Iris flower predicition

importing libraries

In [1]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
pd.options.display.max_rows=5000
```

In [2]:

```
aggtype = ['mean','std']
```

In [3]:

```
df=pd.read_csv("C:/Users/ayith/Downloads/Iris.csv")    # reading dataset
```

In [4]:

```
df.head()
```

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

checking unique values

In [5]:

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

Out[5]:

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

In [6]:

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

Out[6]:

```
array([1.4, 1.3, 1.5, 1.7, 1.6, 1.1, 1.2, 1. , 1.9, 4.7, 4.5, 4.9, 4. ,
       4.6, 3.3, 3.9, 3.5, 4.2, 3.6, 4.4, 4.1, 4.8, 4.3, 5. , 3.8, 3.7,
       5.1, 3. , 6. , 5.9, 5.6, 5.8, 6.6, 6.3, 6.1, 5.3, 5.5, 6.7, 6.9,
       5.7, 6.4, 5.4, 5.2])
```

In [7]:

```
df[df.duplicated()]    #checking duplicates
```

Out[7]:

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species

In [8]:

```
df.isnull().sum()    #finding null values
```

Out[8]:

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

In [9]:

```
df.info
```

Out[9]:

```
<bound method DataFrame.info of       Id  SepalLengthCm  SepalWidthCm  
PetalLengthCm  PetalWidthCm  \
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
5      6            5.4           3.9            1.7           0.4
6      7            4.6           3.4            1.4           0.3
7      8            5.0           3.4            1.5           0.2
8      9            4.4           2.9            1.4           0.2
9     10            4.9           3.1            1.5           0.1
10    11            5.4           3.7            1.5           0.2
11    12            4.8           3.4            1.6           0.2
12    13            4.8           3.0            1.4           0.1
13    14            4.3           3.0            1.1           0.1
14    15            5.8           4.0            1.2           0.2
15    16            5.7           4.4            1.5           0.4
```

In [10]:

```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
Id	150.0	75.500000	43.445368	1.0	38.25	75.50	112.75	150.0
SepalLengthCm	150.0	5.843333	0.828066	4.3	5.10	5.80	6.40	7.9
SepalWidthCm	150.0	3.054000	0.433594	2.0	2.80	3.00	3.30	4.4
PetalLengthCm	150.0	3.758667	1.764420	1.0	1.60	4.35	5.10	6.9
PetalWidthCm	150.0	1.198667	0.763161	0.1	0.30	1.30	1.80	2.5

In [11]:

```
df.groupby('Species').agg(aggtype).T
```

Out[11]:

	Species	Iris-setosa	Iris-versicolor	Iris-virginica
Id	mean	25.500000	75.500000	125.500000
	std	14.577380	14.577380	14.577380
SepalLengthCm	mean	5.006000	5.936000	6.588000
	std	0.352490	0.516171	0.635880
SepalWidthCm	mean	3.418000	2.770000	2.974000
	std	0.381024	0.313798	0.322497
PetalLengthCm	mean	1.464000	4.260000	5.552000
	std	0.173511	0.469911	0.551895
PetalWidthCm	mean	0.244000	1.326000	2.026000
	std	0.107210	0.197753	0.274650

In [12]:

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

Out[12]:

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

In [13]:

```
pd.crosstab(df['Species'],df['SepalLengthCm']).sum()
```

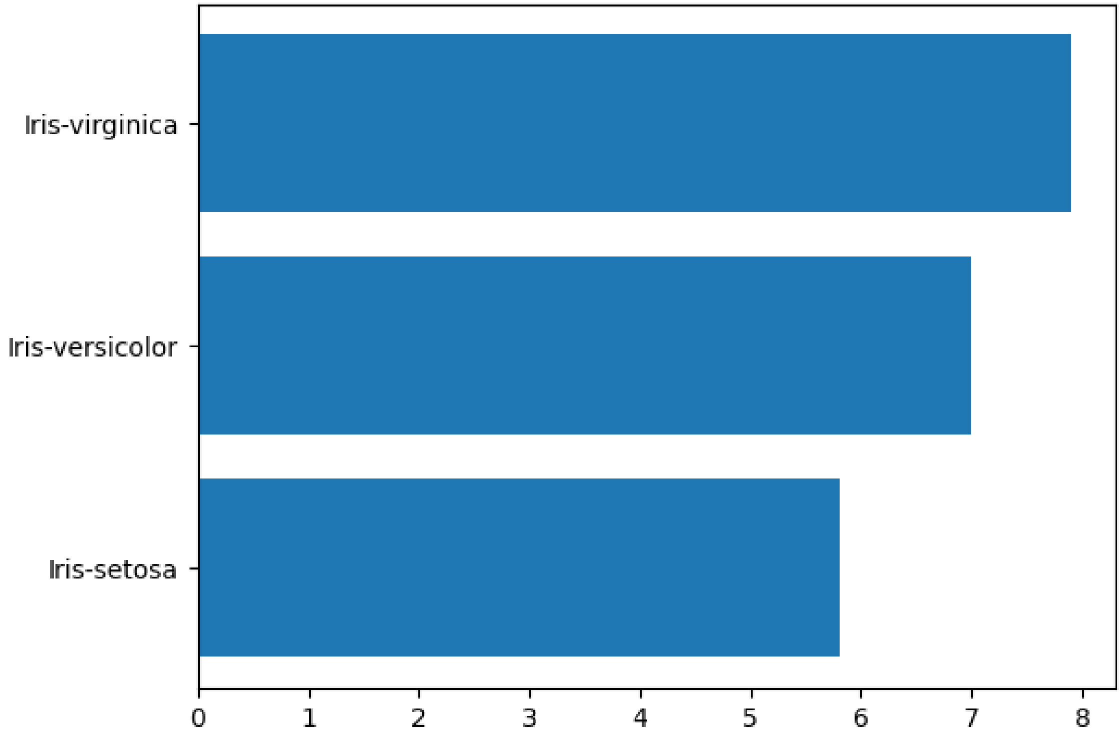
Out[13]:

```
SepalLengthCm
4.3     1
4.4     3
4.5     1
4.6     4
4.7     2
4.8     5
4.9     6
5.0    10
5.1     9
5.2     4
5.3     1
5.4     6
5.5     7
5.6     6
5.7     8
5.8     7
5.9     3
6.0     6
6.1     6
6.2     4
6.3     9
6.4     7
6.5     5
6.6     2
6.7     8
6.8     3
6.9     4
7.0     1
7.1     1
7.2     3
7.3     1
7.4     1
7.6     1
7.7     4
7.9     1
dtype: int64
```

Data graphs

In [14]:

```
fig,ax=plt.subplots()
y=ax.barh('Species','SepalLengthCm',data=df)

plt.show()
```



In [15]:

```
df
```

Out[15]:

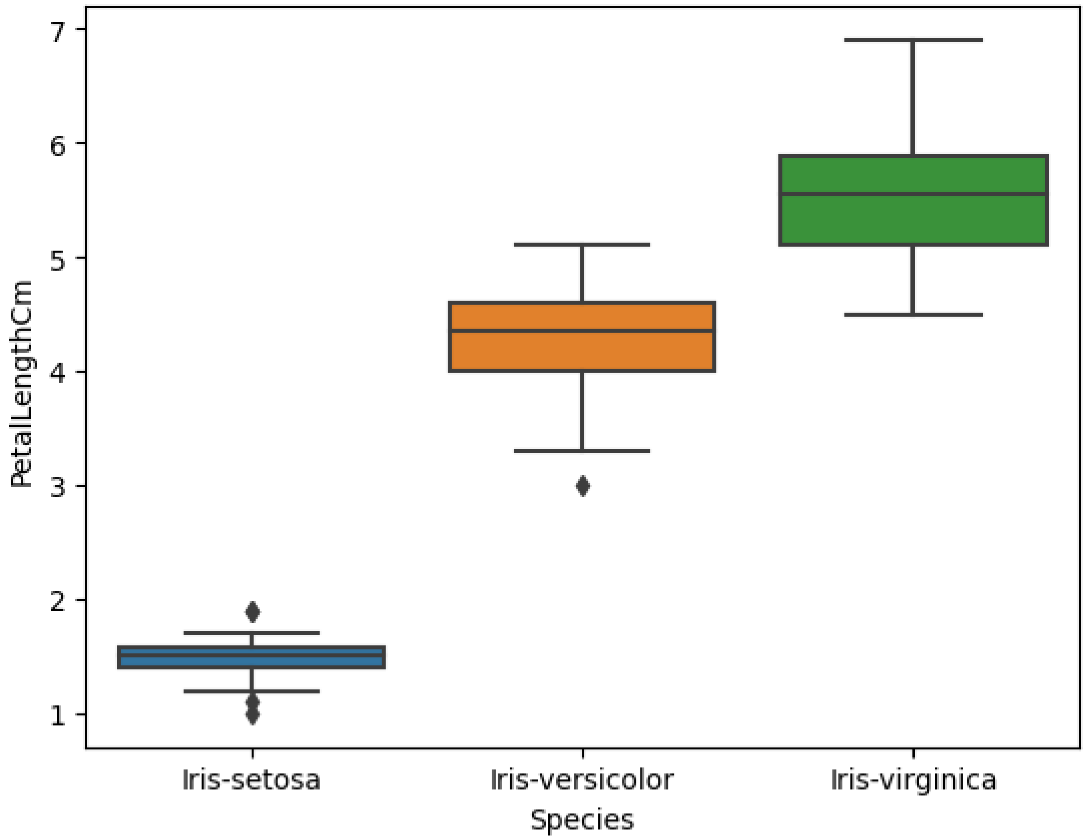
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	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
10	11	5.4	3.7	1.5	0.2	Iris-setosa

In [16]:

```
sns.boxplot(x ='Species', y ='PetalLengthCm',data=df)
```

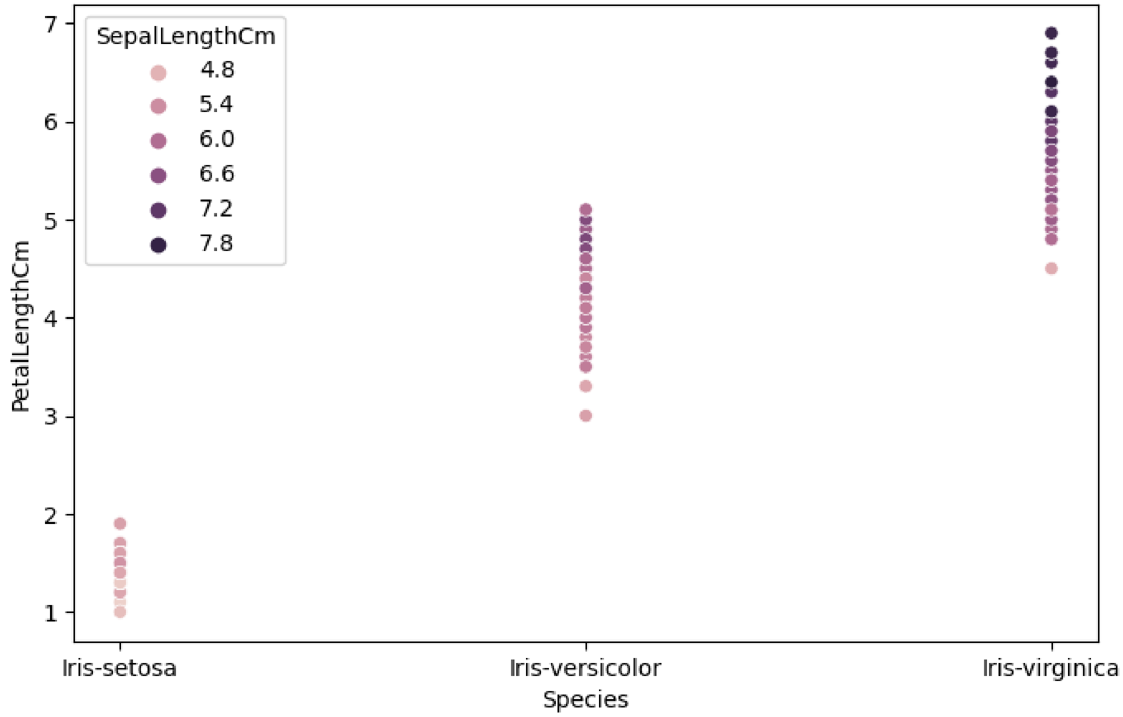
Out[16]:

```
<AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>
```



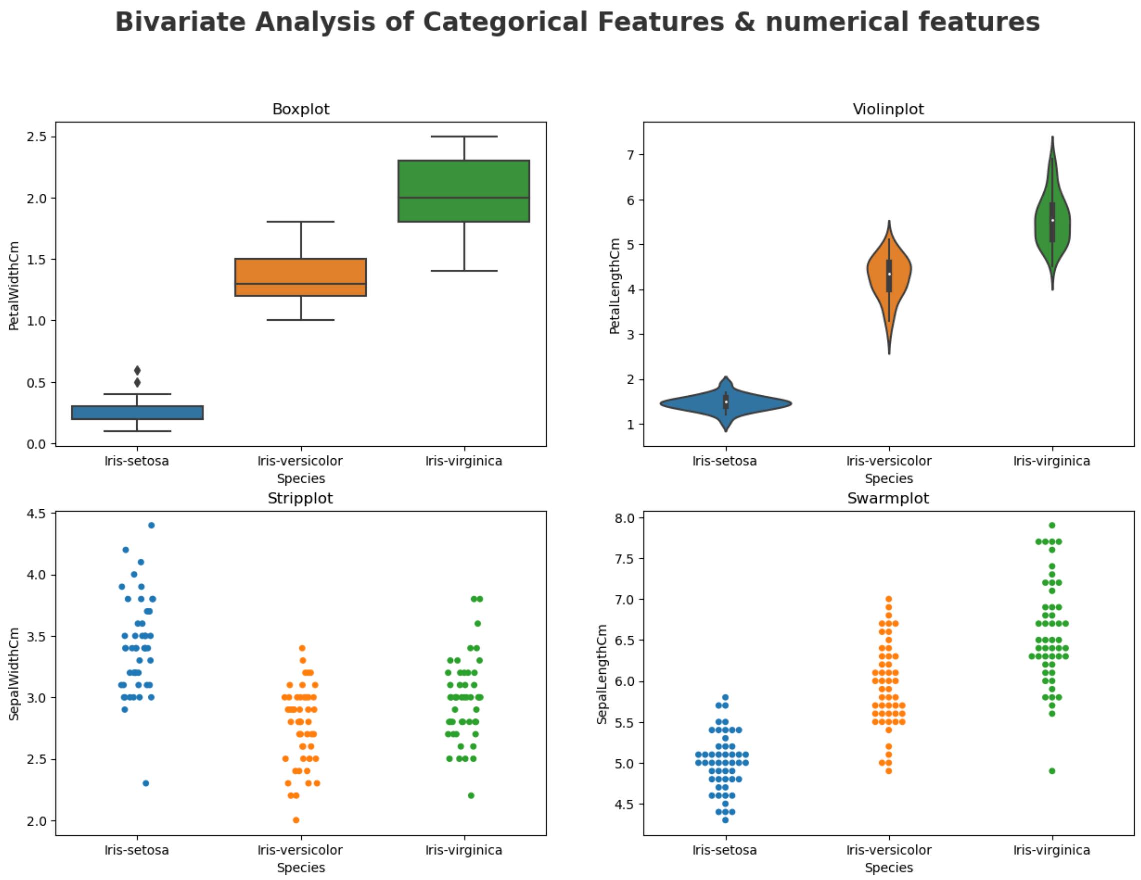
In [17]:

```
plt.figure(figsize=(8,5))
sns.scatterplot(x = 'Species', y = 'PetalLengthCm', hue = 'SepalLengthCm', data = df)
plt.show()
```



In [18]:

```
plt.figure(figsize=(15,10))
plt.suptitle('Bivariate Analysis of Categorical Features & numerical features', fontsize

plt.subplot(2,2,1)
sns.boxplot(x = df['Species'], y = df['PetalWidthCm'])
plt.title("Boxplot")

plt.subplot(2,2,2)
sns.violinplot(x = df['Species'], y = df['PetalLengthCm'])
plt.title("Violinplot")

plt.subplot(2,2,3)
sns.stripplot(x = df['Species'], y = df['SepalWidthCm'])
plt.title("Stripplot")

plt.subplot(2,2,4)
sns.swarmplot(x = df['Species'], y = df['SepalLengthCm'])
plt.title("Swarmplot")

plt.show()
```



In [19]:

```
num_features=[feature for feature in df.columns if df[feature].dtype != 'O']
```

In [20]:

```
num_features
```

Out[20]:

```
['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
```

In [21]:

```
cat_features=[feature for feature in df.columns if df[feature].dtype == 'O']
```

In [22]:

```
cat_features
```

Out[22]:

```
['Species']
```

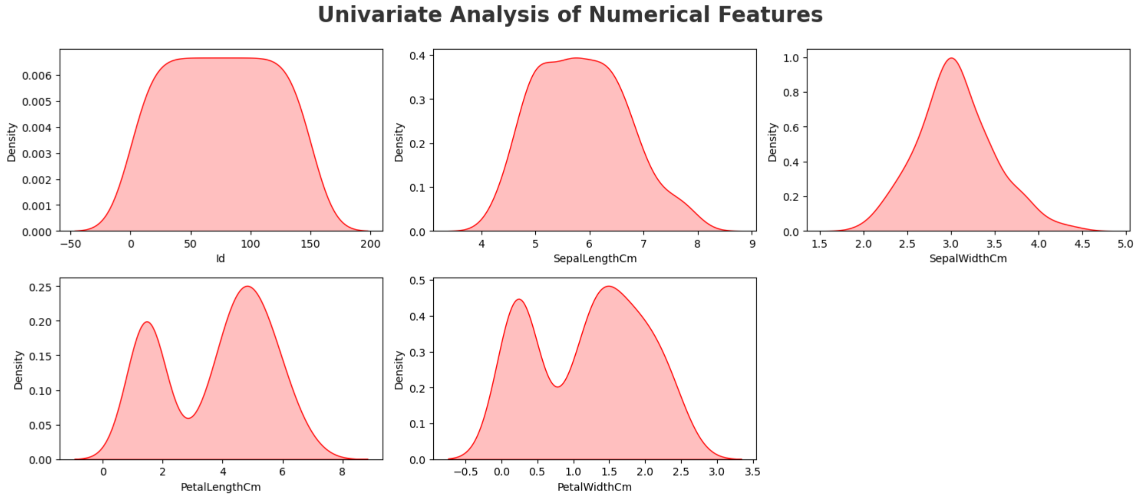
In [23]:

```
for col in num_features:
    print(f"{col}:{df[col].value_counts(normalize=True)*100}")
    print("=============================================")
```

```
Id:1      0.666667
95     0.666667
97     0.666667
98     0.666667
99     0.666667
100    0.666667
101    0.666667
102    0.666667
103    0.666667
104    0.666667
105    0.666667
106    0.666667
107    0.666667
108    0.666667
109    0.666667
110    0.666667
111    0.666667
96     0.666667
94     0.666667
```

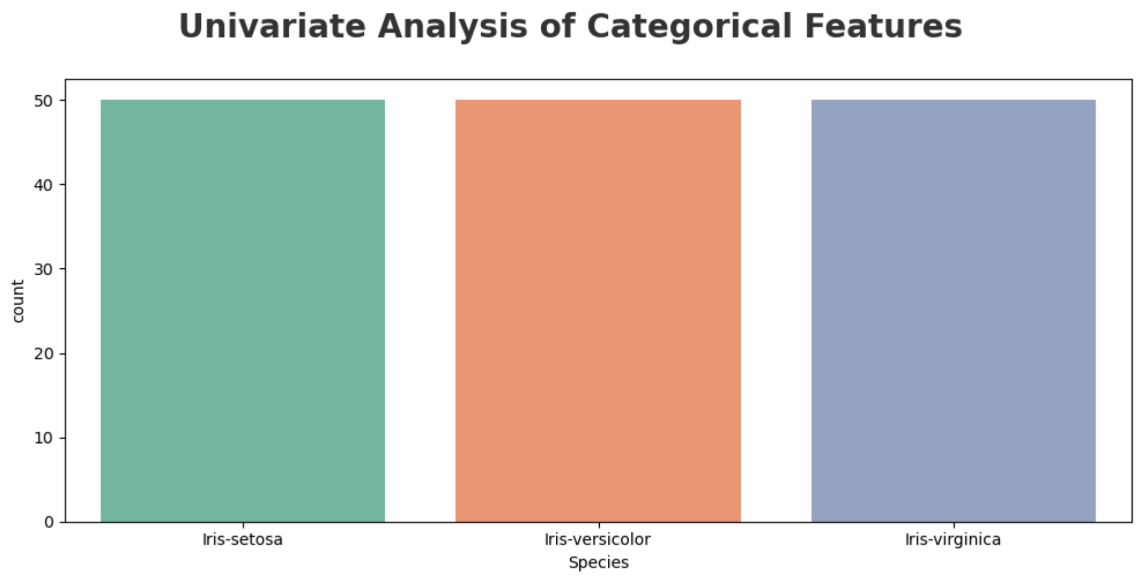
In [24]:

```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold'

for i in range(0, len(num_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[num_features[i]],shade=True, color='r')
    plt.xlabel(num_features[i])
    plt.tight_layout()
```



In [25]:

```
plt.figure(figsize=(10,5))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight='bol
category = [ 'Species']
for i in range(0, len(category)):

    sns.countplot(x=df[category[i]],palette="Set2")
    plt.xlabel(category[i])
    plt.xticks(rotation=0)
    plt.tight_layout()
```



Lables encodeing

In [26]:

```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df['Species']= label_encoder.fit_transform(df['Species'])

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

Out[26]:

```
array([0, 1, 2])
```

In [27]:

```
from sklearn import preprocessing
```

In [28]:

```
lable_encodeing= preprocessing.LabelEncoder()
df['Species']=lable_encodeing.fit_transform(df['Species'])
df['Species'].unique()
```

Out[28]:

```
array([0, 1, 2], dtype=int64)
```

In [29]:

```
df.sample(5)
```

Out[29]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
37	38	4.9	3.1	1.5	0.1	0
132	133	6.4	2.8	5.6	2.2	2
102	103	7.1	3.0	5.9	2.1	2
2	3	4.7	3.2	1.3	0.2	0
33	34	5.5	4.2	1.4	0.2	0

In [30]:

```
df
```

Out[30]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
5	6	5.4	3.9	1.7	0.4	0
6	7	4.6	3.4	1.4	0.3	0
7	8	5.0	3.4	1.5	0.2	0
8	9	4.4	2.9	1.4	0.2	0
9	10	4.9	3.1	1.5	0.1	0
10	11	5.4	3.7	1.5	0.2	0

Data spliting

In [31]:

```
x = df.iloc[:, [1,2,3, 4]].values
y=df.iloc[:,[5]].values
```

In [32]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state
```

In [33]:

```
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[33]:

```
((112, 4), (38, 4), (112, 1), (38, 1))
```

In [34]:

```
type(y_test)
```

Out[34]:

```
numpy.ndarray
```

scaleing data

In [35]:

```
from sklearn.preprocessing import StandardScaler
```

In [36]:

```
sc_X=StandardScaler()
```

In [37]:

```
xtrain = sc_X.fit_transform(X_train)
```

In [38]:

```
xtest = sc_X.fit_transform(X_test)
```

creating model

In [39]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[39]:

```
LogisticRegression(random_state=0)
```

In [40]:

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

In [41]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [42]:

```
print('accuracy of model: ',classifier.score(X_test, y_test))
```

```
accuracy of model:  1.0
```

In [43]:

```
from sklearn.model_selection import cross_val_score
cvs = cross_val_score(classifier, X_train, y_train, cv=10)
print('accuracy of model after CVS: ',np.mean(cvs))
```

```
accuracy of model after CVS:  0.9469696969696969
```

checking the predict values with original values

In [44]:

```
y_test=y_test.flatten()
predict = classifier.predict(X_test)
compar = pd.DataFrame({'actual':y_test, 'predicted': predict})
compar = compar.reset_index(drop = True)
compar[:10]
```

Out[44]:

	actual	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	1	1
5	1	1
6	1	1
7	0	0
8	1	1
9	2	2

In [45]:

```
from sklearn.metrics import accuracy_score
print ("Accuracy : ", accuracy_score(y_test, y_pred))
```

```
Accuracy :  1.0
```

In [46]:

```
from sklearn.metrics import classification_report
```

In [47]:

```
print(classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        12

    accuracy                           1.00        38
   macro avg       1.00      1.00      1.00        38
weighted avg       1.00      1.00      1.00        38
```

In [ ]:

In [ ]: