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

```
dataset=pd.read_csv('/content/drive/MyDrive/IRIS.csv')
```

```
dataset.head()
```

	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

```
dataset.describe()
```

	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

```
dataset.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
```

```
dataset['species'].value_counts()
```

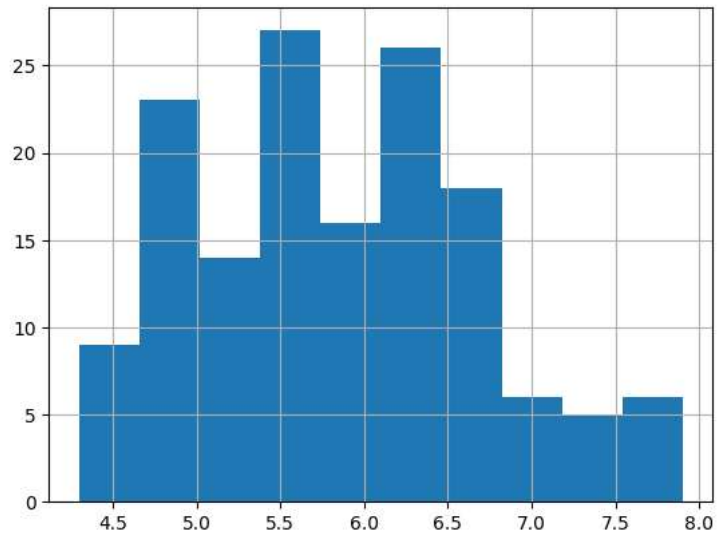
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

```
#to check whether the data have any null values
dataset.isnull().sum()
```

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

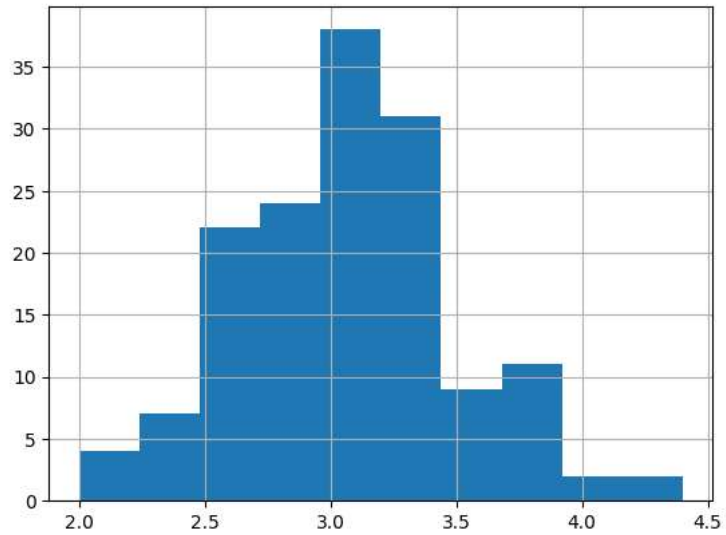
```
dataset['sepal_length'].hist()
```

```
<Axes: >
```



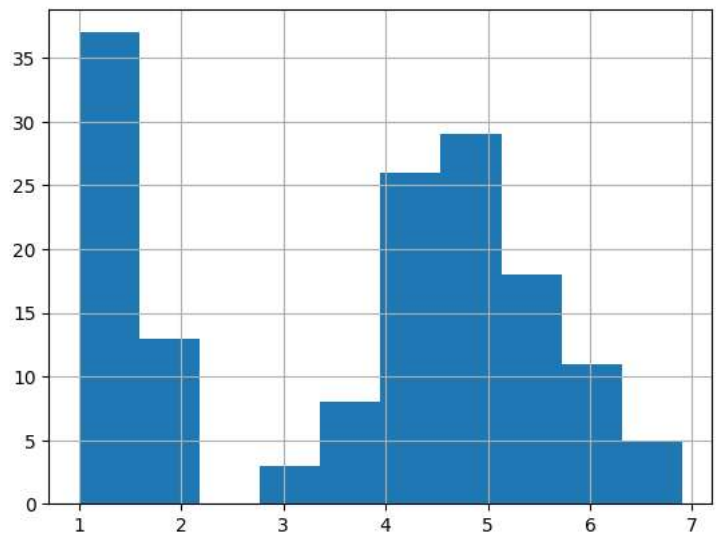

```
dataset['sepal_width'].hist()
```

```
<Axes: >
```



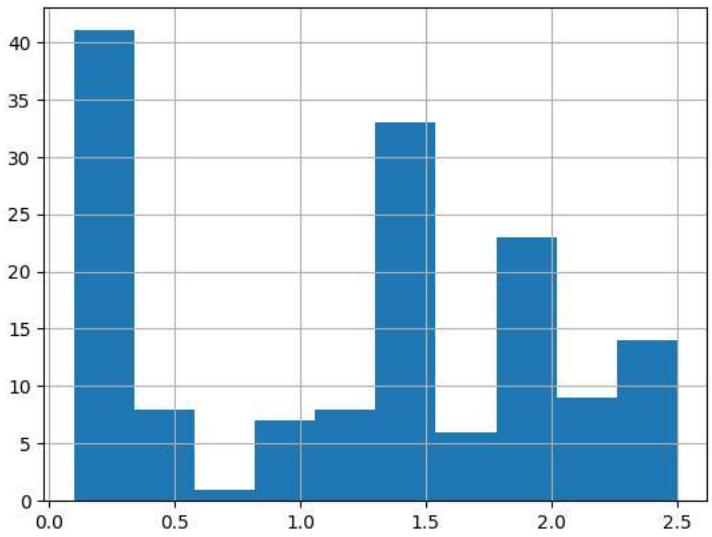

```
dataset['petal_length'].hist()
```

```
<Axes: >
```



```
dataset['petal_width'].hist()
```

```
<Axes: >
```

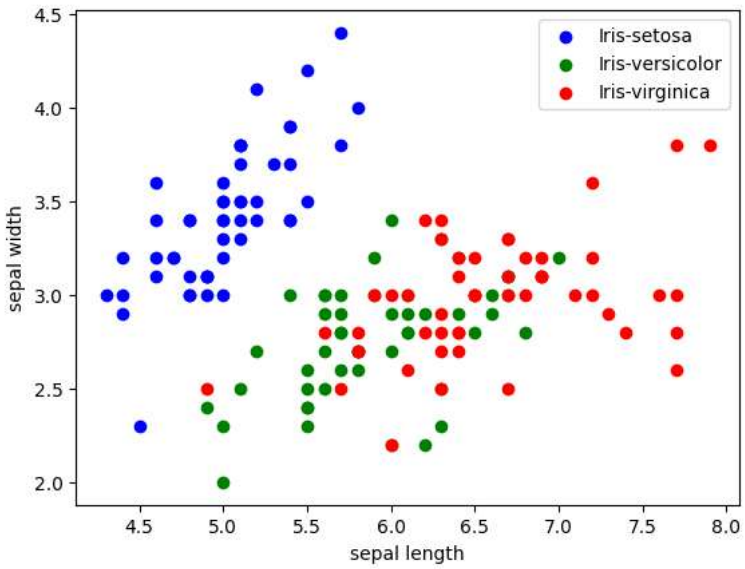



```
#scaterplot
colors=['blue','green','red']
species=['Iris-setosa','Iris-versicolor','Iris-virginica']


for i in range(3):
  x=dataset[dataset['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()
```

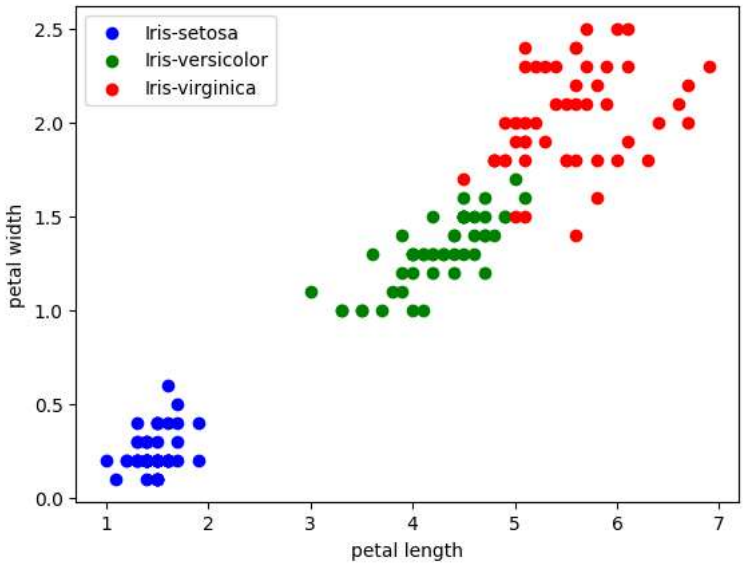
```
<matplotlib.legend.Legend at 0x7f71cd315030>
```




```
for i in range(3):
  x=dataset[dataset['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()
```

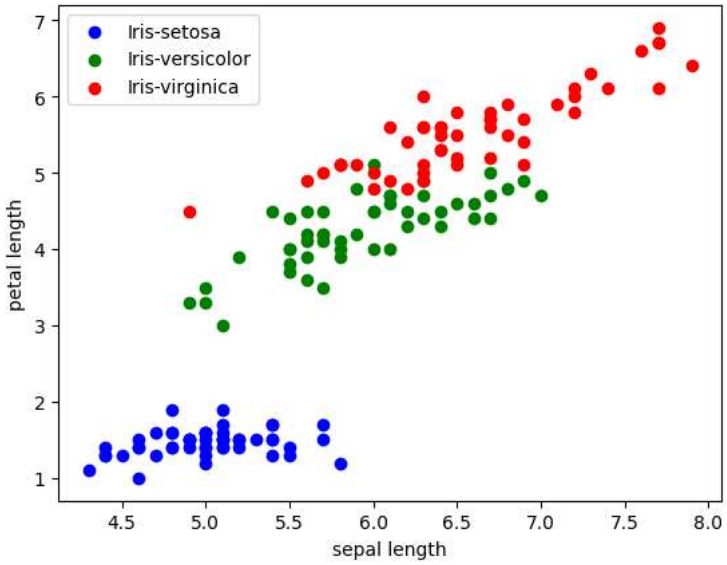
```
<matplotlib.legend.Legend at 0x7f71cd303e50>
```




```
for i in range(3):
  x=dataset[dataset['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()
```

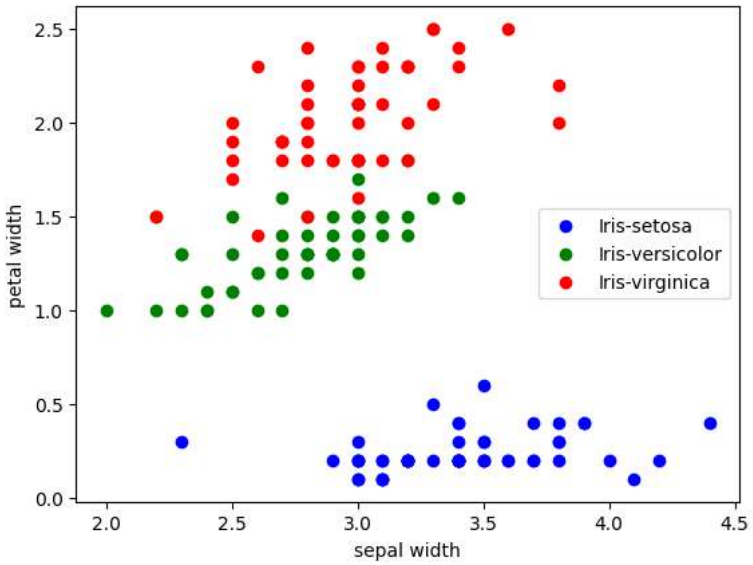
```
<matplotlib.legend.Legend at 0x7f71cd1cb280>
```




```
for i in range(3):
  x=dataset[dataset['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()
```

```
<matplotlib.legend.Legend at 0x7f71cd252c80>
```



CORRLETION MATRIX

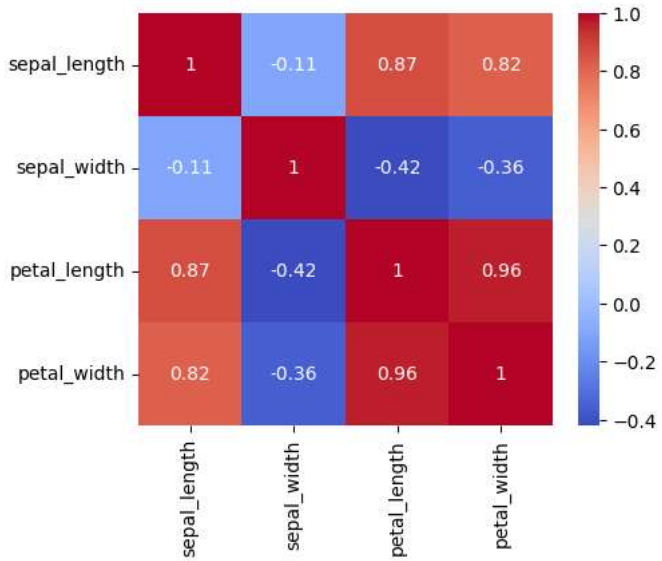
```
dataset.corr()
```

```
<ipython-input-17-c187c74d1e71>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  dataset.corr()
```

	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

```
corr=dataset.corr()
fig,ax=plt.subplots(figsize=(5,4))
sns.heatmap(corr,annot=True, ax=ax, cmap='coolwarm')
```

```
<ipython-input-18-adfd7af69261>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version
  corr=dataset.corr()
<Axes: >
```



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

```
dataset['species']=le.fit_transform(dataset['species'])
dataset.head()
```

	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

```
from sklearn.model_selection import train_test_split
#train 70%
#test 30%
X=dataset.drop(columns=['species'])
y=dataset['species']
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.30)
```

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

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

```
▾ LogisticRegression
LogisticRegression()
```

```
#print metric to get performance
print('accuracy: ',model.score(X_test,y_test)*100)
```

```
accuracy:  95.55555555555556
```

```
#knn-K nearest neighbors
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier()
```

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

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
#print metric to get performance
print('accuracy: ',model.score(X_test,y_test)*100)
```

```
accuracy:  95.55555555555556
```

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

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score,roc_curve
```

```
y_pred_train=model.predict(X_train)
```

```
accuracy_score(y_train,y_pred_train)
confusion_matrix(y_train,y_pred_train)
print(classification_report(y_train,y_pred_train))
```

```
                   precision    recall  f1-score   support

                0       1.00      1.00      1.00        34
                1       1.00      1.00      1.00        36
                2       1.00      1.00      1.00        35

         accuracy                           1.00       105
        macro avg       1.00      1.00      1.00       105
     weighted avg       1.00      1.00      1.00       105
```

```
y_pred_test=model.predict(X_test)
```

```
accuracy_score(y_test,y_pred_test)
confusion_matrix(y_test,y_pred_test)
print(classification_report(y_test,y_pred_test))
```

```
                   precision    recall  f1-score   support

                0       1.00      1.00      1.00        16
                1       0.92      0.86      0.89        14
                2       0.88      0.93      0.90        15

         accuracy                           0.93        45
        macro avg       0.93      0.93      0.93        45
     weighted avg       0.93      0.93      0.93        45
```

```
#print metric to get performance
print('accuracy: ',model.score(X_test,y_test)*100)
```

```
     accuracy:  93.33333333333333
```