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:

sepal length in cm
sepal width in cm
petal length in cm
petal width in cm
class:
-- Iris Setosa -- Iris Versicolour -- Iris Virginica
```

Import modules

In [2]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [3]:
```
df = pd.read_csv('Iris.csv')
df.head()
```

Out[3]:

	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 [4]:
```
# delete a column
df = df.drop(columns = ['Id'])
df.head()
```

Out[4]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	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 [5]:
```
# to display stats about data
df.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 basic info about datatype
df.info()
```

```
<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
```

In [7]:
```
# to display no. of samples on each class
df['Species'].value_counts()
```

Out[7]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

Preprocessing the dataset

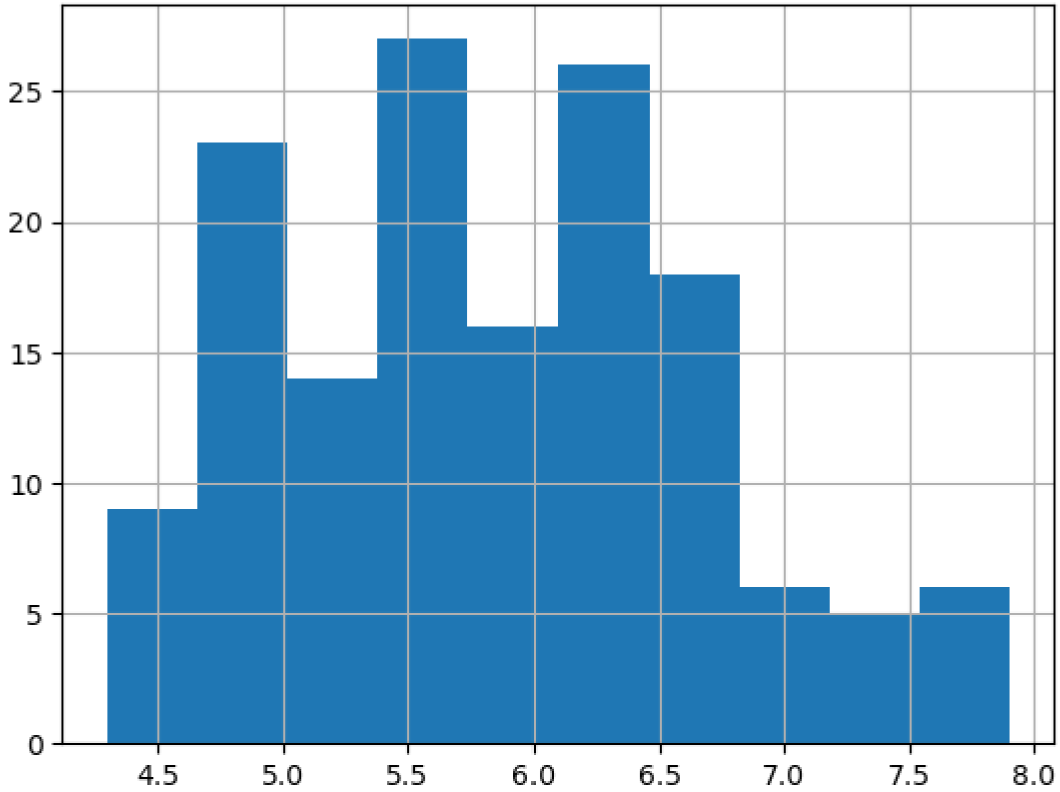
In [8]:
```
# check for null values
df.isnull().sum()
```

Out[8]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Exploratory Data Analysis

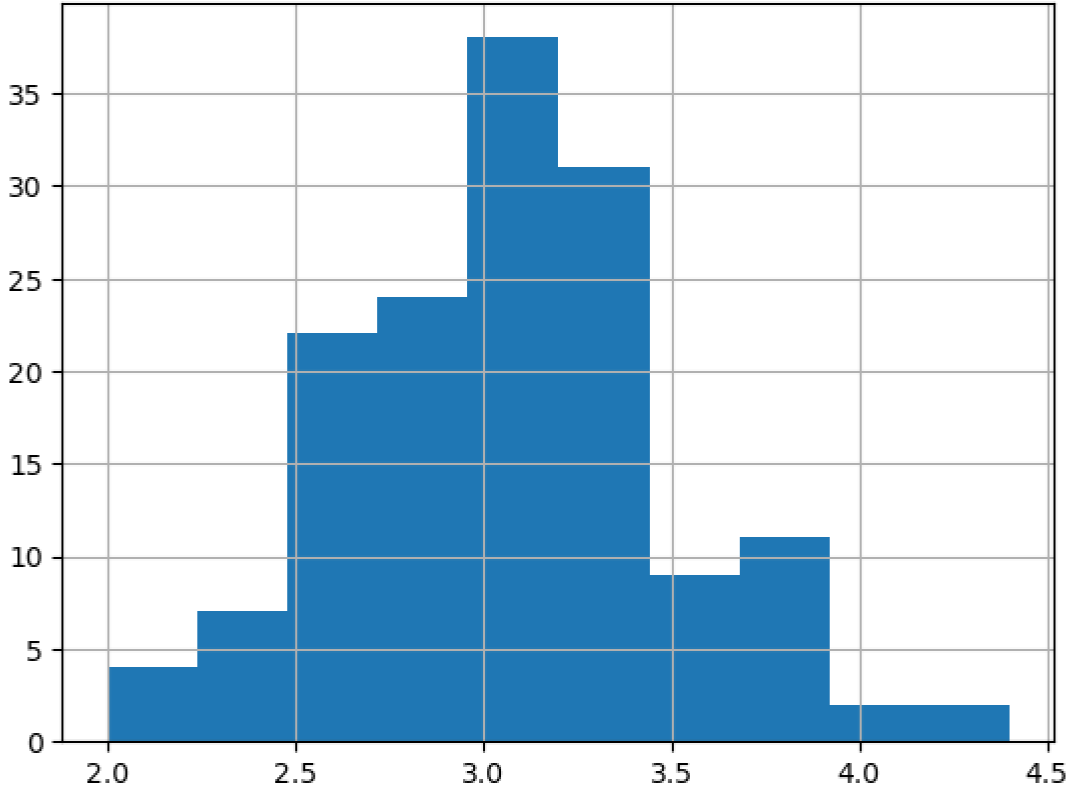
In [9]:
```
# histograms
df['SepalLengthCm'].hist()
```

Out[9]: <Axes: >



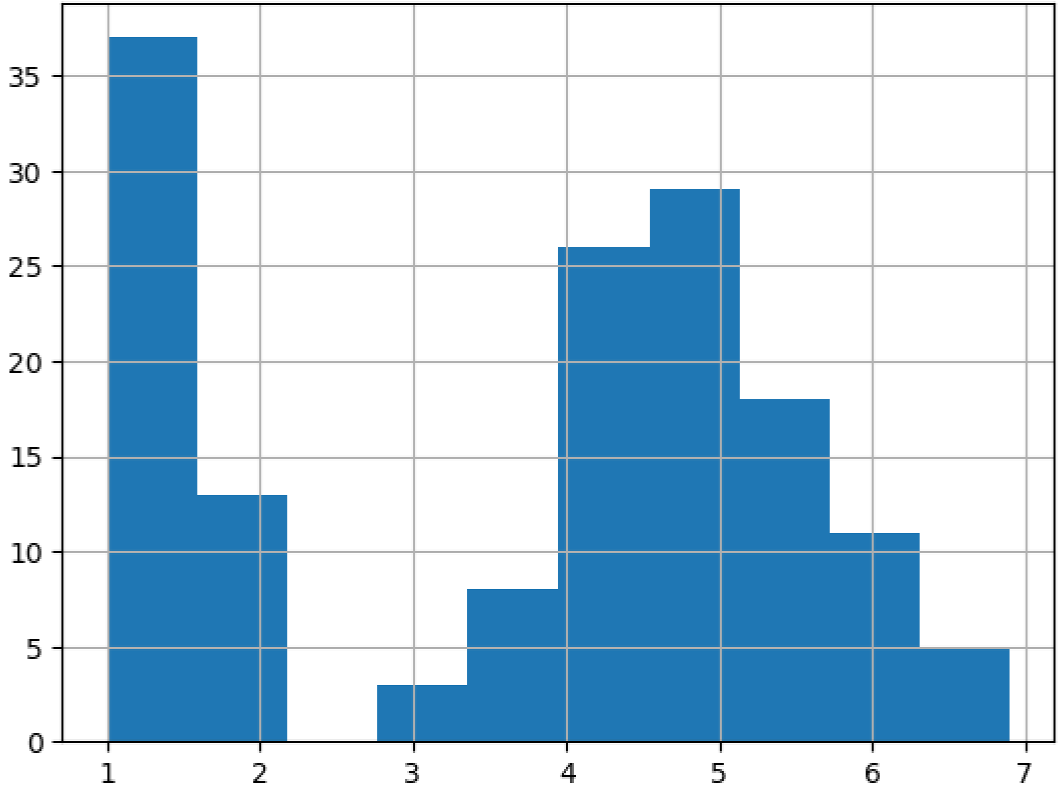
In [11]:
```
df['SepalWidthCm'].hist()
```

Out[11]: <Axes: >



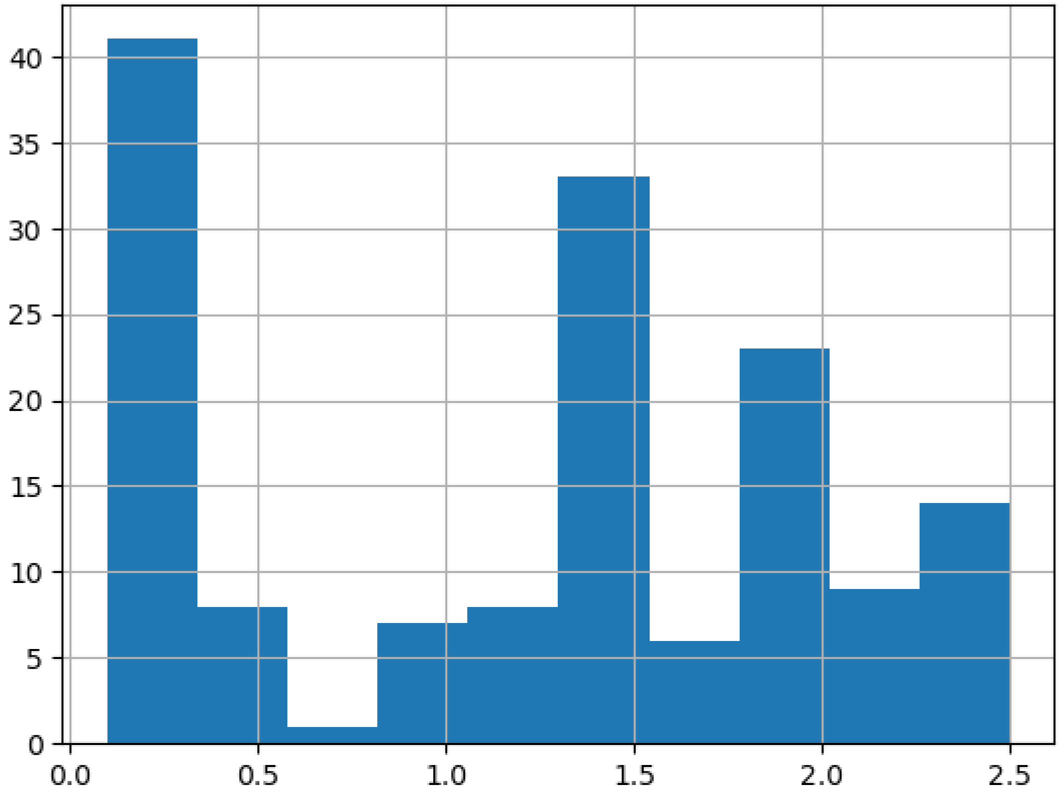
In [12]:
```
df['PetalLengthCm'].hist()
```

Out[12]: <Axes: >



In [13]:
```
df['PetalWidthCm'].hist()
```

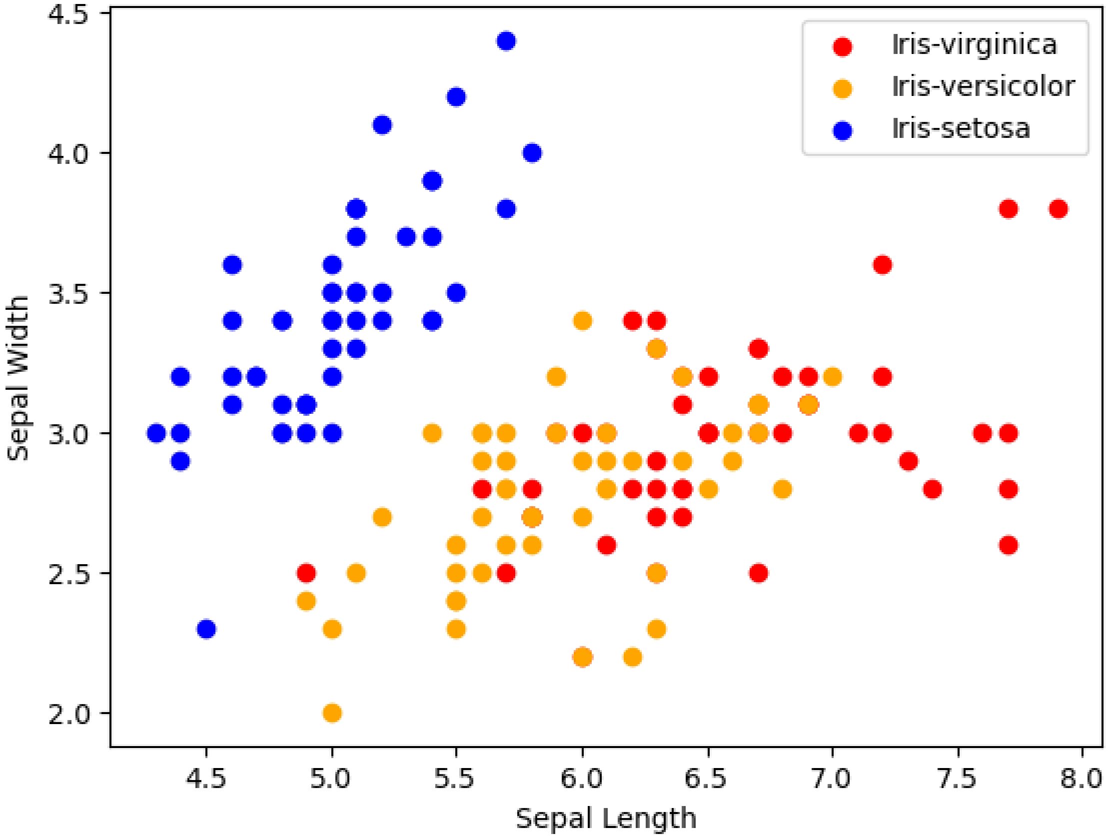
Out[13]: <Axes: >



In [14]:
```
# scatterplot
colors = ['red', 'orange', 'blue']
species = ['Iris-virginica','Iris-versicolor','Iris-setosa']
```

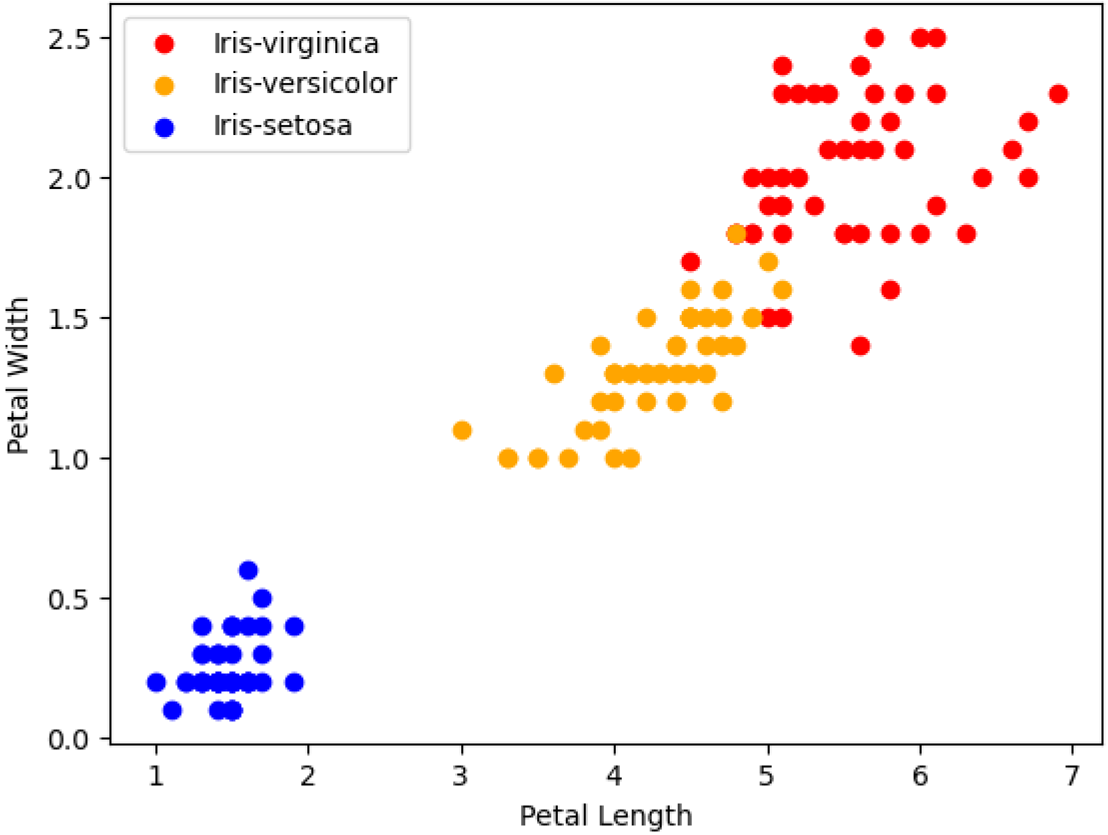
In [15]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x210f9f47d00>
```



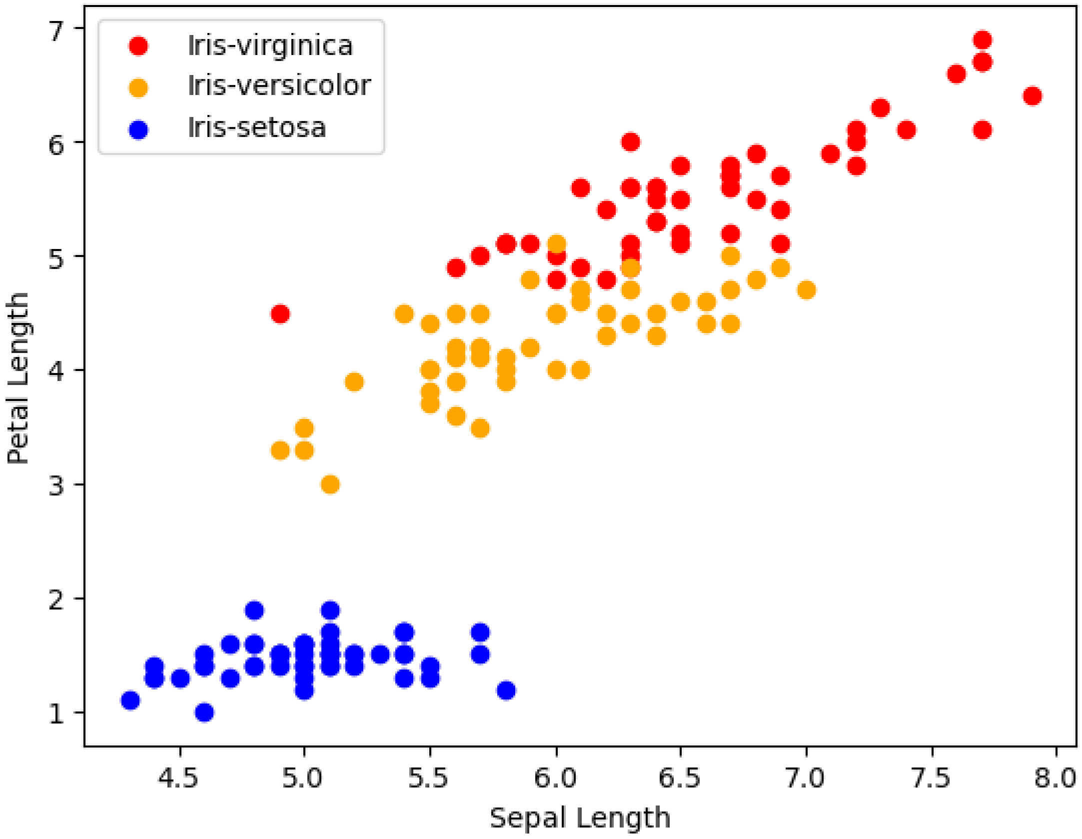
In [16]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['PetalLengthCm'], x['PetalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.legend()
```

Out[16]: <matplotlib.legend.Legend at 0x210fa0db130>



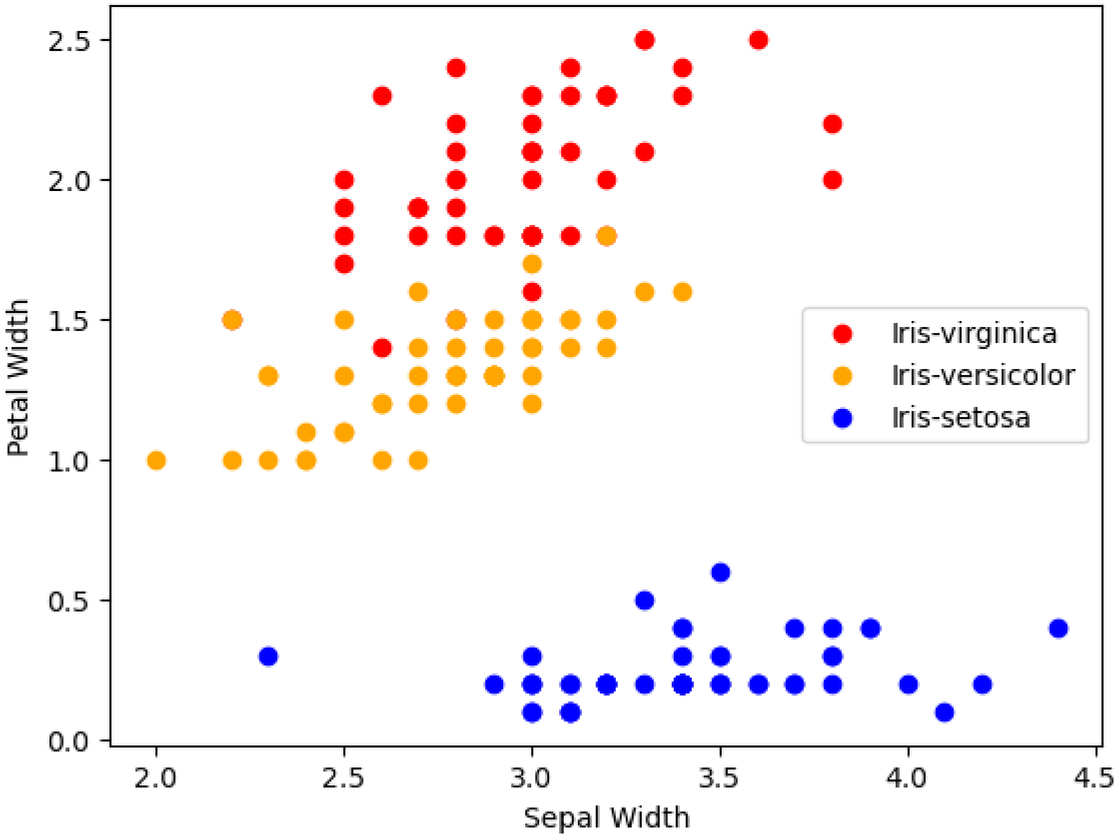
In [17]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x210f9d59210>



In [18]:
```
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[18]:
```
<matplotlib.legend.Legend at 0x210fb25cbb0>
```



In [ ]:

In [ ]:

Coorelation Matrix

In [ ]:

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 [ ]:

In [19]:

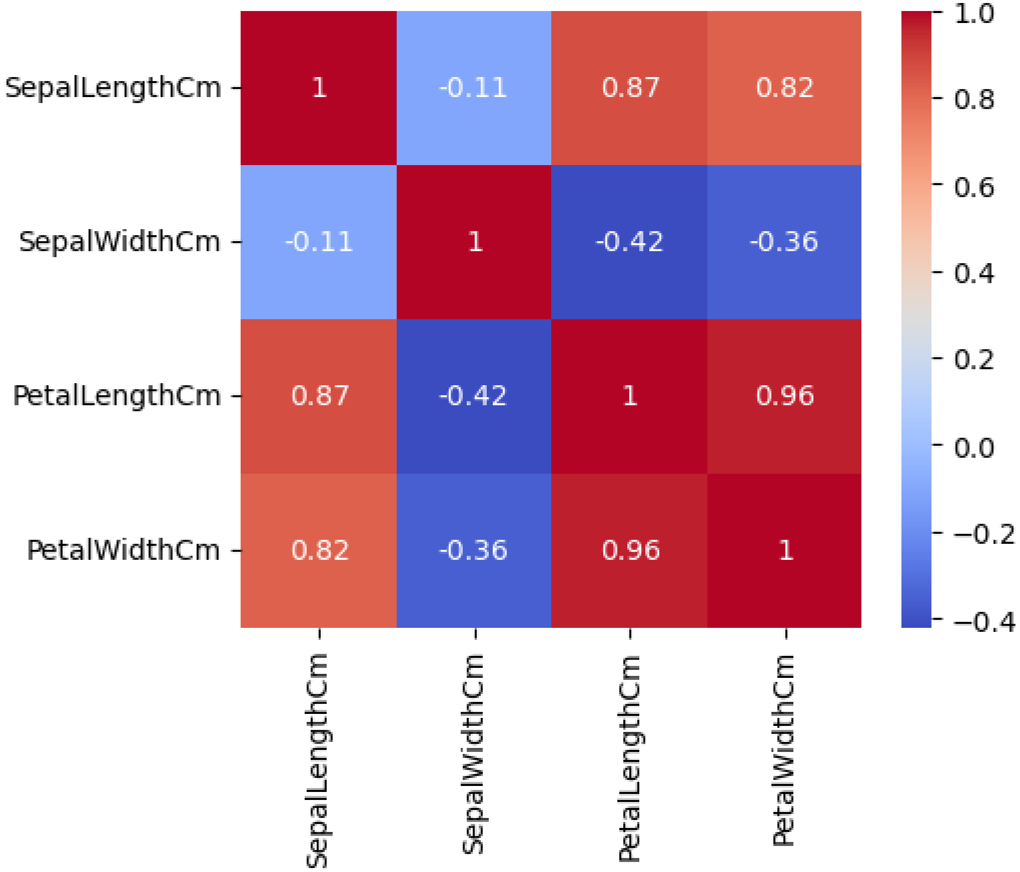
```
df.corr()
```

Out[19]:

	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 [20]:
```
corr = df.corr()
fig, ax = plt.subplots(figsize=(5,4))
sns.heatmap(corr, annot=True, ax=ax, cmap = 'coolwarm')
```

Out[20]: <Axes: >



Label Encoder

In [ ]:

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 [ ]:

In [22]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [23]:
```
df['Species'] = le.fit_transform(df['Species'])
df.head()
```

Out[23]:

	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

In [ ]:

Model Training

In [24]:
```
from sklearn.model_selection import train_test_split
# train - 70
# test - 30
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)
```

In [25]:
```
# logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [26]:
```
# model training
model.fit(x_train, y_train)
```

Out[26]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [27]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  95.55555555555556
```

In [29]:
```
# knn - k-nearest neighbours
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
```

In [30]:
```
model.fit(x_train, y_train)
```

Out[30]:
```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  95.55555555555556
```

In [32]:
```
# decision tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

In [33]:
```
model.fit(x_train, y_train)
```

Out[33]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```
# print metric to get performance
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  95.55555555555556
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
```