About the Project

Iris flower classification is a very popular machine learning project which uses supervised machine learning(uses labelled data) for the classification of species of the Iris flower. The algorithms used are Logistic Regression and KNN

Dataset Information

The data set contains 3 classes of 50 instances each, where each class refers to a type of ***IRIS PLANT*. Using the below four mentioned input attributes we've to predict the class of iris flower.**
ATTRIBUTE INFORMATION:

1.Sepal length in cm
2.Sepal width in cm
3.Petal length in cm
4.Petal width in cm

CLASS:
__*Iris Setosa*__
__*Iris Versicolour*__
__*Iris Virginica*__

Importing modules

In [1]:
```
import pandas as pd             #Pandas is to read the dataset of various file formats such
 as comma-separated values, JSON, Excel, etc.
import numpy as np              #NumPy is used for working with arrays
import os                       #For adding files
import matplotlib.pyplot as plt #Matplotlib is to visualise data in form of graphs
import seaborn as sns           #Seaborn is built on top of matplotlib. It is used for data
 visualization and exploratory data analysis
```

Uploading dataset

In [2]:
```
df=pd.read_csv('Iris.csv') #The dataset is in the CSV(Comma Separated Value) format, here df
is dataframe in which the CSV file gets stored
```

In [3]:
```
#Displaying the first five rows of the dataset
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]:
```
#Displaying the last five rows of the dataset
df.tail()
```

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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 [5]:
```
#Here the Id column is unwanted
df = df.drop(columns = ['Id']) #To delete the Id column
df.head() #Displaying again the first 5 rows
```

Out[5]:

	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 [6]:
```
#To display the statistics about the dataset
df.describe()
```

Out[6]:

	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 [7]:
```
#To display basic information about the dataset
df.info()
```

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

In [8]:
```
#To display number of samples in each class.
df['Species'].value_counts()
#Output shows each class containing 50 samples
```

Out[8]:
```
Iris-setosa        50
Iris-virginica     50
Iris-versicolor    50
Name: Species, dtype: int64
```

Data Preprocessing

In [9]:
```
#To check if any null values are present
df.isnull().sum()
```

Out[9]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

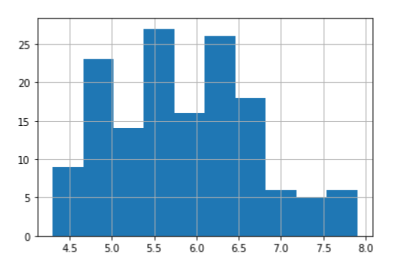
Exploratory Data Analysis

Exploratory data analysis (EDA) is used to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

Plotting Histogram

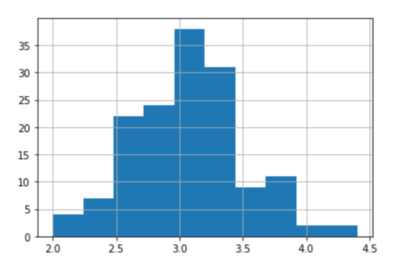
In [10]:
```
df['SepalLengthCm'].hist()
```

Out[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27813d49648>
```



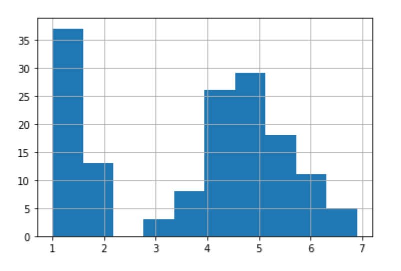
In [11]:
```
df['SepalWidthCm'].hist()
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27814e0b788>
```



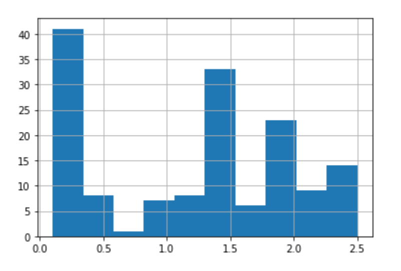
In [12]:
```
df['PetalLengthCm'].hist()
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27814eb6f48>
```



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

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x27814f4a0c8>
```

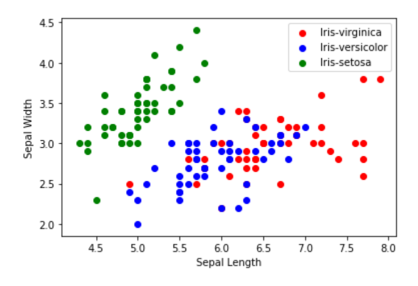


Scatterplot

In [14]:
```
#Assigning colors to the classes respectively
colors = ['red', 'blue', 'green']
species = ['Iris-virginica','Iris-versicolor','Iris-setosa']
```

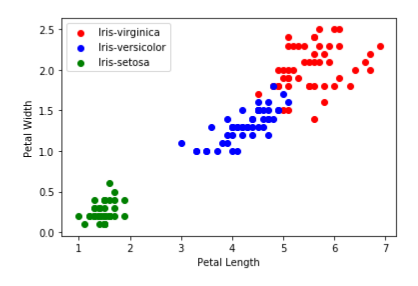
In [15]:
```
#Iterating the classes
for i in range(3):
    x = df[df['Species'] == species[i]]
    #Plotting the data
    plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c = colors[i], label=species[i])
#X-axix
plt.xlabel("Sepal Length")
#Y-axis
plt.ylabel("Sepal Width")
#Index for the graph
plt.legend()
```

Out[15]:
```
<matplotlib.legend.Legend at 0x27814fbb548>
```



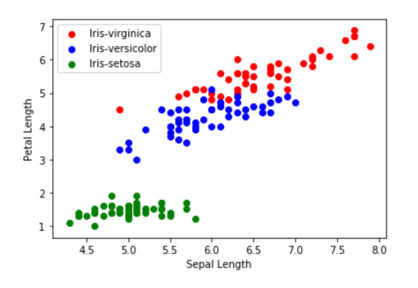
In [16]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['PetalLengthCm'], x['PetalWidthCm'], c = colors[i], label=species[i])
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.legend()
```

Out[16]:
```
<matplotlib.legend.Legend at 0x27815060388>
```



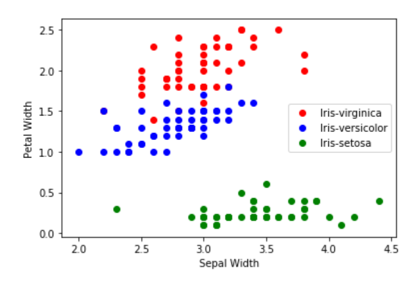
In [17]:
```
for i in range(3):
    x = df[df['Species'] == species[i]]
    plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], label=species[i])
plt.xlabel("Sepal Length")
plt.ylabel("Petal Length")
plt.legend()
```

Out[17]:
```
<matplotlib.legend.Legend at 0x27814f13608>
```



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 0x27815157e48>
```



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.

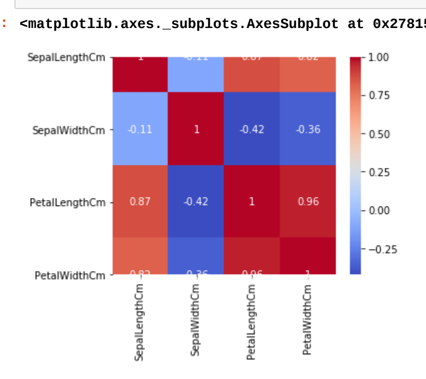
In [19]:
```
df.corr()
```

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]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x278151d3e48>
```



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 [21]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [22]:
```
df['Species'] = le.fit_transform(df['Species'])
df.head()
```

Out[22]:

	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

Training and testing the model

In [23]:
```
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)
```

Logistic Regression

In [24]:
```
# logistic regression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
```

In [25]:
```
# model training
model.fit(x_train, y_train)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: FutureWarning:
Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to sile
nce this warning.
  "this warning.", FutureWarning)
```

Out[25]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [26]:
```
# print answer
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  100.0
```

knn - k nearest neighbours

In [27]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
```

In [28]:
```
model.fit(x_train, y_train)
```

Out[28]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [29]:
```
# print answer
print("Accuracy: ",model.score(x_test, y_test) * 100)
```

```
Accuracy:  100.0
```

In [ ]:

In [ ]: