Iris Dataset Visualization

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
#plt.style.use('fivethirtyeight')
import warnings
warnings.filterwarnings('ignore')  #this will ignore the warnings.it wont displa
```

Importing Iris data set

In [3]:
```
iris = pd.read_csv(r'C:\Users\91939\Desktop\AI&DS\20thAug\19th, 20th\19th, 20th\
```

In [4]:
```
iris
```

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
...	...	...	...	...	...	...
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 [5]:
```
iris.head()
```

Out[5]:

	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 [6]:
```
iris.shape
```

Out[6]: (150, 6)

In [7]:
```
iris.isnull()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows × 6 columns

In [8]:
```
iris[iris.isnull()]
```

Out[8]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...
145	NaN	NaN	NaN	NaN	NaN	NaN
146	NaN	NaN	NaN	NaN	NaN	NaN
147	NaN	NaN	NaN	NaN	NaN	NaN
148	NaN	NaN	NaN	NaN	NaN	NaN
149	NaN	NaN	NaN	NaN	NaN	NaN

150 rows × 6 columns

In [9]:
```
iris.columns
```

Out[9]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [10]:
```
iris.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

In [11]:
```
iris.drop('Id',axis=1,inplace=True)
```

In [12]:
```
iris.head()
```

Out[12]:

	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

Checking if there are any missing values

In [13]:
```
iris.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 [14]:
```
iris['Species'].value_counts()
```

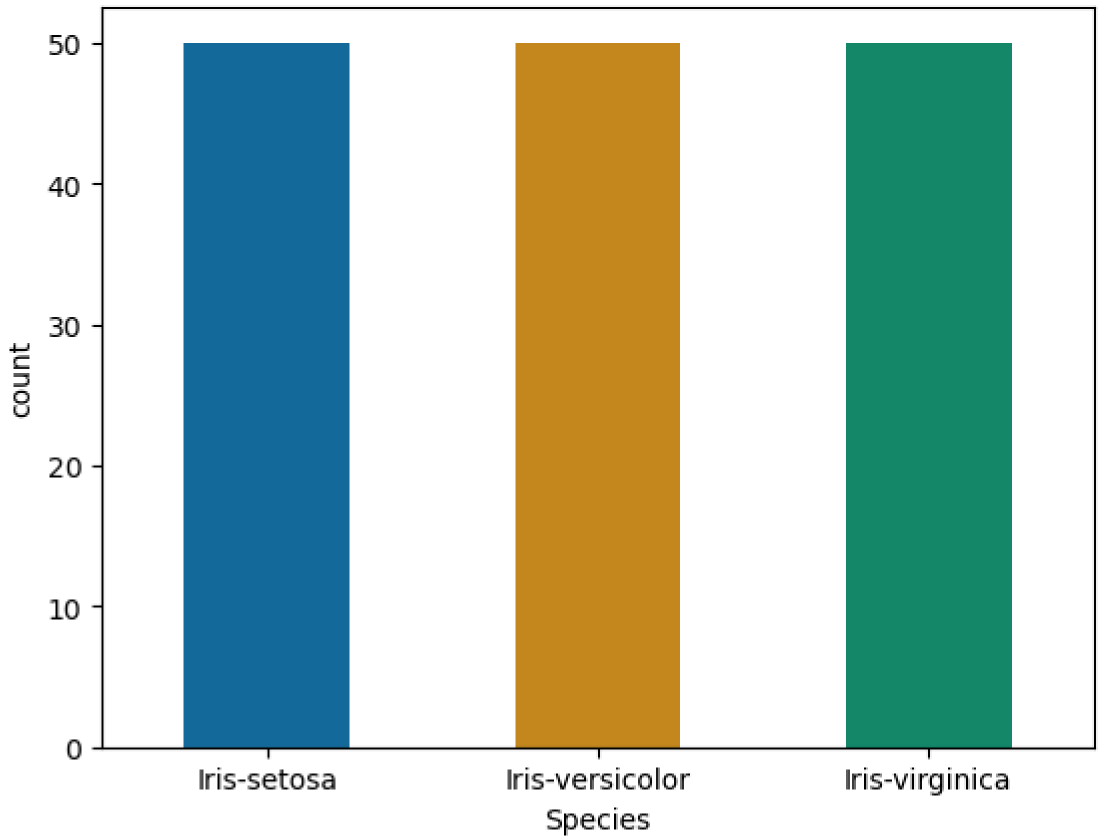
Out[14]:
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

Count Plot :

Show the counts of observations in each categorical bin using bars.

This data set has three varities of Iris plant.

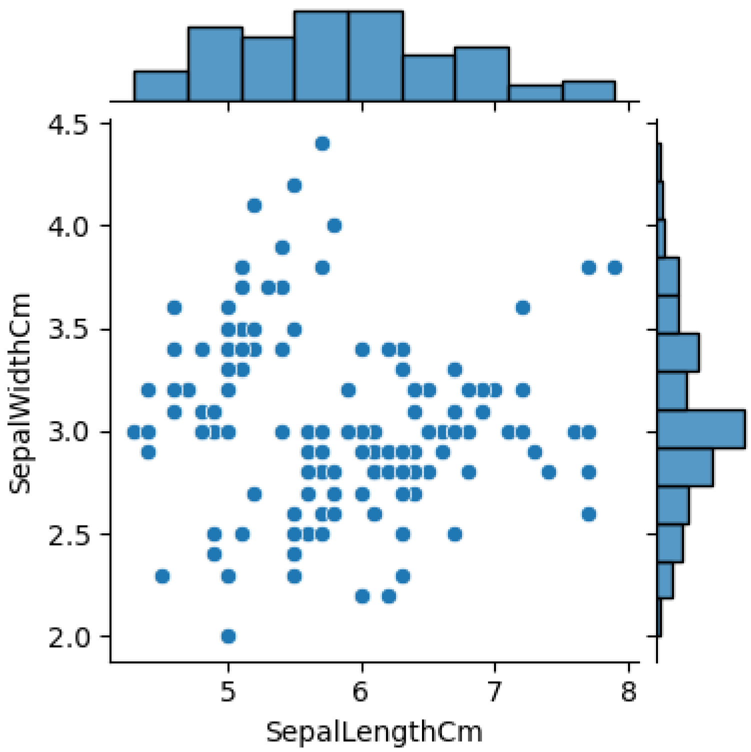
In [15]:
```
sns.countplot(x='Species',data=iris,palette='colorblind',width=0.5)
plt.show()
```



Joint plot:

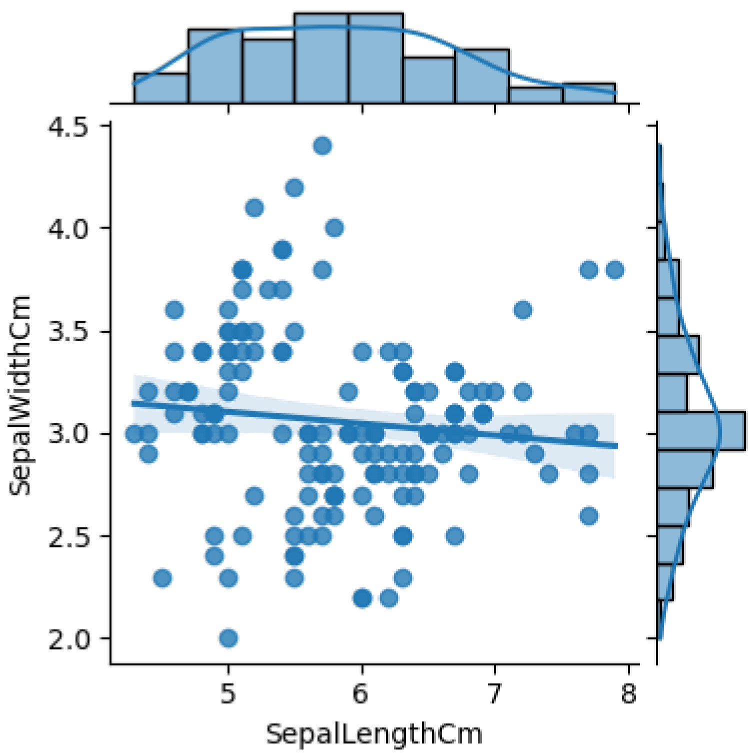
Jointplot is seaborn library specific and can be used to quickly visualize and analyze the relationship between two variables and describe their individual distributions on the same plot.

In [16]:
```
fig=sns.jointplot(x='SepalLengthCm',y='SepalWidthCm',data=iris,height=4)
```

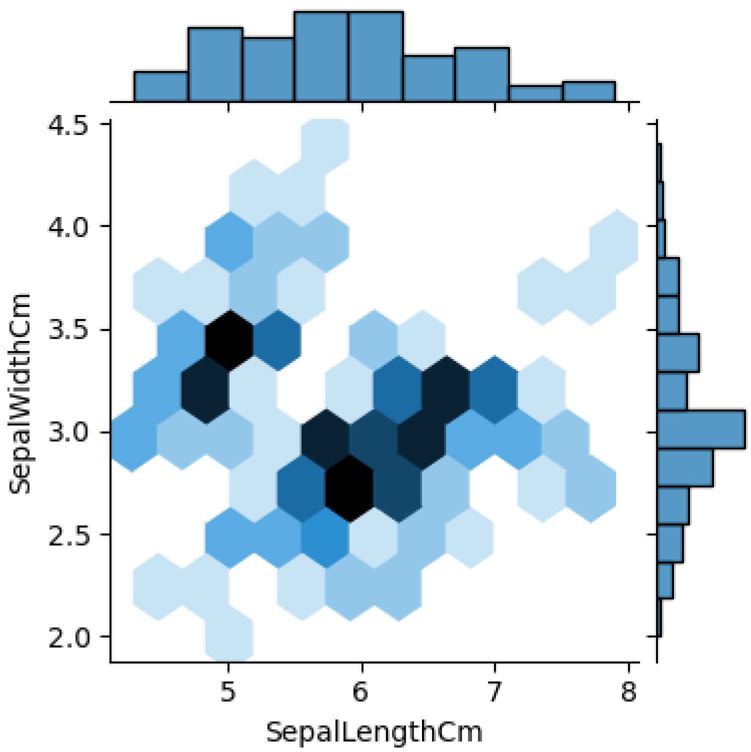


In [17]:
```
sns.jointplot(x="SepalLengthCm", y="SepalWidthCm", data=iris, kind="reg",height=
```

Out[17]: <seaborn.axisgrid.JointGrid at 0x2469f315bb0>



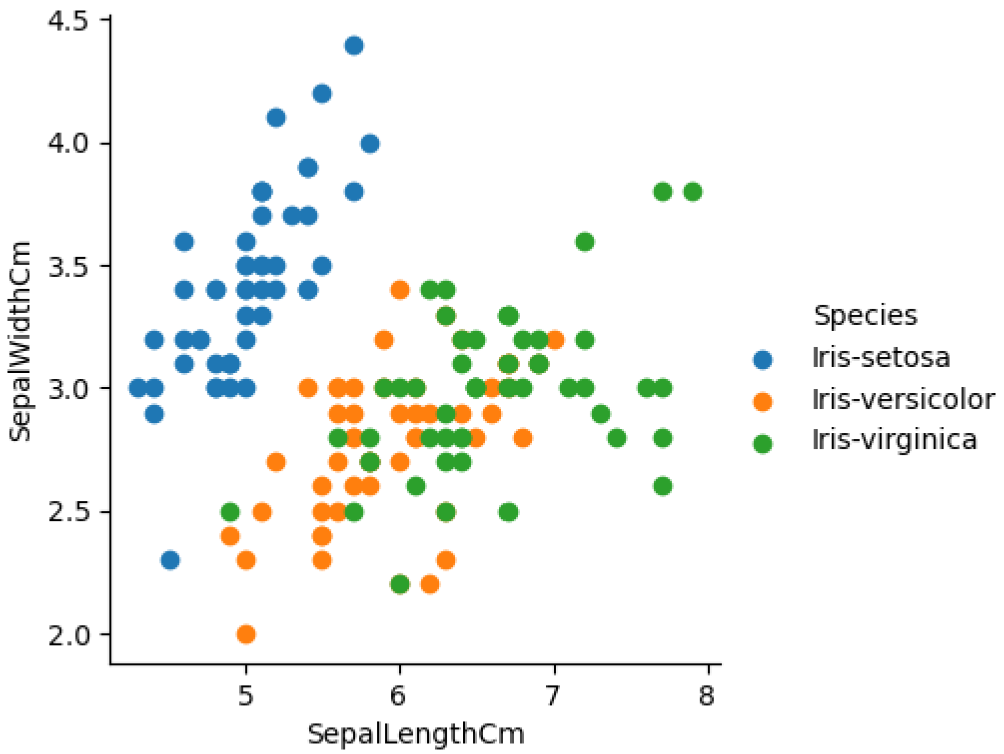
In [18]:
```
fig=sns.jointplot(x='SepalLengthCm',y='SepalWidthCm',kind='hex',data=iris,height
```



In [19]:
```
import matplotlib.pyplot as plt
%matplotlib inline

sns.FacetGrid(iris,hue='Species',height=4)\
.map(plt.scatter,'SepalLengthCm','SepalWidthCm')\
.add_legend()
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x246a46e7560>



Boxplot or Whisker plot

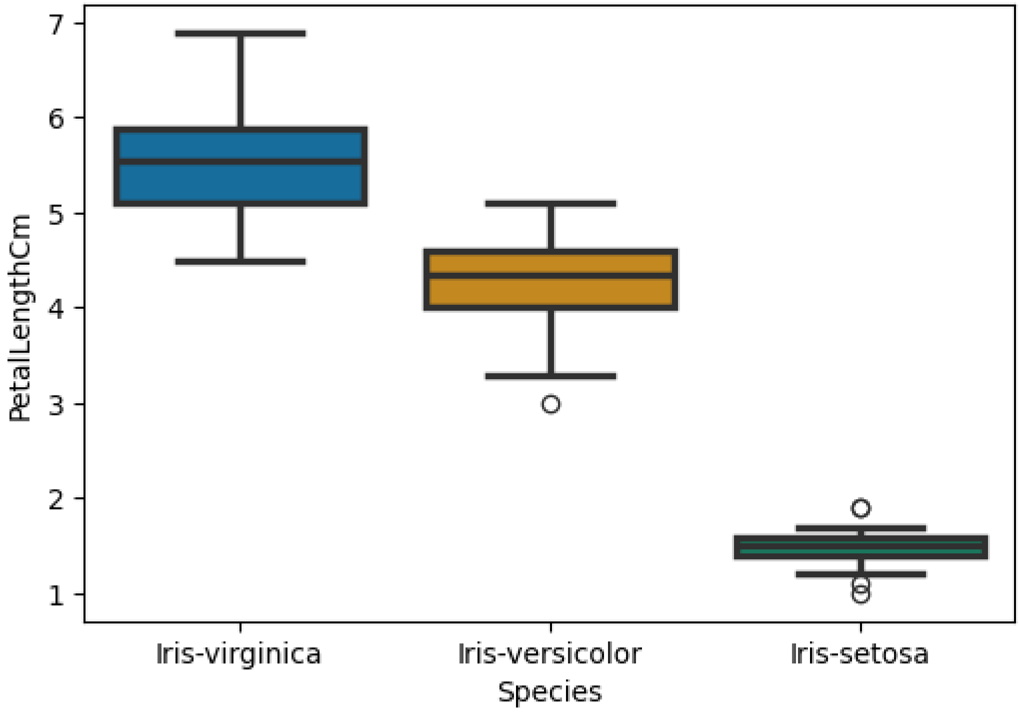
Box plot was was first introduced in year 1969 by Mathematician John Tukey.Box plot give a statical summary of the features being plotted.Top line represent the max value,top edge of box is third Quartile, middle edge represents the median,bottom edge represents the first quartile value.The bottom most line respresent the minimum value of the feature.The height of the box is called as Interquartile range.The black dots on the plot represent the outlier values in the data.

In [20]:
```
iris.head()
```

Out[20]:

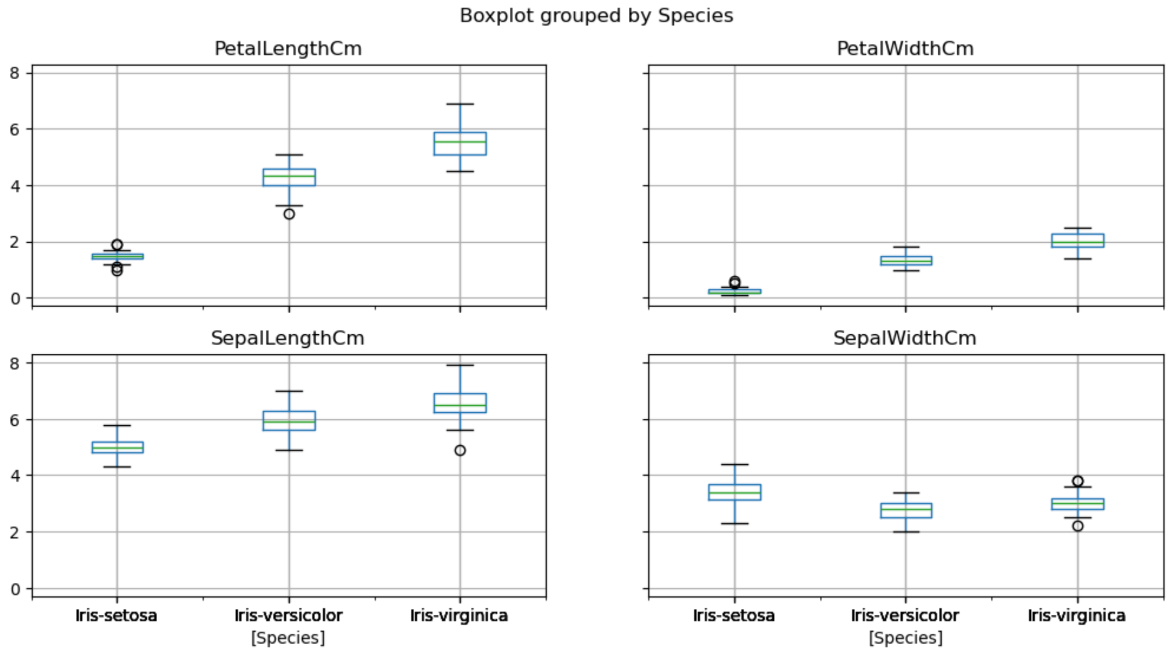
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [21]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
fig=sns.boxplot(x='Species',y='PetalLengthCm',data=iris,order=['Iris-virginica',
```



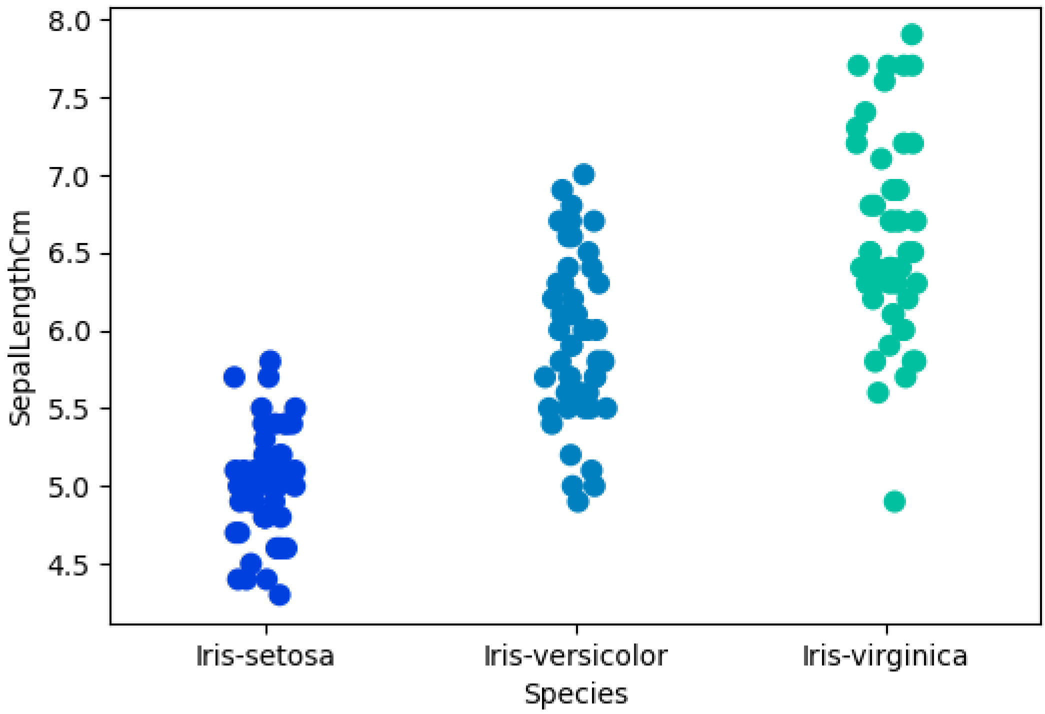
In [22]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
iris.boxplot(by="Species", figsize=(12, 6))
#by : str or array-like, optional
#Column in the DataFrame to :meth:`pandas.DataFrame.groupby`.
#One box-plot will be done per value of columns in `by`.
```

Out[22]:
```
array([[<Axes: title={'center': 'PetalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'PetalWidthCm'}, xlabel='[Species]'>],
       [<Axes: title={'center': 'SepalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'SepalWidthCm'}, xlabel='[Species]'>]],
      dtype=object)
<Figure size 600x400 with 0 Axes>
```



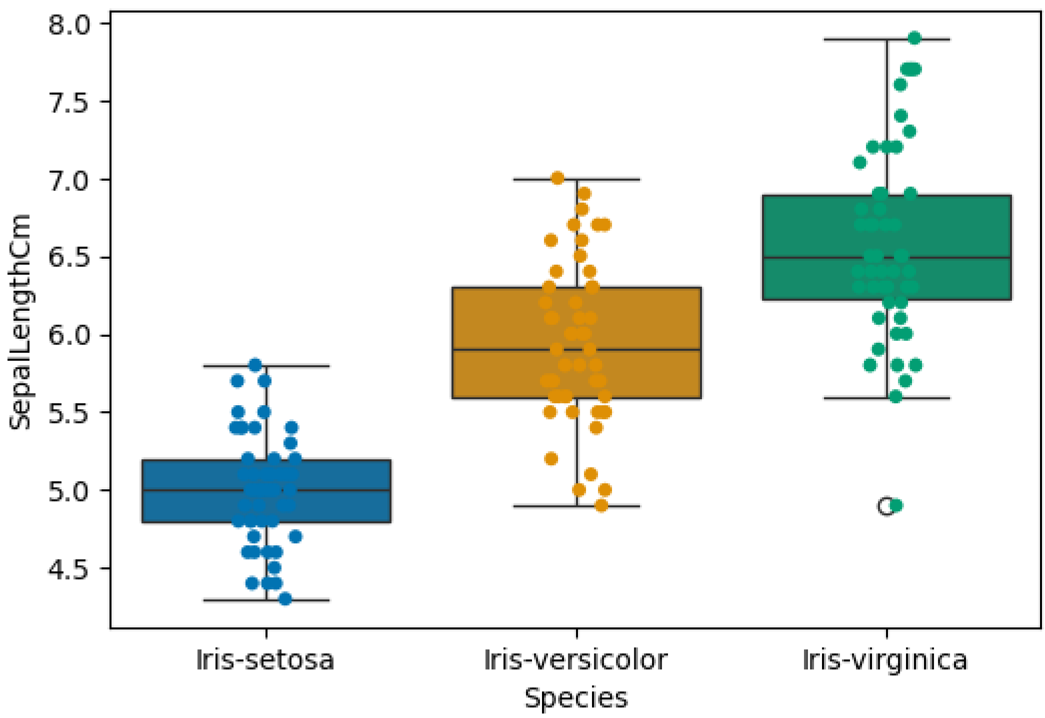
Strip plot

In [23]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
fig=sns.stripplot(x='Species',y='SepalLengthCm',data=iris,jitter=True,edgecolor=
```



Combining Box and Strip Plots

In [24]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
fig=sns.boxplot(x='Species',y='SepalLengthCm',palette='colorblind',data=iris)
fig=sns.stripplot(x='Species',y='SepalLengthCm',data=iris,jitter=True,palette='c
```

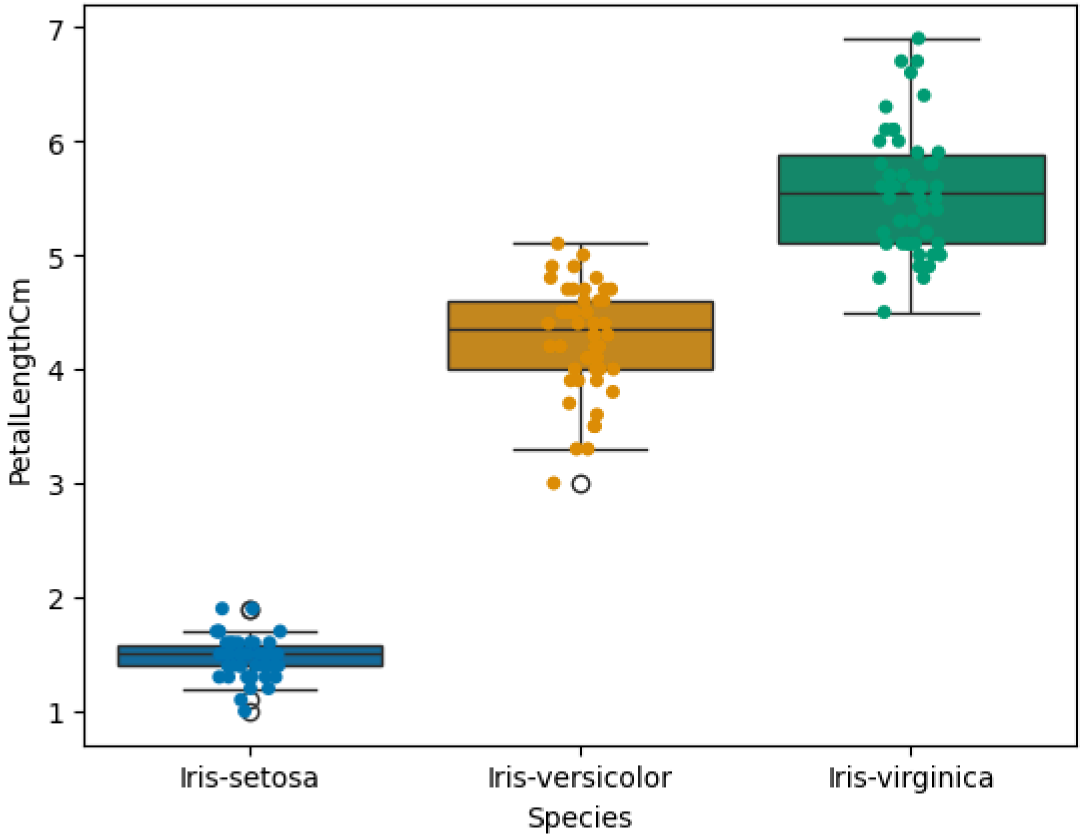


In [25]:
```
# Create the boxplot
ax = sns.boxplot(x="Species", y="PetalLengthCm", palette='colorblind', data=iris
```

```
# Create the stripplot on the same axis
sns.stripplot(x="Species", y="PetalLengthCm", palette='colorblind', data=iris, j

# Iterate through artists and modify boxes (safer approach)
num_boxes = len([artist for artist in ax.artists if isinstance(artist, matplotli
for i in range(num_boxes):
    box = ax.artists[i]
    if i == 0:
        box.set_facecolor('green')
    elif i == 1:
        box.set_facecolor('red')
    elif i == 2:
        box.set_facecolor('yellow')
    box.set_edgecolor('black')

plt.show()
```

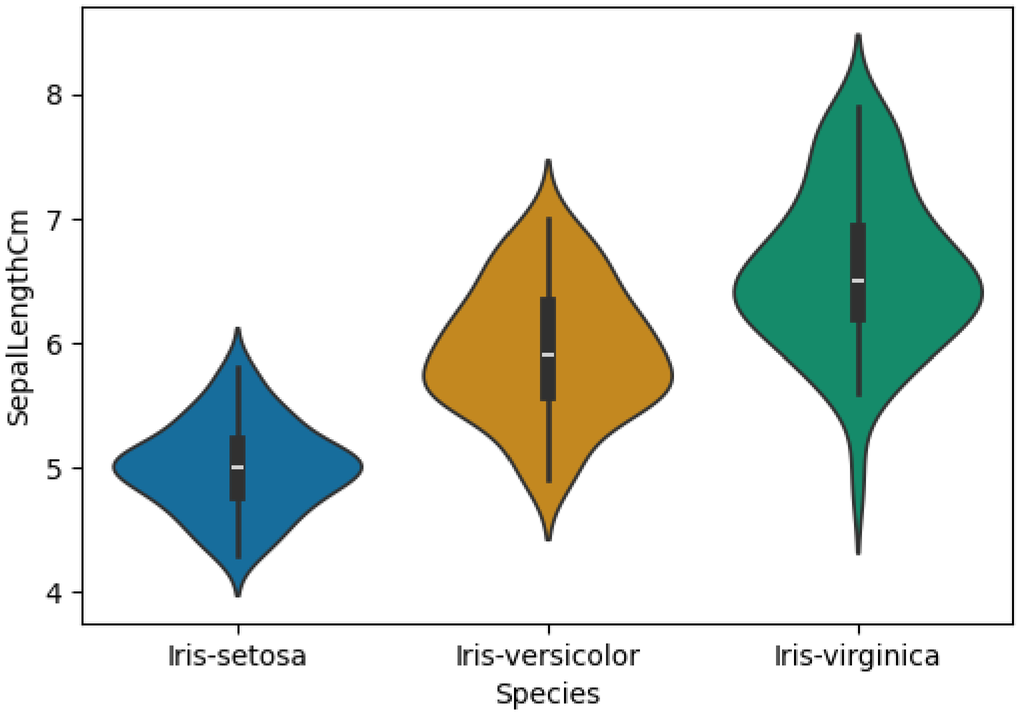


Violin Plot

It is used to visualize the distribution of data and its probability distribution.This chart is a combination of a Box Plot and a Density Plot that is rotated and placed on each side, to show the distribution shape of the data. The thick black bar in the centre represents the interquartile range, the thin black line extended from it represents the 95% confidence intervals, and the white dot is the median.Box Plots are limited in their display of the data, as their visual simplicity tends to hide significant details about how values in the data are distributed

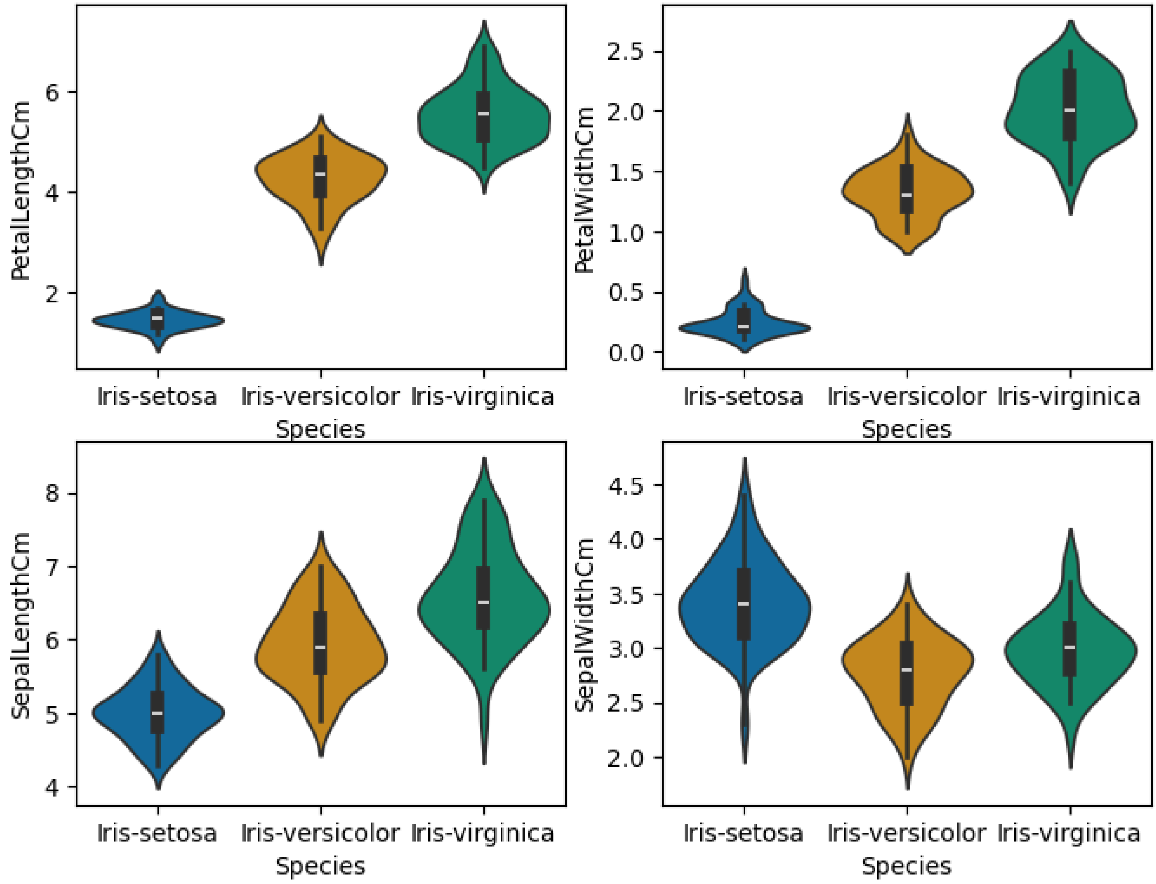
In [26]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
```

```
fig=sns.violinplot(x='Species',y='SepalLengthCm',palette='colorblind',data=iris)
```



In [27]:
```
plt.figure(figsize=(8,6))
plt.subplot(2,2,1)
sns.violinplot(x='Species',y='PetalLengthCm',palette='colorblind',data=iris)
plt.subplot(2,2,2)
sns.violinplot(x='Species',y='PetalWidthCm',palette='colorblind',data=iris)
plt.subplot(2,2,3)
sns.violinplot(x='Species',y='SepalLengthCm',palette='colorblind',data=iris)
plt.subplot(2,2,4)
sns.violinplot(x='Species',y='SepalWidthCm',palette='colorblind',data=iris)
```

Out[27]:
```
<Axes: xlabel='Species', ylabel='SepalWidthCm'>
```

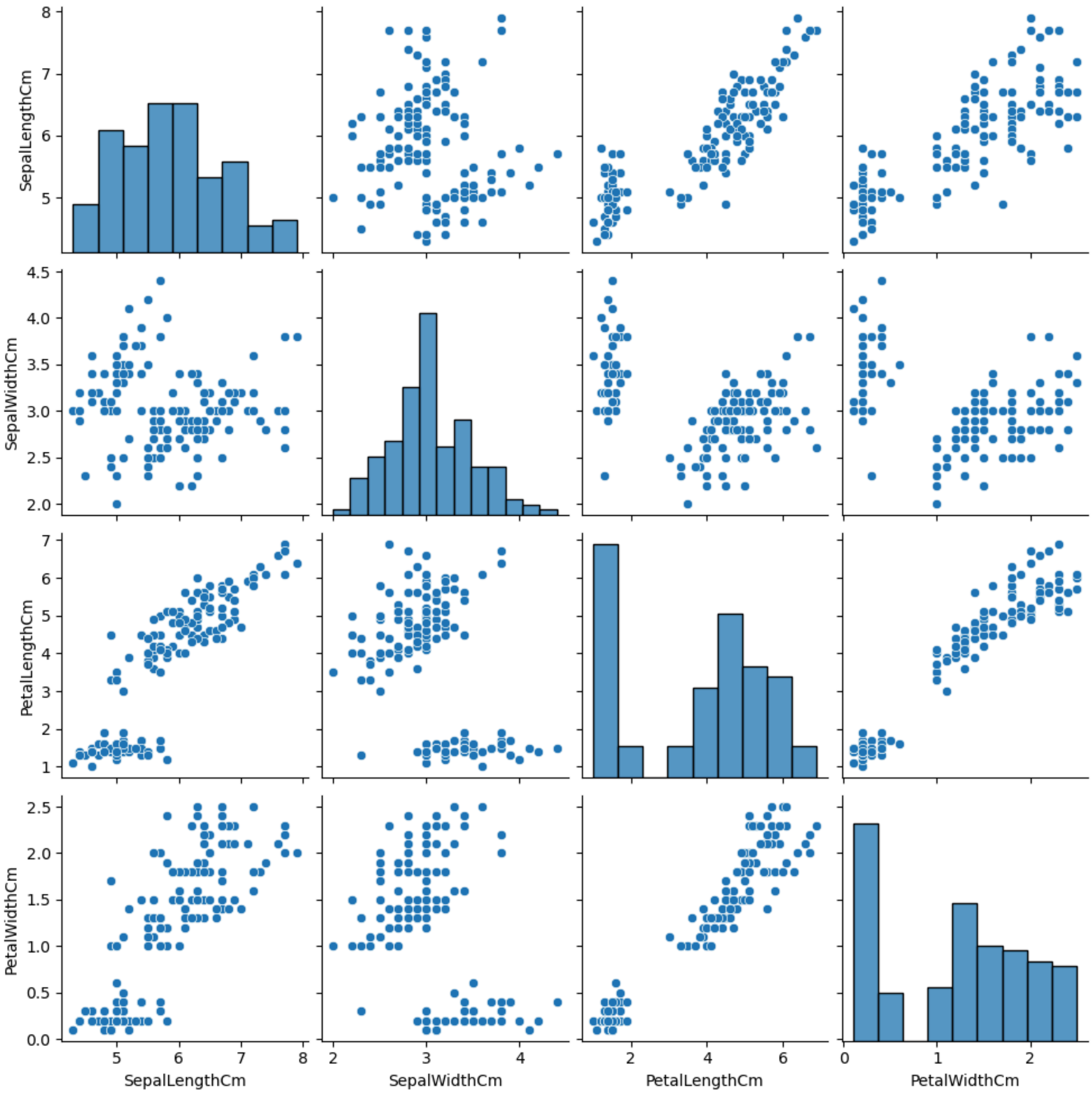


Pair Plot:

A “pairs plot” is also known as a scatterplot, in which one variable in the same data row is matched with another variable's value, like this: Pairs plots are just elaborations on this, showing all variables paired with all the other variables.

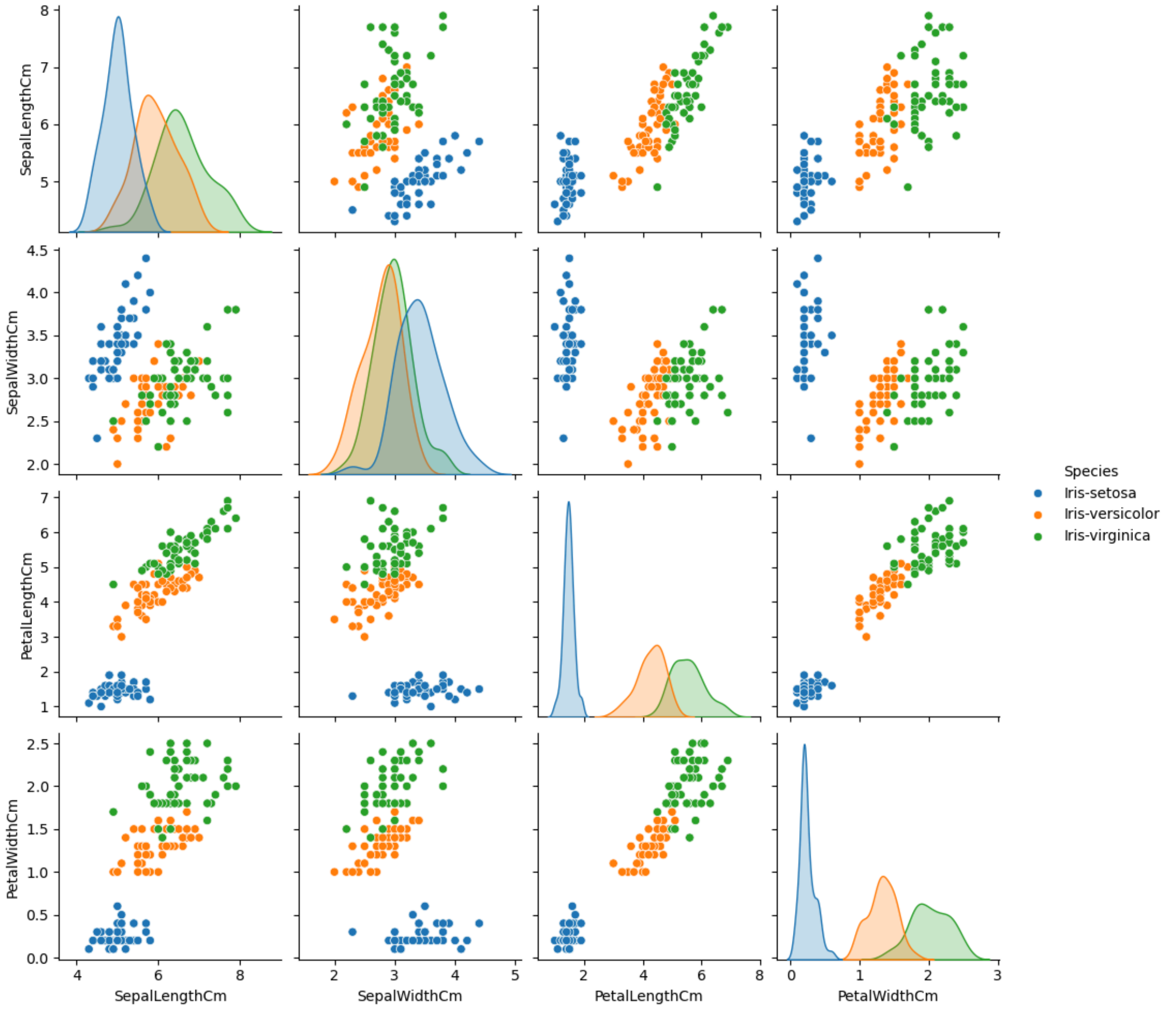
In [28]:
```
sns.pairplot(data=iris,kind='scatter')
```

Out[28]: <seaborn.axisgrid.PairGrid at 0x246a7024fe0>



In [29]:
```python
sns.pairplot(iris,hue='Species')
```

Out[29]: <seaborn.axisgrid.PairGrid at 0x246a564b980>



Heat map

Heat map is used to find out the correlation between different features in the dataset.High positive or negative value shows that the features have high correlation.This helps us to select the parmeters for machine learning.

In [30]:
```
iris.columns
```

Out[30]:
```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [31]:
```
iris1=iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
```

In [32]:
```
iris1.head()
```

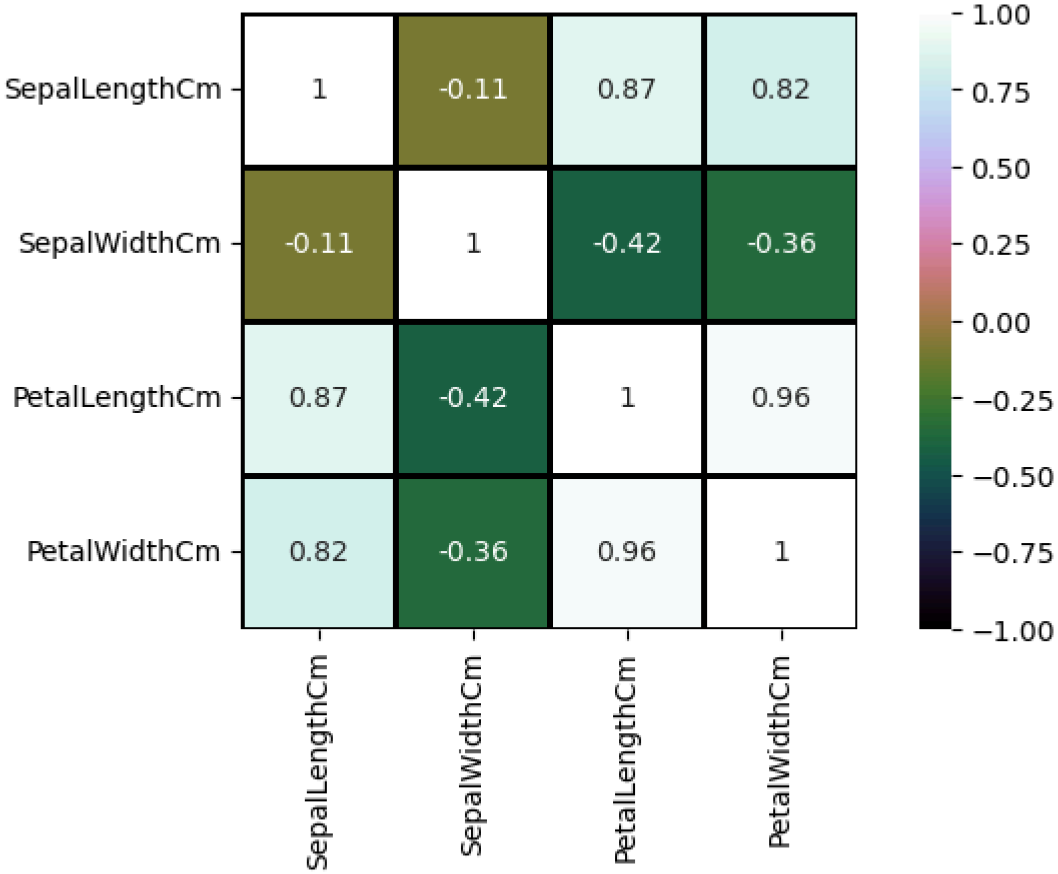
Out[32]:

	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

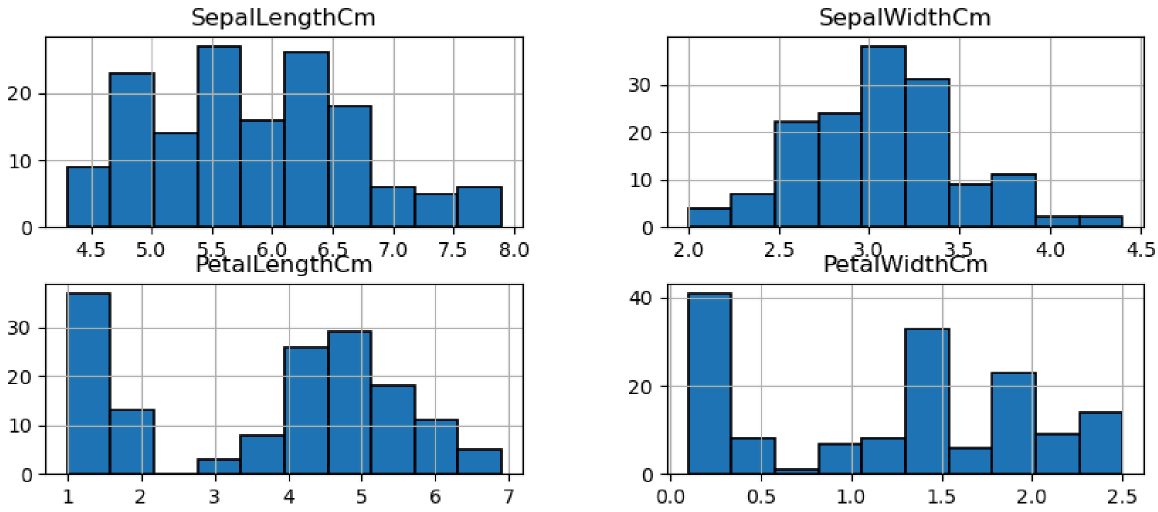
In [33]:
```
iris1.columns
```

Out[33]:
```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm'], dtype
='object')
```

In [34]:
```
fig=plt.gcf()
fig.set_size_inches(8,4)
fig=sns.heatmap(iris1.corr(),annot=True,cmap='cubehelix',linewidths=1,linecolor=
```



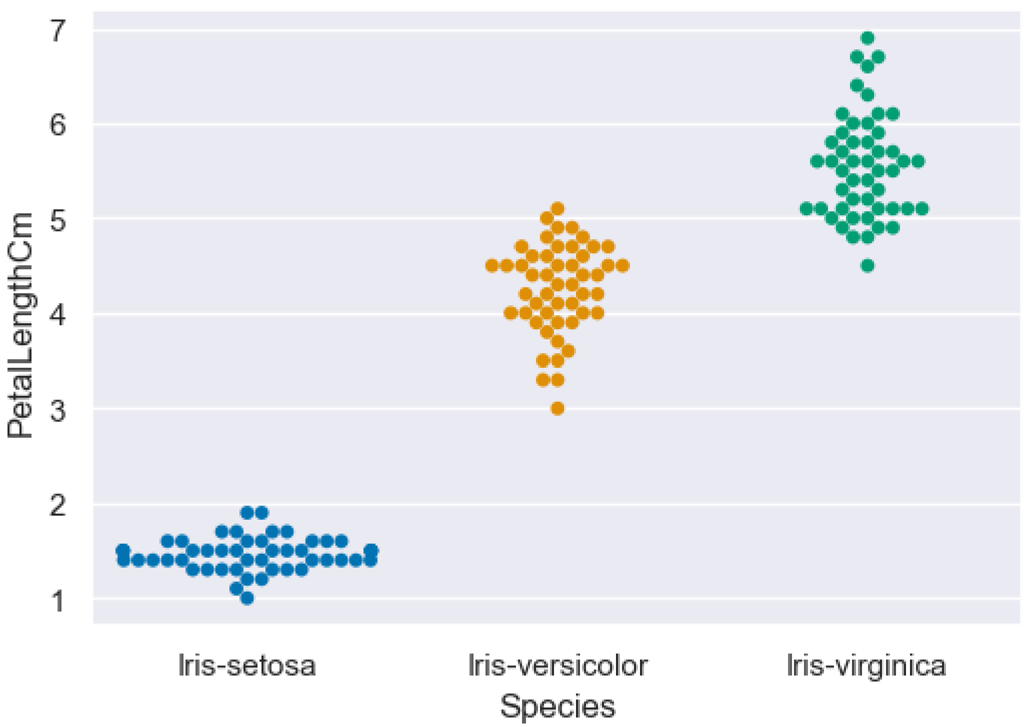
In [35]:
```
iris.hist(edgecolor='black', linewidth=1.2)
fig=plt.gcf()
fig.set_size_inches(10,4)
```



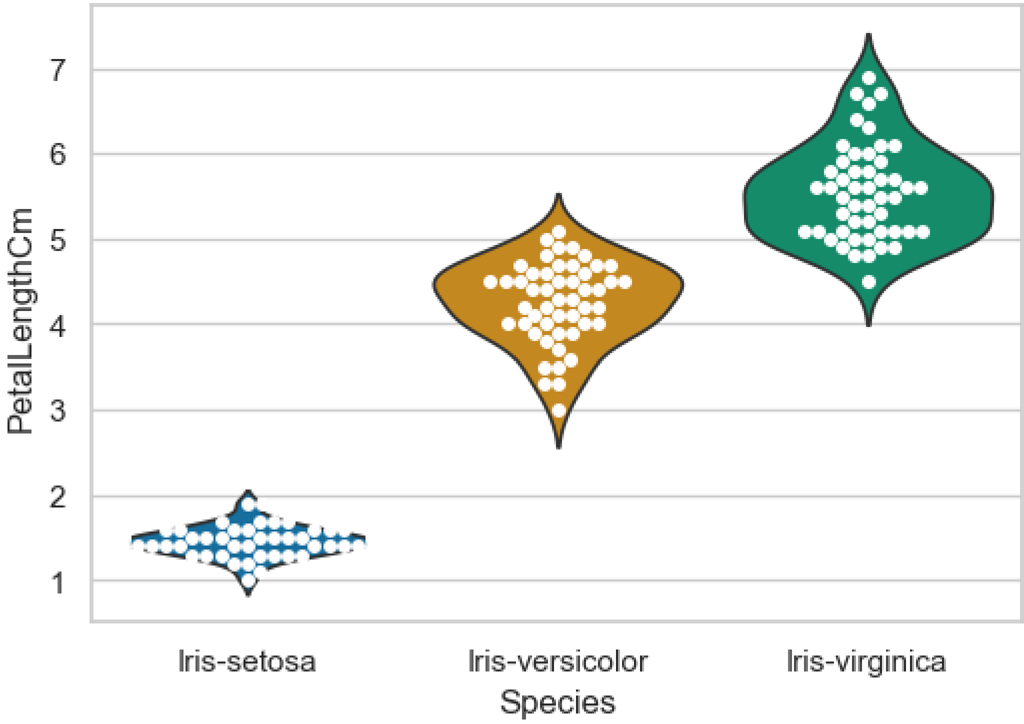
Swarm plot

It looks a bit like a friendly swarm of bees buzzing about their hive. More importantly, each data point is clearly visible and no data are obscured by overplotting.A beeswarm plot improves upon the random jittering approach to move data points the minimum distance away from one another to avoid overlays. The result is a plot where you can see each distinct data point, like shown in below plot

In [36]:
```
sns.set(style="darkgrid")
fig=plt.gcf()
fig.set_size_inches(6,4)
fig = sns.swarmplot(x="Species", y="PetalLengthCm",palette='colorblind', data=ir
```

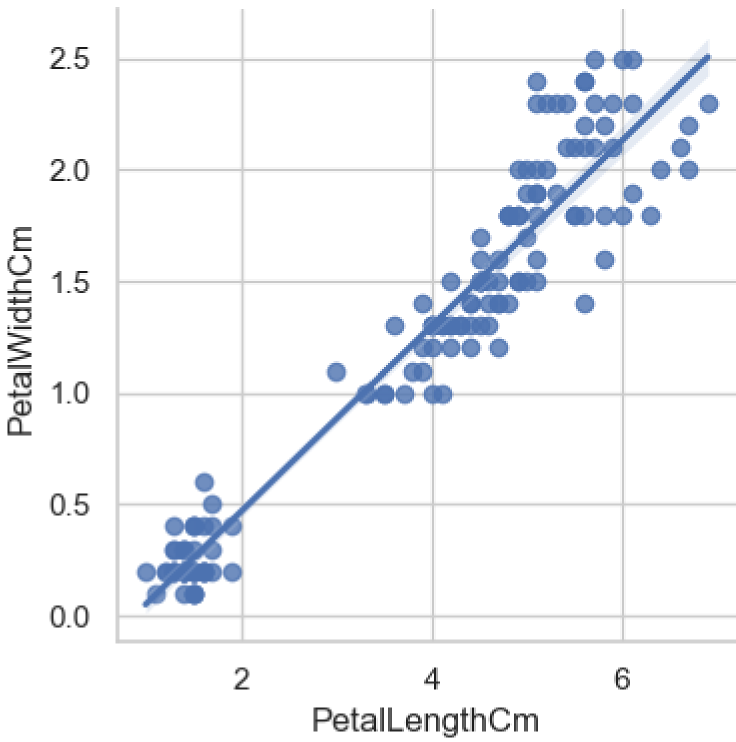


In [37]:
```
sns.set(style="whitegrid")
fig=plt.gcf()
fig.set_size_inches(6,4)
ax = sns.violinplot(x="Species", y="PetalLengthCm", palette='colorblind',data=ir
ax = sns.swarmplot(x="Species", y="PetalLengthCm", data=iris,color="white", edge
```



LM Plot

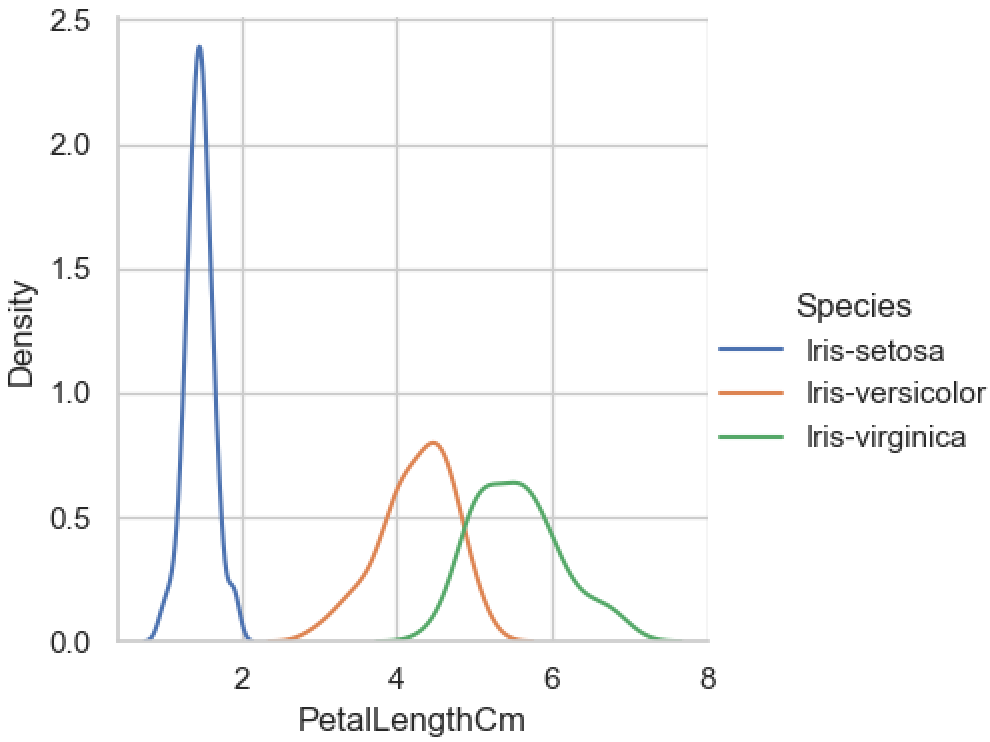
In [38]:
```
fig=sns.lmplot(x="PetalLengthCm", y="PetalWidthCm",data=iris,height=4)
```



FacetGrid

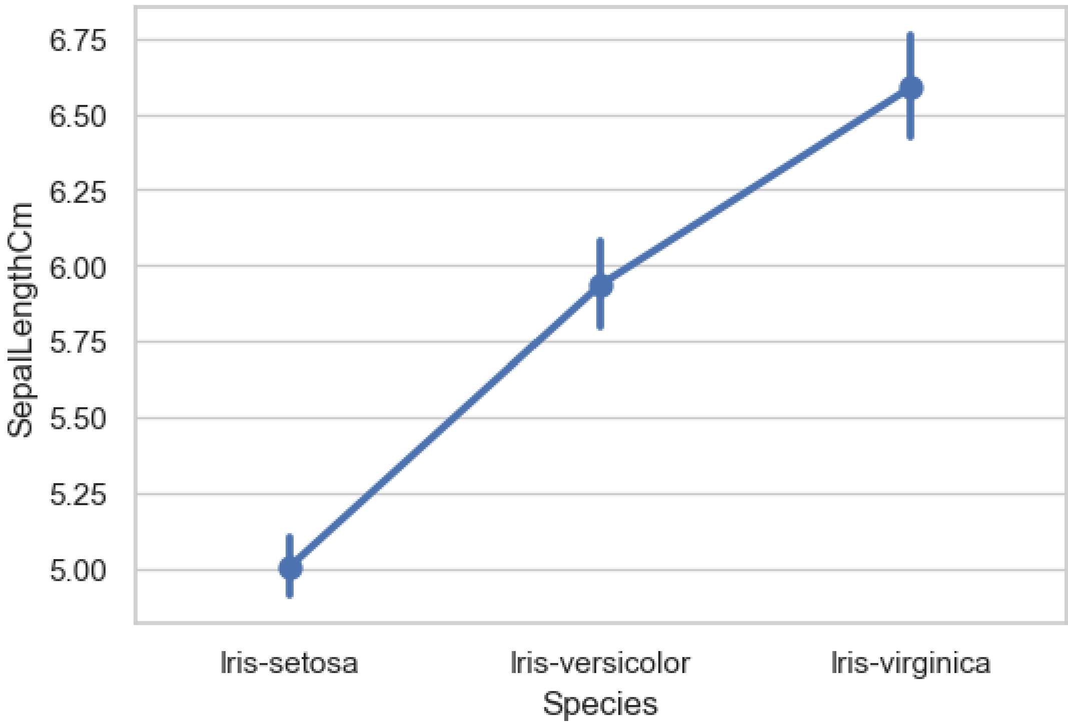
In [39]:
```
sns.FacetGrid(iris, hue="Species", height=4) \
   .map(sns.kdeplot, "PetalLengthCm") \
   .add_legend()
plt.ioff()
```

Out[39]: <contextlib.ExitStack at 0x246a7104b60>



Factor Plot

In [40]:
```
#f,ax=plt.subplots(1,2,figsize=(18,8))
fig=plt.gcf()
fig.set_size_inches(6,4)
sns.pointplot(x='Species',y='SepalLengthCm',data=iris)
plt.ioff()
plt.show()
#sns.factorplot('Species','SepalLengthCm',data=iris,ax=ax[0][0])
#sns.factorplot('Species','SepalWidthCm',data=iris,ax=ax[0][1])
#sns.factorplot('Species','PetalLengthCm',data=iris,ax=ax[1][0])
#sns.factorplot('Species','PetalWidthCm',data=iris,ax=ax[1][1])
```



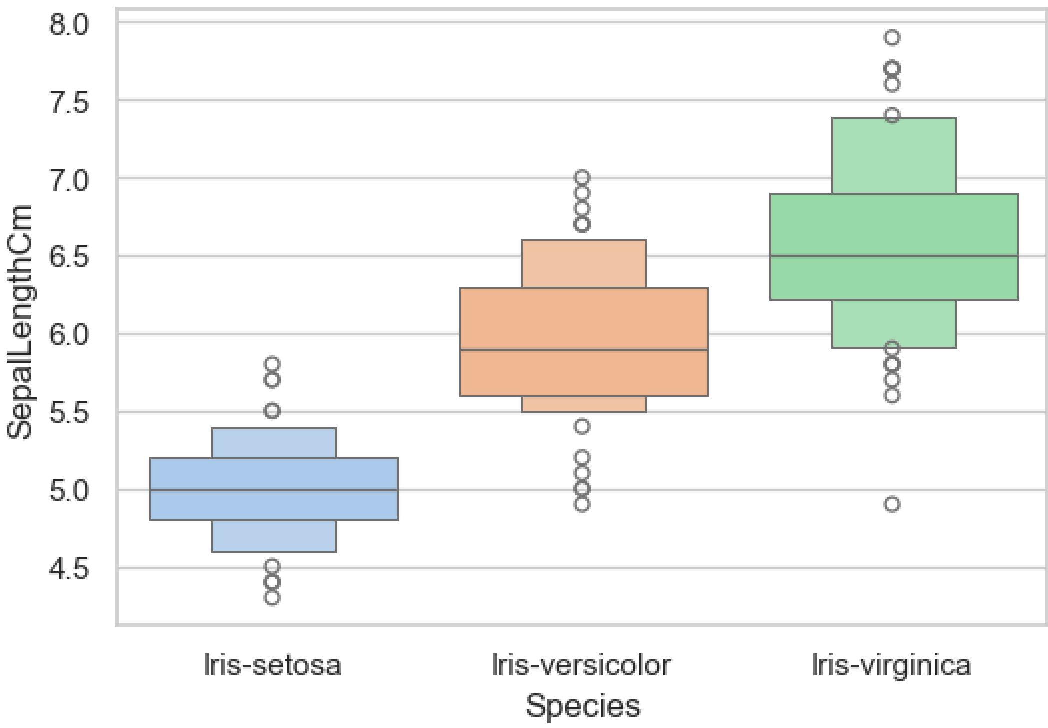
Boxen Plot

In [41]:
```
fig=plt.gcf()
fig.set_size_inches(6,4)
sns.boxenplot(x='Species',y='SepalLengthCm',palette='pastel',data=iris)
```

Out[41]: `<Axes: xlabel='Species', ylabel='SepalLengthCm'>`

In [42]:
```
fig
```

Out[42]:



In [43]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [44]:
```
le=LabelEncoder()
```

In [45]:
```
iris['species']=le.fit_transform(iris['Species'])
iris.head(16)
```

Out[45]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	species
0	5.1	3.5	1.4	0.2	Iris-setosa	0
1	4.9	3.0	1.4	0.2	Iris-setosa	0
2	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5.0	3.6	1.4	0.2	Iris-setosa	0
5	5.4	3.9	1.7	0.4	Iris-setosa	0
6	4.6	3.4	1.4	0.3	Iris-setosa	0
7	5.0	3.4	1.5	0.2	Iris-setosa	0
8	4.4	2.9	1.4	0.2	Iris-setosa	0
9	4.9	3.1	1.5	0.1	Iris-setosa	0
10	5.4	3.7	1.5	0.2	Iris-setosa	0
11	4.8	3.4	1.6	0.2	Iris-setosa	0
12	4.8	3.0	1.4	0.1	Iris-setosa	0
13	4.3	3.0	1.1	0.1	Iris-setosa	0
14	5.8	4.0	1.2	0.2	Iris-setosa	0
15	5.7	4.4	1.5	0.4	Iris-setosa	0

In [46]:
```
x=iris.drop(columns='Species')
y=iris['Species']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

```
LR=LogisticRegression()
LR.fit(x_train,y_train)
```

Out[46]:

```
LogisticRegression
LogisticRegression()
```

In [47]:

```
KNN = KNeighborsClassifier()
```

In [48]:

```
KNN.fit(x_train,y_train)
```

Out[48]:

```
KNeighborsClassifier
KNeighborsClassifier()
```

In [49]:

```
DT=DecisionTreeClassifier()
```

In [50]:

```
DT.fit(x_train,y_train)
```

Out[50]:

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [51]:

```
LR_accuracy=LR.score(x_test,y_test)*100
KNN_accuracy=KNN.score(x_test,y_test)*100
DT_accuracy=DT.score(x_test,y_test)*100
```

In [52]:

```
print(f"Accuracy by using Logistic Regression: {LR_accuracy}%")
```

```
Accuracy by using Logistic Regression: 100.0%
```

In [53]:

```
print(f"Accuracy by using K Nearest Neighbors Algorithm: {KNN_accuracy}%")
```

```
Accuracy by using K Nearest Neighbors Algorithm: 100.0%
```

In [54]:

```
print(f"Accuracy by using Decision Tree Classifier: {DT_accuracy}%")
```

```
Accuracy by using Decision Tree Classifier: 100.0%
```