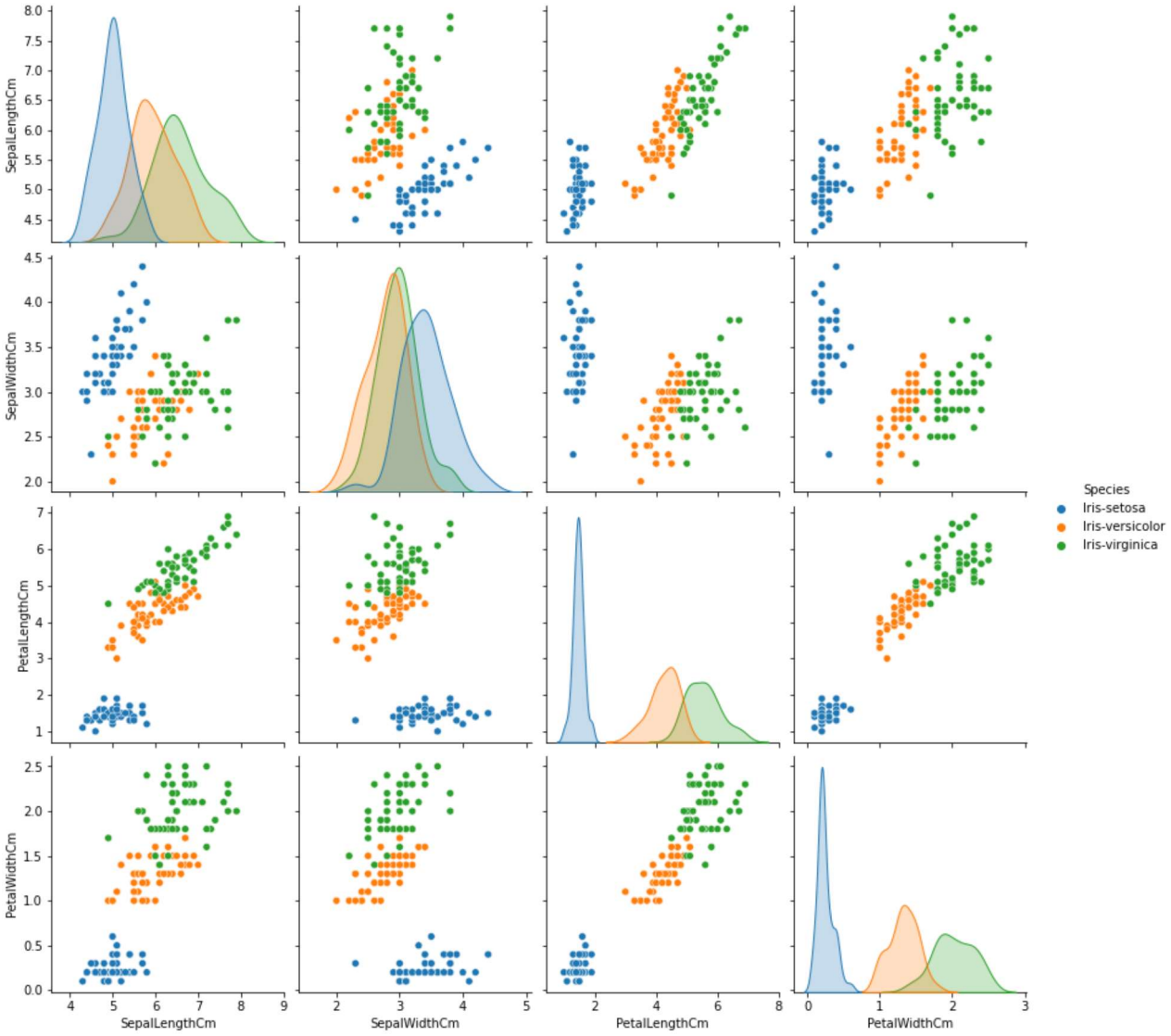

In [23]:
```
df['PetalWidthCm'].hist()
```

Out[23]:
```
<AxesSubplot:>
```




In [24]:
```
#from Pairplot, we'll see that the Iris-setosa species is separated frpm the other
#Two across all feature combinations
sns.pairplot(df.drop("Id",axis=1), hue="Species", size=3)
```

Out[24]:
```
<seaborn.axisgrid.PairGrid at 0x507fb80>
```

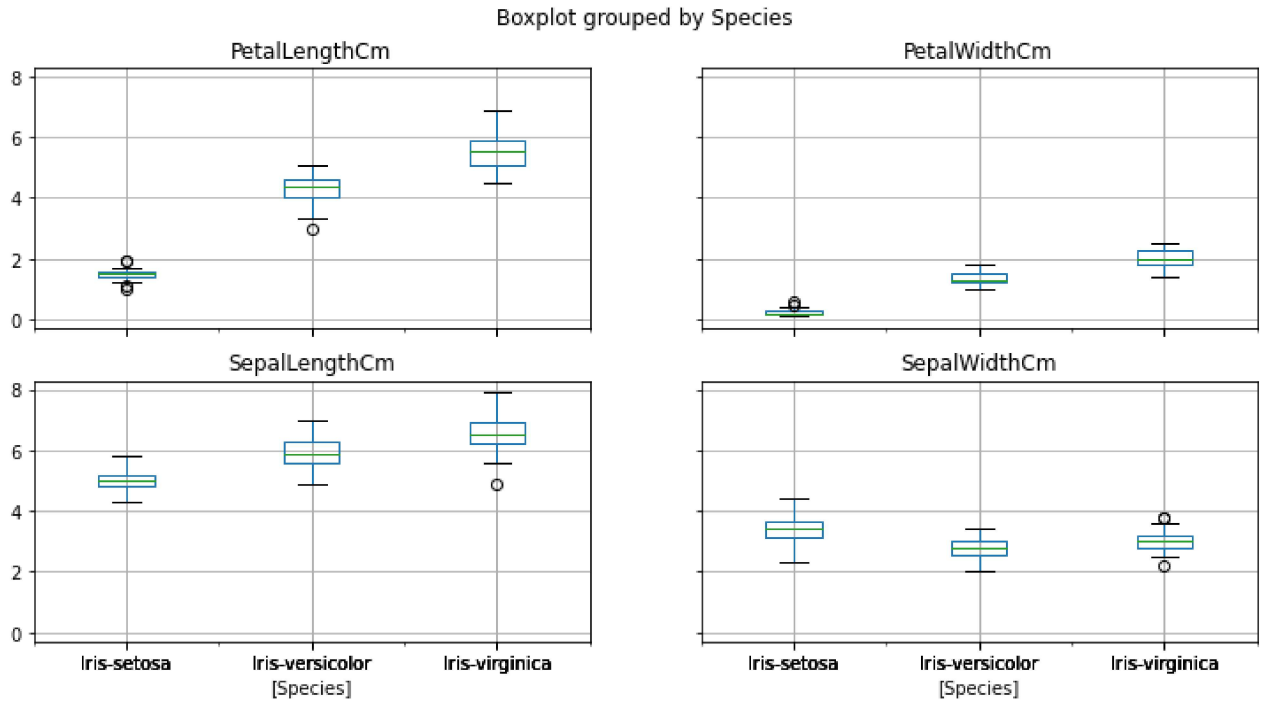



In [29]:

```
# Box plot grid
df.drop("Id",axis=1).boxplot(by="Species", figsize=(12,6))
```

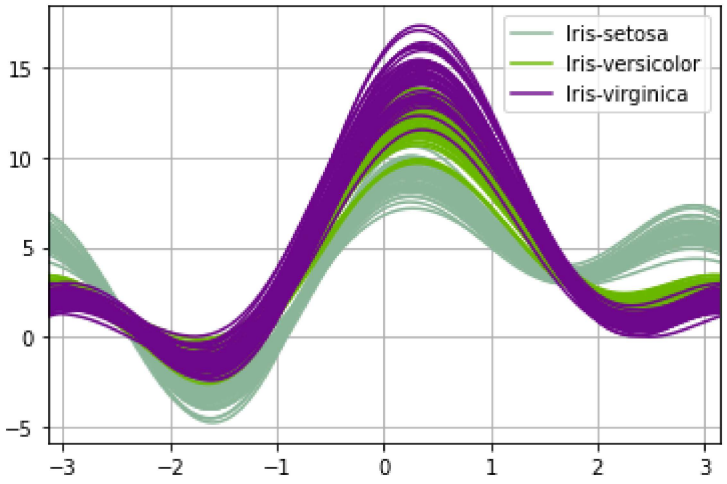
Out[29]:

```
array([[<AxesSubplot:title={'center':'PetalLengthCm'}, xlabel='[Species]'>,
        <AxesSubplot:title={'center':'PetalWidthCm'}, xlabel='[Species]'>],
       [<AxesSubplot:title={'center':'SepalLengthCm'}, xlabel='[Species]'>,
        <AxesSubplot:title={'center':'SepalWidthCm'}, xlabel='[Species]'>]],
      dtype=object)
```



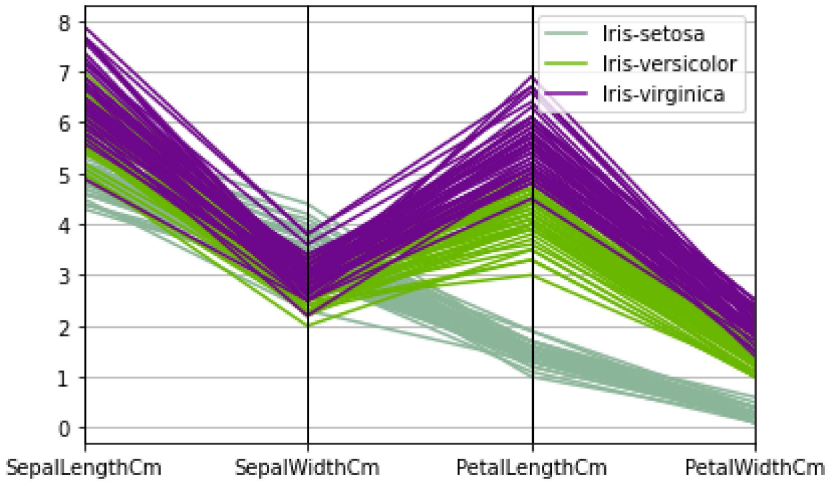

In [30]:
```
# Andrew Curves involve using attributes of sample as coefficients for Fourier series
#and then plotting these
from pandas.plotting import andrews_curves
andrews_curves(df.drop("Id", axis=1), "Species")
```

Out[30]: <AxesSubplot:>



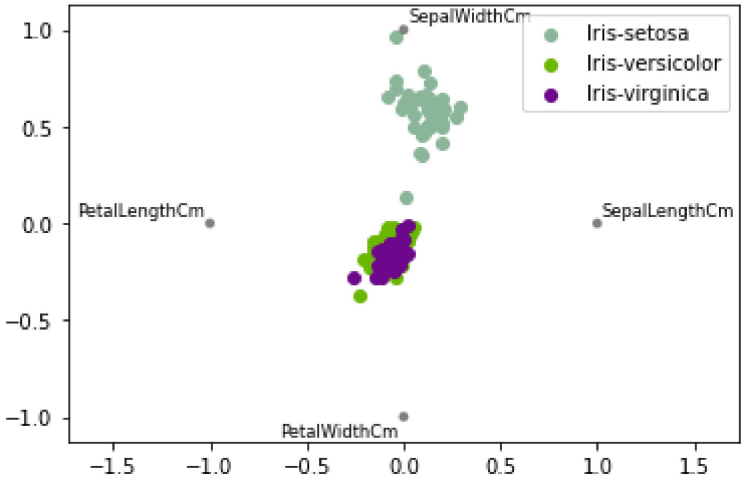

In [31]:
```
# Technique pandas has Parallel_Coordinates
# Parallel coordinates plots each feature on a separate column & then draws lines
# connecting the features for each data sample
from pandas.plotting import parallel_coordinates
parallel_coordinates(df.drop("Id",axis=1),"Species")
```

Out[31]: <AxesSubplot:>




In [32]:
```
# with radviz
from pandas.plotting import radviz
radviz(df.drop("Id",axis=1),"Species")
```

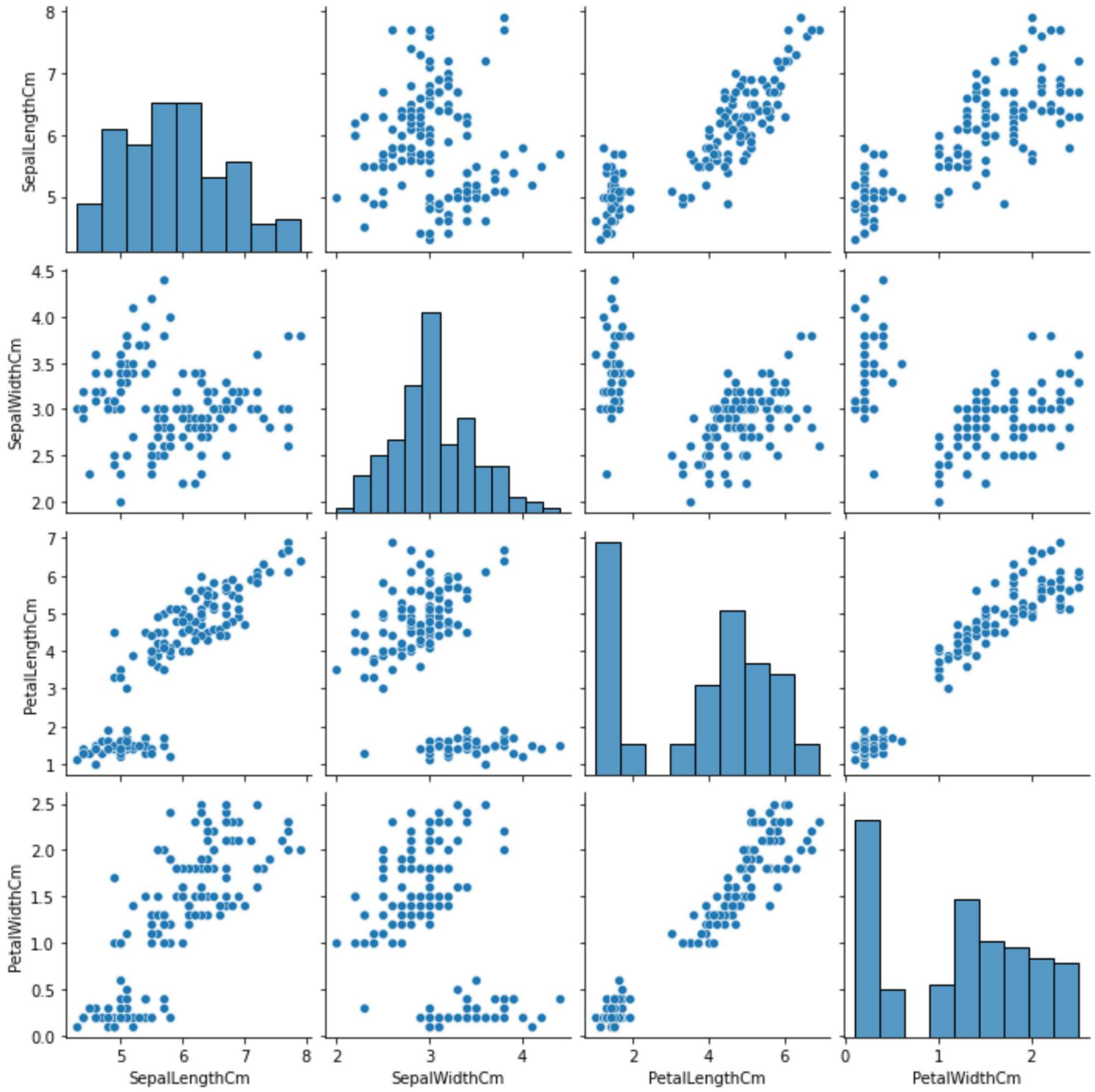
Out[32]: <AxesSubplot:>




In [35]:
```
#for deleting column Id
df = df.drop(columns = ['Id'])
```

In [36]:
```
sns.pairplot(df)
```

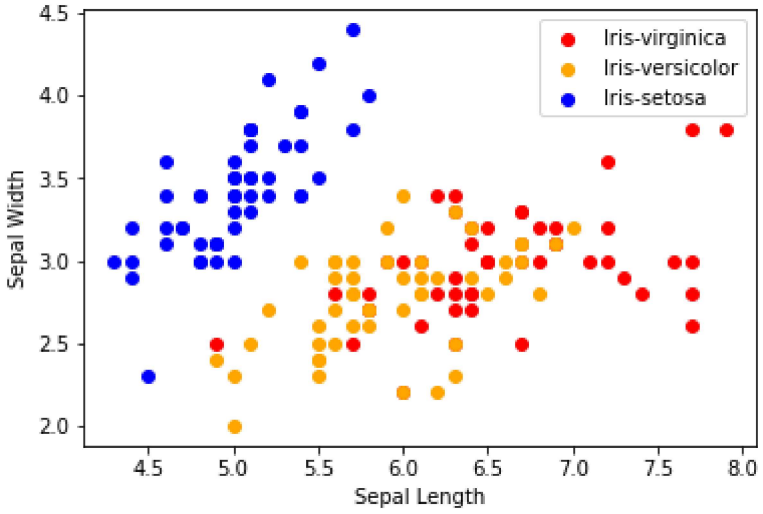
Out[36]: <seaborn.axisgrid.PairGrid at 0xf25ce50>




In [37]:
```
# scatterplot
colors = ['red', 'orange', 'blue']
species = ['Iris-virginica','Iris-versicolor','Iris-setosa']
```

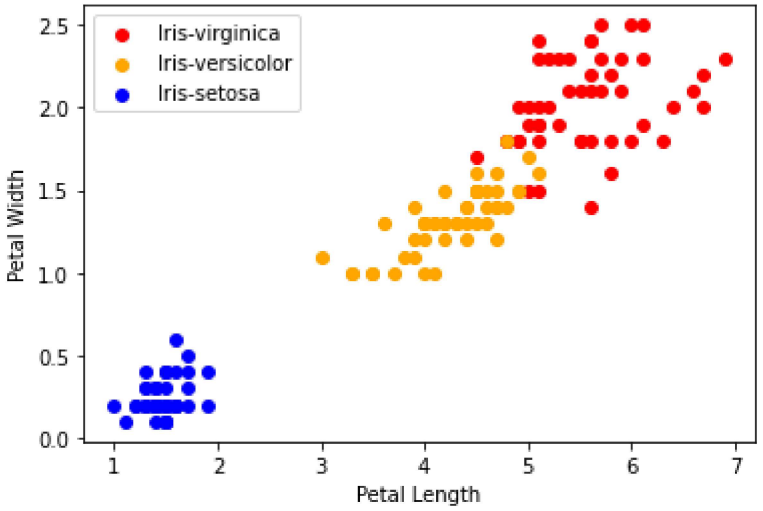
In [38]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
```

Out[38]:
```
<matplotlib.legend.Legend at 0xfb6efd0>
```



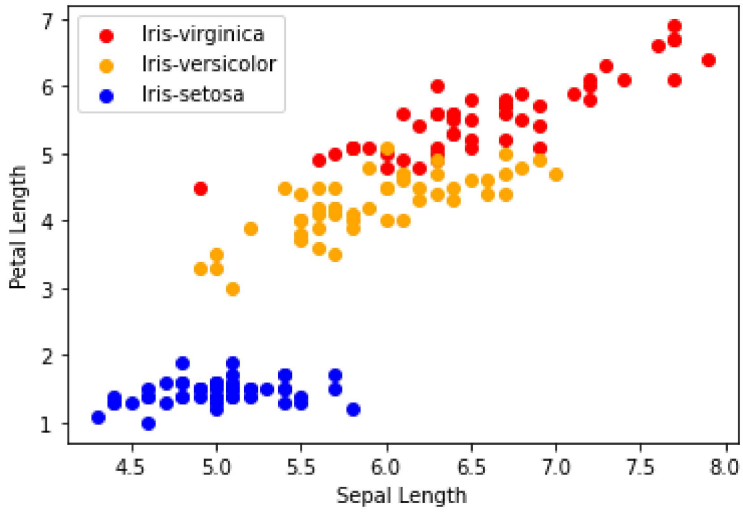

In [39]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['PetalLengthCm'], x['PetalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.legend()
```

Out[39]:
```
<matplotlib.legend.Legend at 0xfdc1100>
```



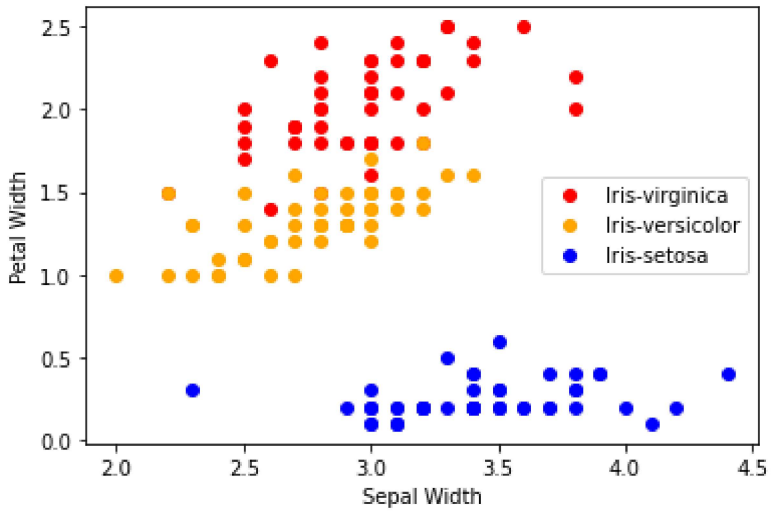

In [40]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], label=species[i]
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.legend()
```

Out[40]:
```
<matplotlib.legend.Legend at 0xfe072f8>
```




In [41]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalWidthCm'], x['PetalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Sepal Width")
plt.ylabel("Petal Width")
plt.legend()
```

Out[41]: <matplotlib.legend.Legend at 0xfe4a610>




In [ ]:

In [ ]:

# Coorelation Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The value is in the range of -1 to 1. If two varibles have high correlation, we can neglect one variable from those two.

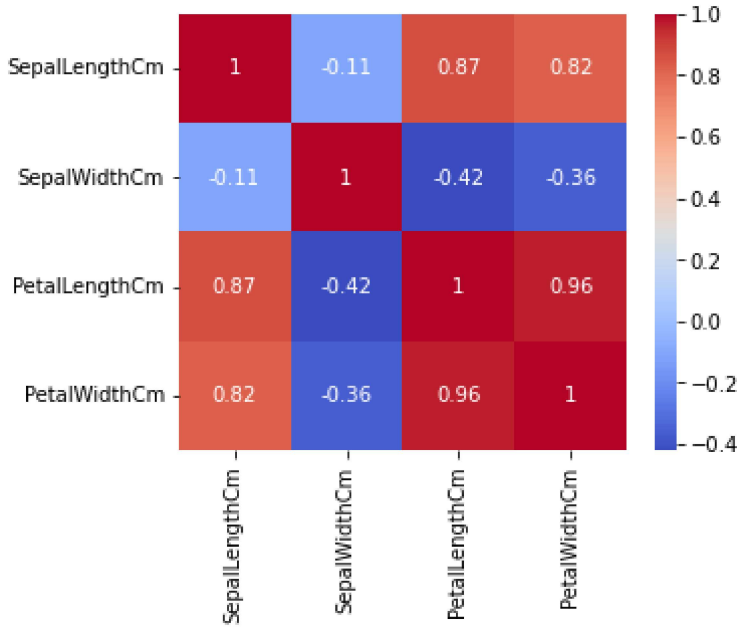
In [42]:
```
df.corr()
```

Out[42]:

| | 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 [43]:
```
corr = df.corr()
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(corr, annot=True, ax=ax, cmap = 'coolwarm')
```

Out[43]:
```
<AxesSubplot:>
```




# Label Encoder

In machine learning, we usually deal with datasets which contains multiple labels in one or more than one columns. These labels can be in the form of words or numbers. Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form

In [44]:
```
#from sklearn import preprocessing
#le = preprocessing.LabelEncoder()
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [45]:
```
df['Species'] = le.fit_transform(df['Species'])
df.head()
```

Out[45]:

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | 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 |

# Model Training

In [75]:
```
from sklearn.model_selection import train_test_split
X = df.drop(columns=['Species'])
Y = df['Species']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.30) # train-70, t

#OR

#from sklearn import model_selection
#te = model_selection.train_test_split
#X = df.drop(columns=['Species'])
#Y = df['Species']
#x_train, x_test, y_train, y_test = te(X, Y, test_size=0.30)''''
```

In [76]:
```
# Logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(max_iter=700)
#OR

#from sklearn import linear_model
#model = linear_model.LogisticRegression()
```

In [77]:
```
# model training
model.fit(x_train, y_train)
```

Out[77]:
```
LogisticRegression(max_iter=700)
```

In [78]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  97.77777777777777
```

In [79]:
```
# knn - k-nearest neighbours
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
```

In [80]:
```
model.fit(x_train, y_train)
```

Out[80]:
```
KNeighborsClassifier()
```

In [81]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  95.55555555555556
```

In [82]:
```
# decision tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

In [83]:
```
model.fit(x_train, y_train)
```

Out[83]:
```
DecisionTreeClassifier()
```

In [84]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  97.77777777777777
```

In [ ]:

In [ ]:

In [ ]: