Name of the Intern : SHAHNAWAZ ALAM

Data Science Internship

Oasis Infobyte

Task 1: Iris Flower Classification using Machine Learning

Batch- April Phase 1 OIBSIP

Steps to build a ML Model:

```
1.Import dataset
2.Visualizing the dataset
3.Data preparation
4.Training the algorithms
5.Making Prediction
6.Model Evolution
```

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
```

1. Importing Dataset

In [34]:

```
#loading data
print("Importing data...")
df=pd.read_csv(r"C:\Users\md naiyer azam\Desktop\OIBSIP_Internship\Data Science\Iris.csv")
print("Sucessfully imported.")
```

```
Importing data...
Sucessfully imported.
```

In [35]:

```
df.head()  # #to check sucessful importation of dataset.
```

Out[35]:

	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]:

```
df.head(10)
```

Out[4]:

	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
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [5]:

```
df.tail()
```

Out[5]:

	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 [6]:

```
df.shape   ##to get no. of rows and column(rows,column)
```

Out[6]:

```
(150, 6)
```

In [36]:

```
#info of data
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: 7.2+ KB
```

In [37]:

```
#description of data
df.describe()
```

Out[37]:

	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

From the above discription count tells that all the 4 features have 150 rows and from Mean we can say that sepal is larger than petal.

In [38]:

```
df['Species'].value_counts()
```

Out[38]:

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

we can observe all three classes are equally distributed in terms of the number of counts of each class.

In [41]:

```
#Create 3 DataFrame for each Species
setosa=df[df['Species']=='Iris-setosa']
versicolor =df[df['Species']=='Iris-versicolor']
virginica =df[df['Species']=='Iris-virginica']

print("SETOSA:\n",setosa.describe())
print("\nVERSICOLOR:\n",versicolor.describe())
print("\nVIRGINICA:\n",virginica.describe())
```

```
SETOSA:
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count  50.00000       50.00000     50.000000      50.000000      50.00000
mean   25.50000        5.00600      3.418000       1.464000       0.24400
std    14.57738        0.35249      0.381024       0.173511       0.10721
min     1.00000        4.30000      2.300000       1.000000       0.10000
25%    13.25000        4.80000      3.125000       1.400000       0.20000
50%    25.50000        5.00000      3.400000       1.500000       0.20000
75%    37.75000        5.20000      3.675000       1.575000       0.30000
max    50.00000        5.80000      4.400000       1.900000       0.60000

VERSICOLOR:
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000      50.000000     50.000000      50.000000     50.000000
mean    75.50000       5.936000      2.770000       4.260000      1.326000
std     14.57738       0.516171      0.313798       0.469911      0.197753
min     51.00000       4.900000      2.000000       3.000000      1.000000
25%     63.25000       5.600000      2.525000       4.000000      1.200000
50%     75.50000       5.900000      2.800000       4.350000      1.300000
75%     87.75000       6.300000      3.000000       4.600000      1.500000
max    100.00000       7.000000      3.400000       5.100000      1.800000

VIRGINICA:
               Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count   50.00000       50.00000     50.000000      50.000000      50.00000
mean   125.50000        6.58800      2.974000       5.552000       2.02600
std     14.57738        0.63588      0.322497       0.551895       0.27465
min    101.00000        4.90000      2.200000       4.500000       1.40000
25%    113.25000        6.22500      2.800000       5.100000       1.80000
50%    125.50000        6.50000      3.000000       5.550000       2.00000
75%    137.75000        6.90000      3.175000       5.875000       2.30000
max    150.00000        7.90000      3.800000       6.900000       2.50000
```

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [8]:

```
df.dtypes
```

Out[8]:

```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [9]:

```
data=df.groupby('Species')
```

In [10]:

```
data.head()
```

Out[10]:

	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
50	51	7.0	3.2	4.7	1.4	Iris-versicolor
51	52	6.4	3.2	4.5	1.5	Iris-versicolor
52	53	6.9	3.1	4.9	1.5	Iris-versicolor
53	54	5.5	2.3	4.0	1.3	Iris-versicolor
54	55	6.5	2.8	4.6	1.5	Iris-versicolor
100	101	6.3	3.3	6.0	2.5	Iris-virginica
101	102	5.8	2.7	5.1	1.9	Iris-virginica
102	103	7.1	3.0	5.9	2.1	Iris-virginica
103	104	6.3	2.9	5.6	1.8	Iris-virginica
104	105	6.5	3.0	5.8	2.2	Iris-virginica

In [11]:

```
df['Species'].unique()
```

Out[11]:

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [12]:

```
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: 7.2+ KB
```

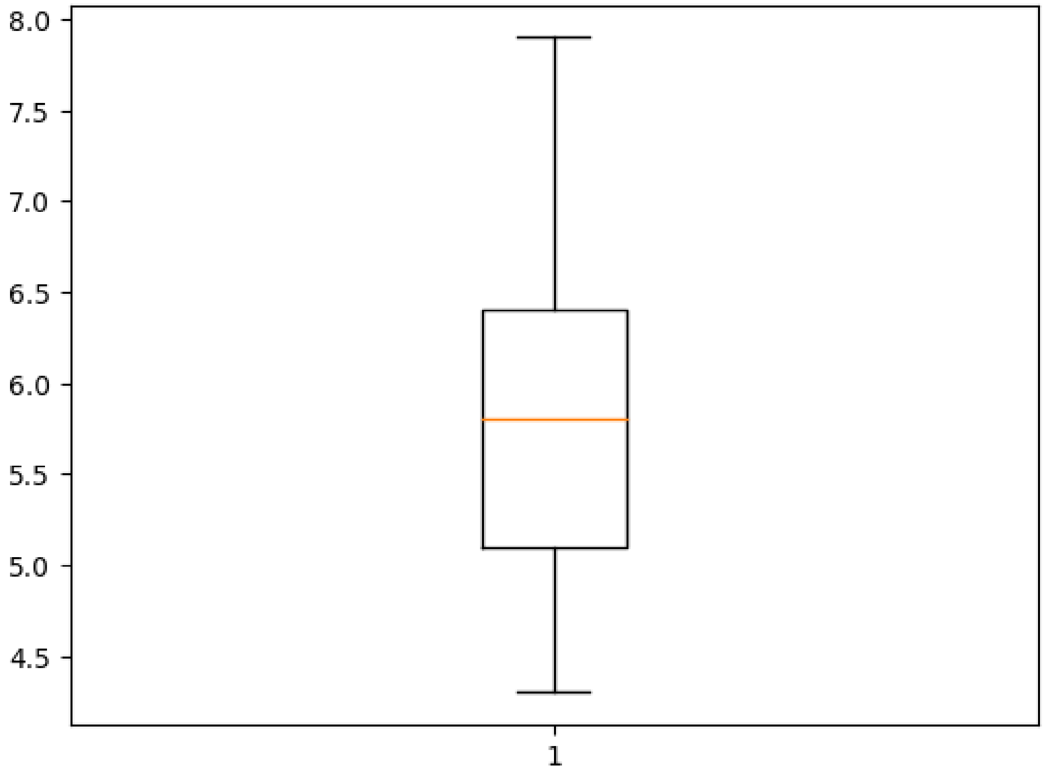
2. visualizing the dataset

In [13]:

```
plt.boxplot(df['SepalLengthCm'])
```

Out[13]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x222dadd4f10>,
  <matplotlib.lines.Line2D at 0x222dadd51b0>],
 'caps': [<matplotlib.lines.Line2D at 0x222dadd5450>,
  <matplotlib.lines.Line2D at 0x222dadd56f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x222dadd4c70>],
 'medians': [<matplotlib.lines.Line2D at 0x222dadd5990>],
 'fliers': [<matplotlib.lines.Line2D at 0x222dadd5c30>],
 'means': []}
```

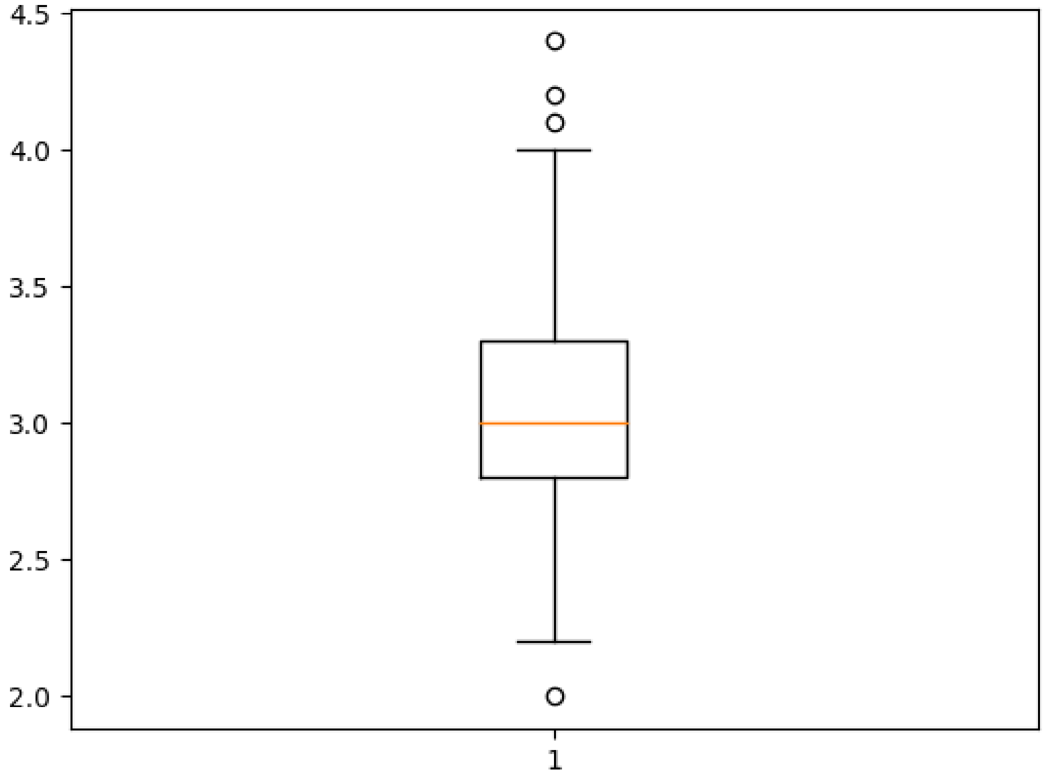


In [14]:

```
plt.boxplot(df['SepalWidthCm'])
```

Out[14]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x222db03c790>,
  <matplotlib.lines.Line2D at 0x222db03ca30>],
 'caps': [<matplotlib.lines.Line2D at 0x222db03cbb0>,
  <matplotlib.lines.Line2D at 0x222db03ce50>],
 'boxes': [<matplotlib.lines.Line2D at 0x222db03c4f0>],
 'medians': [<matplotlib.lines.Line2D at 0x222db03d0f0>],
 'fliers': [<matplotlib.lines.Line2D at 0x222db03d390>],
 'means': []}
```

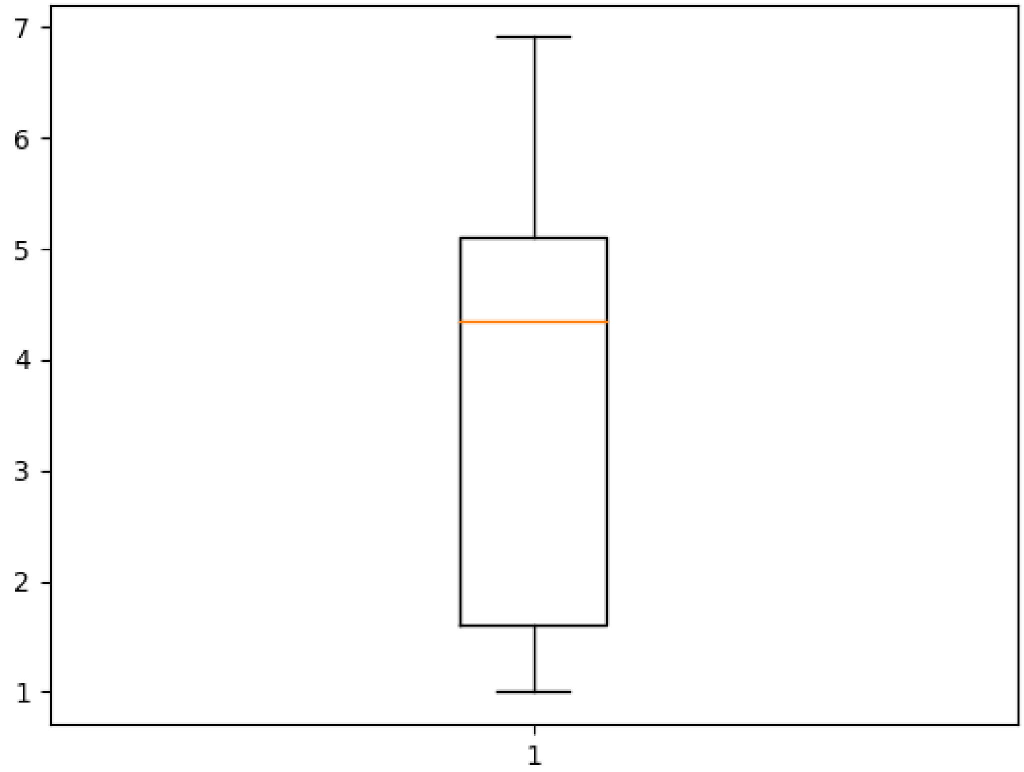


In [15]:

```
plt.boxplot(df['PetalLengthCm'])
```

Out[15]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x222dd093100>,
  <matplotlib.lines.Line2D at 0x222dd0933a0>],
 'caps': [<matplotlib.lines.Line2D at 0x222dd093640>,
  <matplotlib.lines.Line2D at 0x222dd0938e0>],
 'boxes': [<matplotlib.lines.Line2D at 0x222dd092e60>],
 'medians': [<matplotlib.lines.Line2D at 0x222dd093b80>],
 'fliers': [<matplotlib.lines.Line2D at 0x222dd093e20>],
 'means': []}
```

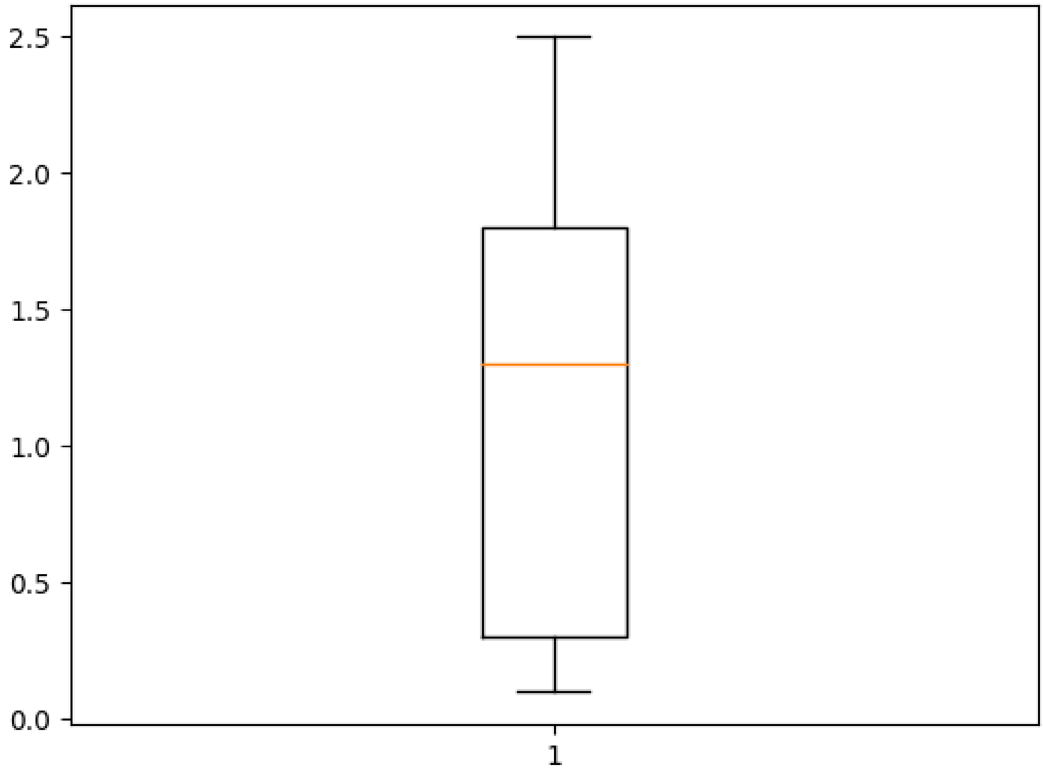


In [16]:

```
plt.boxplot(df['PetalWidthCm'])
```

Out[16]:


```
{'whiskers': [<matplotlib.lines.Line2D at 0x222dd10a4a0>,
  <matplotlib.lines.Line2D at 0x222dd10a740>],
 'caps': [<matplotlib.lines.Line2D at 0x222dd10a9e0>,
  <matplotlib.lines.Line2D at 0x222dd10ac80>],
 'boxes': [<matplotlib.lines.Line2D at 0x222dd10a200>],
 'medians': [<matplotlib.lines.Line2D at 0x222dd10af20>],
 'fliers': [<matplotlib.lines.Line2D at 0x222dd10b1c0>],
 'means': []}
```

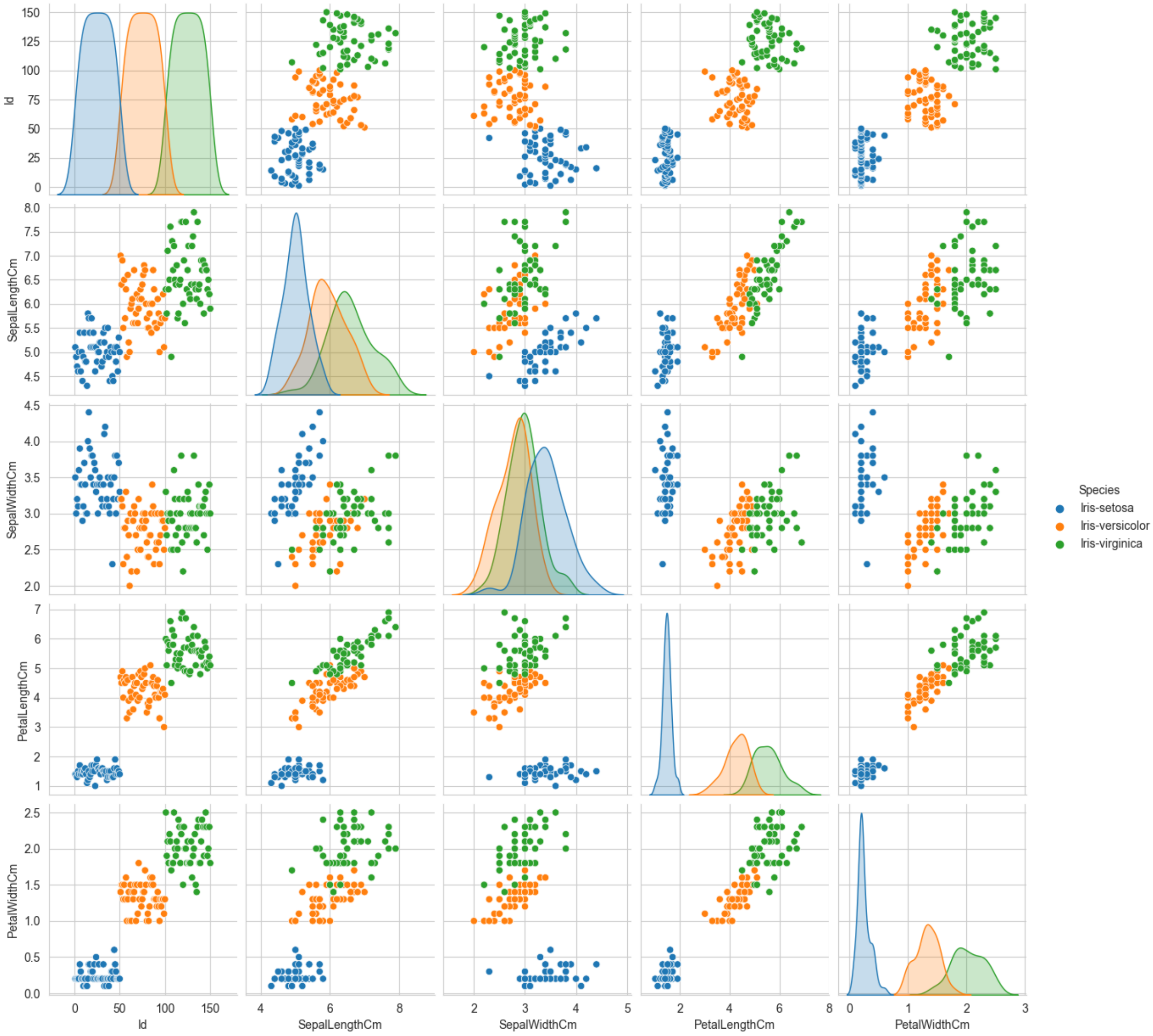



In [42]:

```
#ploting graph usning seaborn
sns.set_style('whitegrid')
sns.pairplot(data = df, hue='Species')
```

Out[42]:

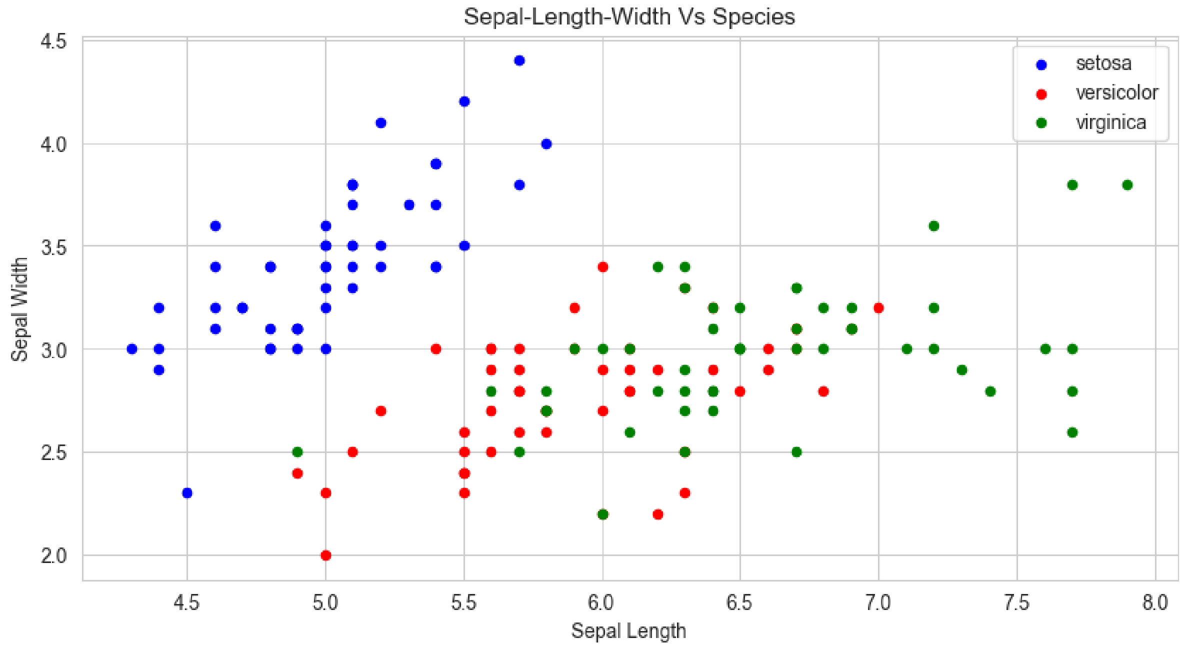
```
<seaborn.axisgrid.PairGrid at 0x222dad1d270>
```



Scatter Plot - Sepal_Length_Width Vs Species.

In [43]:

```
pet_len_wid = df[df.Species == 'Iris-setosa'].plot(kind = 'scatter', x = 'SepalLengthCm', y =
,color = 'blue', label = 'setosa')
df[df.Species == 'Iris-versicolor'].plot(kind = 'scatter', x = 'SepalLengthCm', y = 'SepalWid
,label = 'versicolor', ax = pet_len_wid)
df[df.Species == 'Iris-virginica'].plot(kind = 'scatter', x = 'SepalLengthCm', y = 'SepalWidtl
,label = 'virginica', ax = pet_len_wid)
pet_len_wid.set_xlabel('Sepal Length')
pet_len_wid.set_ylabel('Sepal Width')
pet_len_wid.set_title('Sepal-Length-Width Vs Species')
pet_len_wid = plt.gcf()
pet_len_wid.set_size_inches(10, 5)
plt.show()
```



Scatter Plot - Petal_Length_Width Vs Species.

In [44]:

```
pet_len_wid = df[df.Species == 'Iris-setosa'].plot(kind = 'scatter', x = 'PetalLengthCm', y =
,color = 'blue', label = 'setosa')
df[df.Species == 'Iris-versicolor'].plot(kind = 'scatter', x = 'PetalLengthCm', y = 'PetalWid
,label = 'versicolor', ax = pet_len_wid)
df[df.Species == 'Iris-virginica'].plot(kind = 'scatter', x = 'PetalLengthCm', y = 'PetalWidtl
,label = 'virginica', ax = pet_len_wid)
pet_len_wid.set_xlabel('Petal Length')
pet_len_wid.set_ylabel('Petal Width')
pet_len_wid.set_title('Petal-Length-Width Vs Species')
pet_len_wid = plt.gcf()
pet_len_wid.set_size_inches(10, 5)
plt.show()
```

3. Data Preparation

In [18]:

```
df.drop('Id',axis=1,inplace=True)
```

In [19]:

```
sp={'Iris-setosa':1,'Iris-versicolor':2,'Iris-virginica':3}
```

In [20]:

```
df.Species=[sp[i] for i in df.Species]
```

In [21]:

```
df
```

Out[21]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	3
146	6.3	2.5	5.0	1.9	3
147	6.5	3.0	5.2	2.0	3
148	6.2	3.4	5.4	2.3	3
149	5.9	3.0	5.1	1.8	3

150 rows × 5 columns

In [22]:

```
X=df.iloc[:,0:4]
```

In [23]:

```
X
```

Out[23]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [24]:

```
y=df.iloc[:,4]
```

In [25]:

```
y
```

Out[25]:

```
0      1
1      1
2      1
3      1
4      1
      ..
145    3
146    3
147    3
148    3
149    3
Name: Species, Length: 150, dtype: int64
```

In [26]:

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=42)
```

4.Traning Model (Machine Learning Algorithm)

In [46]:

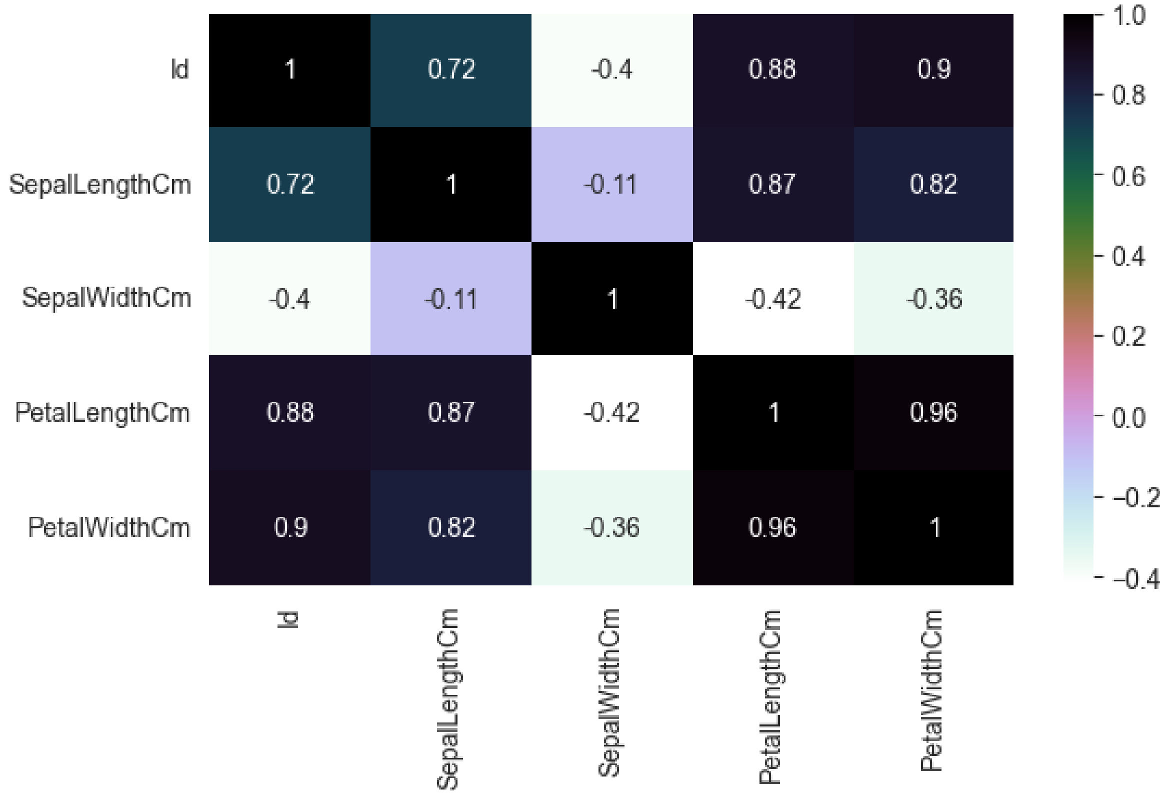
```
#importing required modules
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn import metrics #for checking the model accuracy
from sklearn.tree import DecisionTreeClassifier
```

Corelation Matrix

In [47]:

```
plt.figure(figsize=(7,4))
sns.heatmap(df.corr(),annot=True,cmap='cubehelix_r')
plt.show()
```

```
C:\Users\md naiyer azam\AppData\Local\Temp\ipykernel_18796\3227061104.py:2: Fut
ureWarning: The default value of numeric_only in DataFrame.corr is deprecated.
In a future version, it will default to False. Select only valid columns or spe
cify the value of numeric_only to silence this warning.
  sns.heatmap(df.corr(),annot=True,cmap='cubehelix_r')
```



The Sepal Width and Length are not correlated The Petal Width and Length are highly correlated

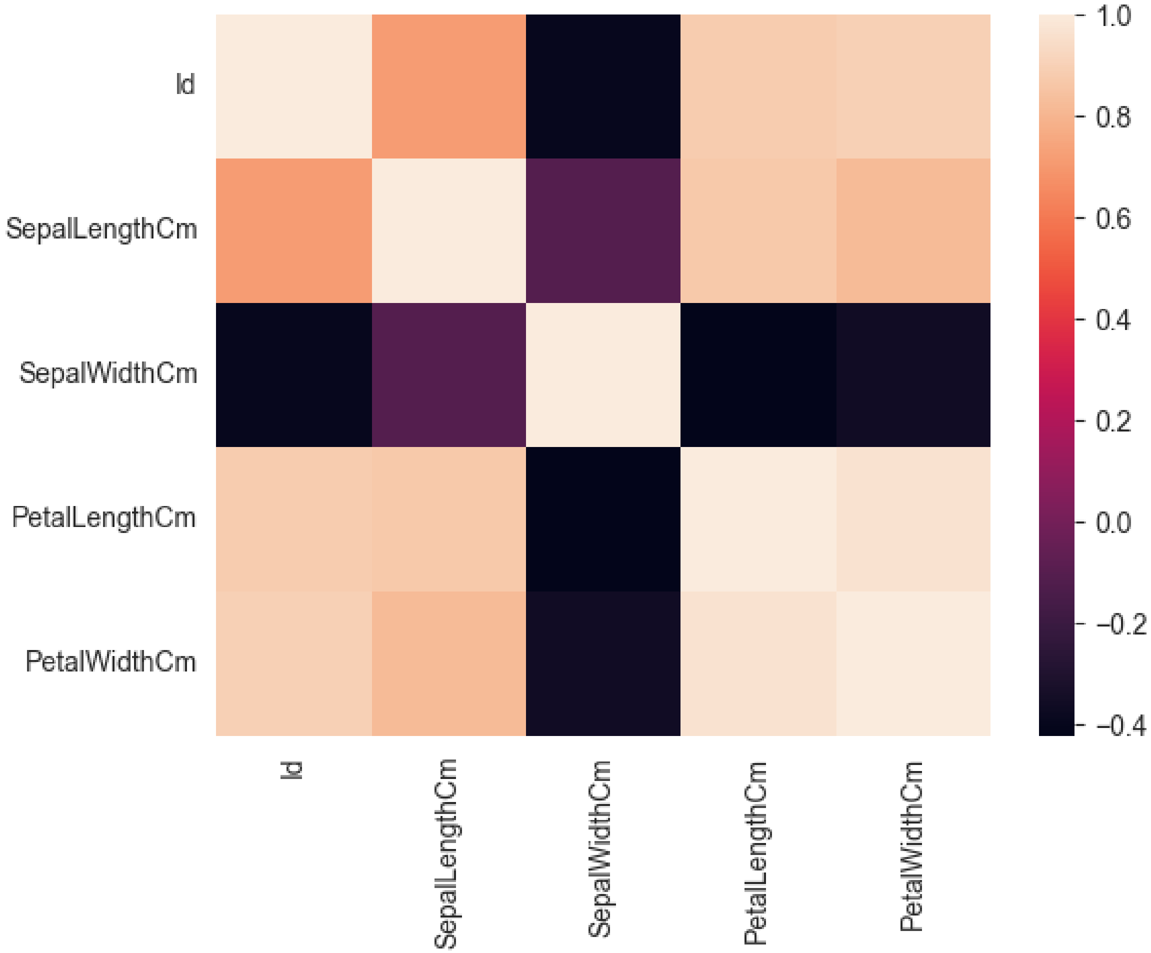
In [49]:

```
sns.heatmap(df.corr())
```

```
C:\Users\md naiyer azam\AppData\Local\Temp\ipykernel_18796\58359773.py:1: Futur
eWarning: The default value of numeric_only in DataFrame.corr is deprecated. In
a future version, it will default to False. Select only valid columns or specif
y the value of numeric_only to silence this warning.
  sns.heatmap(df.corr())
```

Out[49]:

```
<AxesSubplot: >
```



Logistic Regression

In [51]:

```
model=LinearRegression()
```

In [52]:

```
model.fit(X,y)
```

Out[52]:

```
▾ LinearRegression
LinearRegression()
```

In [53]:

```
model.score(X,y) #coef of prediction
```

Out[53]:

```
0.9304223675331595
```

In [54]:

```
model.coef_
```

Out[54]:

```
array([-0.10974146, -0.04424045,  0.22700138,  0.60989412])
```

In [55]:

```
model.intercept_
```

Out[55]:

```
1.1920839948281392
```

In [57]:

```
lR = LogisticRegression()
lR.fit(X_train, y_train)
prediction=lR.predict(X_test)
#score method to get accuracy of model
score = lR.score(X_test, y_test)
print(score,"\n\n")
#Confussion Matrix: used to describe the performance of a classification model
metrics.confusion_matrix(y_test,prediction)
```

```
1.0
```

Out[57]:

```
array([[19,  0,  0],
       [ 0, 15,  0],
       [ 0,  0, 16]], dtype=int64)
```

SVM

In [59]:

```
model = svm.SVC()
model.fit(X_train,y_train)
prediction=model.predict(X_test)
score = model.score(X_test, y_test)
print(score,"\n\n")
#Confussion Matrix: used to describe the performance of a classification model
metrics.confusion_matrix(y_test,prediction)
```

```
1.0
```

Out[59]:

```
array([[19,  0,  0],
       [ 0, 15,  0],
       [ 0,  0, 16]], dtype=int64)
```

KNN(K-Nearest Neighbours)

In [60]:

```
model=KNeighborsClassifier(n_neighbors=3)#this examines 3 neighbours for putting the new data
model.fit(X_train,y_train)
prediction=model.predict(X_test)
score = model.score(X_test, y_test)
print(score,"\n\n")
#Confussion Matrix: used to describe the performance of a classification model
metrics.confusion_matrix(y_test,prediction)
```

```
0.98
```

Out[60]:

```
array([[19,  0,  0],
       [ 0, 15,  0],
       [ 0,  1, 15]], dtype=int64)
```

Decission Tree

In [61]:

```
model=DecisionTreeClassifier()
model.fit(X_train,y_train)
prediction=model.predict(X_test)
score = model.score(X_test, y_test)
print(score,"\n\n")
#Confussion Matrix: used to describe the performance of a classification model
metrics.confusion_matrix(y_test,prediction)
```

```
0.98
```

Out[61]:

```
array([[19,  0,  0],
       [ 0, 15,  0],
       [ 0,  1, 15]], dtype=int64)
```

5.Making Predictions

In [32]:

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

6.Model Evolution

In [33]:

```
print("Mean squared error: %.2f" % np.mean((y_pred - y_test) ** 2))
```

```
Mean squared error: 0.04
```