In [1]:

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

In [2]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [3]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
```

In [4]:

```
#ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

Collecting Data

In [5]:

```
Idata=pd.read_csv("Iris (1).csv")
```

In [6]:

```
Idata
```

Out[6]:

	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
...	...	...	...	...	...	...
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

150 rows × 6 columns

In [7]:

```
Idata.head()
```

Out[7]:

	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 [8]:

```
Idata.isnull().sum()
```

Out[8]:

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

In [9]:

```
Idata.shape
```

Out[9]:

```
(150, 6)
```

In [10]:

```
Idata.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 [11]:

```
Idata.describe()
```

Out[11]:

	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

In [12]:

```
Idata.columns
```

Out[12]:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [ ]:

Feature Engineering

In [13]:

```
Idata.head(10)
```

Out[13]:

	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

Feature Engineering

In [15]:

```
#creating instance of a label encoder
encode = LabelEncoder()

#Assigning numerical values and storing in the same name column 'species'
Idata.Species = encode.fit_transform(Idata.Species)
```

In [17]:
```
print(Idata.head(10))
```

```
   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
```

In [18]:
```
# species column values are changed to numerical form
Idata.Species.unique()
```

Out[18]:
```
array([0, 1, 2])
```

In [20]:
```
#train test split
train, test = train_test_split(Idata, test_size = 0.2, random_state = 0)
```

In [21]:
```
print(train.shape)
```

```
(120, 6)
```

In [22]:
```
#Seperate the target and independent variable
train_X = train.drop(columns = ['Species'], axis = 1)
train_Y = train['Species']

test_X = test.drop(columns = ['Species'], axis = 1)
test_Y = test['Species']
```

In [23]:
```
print(train_X.shape, train_Y.shape)
```

```
(120, 5) (120,)
```

In [24]:
```
print(test_X.shape, test_Y.shape)
```

```
(30, 5) (30,)
```

In [26]:
```
Idata['Species'].value_counts()
```

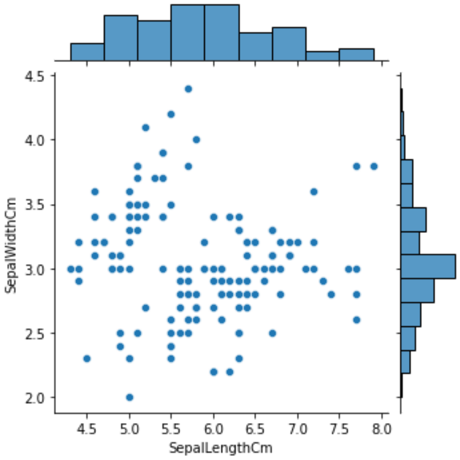
Out[26]:
```
0    50
1    50
2    50
Name: Species, dtype: int64
```

In [30]:
```
d=Idata.copy()
```

Joint plot

In [31]:
```
sns.jointplot(x='SepalLengthCm',y='SepalWidthCm',data=d,size=5)
```

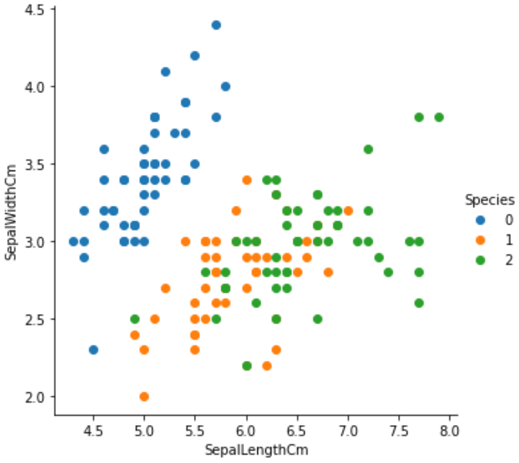
Out[31]:
```
<seaborn.axisgrid.JointGrid at 0x1de196ef280>
```



FacetGrid Plot

```
In [33]: import matplotlib.pyplot as plt
         %matplotlib inline
         sns.FacetGrid(d,hue='Species',size=5)\
         .map(plt.scatter,'SepalLengthCm','SepalWidthCm')\
         .add_legend()
```

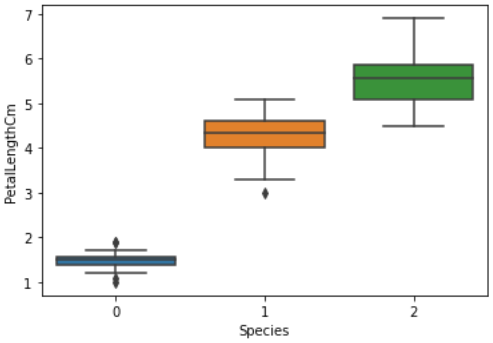
```
Out[33]: <seaborn.axisgrid.FacetGrid at 0x1de1990a670>
```



Boxplot

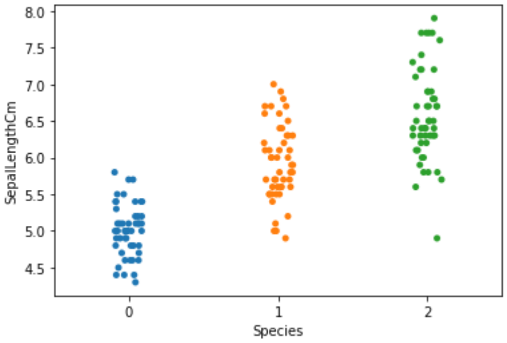
```
In [34]: sns.boxplot(x='Species',y='PetalLengthCm',data=d)
```

```
Out[34]: <AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>
```



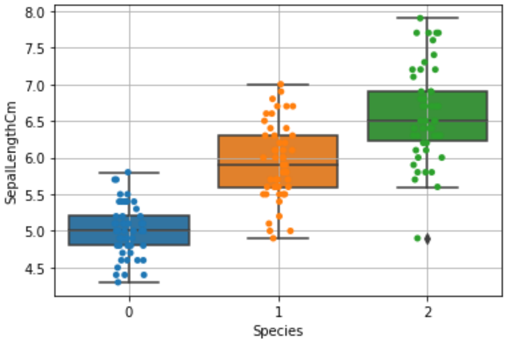
Strip plot

```
In [35]: ax=sns.stripplot(x='Species',y='SepalLengthCm',data=d,jitter=True,edgecolor='gray')
```



Combining Box and Strip Plots

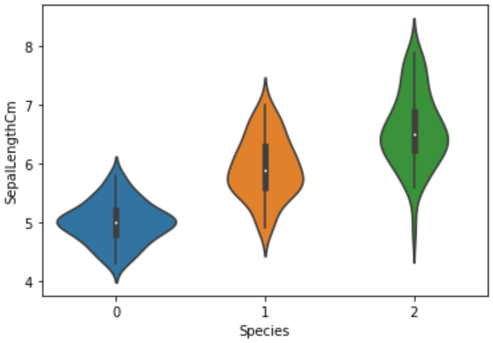
```
In [36]: ax=sns.boxplot(x='Species',y='SepalLengthCm',data=d)
         ax=sns.stripplot(x='Species',y='SepalLengthCm',data=d,jitter=True,edgecolor='gray')
         plt.grid()
```



Violin Plot

```
In [37]: sns.violinplot(x='Species',y='SepalLengthCm',data=d,size=8)
```

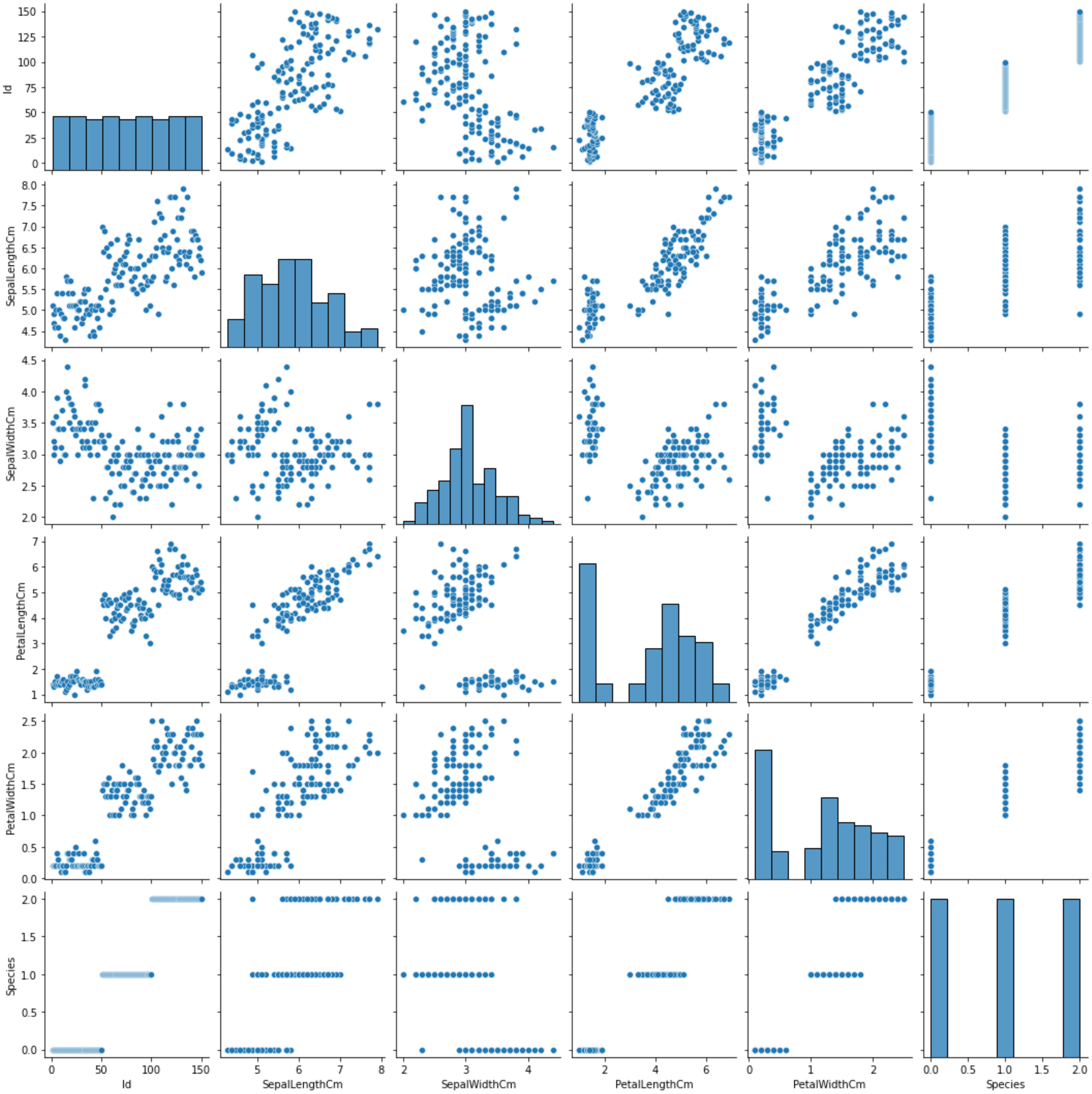
```
Out[37]: <AxesSubplot:xlabel='Species', ylabel='SepalLengthCm'>
```



Pair Plot

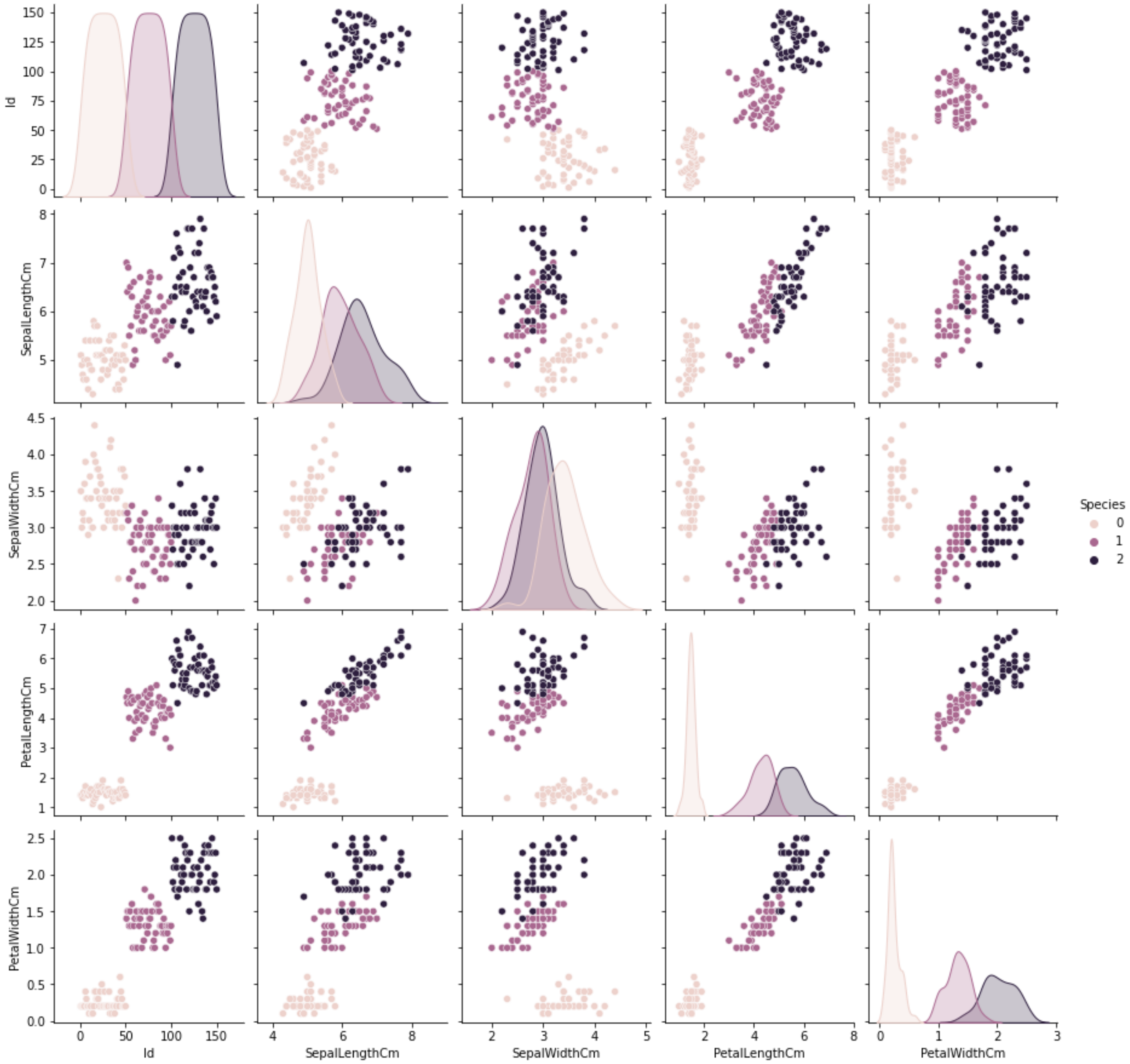
```
In [38]: sns.pairplot(data=d,kind='scatter')
```

```
Out[38]: <seaborn.axisgrid.PairGrid at 0x1de19bb65e0>
```



```
In [39]: sns.pairplot(d,hue='Species')
```

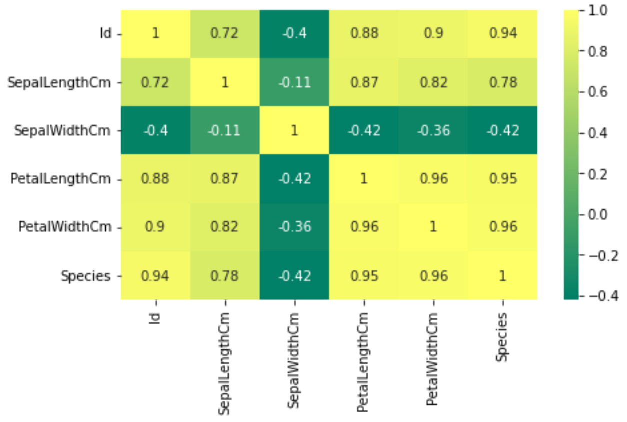
```
Out[39]: <seaborn.axisgrid.PairGrid at 0x1de19acdfd0>
```



Plotting heat map

```
In [42]: plt.figure(figsize=(7,4))
         sns.heatmap(d.corr(),annot=True,cmap='summer')
```

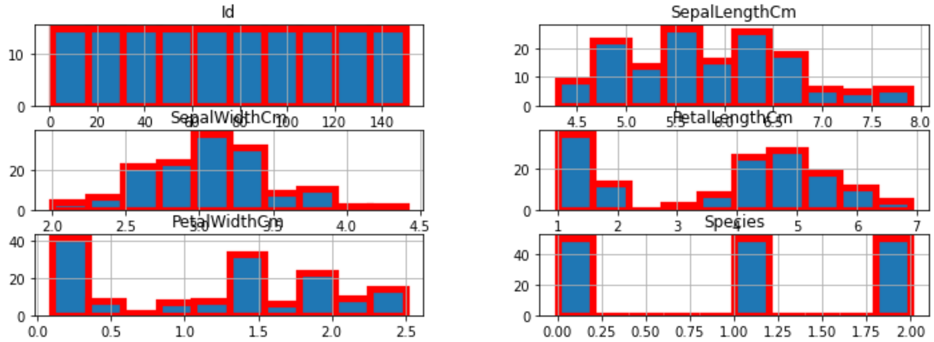
```
Out[42]: <AxesSubplot:>
```



Distribution plot

In [48]:

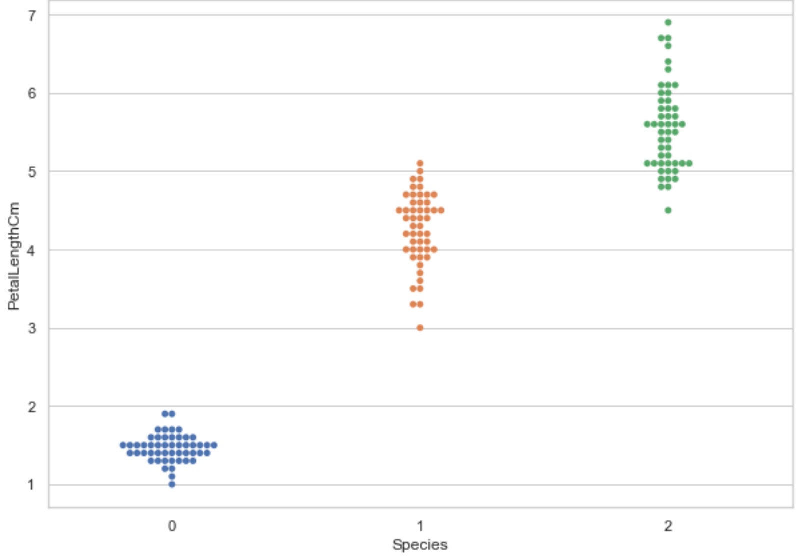
```
d.hist(edgecolor='red', linewidth=5.2)
fig=plt.gcf()
fig.set_size_inches(12,4)
plt.show()
```



Swarm plot

In [49]:

```
sns.set(style="whitegrid")
fig=plt.gcf()
fig.set_size_inches(10,7)
fig = sns.swarmplot(x="Species", y="PetalLengthCm", data=d)
```



In [51]:

```
d
```

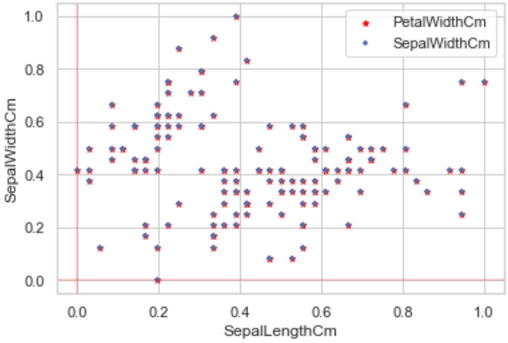
Out[51]:

	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
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

150 rows × 6 columns

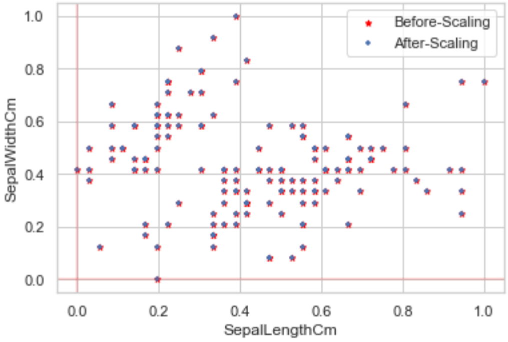
In [57]:
```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
d=pd.DataFrame(scaler.fit_transform(d),columns=['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm','Species']
ax=d.plot.scatter(x="SepalLengthCm",y="SepalWidthCm",marker="*",label="PetalWidthCm",color="red")
d.plot.scatter(x="SepalLengthCm",y="SepalWidthCm",marker="+",label="SepalWidthCm",ax=ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2)
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```



In [59]:
```
from sklearn.preprocessing import MaxAbsScaler
maxabsscaler=MaxAbsScaler()
d=pd.DataFrame(maxabsscaler.fit_transform(d),columns=['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm','Spe
ax=d.plot.scatter(x="SepalLengthCm",y="SepalWidthCm",marker="*",label="Before-Scaling",color="red")
d.plot.scatter(x="SepalLengthCm",y="SepalWidthCm",marker="+",label="After-Scaling",ax=ax)
plt.axhline(0, color='red',alpha=0.2)
plt.axvline(0, color='red',alpha=0.2);
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in
case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you
intend to specify the same RGB or RGBA value for all points.
```



In [60]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, [0, 2]]
Y = iris.target
```

In [61]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
```

In [62]:
```
from sklearn.linear_model import Perceptron

prcptrn = Perceptron(eta0=0.1, random_state=1)
prcptrn.fit(X_train, Y_train)
```

Out[62]:
```
Perceptron(eta0=0.1, random_state=1)
```

In [63]:

```
from sklearn.metrics import accuracy_score
Y_predict = prcptrn.predict(X_test)
print("Misclassified examples %d" %(Y_test != Y_predict).sum())
print("Accuracy Score %.3f" %accuracy_score(Y_test, Y_predict))
```

```
Misclassified examples 19
Accuracy Score 0.578
```

In [64]:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

In [65]:

```
prcptrnFS = Perceptron(eta0=0.1, random_state=1)
prcptrnFS.fit(X_train_std, Y_train)

Y_predict_std = prcptrnFS.predict(X_test_std)
print("Misclassified examples %d" %(Y_test != Y_predict_std).sum())

from sklearn.metrics import accuracy_score
print("Accuracy Score %0.3f" % accuracy_score(Y_test, Y_predict_std))
```

```
Misclassified examples 1
Accuracy Score 0.978
```

In [ ]: