In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import numpy as np
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ro
from sklearn.metrics import roc_auc_score

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import RandomizedSearchCV
from xgboost import XGBClassifier as xgb

import shap
import lime
import lime.lime_tabular
```

In [2]:
```
df = pd.read_csv("Iris Flower - Iris.csv")
df.head()
```

Out[2]:

	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 [3]:
```
df = df.drop('Id',axis=1)
df.head()
```

Out[3]:

	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 [4]:
```
df1 =df.copy()
df2 =df.copy()
df3 =df.copy()
```

In [5]:
```
df.shape
```

Out[5]:
```
(150, 5)
```

In [6]:
```
df.info()
```

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

In [7]:
```python
df.describe()
```

Out[7]:

	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

The target Variable is PetalLengthCm

In [8]:
```python
df.rename(columns={'Species':'Species_Value'},inplace=True)
df.head()
```

Out[8]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Value
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

Chack Missing Values

In [9]:
```
df.isnull().sum()
```

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

In [10]:
```
df.dtypes
```

Out[10]:
```
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species_Value     object
dtype: object
```

Data inconsistencies & Data Preprocessing:

In [11]:
```
for col in df.columns:
    print(f"---###*** {col} ---###***")
    print(df[col].value_counts())
```

```
5.8     7
5.5     7
6.4     7
4.9     6
5.4     6
6.1     6
6.0     6
5.6     6
4.8     5
6.5     5
6.2     4
7.7     4
6.9     4
4.6     4
5.2     4
5.9     3
4.4     3
7.2     3
6.8     3
6.6     2
```

Handling NaN Values

In [12]:
```
Num_cols = []
cat_cols = []
for col in df.columns:
    if df[col].dtypes == 'object':
        cat_cols.append(col)
    else:
        Num_cols.append(col)
```

In [13]:
```
#handle null values of numerical columns

for col in Num_cols:
    if df[col].isna:
        df[col].fillna(df[col].median(), inplace=True)
```

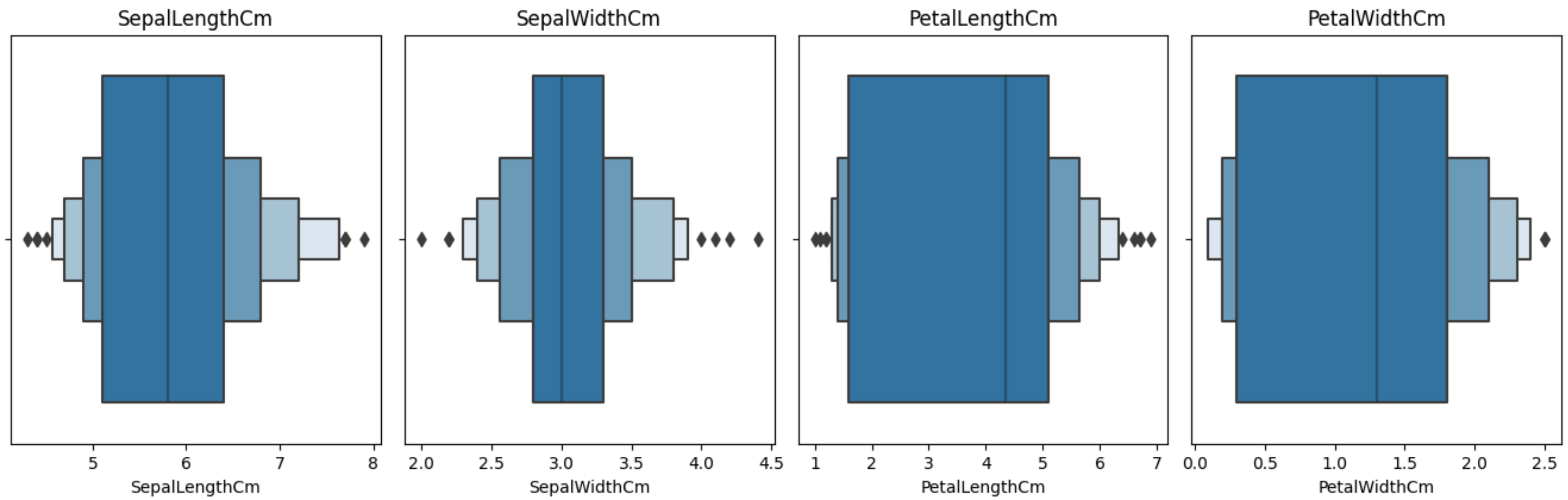
In [14]:
```
#handle null values of categorical columns

for col in cat_cols:
    if df[col].isnull:
        df[col].fillna(df[col].mode()[0], inplace=True)
```

In [15]:
```
df.isnull().sum()
```

Out[15]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species_Value    0
dtype: int64
```

Outliers

In [16]:
```
plt.figure(figsize=(20,20))
for ax, col in enumerate(Num_cols):
    plt.subplot(5,6, int(ax+1))
    plt.title(col)
    sns.boxenplot(x=df[col],hue=df['Species_Value'])
plt.tight_layout()
plt.show()
```



Encoding target value

In [17]:
```
from sklearn.preprocessing import LabelEncoder
led =LabelEncoder()
led.fit_transform(df['Species_Value'])
df['Species_Value'] =led.fit_transform(df['Species_Value'])
df.head()
```

Out[17]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Value
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 [18]:
```
df.value_counts('Species_Value')
```

Out[18]:
```
Species_Value
0    50
1    50
2    50
dtype: int64
```

Model_Selection of ML

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

Out[19]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Value
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

Decision Tree Classifier

In [20]:
```
from sklearn import tree
```

In [21]:
```
df.head()
```

Out[21]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Value
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 [22]:
```
x =df.drop('Species_Value',axis=1)
y =df['Species_Value']
```

DecisionTreeClassifier

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
dtc =DecisionTreeClassifier()
```

In [25]:
```
dtc.fit(x,y)
```

Out[25]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [26]:
```
dtc.predict([[1,0,0,0]])
```

```
X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
```

Out[26]:
```
array([0])
```

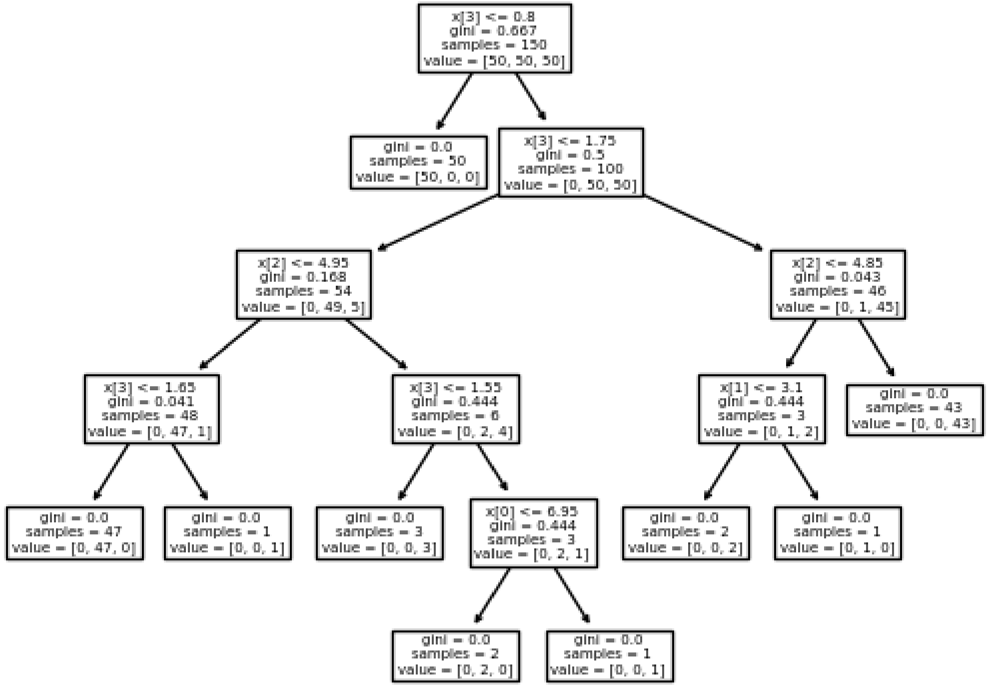
In [27]:
```
dtc.predict([[0,0,0,1]])
```

```
X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
```

Out[27]:
```
array([1])
```

In [28]:
```
tree.plot_tree(dtc)
```

Out[28]:
```
[Text(0.5, 0.9166666666666666, 'x[3] <= 0.8\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 50]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'x[3] <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 50]'),
 Text(0.3076923076923077, 0.5833333333333334, 'x[2] <= 4.95\ngini = 0.168\nsamples = 54\nvalue = [0, 4
9, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'x[3] <= 1.65\ngini = 0.041\nsamples = 48\nvalue = [0, 4
7, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'x[3] <= 1.55\ngini = 0.444\nsamples = 6\nvalue = [0, 2,
4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'x[0] <= 6.95\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'x[2] <= 4.85\ngini = 0.043\nsamples = 46\nvalue = [0, 1,
45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'x[1] <= 3.1\ngini = 0.444\nsamples = 3\nvalue = [0, 1,
2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue = [0, 0, 43]')]
```



x[3] <= 0.8
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
x[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
x[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
x[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
x[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
x[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
x[1] <= 3.1
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
x[0] <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

In [29]:
```
tree.plot_tree(dtc, rounded=True, filled=True,feature_names = x.columns )
```

Out[29]:
```
[Text(0.5, 0.9166666666666666, 'PetalWidthCm <= 0.8\ngini = 0.667\nsamples = 150\nvalue = [50, 50, 5
0]'),
 Text(0.4230769230769231, 0.75, 'gini = 0.0\nsamples = 50\nvalue = [50, 0, 0]'),
 Text(0.5769230769230769, 0.75, 'PetalWidthCm <= 1.75\ngini = 0.5\nsamples = 100\nvalue = [0, 50, 5
0]'),
 Text(0.3076923076923077, 0.5833333333333334, 'PetalLengthCm <= 4.95\ngini = 0.168\nsamples = 54\nvalue
= [0, 49, 5]'),
 Text(0.15384615384615385, 0.4166666666666667, 'PetalWidthCm <= 1.65\ngini = 0.041\nsamples = 48\nvalue
= [0, 47, 1]'),
 Text(0.07692307692307693, 0.25, 'gini = 0.0\nsamples = 47\nvalue = [0, 47, 0]'),
 Text(0.23076923076923078, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.46153846153846156, 0.4166666666666667, 'PetalWidthCm <= 1.55\ngini = 0.444\nsamples = 6\nvalue
= [0, 2, 4]'),
 Text(0.38461538461538464, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(0.5384615384615384, 0.25, 'SepalLengthCm <= 6.95\ngini = 0.444\nsamples = 3\nvalue = [0, 2, 1]'),
 Text(0.46153846153846156, 0.08333333333333333, 'gini = 0.0\nsamples = 2\nvalue = [0, 2, 0]'),
 Text(0.6153846153846154, 0.08333333333333333, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(0.8461538461538461, 0.5833333333333334, 'PetalLengthCm <= 4.85\ngini = 0.043\nsamples = 46\nvalue
= [0, 1, 45]'),
 Text(0.7692307692307693, 0.4166666666666667, 'SepalWidthCm <= 3.1\ngini = 0.444\nsamples = 3\nvalue =
[0, 1, 2]'),
 Text(0.6923076923076923, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.8461538461538461, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(0.9230769230769231, 0.4166666666666667, 'gini = 0.0\nsamples = 43\nvalue = [0, 0, 43]')]
```

K-Nearest Neighbors (KNN)

In [30]:
```
from sklearn.neighbors import KNeighborsRegressor
```

In [31]:
```
knn = KNeighborsRegressor(n_neighbors=3)
```

In [32]:
```
knn.fit(df[["PetalLengthCm"]],df["Species_Value"])
```

Out[32]:
```
▾ KNeighborsRegressor
KNeighborsRegressor(n_neighbors=3)
```

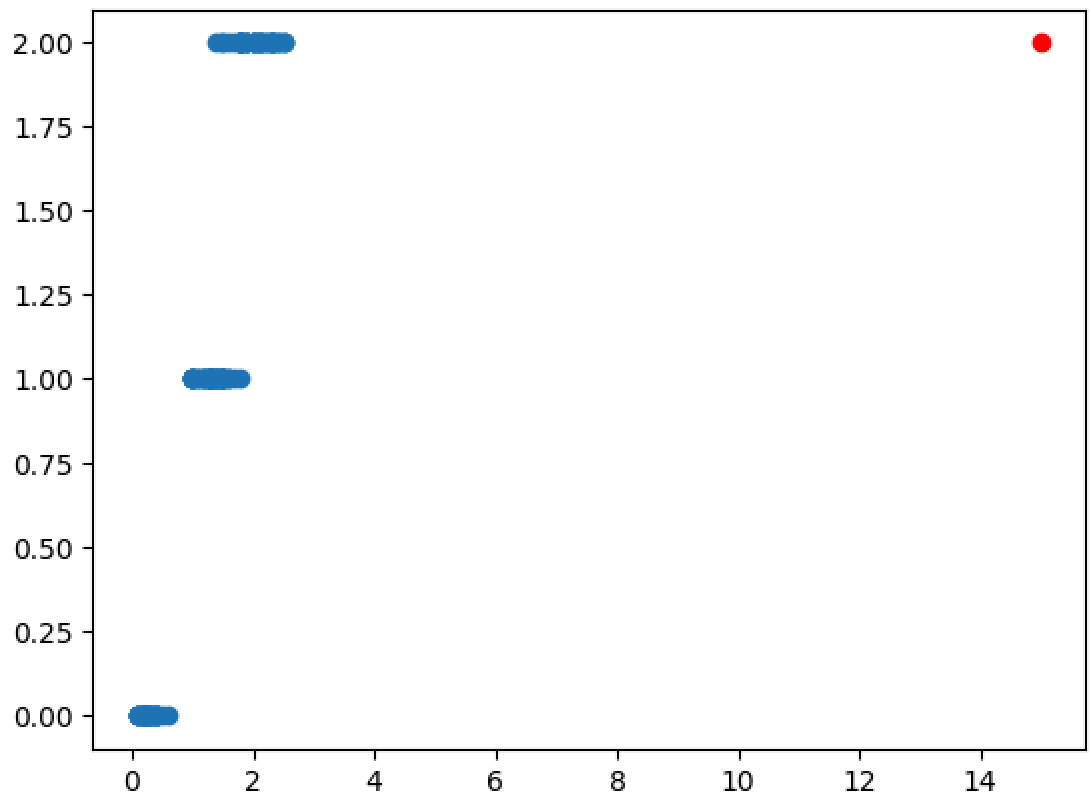
In [33]:
```
knn.predict([[15]])
```

X does not have valid feature names, but KNeighborsRegressor was fitted with feature names

Out[33]:
```
array([2.])
```

In [34]:
```
plt.scatter(df.PetalWidthCm, df.Species_Value)
plt.scatter(15,2, color="red")
```

Out[34]:
```
<matplotlib.collections.PathCollection at 0x206fcbecfd0>
```



Logistic Regression

In [35]:
```
df2.head()
```

Out[35]:

	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 [36]:
```
df2.value_counts('Species')
```

Out[36]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [37]:
```
df2['Species'] = df2['Species'].replace(['Iris-setosa','Iris-versicolor','Iris-virginica'],[0,1,2])
```

In [38]:
```
df2.head()
```

Out[38]:

	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

In [39]:
```
x=df2.drop('Species', axis=1)
```

In [40]:
```
y =df2[['Species']]
```

In [41]:
```
from sklearn.linear_model import LogisticRegression
```

In [42]:
```
reg =LogisticRegression()
```

In [43]:
```
reg.fit(x,y)
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
```

Out[43]:
```
▾ LogisticRegression
LogisticRegression()
```

In [44]:
```
reg.coef_
```

Out[44]:
```
array([[-0.4234002 ,  0.9616903 , -2.51936288, -1.08612842],
       [ 0.53407785, -0.31789737, -0.20537408, -0.93954425],
       [-0.11067765, -0.64379293,  2.72473696,  2.02567267]])
```

In [45]:
```
reg.intercept_
```

Out[45]:
```
array([  9.88097006,   2.21928156, -12.10025163])
```

In [46]:
```
reg.predict(x)
```

Out[46]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int64)
```

In [47]:
```
reg.predict_proba(x)
```

```
Out[47]: array([[9.81804276e-01, 1.81957100e-02, 1.43460484e-08],
                [9.71802160e-01, 2.81978107e-02, 2.97679380e-08],
                [9.85495978e-01, 1.45040094e-02, 1.21748649e-08],
                [9.76424964e-01, 2.35749969e-02, 3.91821962e-08],
                [9.85398356e-01, 1.46016317e-02, 1.18853668e-08],
                [9.70381261e-01, 2.96186651e-02, 7.36250444e-08],
                [9.86930605e-01, 1.30693757e-02, 1.97677029e-08],
                [9.76445341e-01, 2.35546320e-02, 2.74313737e-08],
                [9.79977769e-01, 2.00222013e-02, 3.01443225e-08],
                [9.69264678e-01, 3.07352902e-02, 3.12962705e-08],
                [9.76465697e-01, 2.35342841e-02, 1.92046470e-08],
                [9.75509897e-01, 2.44900592e-02, 4.34927557e-08],
                [9.74675031e-01, 2.53249478e-02, 2.11985210e-08],
                [9.92023518e-01, 7.97647851e-03, 3.82662375e-09],
                [9.88115480e-01, 1.18845174e-02, 2.82138058e-09],
                [9.86674116e-01, 1.33258709e-02, 1.29085668e-08],
                [9.88048548e-01, 1.19514428e-02, 9.20169859e-09],
                [9.81540546e-01, 1.84594348e-02, 1.95775987e-08],
                [9.56425841e-01, 4.35740903e-02, 6.85582969e-08],
                [9.84109627e-01, 1.58903526e-02, 2.04864764e-08],
                [9.46788727e-01, 5.32111874e-02, 8.60349466e-08],
                [9.81718341e-01, 1.82816258e-02, 3.27551617e-08],
                [9.96012118e-01, 3.98788090e-03, 1.30120720e-09],
                [9.52307406e-01, 4.76923584e-02, 2.35294370e-07],
                [9.52142739e-01, 4.78570566e-02, 2.04706858e-07],
                [9.51724920e-01, 4.82749940e-02, 8.58539787e-08],
                [9.69642321e-01, 3.03575937e-02, 8.57525714e-08],
                [9.74938912e-01, 2.50610632e-02, 2.48321606e-08],
                [9.77345878e-01, 2.26541046e-02, 1.73006443e-08],
                [9.71377478e-01, 2.86224638e-02, 5.78685246e-08],
                [9.64456751e-01, 3.55431792e-02, 6.96055318e-08],
                [9.64847917e-01, 3.51520261e-02, 5.72363014e-08],
                [9.88381497e-01, 1.16184958e-02, 7.03827774e-09],
                [9.89023113e-01, 1.09768821e-02, 5.32307431e-09],
                [9.69264678e-01, 3.07352902e-02, 3.12962705e-08],
                [9.84675735e-01, 1.53242569e-02, 7.90857558e-09],
                [9.78888779e-01, 2.11112114e-02, 9.59440027e-09],
                [9.69264678e-01, 3.07352902e-02, 3.12962705e-08],
                [9.85936993e-01, 1.40629917e-02, 1.52884991e-08],
                [9.74139741e-01, 2.58602304e-02, 2.82359399e-08],
                [9.86622239e-01, 1.33777494e-02, 1.12894387e-08],
                [9.62454256e-01, 3.75456797e-02, 6.46692339e-08],
                [9.89077657e-01, 1.09223321e-02, 1.11248863e-08],
```

```
[9.72415189e-01, 2.75846748e-02, 1.36473924e-07],
[9.60295084e-01, 3.97046941e-02, 2.22313454e-07],
[9.73941209e-01, 2.60587512e-02, 3.94699041e-08],
[9.80339423e-01, 1.96605521e-02, 2.52583405e-08],
[9.83423725e-01, 1.65762546e-02, 1.98937643e-08],
[9.78568486e-01, 2.14314958e-02, 1.86534512e-08],
[9.78710783e-01, 2.12891974e-02, 1.91086885e-08],
[2.10862668e-03, 8.73911720e-01, 1.23979654e-01],
[5.76823060e-03, 8.59784431e-01, 1.34447338e-01],
[1.05081764e-03, 7.25004743e-01, 2.73944440e-01],
[1.54491926e-02, 9.39605856e-01, 4.49449509e-02],
[2.36330137e-03, 8.15138141e-01, 1.82498557e-01],
[6.96159722e-03, 8.60065758e-01, 1.32972645e-01],
[3.73034137e-03, 7.16634303e-01, 2.79635356e-01],
[1.48642786e-01, 8.48302043e-01, 3.05517120e-03],
[2.76433824e-03, 8.96565673e-01, 1.00669988e-01],
[4.14097366e-02, 9.11964289e-01, 4.66259747e-02],
[5.63081930e-02, 9.37213785e-01, 6.47802175e-03],
[1.50880019e-02, 8.98923592e-01, 8.59884062e-02],
[9.14621362e-03, 9.76491713e-01, 1.43620734e-02],
[3.03185326e-03, 7.79195995e-01, 2.17772151e-01],
[7.43534625e-02, 9.15191810e-01, 1.04547279e-02],
[5.24771741e-03, 9.26352106e-01, 6.84001765e-02],
[8.65632963e-03, 7.74751407e-01, 2.16592263e-01],
[1.64699180e-02, 9.65138181e-01, 1.83919005e-02],
[1.80985911e-03, 8.00836957e-01, 1.97353184e-01],
[2.40284995e-02, 9.59352815e-01, 1.66186855e-02],
[2.27943755e-03, 4.40397579e-01, 5.57322984e-01],
[1.67935233e-02, 9.56795838e-01, 2.64106382e-02],
[7.12108592e-04, 5.96065484e-01, 4.03222407e-01],
[3.03114082e-03, 8.59773974e-01, 1.37194885e-01],
[7.04928838e-03, 9.42941928e-01, 5.00087837e-02],
[5.04702753e-03, 9.20090125e-01, 7.48628470e-02],
[1.11365773e-03, 8.01376342e-01, 1.97510000e-01],
[5.72573050e-04, 4.81147340e-01, 5.18280087e-01],
[5.45016575e-03, 8.13094564e-01, 1.81455270e-01],
[6.19098607e-02, 9.34706775e-01, 3.38336387e-03],
[2.92562226e-02, 9.57114156e-01, 1.36296213e-02],
[3.73398416e-02, 9.55117128e-01, 7.54303014e-03],
[2.51785108e-02, 9.56466417e-01, 1.83550721e-02],
[4.46077656e-04, 3.49641519e-01, 6.49912404e-01],
[1.01618347e-02, 7.50991852e-01, 2.38846313e-01],
[9.88064442e-03, 7.88903270e-01, 2.01216085e-01],
```

```
[2.24498796e-03, 8.05140196e-01, 1.92614816e-01],
[2.76574182e-03, 9.13049949e-01, 8.41843090e-02],
[2.69596896e-02, 9.28603489e-01, 4.44368212e-02],
[1.99216288e-02, 9.38039653e-01, 4.20387180e-02],
[8.71455579e-03, 8.97810703e-01, 9.34747407e-02],
[4.61617435e-03, 8.28235221e-01, 1.67148604e-01],
[1.75871757e-02, 9.56979931e-01, 2.54328934e-02],
[1.22509266e-01, 8.74440260e-01, 3.05047361e-03],
[1.44435519e-02, 9.20449668e-01, 6.51067802e-02],
[1.99267378e-02, 9.38131157e-01, 4.19421055e-02],
[1.70447215e-02, 9.25454598e-01, 5.75006806e-02],
[8.46623899e-03, 9.35114317e-01, 5.64194440e-02],
[2.44641374e-01, 7.54068127e-01, 1.29049945e-03],
[1.91187811e-02, 9.36056233e-01, 4.48249863e-02],
[8.83945976e-07, 3.92380755e-03, 9.96075309e-01],
[2.40478186e-04, 1.62637592e-01, 8.37121929e-01],
[2.43989458e-06, 2.55900308e-02, 9.74407529e-01],
[3.08462316e-05, 8.17010657e-02, 9.18268088e-01],
[3.67341536e-06, 1.74640166e-02, 9.82532310e-01],
[5.42581633e-08, 4.64029239e-03, 9.95359653e-01],
[5.74744241e-03, 5.13824824e-01, 4.80427733e-01],
[6.12087217e-07, 2.13369896e-02, 9.78662398e-01],
[5.17296248e-06, 5.32566160e-02, 9.46738211e-01],
[6.38000895e-07, 5.75609365e-03, 9.94243268e-01],
[2.97721865e-04, 2.10640717e-01, 7.89061562e-01],
[7.19637222e-05, 1.37323809e-01, 8.62604227e-01],
[2.09423186e-05, 6.53123413e-02, 9.34666716e-01],
[2.27360977e-04, 1.45309102e-01, 8.54463537e-01],
[6.80627890e-05, 4.35826418e-02, 9.56349295e-01],
[5.07181450e-05, 5.41253766e-02, 9.45823905e-01],
[5.48969877e-05, 1.22934156e-01, 8.77010947e-01],
[8.24635299e-08, 3.56625143e-03, 9.96433666e-01],
[3.09054643e-09, 1.00046829e-03, 9.98999529e-01],
[3.88669705e-04, 4.51638362e-01, 5.47972968e-01],
[5.49480546e-06, 2.38827666e-02, 9.76111739e-01],
[6.07239782e-04, 1.90609152e-01, 8.08783608e-01],
[3.08193167e-08, 4.65327551e-03, 9.95346694e-01],
[5.80434319e-04, 3.93056939e-01, 6.06362627e-01],
[1.26038590e-05, 3.86515075e-02, 9.61335889e-01],
[4.77771806e-06, 5.14939827e-02, 9.48501240e-01],
[1.06269412e-03, 4.56521122e-01, 5.42416184e-01],
[1.01199819e-03, 3.85479491e-01, 6.13508511e-01],
[1.05051565e-05, 3.63647722e-02, 9.63624723e-01],
```

```
       [1.66741235e-05, 1.41905719e-01, 8.58077606e-01],
       [1.04799974e-06, 2.91884542e-02, 9.70810498e-01],
       [6.86347607e-07, 1.74357944e-02, 9.82563519e-01],
       [7.76839814e-06, 2.72882629e-02, 9.72703969e-01],
       [5.24875293e-04, 4.75421735e-01, 5.24053390e-01],
       [6.22412534e-05, 1.88447131e-01, 8.11490627e-01],
       [3.85676613e-07, 1.17528221e-02, 9.88246792e-01],
       [1.13862839e-05, 1.73678116e-02, 9.82620802e-01],
       [6.67619232e-05, 1.19535901e-01, 8.80397337e-01],
       [1.60407457e-03, 4.40522497e-01, 5.57873429e-01],
       [3.90465598e-05, 9.35554232e-02, 9.06405530e-01],
       [6.18835283e-06, 2.03228444e-02, 9.79670967e-01],
       [9.80090824e-05, 1.20781073e-01, 8.79120918e-01],
       [2.40478186e-04, 1.62637592e-01, 8.37121929e-01],
       [2.00921845e-06, 1.26057079e-02, 9.87392283e-01],
       [3.73036310e-06, 1.21304884e-02, 9.87865781e-01],
       [5.50264541e-05, 8.02057758e-02, 9.19739198e-01],
       [2.25169948e-04, 2.51893935e-01, 7.47880895e-01],
       [1.36503465e-04, 1.57222283e-01, 8.42641214e-01],
       [4.47860511e-05, 3.85093811e-02, 9.61445833e-01],
       [4.70348664e-04, 2.34999252e-01, 7.64530399e-01]])
```

In [48]:
```
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=.3, random_state=42)
```

LogisticRegression

In [49]:
```
LR_model = LogisticRegression()
```

In [50]:
```
LR_model.fit(xtrain, ytrain)
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
```

Out[50]:
```
▾ LogisticRegression
LogisticRegression()
```

In [51]:
```
smote = SMOTE(random_state=42)
xtrain, ytrain = smote.fit_resample(xtrain, ytrain)
```

RandomForestClassifier

In [52]:
```
RFC = RandomForestClassifier()
```

In [53]:
```
RFC.fit(xtrain, ytrain)
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
```

Out[53]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

XGboost

In [54]:
```
XGB = xgb()
```

In [55]:
```
XGB.fit(xtrain, ytrain)
```

Out[55]:
```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
```

SVC

In [56]:
```
svc = SVC(kernel='rbf', gamma='scale', probability=True)
```

In [57]:
```
svc.fit(xtrain, ytrain)
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
```

Out[57]:
```
SVC
SVC(probability=True)
```

Predcting and Evaluation Metrics

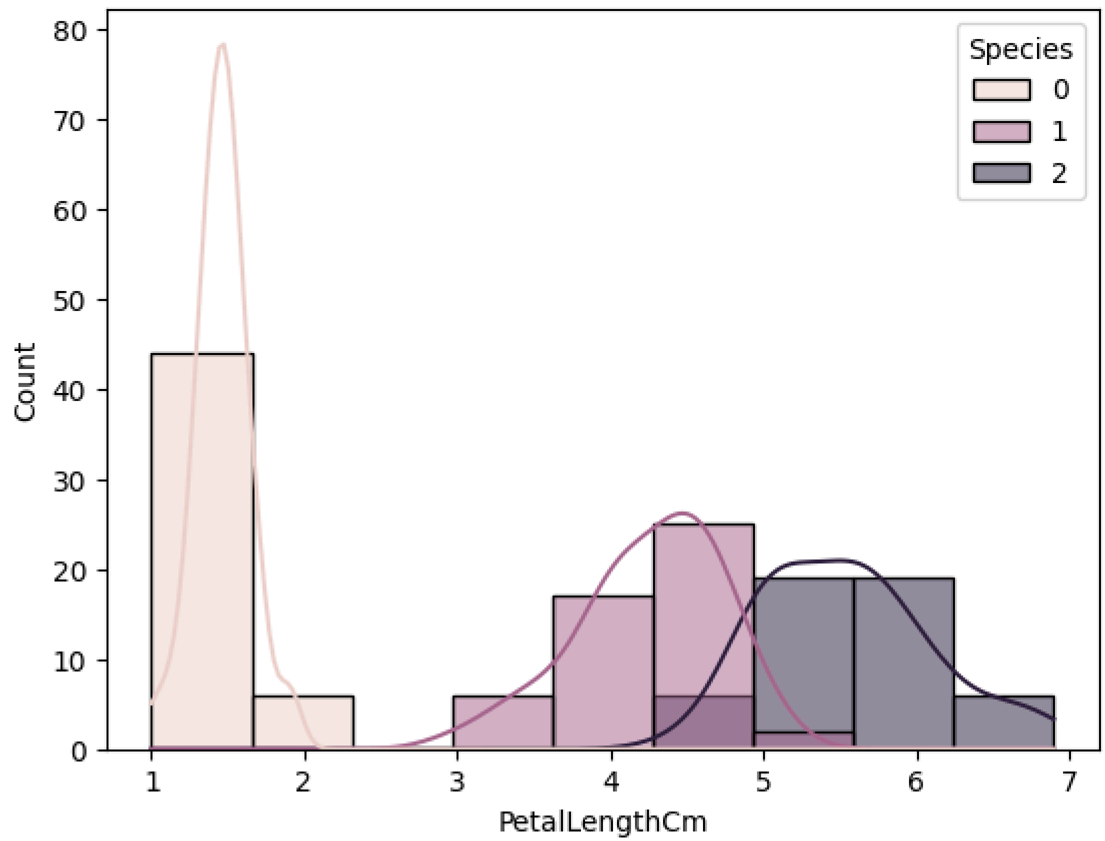
In [58]:
```
trained_models = [
    LR_model,
    RFC,
    XGB,
    svc
]
```

In [59]:
```
for model in trained_models:
    y_pred = model.predict(xtest)
    accuracy = accuracy_score(ytest, y_pred)
    precision = precision_score(ytest, y_pred, average='micro')
    recall = recall_score(ytest, y_pred, average='micro')
    f1 = f1_score(ytest,y_pred, average='micro')
    cm = confusion_matrix(ytest, y_pred)


    proba = model.predict_proba(xtest)
    auc = roc_auc_score(ytest,proba,multi_class='ovr')

    print(f"\t ********* {model.__class__.__name__} *********")
    print("Accuracy Score: ", accuracy)
    print("Precision Score: ", precision)
    print("Recall Score: ", recall)
    print("F1 Score: ", f1)
    print("AUC Score: ", auc)
    print("Confusion Matrix: \n", cm)
    sns.heatmap(cm)
    plt.show()

    fpr, tpr, thresold = roc_curve(ytest, proba[:,1], pos_label=1)
    plt.plot(fpr, tpr, linestyle="--", label="CURVE", )
    plt.title("ROC CURVE")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()
```

```
         ********* LogisticRegression *********
Accuracy Score:  1.0
Precision Score:  1.0
Recall Score:  1.0
F1 Score:  1.0
AUC Score:  1.0
Confusion Matrix: 
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```


```
In [60]: sns.histplot(x = "PetalLengthCm", data = df2, kde=True, hue="Species")
```

```
Out[60]: <Axes: xlabel='PetalLengthCm', ylabel='Count'>
```



RandomForest Classifier has the highest accuracy. So we will use Gradient Boosting Classifier for hyper parameter tuning

Hyperparameter Tuning:

In [61]:

```
params = {
    'n_estimators': [100, 200, 300],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3],
    'max_features': ['auto', 'sqrt', 'log2']
}
```

In [62]:
```
model_param = {
    'svm': {
        'model': SVC(gamma='auto'),
        'params': {
            'C': [1.0, 5.0, 10.0],
            'kernel': ['rbf', 'linear']
        }
    },
    'LogReg': {
        'model': LogisticRegression(solver='liblinear'),
        'params': {
            'C': [1.0, 5.0, 10.0],
            'penalty': ['l1', 'l2'],
        }
    },
    'rf': {
        'model': RandomForestClassifier(),
        'params': {
            'n_estimators': [20, 50, 100],
            'criterion': ['gini', 'entropy'],
            'min_samples_leaf' : [1, 2],
            'max_features': ['sqrt', 'log2']
        }
    },
    'XGboost': {
        'model': xgb(),
        'params': {
            'learning_rate': [0.1, 0.01, 0.2],
            'n_estimators': [20, 50, 100],
        }
    }
}
```

In [63]:

```
scores = []

for name, mp in model_param.items():
    RandomSearch = RandomizedSearchCV(estimator=mp['model'] , param_distributions=mp['params'], return_tr
    RandomSearch.fit(xtrain, ytrain)
    scores.append({
        'model': mp['model'],
        'best_score': RandomSearch.best_score_,
        'best_param': RandomSearch.best_params_
    })
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
```

In [64]:
```
scores
```

Out[64]:
```
[{'model': SVC(gamma='auto'),
  'best_score': 0.9636363636363636,
  'best_param': {'kernel': 'linear', 'C': 1.0}},
 {'model': LogisticRegression(solver='liblinear'),
  'best_score': 0.9458498023715414,
  'best_param': {'penalty': 'l2', 'C': 1.0}},
 {'model': RandomForestClassifier(),
  'best_score': 0.9458498023715414,
  'best_param': {'n_estimators': 100,
   'min_samples_leaf': 2,
   'max_features': 'sqrt',
   'criterion': 'gini'}},
 {'model': XGBClassifier(base_score=None, booster=None, callbacks=None,
                colsample_bylevel=None, colsample_bynode=None,
                colsample_bytree=None, device=None, early_stopping_rounds=None,
                enable_categorical=False, eval_metric=None, feature_types=None,
                gamma=None, grow_policy=None, importance_type=None,
                interaction_constraints=None, learning_rate=None, max_bin=None,
                max_cat_threshold=None, max_cat_to_onehot=None,
                max_delta_step=None, max_depth=None, max_leaves=None,
                min_child_weight=None, missing=nan, monotone_constraints=None,
                multi_strategy=None, n_estimators=None, n_jobs=None,
                num_parallel_tree=None, random_state=None, ...),
  'best_score': 0.9367588932806324,
  'best_param': {'n_estimators': 20, 'learning_rate': 0.1}}]
```

Predict testing values with hyperparameter tuning

In [65]:
```
rf = RandomForestClassifier(n_estimators=50, min_samples_leaf=1, criterion='entropy')
rf.fit(xtrain, ytrain)
b_predicted = rf.predict(xtest)
```

```
A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_sample
s,), for example using ravel().
```

In [66]:
```python
accuracy_score(ytest, b_predicted)
```

Out[66]:
```
1.0
```

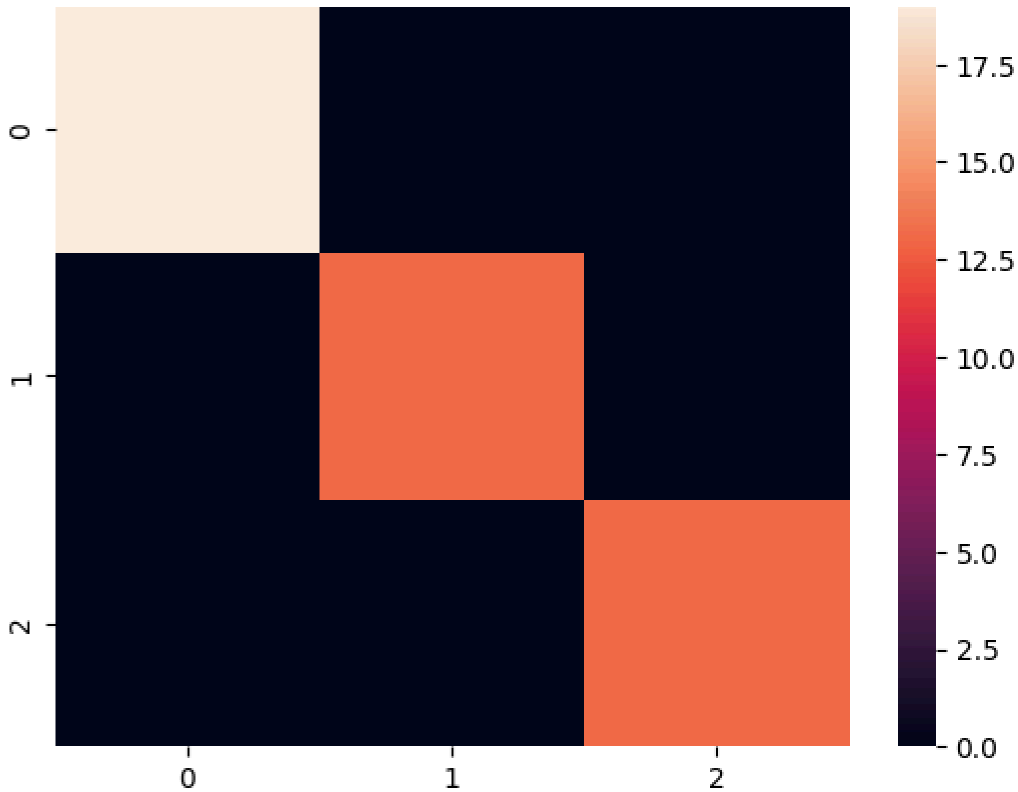
In [67]:
```python
accuracy = accuracy_score(ytest, b_predicted)
precision = precision_score(ytest, b_predicted, average='micro')
recall = recall_score(ytest, b_predicted, average='micro')
f1 = f1_score(ytest, b_predicted, average='micro')
cm = confusion_matrix(ytest, b_predicted)
```

In [68]:
```python
print("Accuracy Score: ", accuracy)
print("Precision Score: ", precision)
print("Recall Score: ", recall)
print("F1 Score: ", f1)
print("AUC Score: ", auc)
print("Confusion Matrix: \n", cm)
```

```
Accuracy Score:  1.0
Precision Score:  1.0
Recall Score:  1.0
F1 Score:  1.0
AUC Score:  1.0
Confusion Matrix: 
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
```

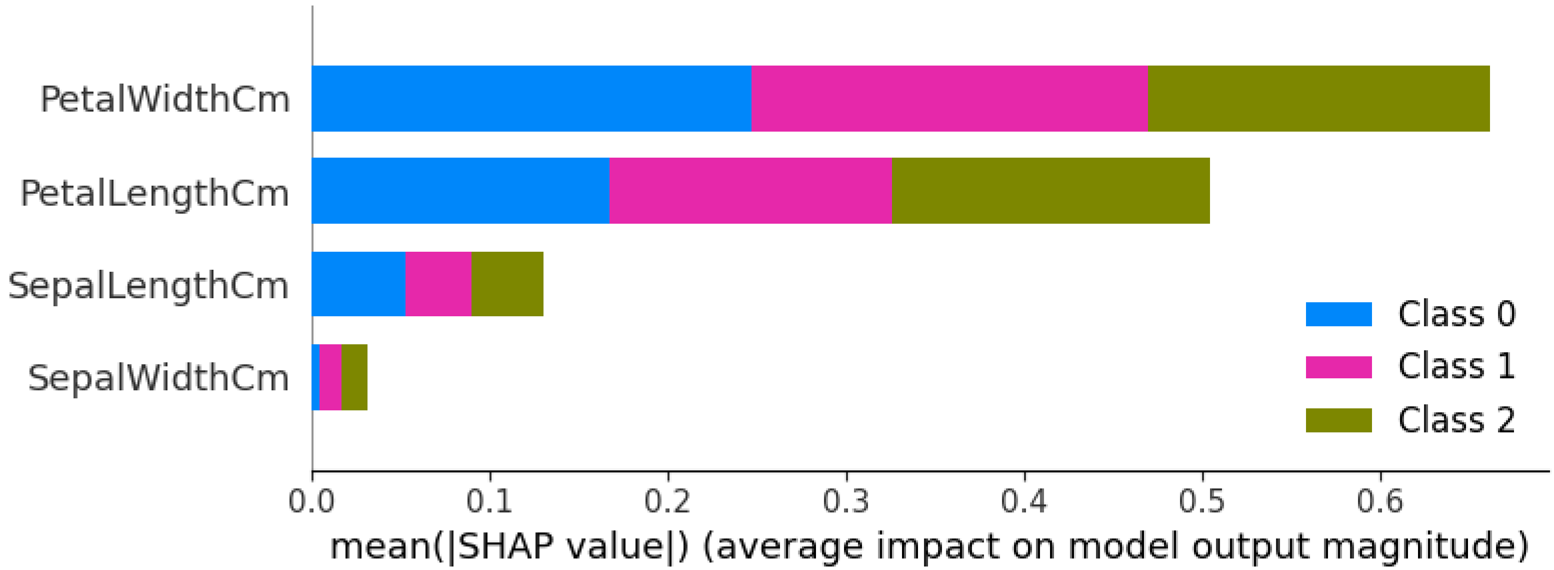
In [69]:
```
sns.heatmap(cm)
```

Out[69]: <Axes: >



Interpretability:

In [70]:
```
explainer = shap.Explainer(rf)
shap_values = explainer.shap_values(xtest)
shap.summary_plot(shap_values, xtest)
```



Visualization :

In [71]:
```
df3.head()
```

Out[71]:

	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 [72]:
```python
x =df.drop('PetalWidthCm',axis=1)
y = df[['PetalWidthCm']]
x.head()
```

Out[72]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	Species_Value
0	5.1	3.5	1.4	0
1	4.9	3.0	1.4	0
2	4.7	3.2	1.3	0
3	4.6	3.1	1.5	0
4	5.0	3.6	1.4	0

Import ML Algorithm

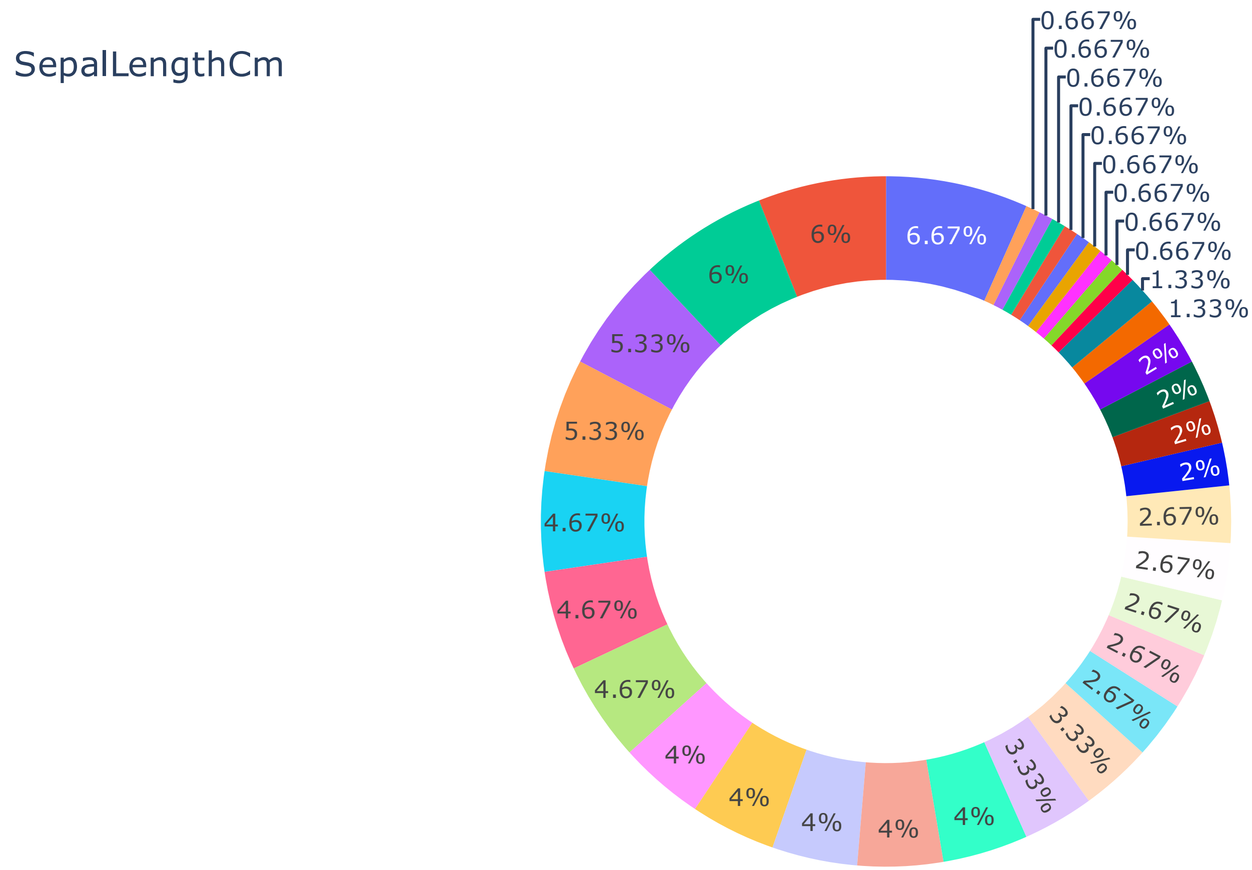
In [73]:
```python
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
```

In [74]:
```python
print('Species_Value in 100%')
round(df.Species_Value.value_counts()*100/len(df),1)
```

```
Species_Value in 100%
```

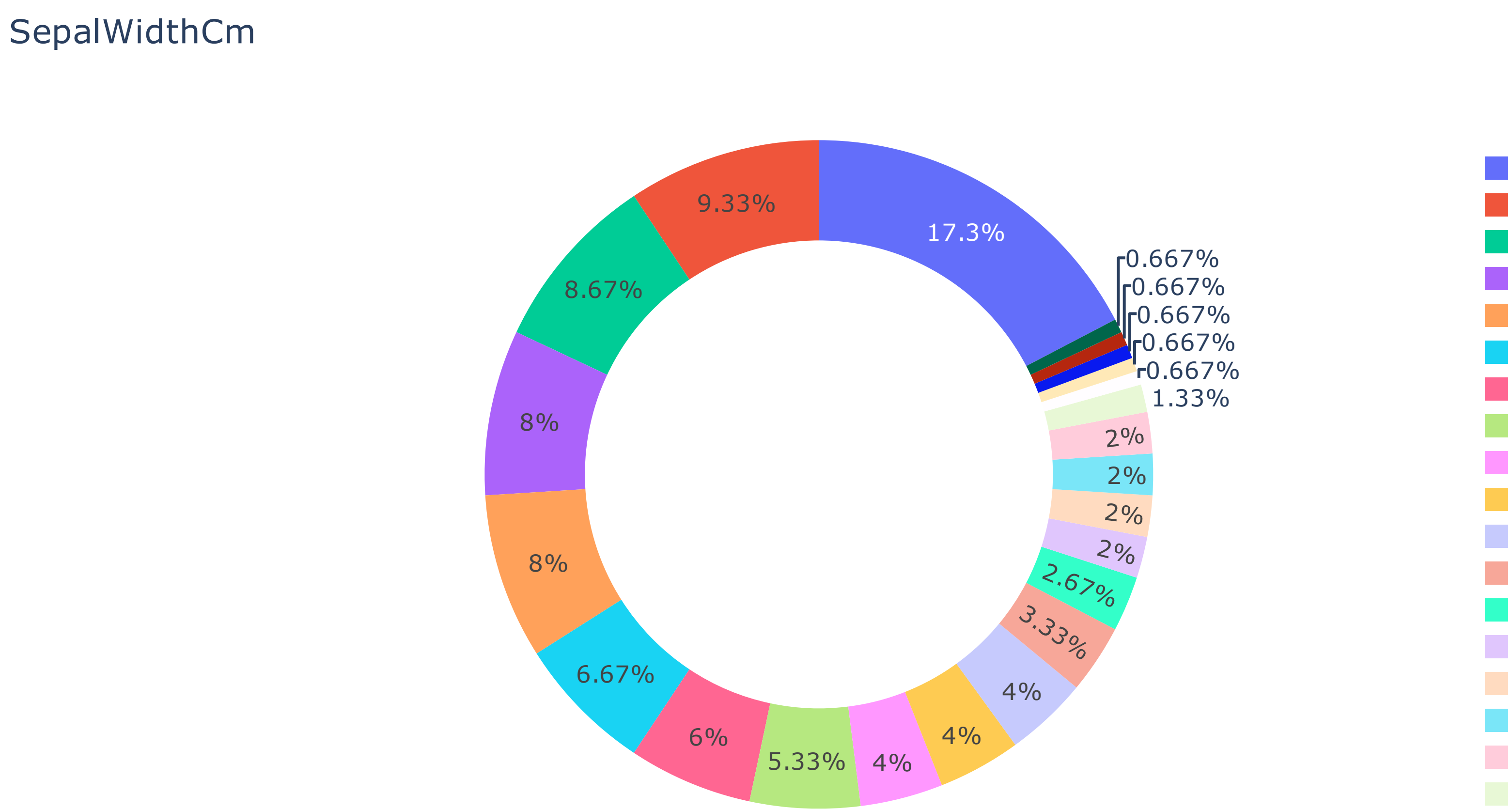
Out[74]:
```
0    33.3
1    33.3
2    33.3
Name: Species_Value, dtype: float64
```

```
In [75]: SepalLengthCm=df3['SepalLengthCm'].value_counts()
         transctions = SepalLengthCm.index
         quantity =SepalLengthCm.values
         figure =px.pie(df3,
                        values=quantity,
                        names =transctions,hole=.70,
                        title=" SepalLengthCm  ")
         figure.show()
```

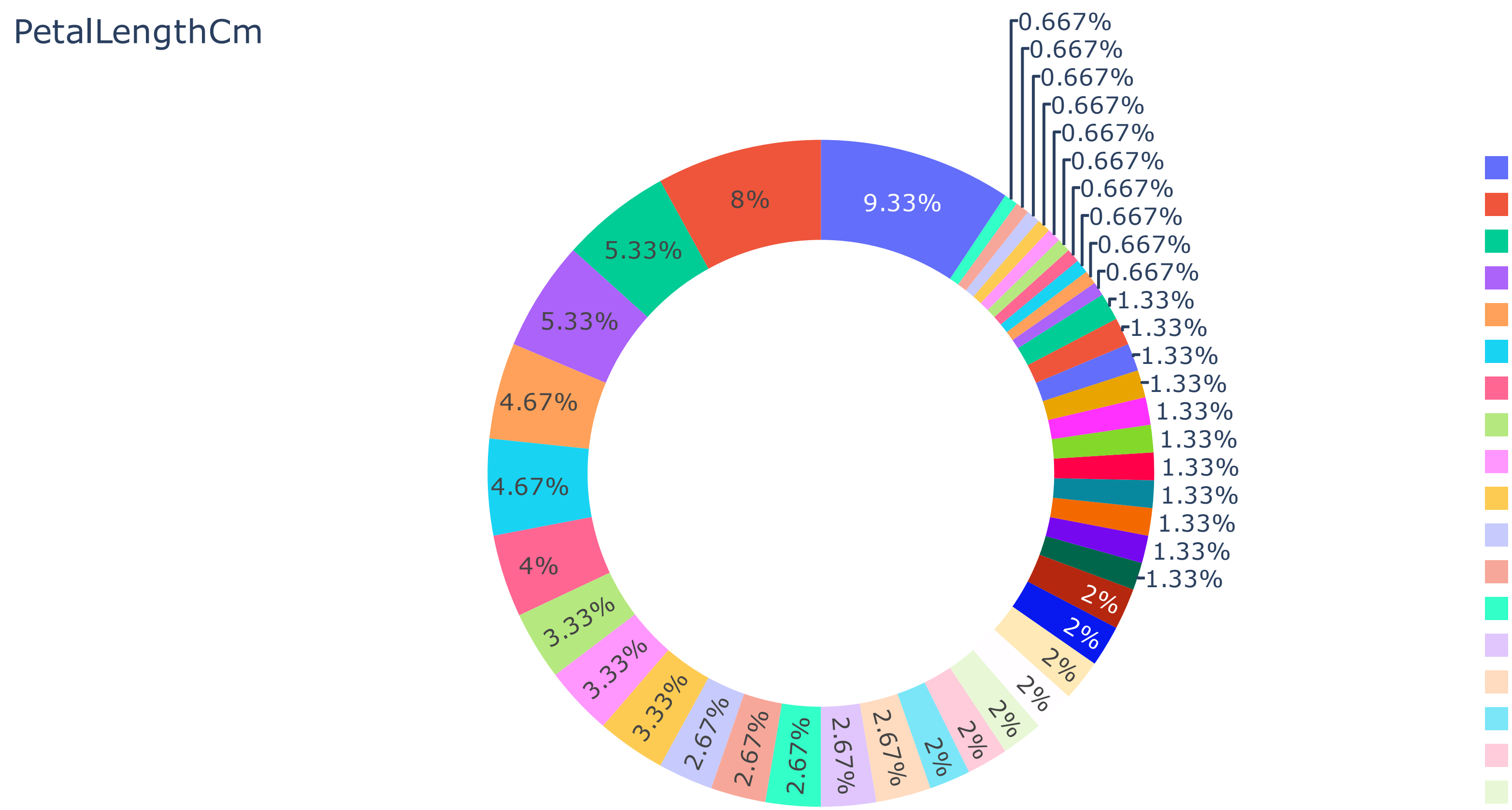


In [76]:

```
SepalWidthCm=df3['SepalWidthCm'].value_counts()
transctions = SepalWidthCm.index
quantity =SepalWidthCm.values
figure =px.pie(df3,
               values=quantity,
               names =transctions,hole=.70,
               title=" SepalWidthCm ")
figure.show()
```



In [77]:
```
PetalLengthCm=df3['PetalLengthCm'].value_counts()
transctions = PetalLengthCm.index
quantity =PetalLengthCm.values
figure =px.pie(df3,
               values=quantity,
               names =transctions,hole=.70,
               title=" PetalLengthCm ")
figure.show()
```



In [ ]:

```

```

Pandas Profiling Report of Iris Flower Classification

In [80]:
```
import pandas as pd
from ydata_profiling import ProfileReport
ProfileReport(x, title="Iris Flower Classification_Report")
```

Summarize dataset: 100% 22/22 [00:02<00:00, 5.04it/s, Completed]

Generate report structure: 100% 1/1 [00:01<00:00, 1.91s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.95it/s]

Overview

Dataset statistics

Number of variables	4
Number of observations	150
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	6
Duplicate rows (%)	4.0%
Total size in memory	4.2 KiB
Average record size in memory	28.9 B

Variable types

Numeric	3
Categorical	1

Alerts

Dataset has 6 (4.0%) duplicate rows	**Duplicates**
`PetalLengthCm` is highly overall correlated with `SepalLengthCm` and 1 other fields (SepalLengthCm, Species_Value)	**High correlation**
`SepalLengthCm` is highly overall correlated with `PetalLengthCm` and 1 other fields (PetalLengthCm, Species_Value)	**High correlation**
`Species_Value` is highly overall correlated with `PetalLengthCm` and 1 other fields (PetalLengthCm, SepalLengthCm)	**High correlation**

(PetalLengthCm, SepalLengthCm)

`Species` `Value` is uniformly distributed Uniform

Out[80]:

In [ ]: