IRIS FLOWER CLASSIFICATION

Iris flower has three species; **setosa, versicolor, and virginica**

Which differs according to their measurements. Now assume that you have the measurements of the iris flowers according to their species, and here your task is to train a machine learning model that can learn from the measurements of the iris species and classify them.

Although the Scikit-learn library provides a dataset for iris flower classification, you can also download the same dataset from here for the task of iris flower classification with Machine Learning.

Import the required modules

In [51]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
%matplotlib inline

from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

Loading the IRIS dataset

In [14]:

```
data = pd.read_csv("IRIS.csv")
data.head()
```

Out[14]:

	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 [15]:

```
data.tail()
```

Out[15]:

	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 [16]:

```
#displaying the stastictics of data
data.describe()
```

Out[16]:

	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 [17]:

```
#basic info about datatype
data.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 [20]:

```
#info about columns of data
data.columns
```

Out[20]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [22]:

```
#to know no.of samples in each class(species varients)
data['species'].value_counts()
```

Out[22]:

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

Data Preprocessing

In [23]:

```
#to know any null values in dataset
data.isnull().sum()
#glad that i dont't have any null values
```

Out[23]:

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

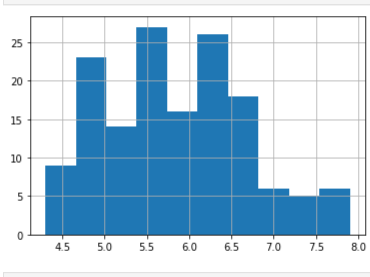
Exploratory Data Analysis

Here, I'll visualize the data in the form of graphs.

1. Histogram

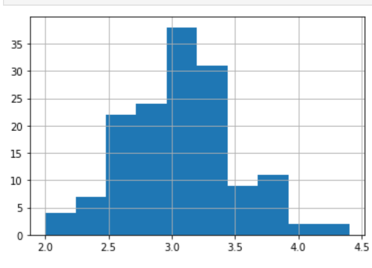
In [26]:

```
data['sepal_length'].hist()
plt.show()
```



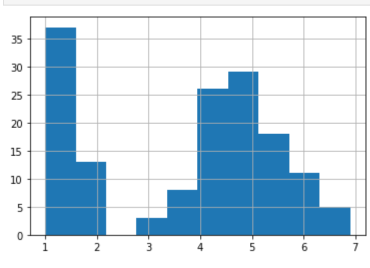
In [27]:

```
data['sepal_width'].hist()
plt.show()
```



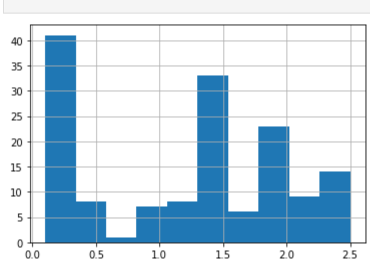
In [28]:

```
data['petal_length'].hist()
plt.show()
```



In [29]:

```
data['petal_width'].hist()
plt.show()
```



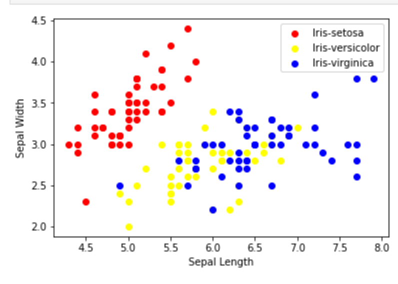
1. Scatterplot

In [36]:

```
colors =['red','yellow', 'blue']
species = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

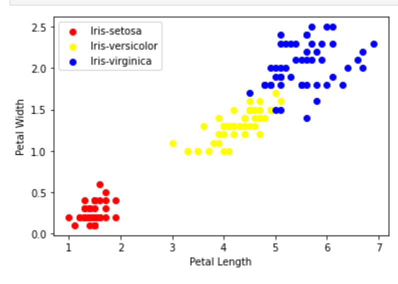
In [39]:

```
for i in range(3):
    x = data[data['species'] == species[i]]
    plt.scatter(x['sepal_length'], x['sepal_width'], c = colors[i], label = species[i])
plt.xlabel("Sepal Length")
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```



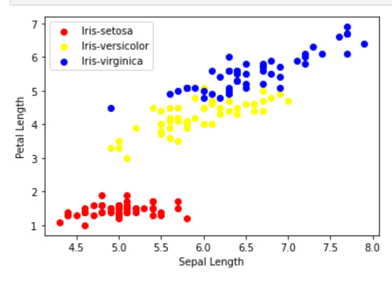
In [41]:

```
for i in range(3):
    x = data[data['species'] == species[i]]
    plt.scatter(x['petal_length'], x['petal_width'], c = colors[i], label = species[i])
plt.xlabel("Petal Length")
plt.ylabel('Petal Width')
plt.legend()
plt.show()
```



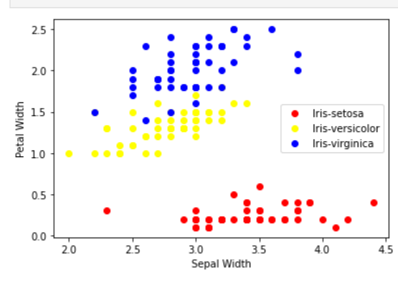
In [42]:

```
for i in range(3):
    x = data[data['species'] == species[i]]
    plt.scatter(x['sepal_length'], x['petal_length'], c = colors[i], label = species[i])
plt.xlabel("Sepal Length")
plt.ylabel('Petal Length')
plt.legend()
plt.show()
```



In [43]:

```
for i in range(3):
    x = data[data['species'] == species[i]]
    plt.scatter(x['sepal_width'], x['petal_width'], c = colors[i], label = species[i])
plt.xlabel("Sepal Width")
plt.ylabel('Petal Width')
plt.legend()
plt.show()
```



Coorelation Matrix

Coorelation matrix is a table which shows the coorelation coefficients between variables.

Each cell in the table shows the coorelation between variables.

Value range of -1 to +1.

If two variables have high coorelation, can neglect one of those.

In [45]:

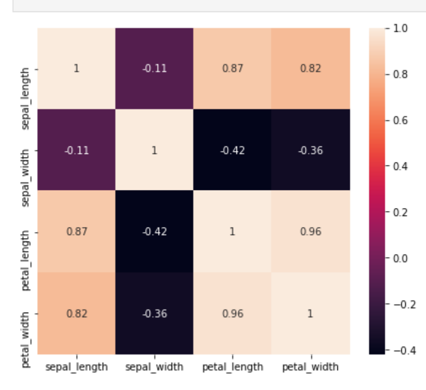
```
coor = data.corr()
coor
```

Out[45]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

In [56]:

```
fig, ax = plt.subplots(figsize =(7,6))
sns.heatmap(coor, annot = True, ax = ax)
plt.show()
```



Label Encoder

converts Categorical into Numeric label

In [59]:

```
from sklearn.preprocessing import LabelEncoder
lab = LabelEncoder()
```

Iris-setosa as 0

Iris-versicolor as 1

Iris-virginica as 2

Name: species, dtype: int64

In [63]:

```
data['species'] = lab.fit_transform(data['species'])
data.head()
```

Out[63]:

	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 [64]:

```
data.tail()
```

Out[64]:

	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 [68]:

```
data.columns
```

Out[68]:

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [85]:

```
from sklearn.model_selection import train_test_split
#test = 30 and train = 70
X = data.drop(columns=['species'])
Y = data['species']
x_train,  x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.30)
```

In [86]:

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

In [87]:

```
#model training
model.fit(x_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[87]:

```
LogisticRegression()
```

In [123…

```
#printing the metric to know the performace of the model trained by LogisticRegression()
print('Accuracy of the model:', model.score(x_test, y_test) * 100)
```

```
Accuracy of the model: 84.44444444444444
```

In [95]:

```
#knn - k nearest neighbours

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
```

In [97]:

```
model.fit(x_train, y_train)
```

Out[97]:

```
KNeighborsClassifier()
```

In [117…

```
#printing the metric to know the performace of the model trained by KNeighborsClassifier()
print('Accuracy of the model:', model.score(x_test, y_test) * 100)
```

```
Accuracy of the model: 84.44444444444444
```

In [109…

```
from sklearn import tree
```

In [114…

```
#descision tree
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
```

In [115…

```
model.fit(x_train, y_train)
```

Out[115…

```
DecisionTreeClassifier()
```

In [125…

```
#printing the metric to know the performace of the model trained by DecisionTreeClassifier()
print('Accuracy of the model:', model.score(x_test, y_test) * 100)
```

```
Accuracy of the model: 84.44444444444444
```