Importing necessary libraries

In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib' , 'inline')

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import metrics

import seaborn as sns
```

In [2]:
```
df=pd.read_csv('Iris.csv')  #reading the iris dataset which is in .CSV format
```

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 [7]:
```
df.shape #returns the pair (no. of rows , no of columns)
```

Out[7]: (150, 6)

In [8]:
```
df.describe()
```

Out[8]:

	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 [9]:
```
df.isnull().values.any()  #returns true for null values else returns false
```

Out[9]:
```
False
```

In [10]:
```
df.info() # returns the information about the dataset like no of rows n columns
```

```
<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]:
```
df.isnull().sum()  # returns no. of missing entries there in in each column
```

Out[11]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

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

Out[12]:

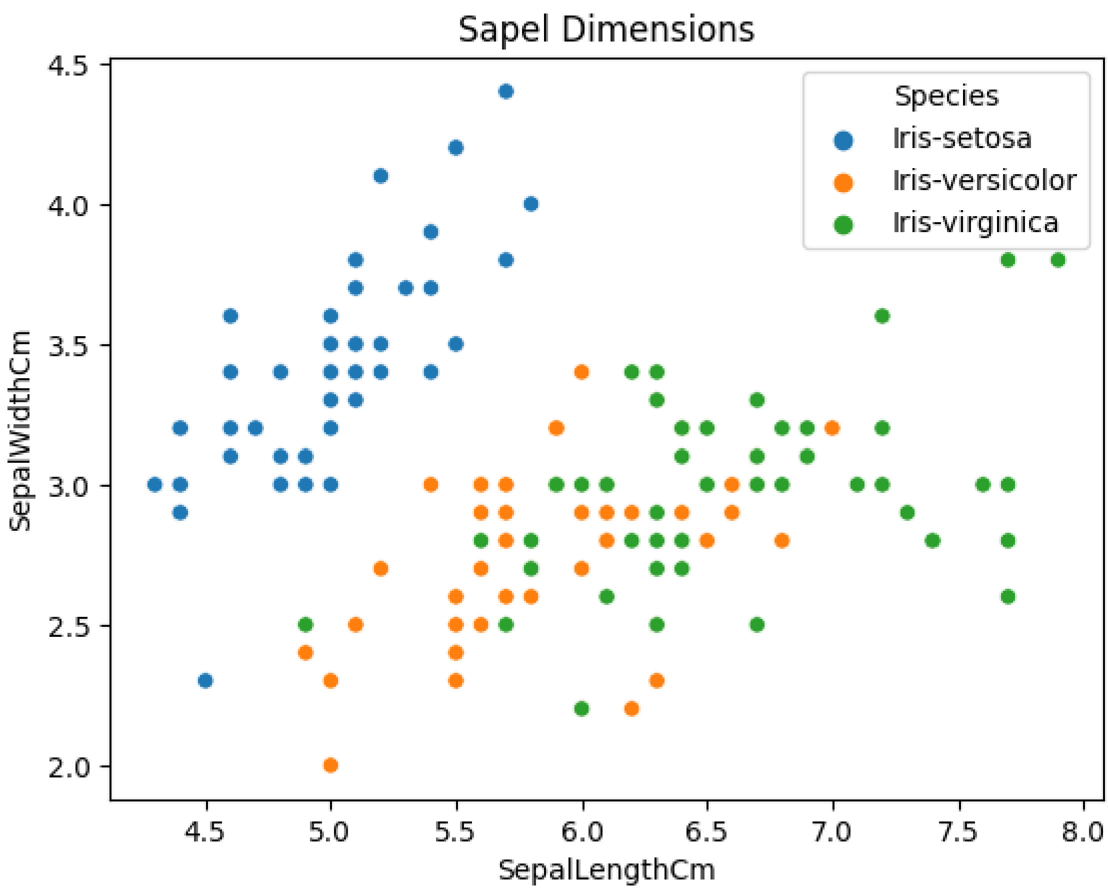
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [13]:

```
plt.title('Sapel Dimensions')  #title of plot
sns.scatterplot(x = 'SepalLengthCm' ,    #seaborn is a visualization library for
                y = 'SepalWidthCm' ,     #scatterplot() plots a scatter plot grap
                hue = 'Species' ,
                data = df)
```

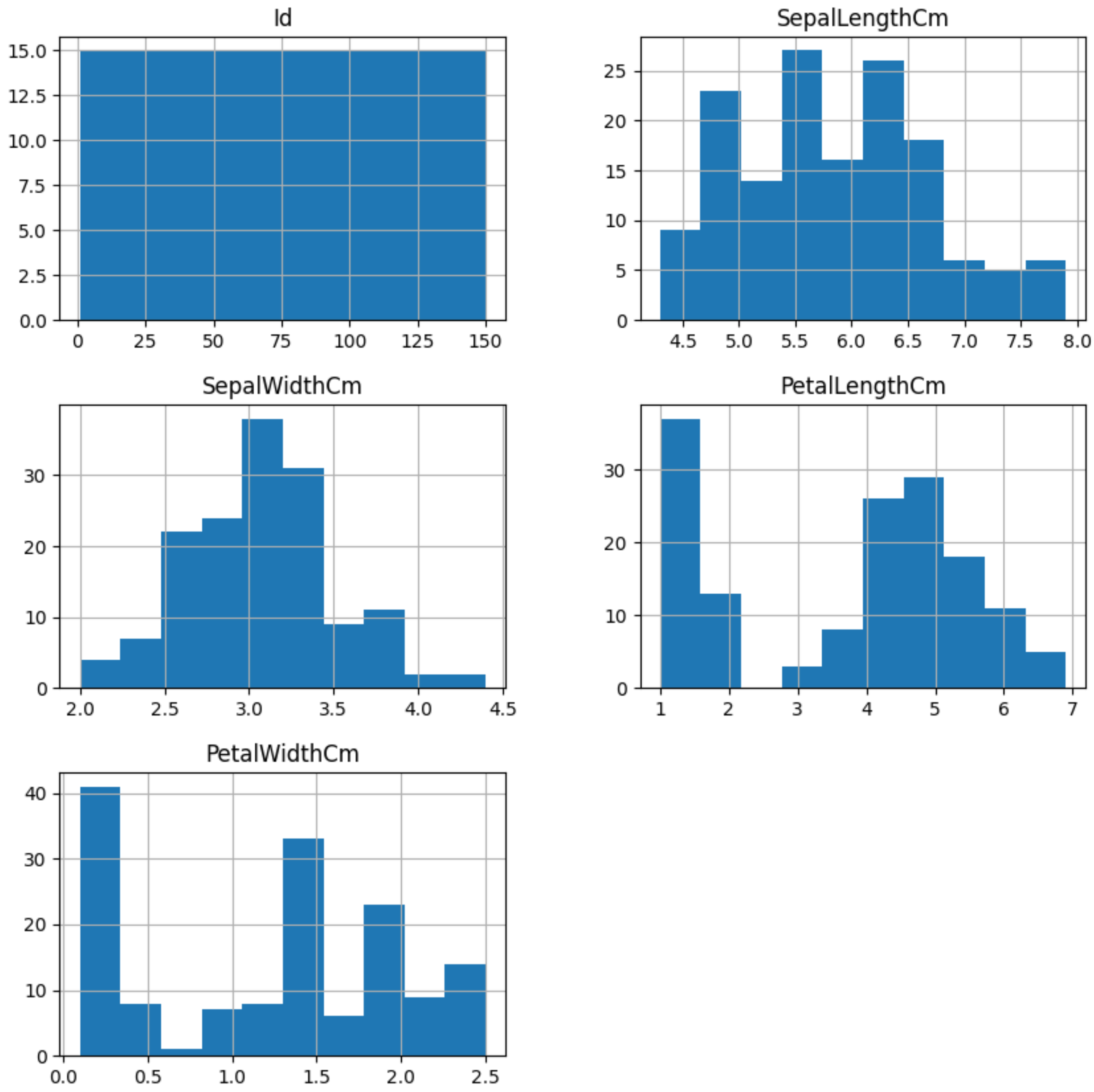
Out[13]:

```
<Axes: title={'center': 'Sapel Dimensions'}, xlabel='SepalLengthCm', ylabel='Se
palWidthCm'>
```



In [14]:

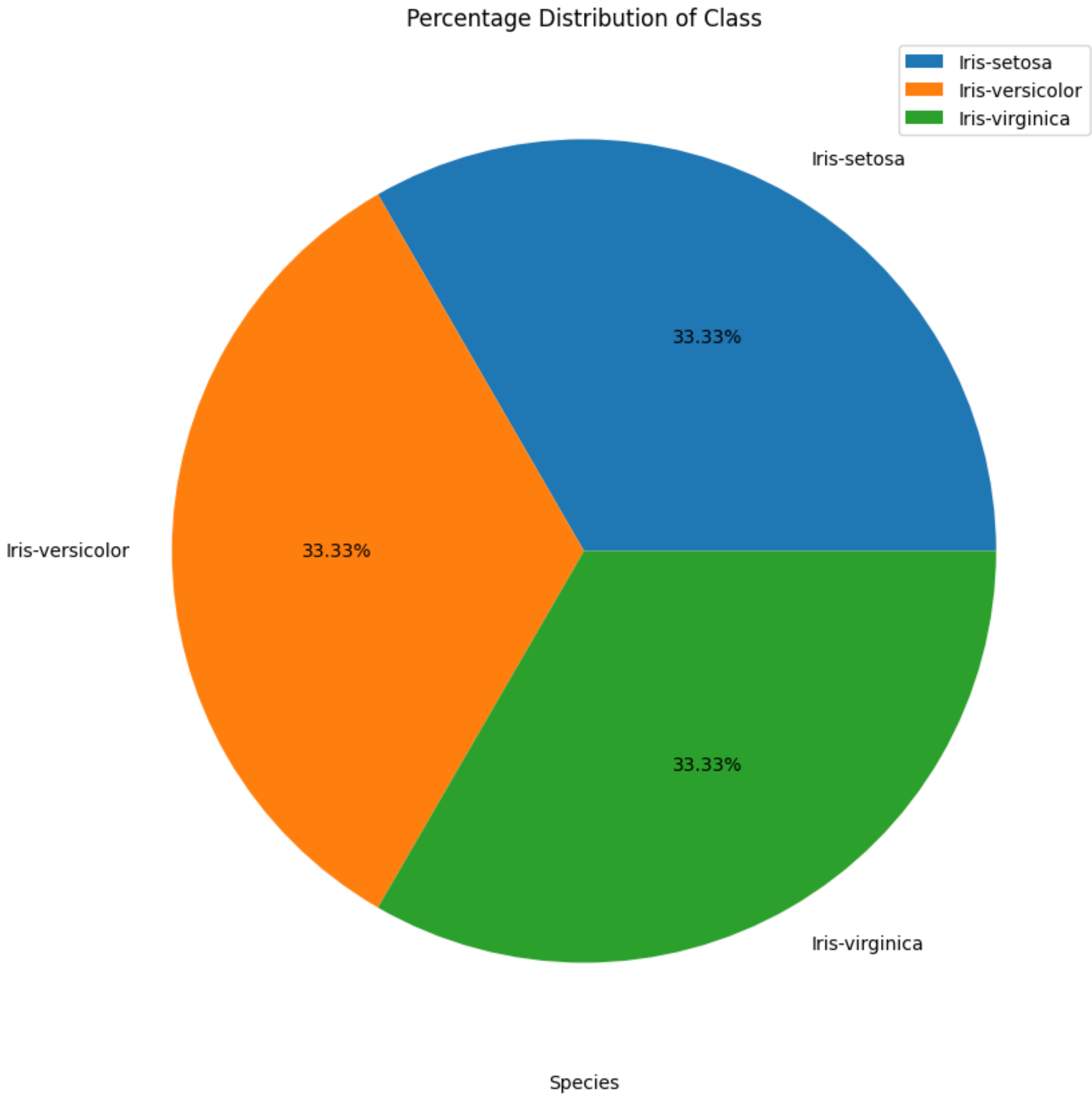
```
df.hist(figsize=(10, 10))
plt.show()
```



In [15]:
```
print("\nClass distribution:")
print(df['Species'].value_counts())
```

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

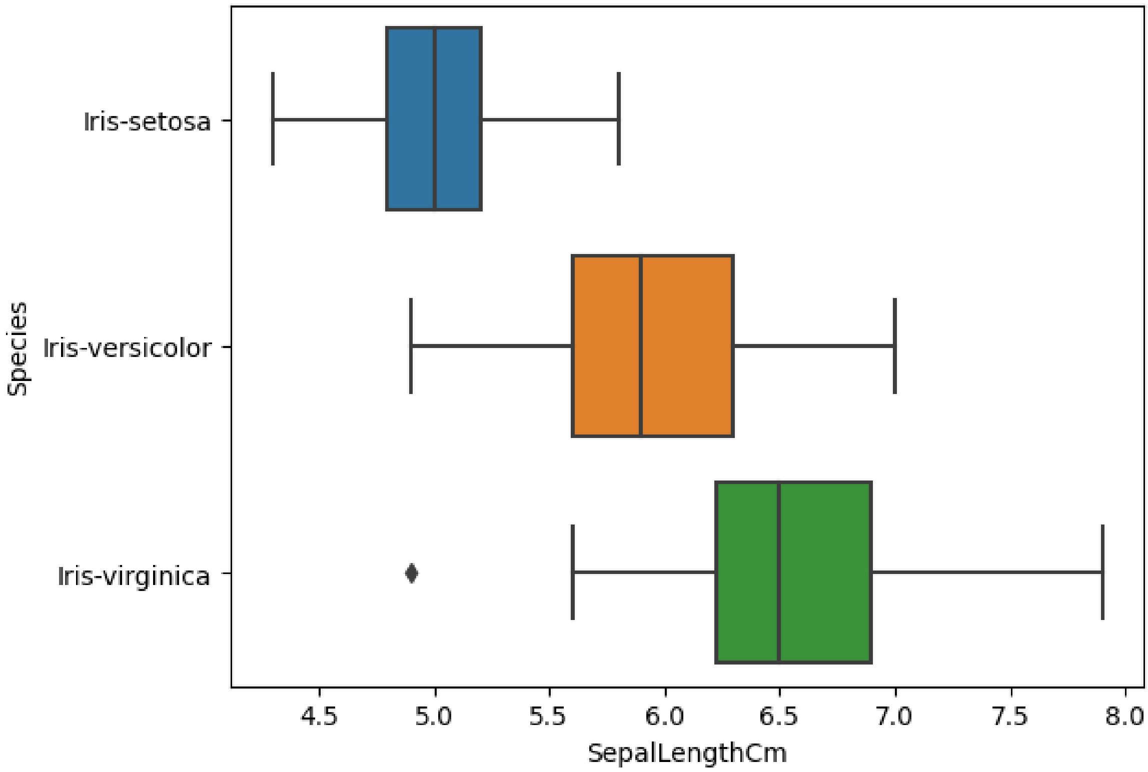
In [16]:
```
# Plot a pie chart to show the percentage distribution of classes
plt.figure(figsize=(20, 10))
df['Species'].value_counts().plot(kind='pie', autopct='%.2f%%')
plt.title('Percentage Distribution of Class')
plt.legend(df['Species'].value_counts().index)
plt.xlabel('Species')
plt.ylabel(None)
plt.show()
```

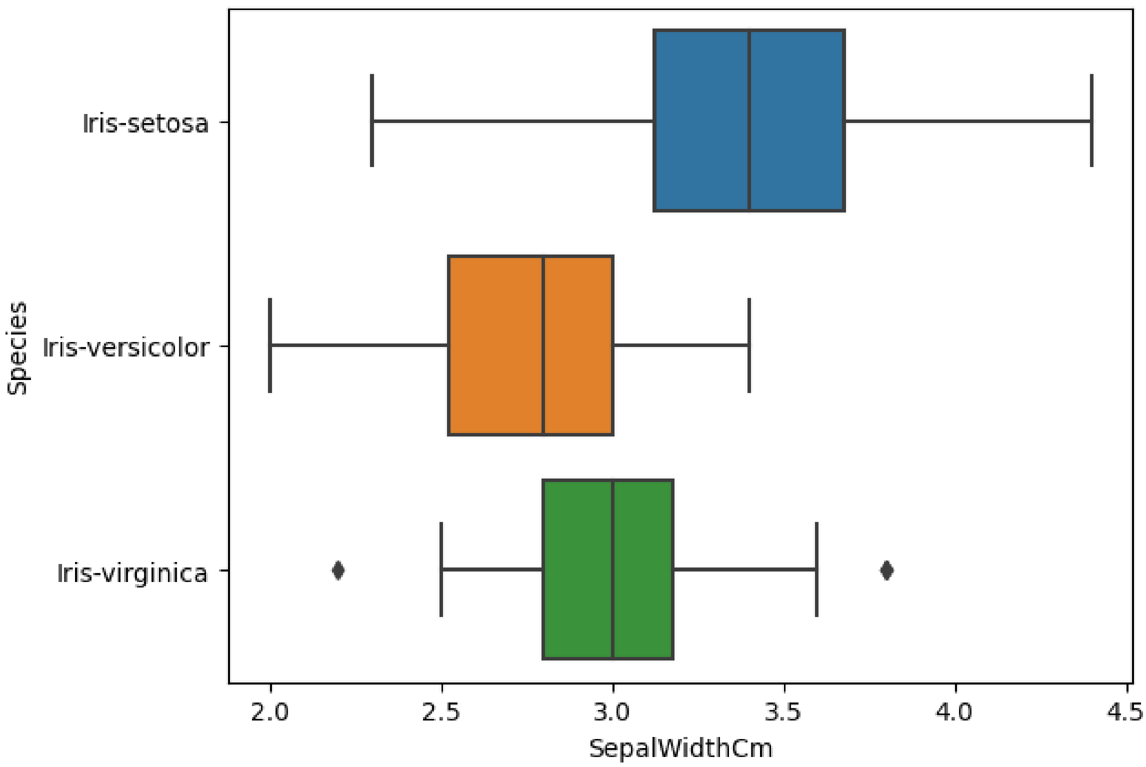


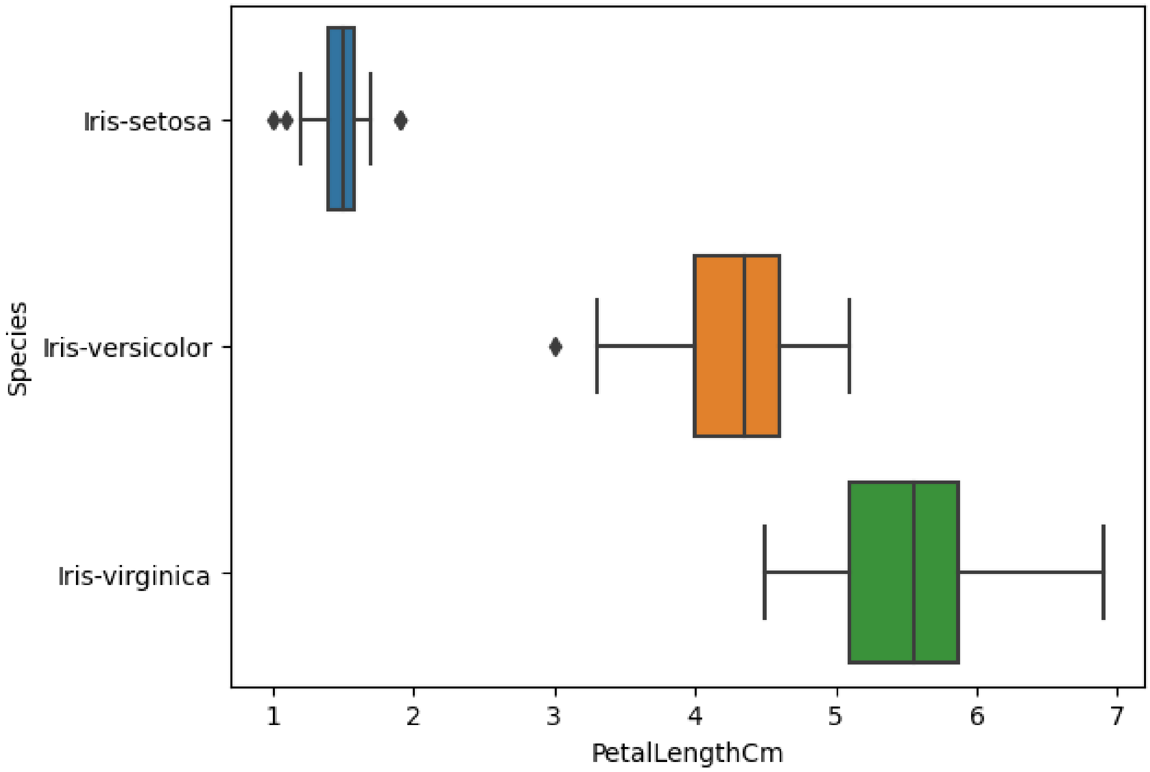
In [17]:
```
df.columns
```

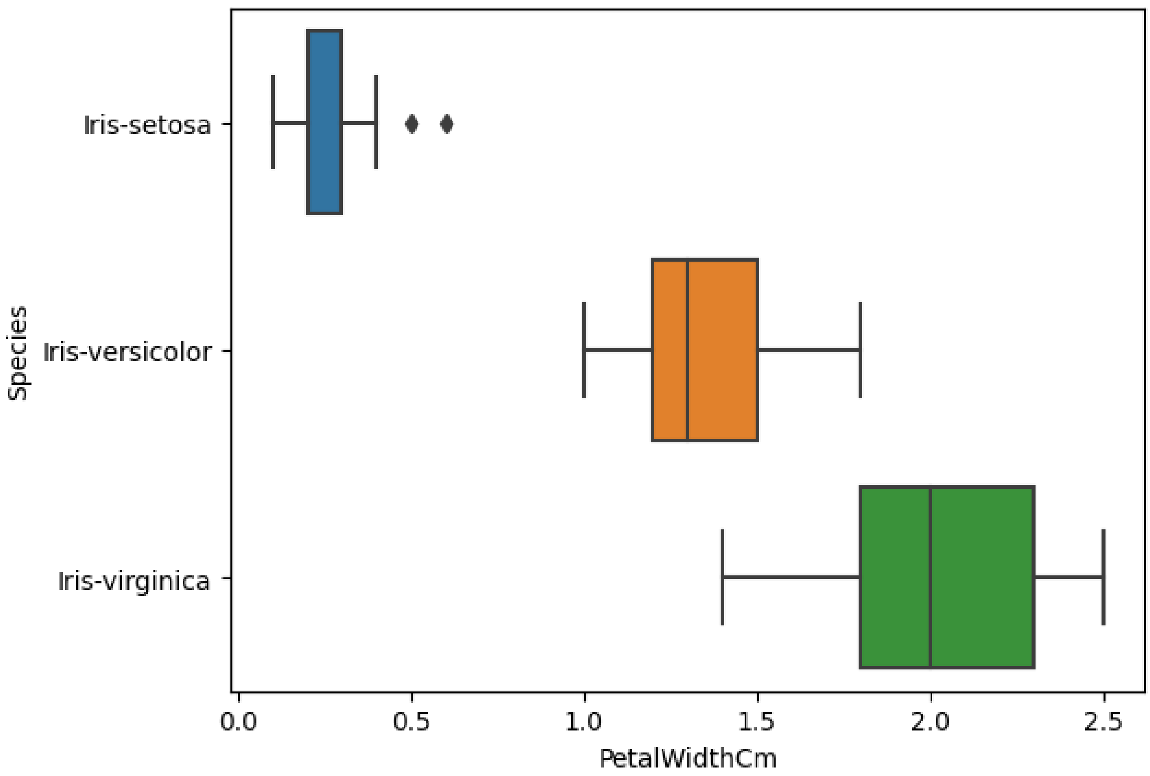
Out[17]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [18]:
```
# Show boxplots for each numerical column against the 'class' column
sns.boxplot(data=df, x='SepalLengthCm', y='Species')
plt.show()
sns.boxplot(data=df, x='SepalWidthCm', y='Species')
plt.show()
sns.boxplot(data=df, x='PetalLengthCm', y='Species')
plt.show()
sns.boxplot(data=df, x='PetalWidthCm', y='Species')
plt.show()
```


Iris-setosa
Iris-versicolor
Iris-virginica
Species
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
SepalLengthCm


Iris-setosa
Iris-versicolor
Iris-virginica
Species
2.0
2.5
3.0
3.5
4.0
4.5
SepalWidthCm



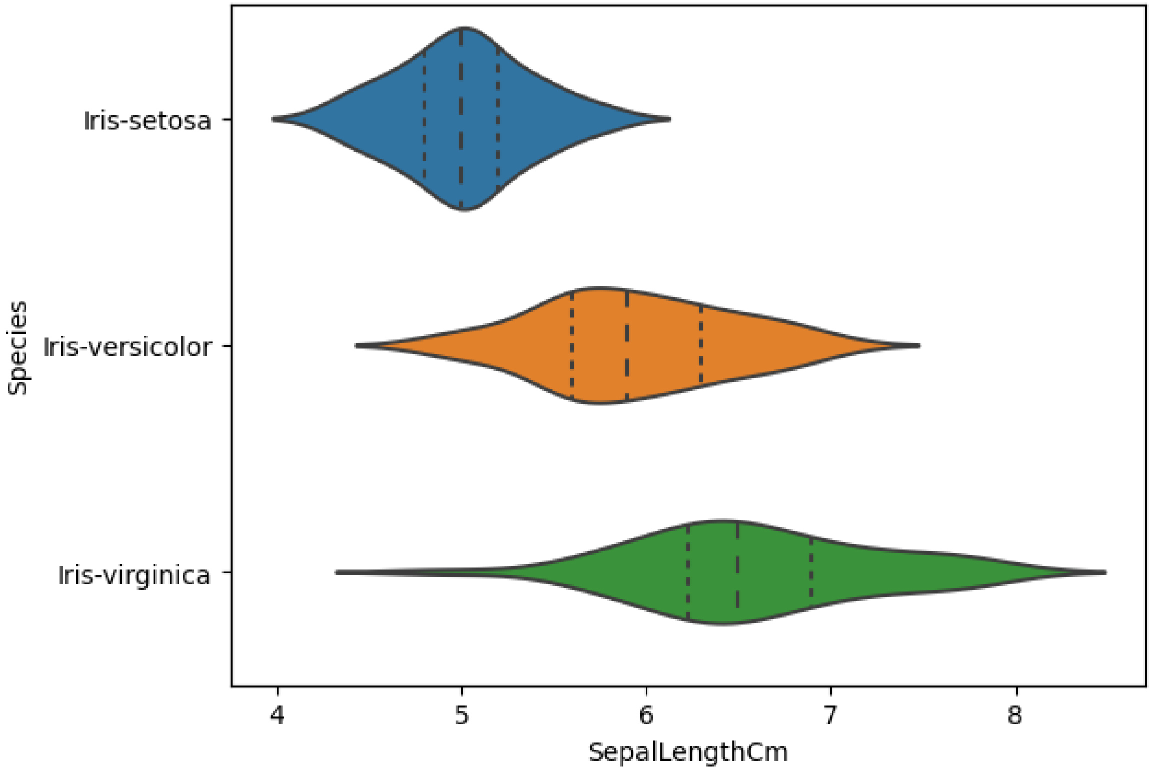


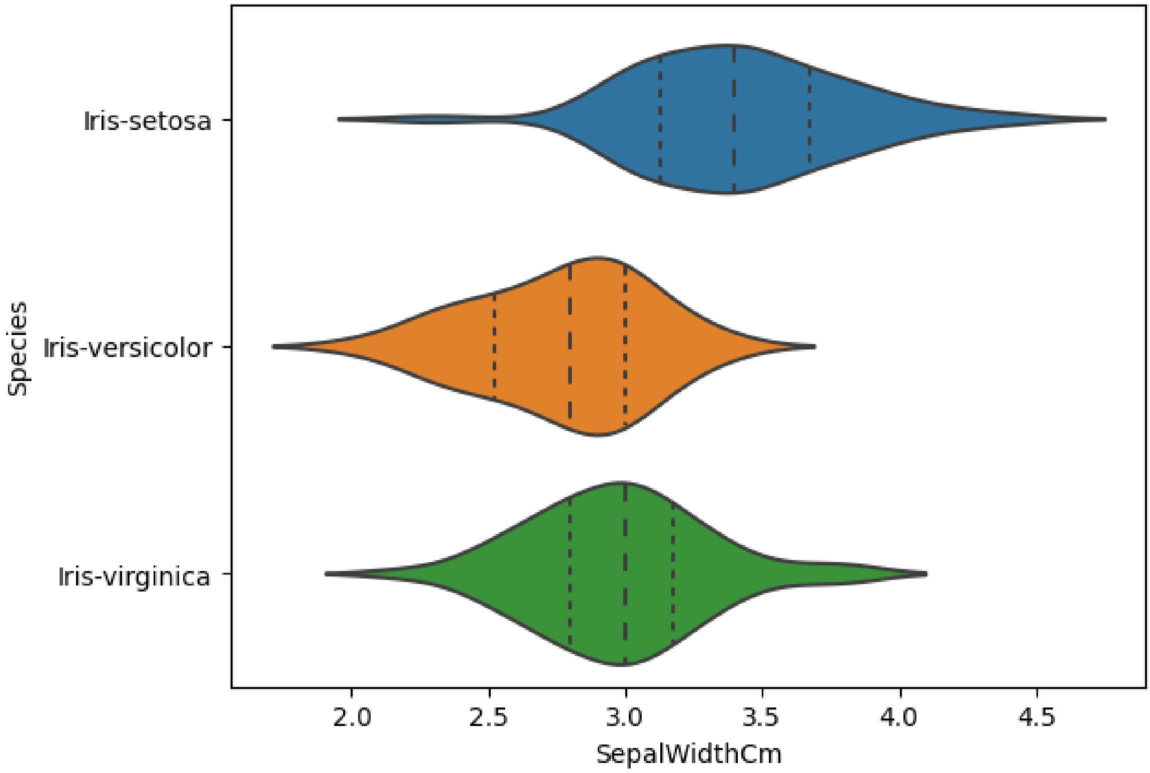
In [21]:
```
df.columns
```

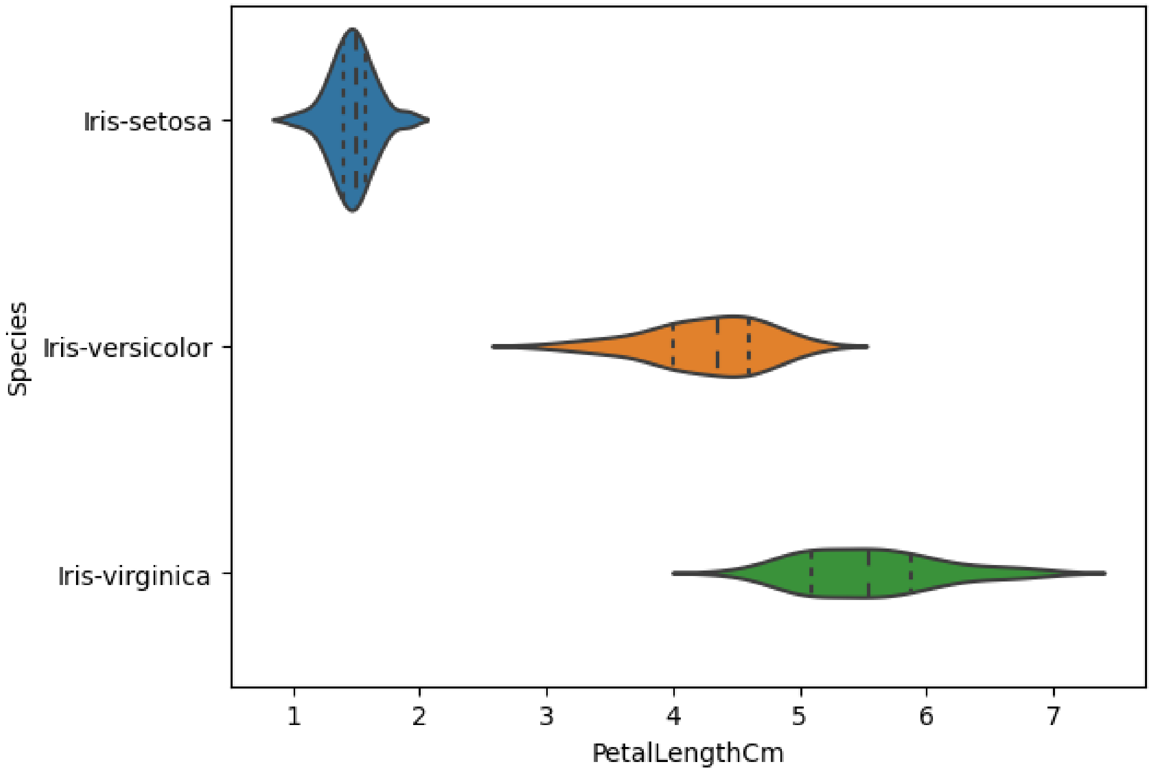
Out[21]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

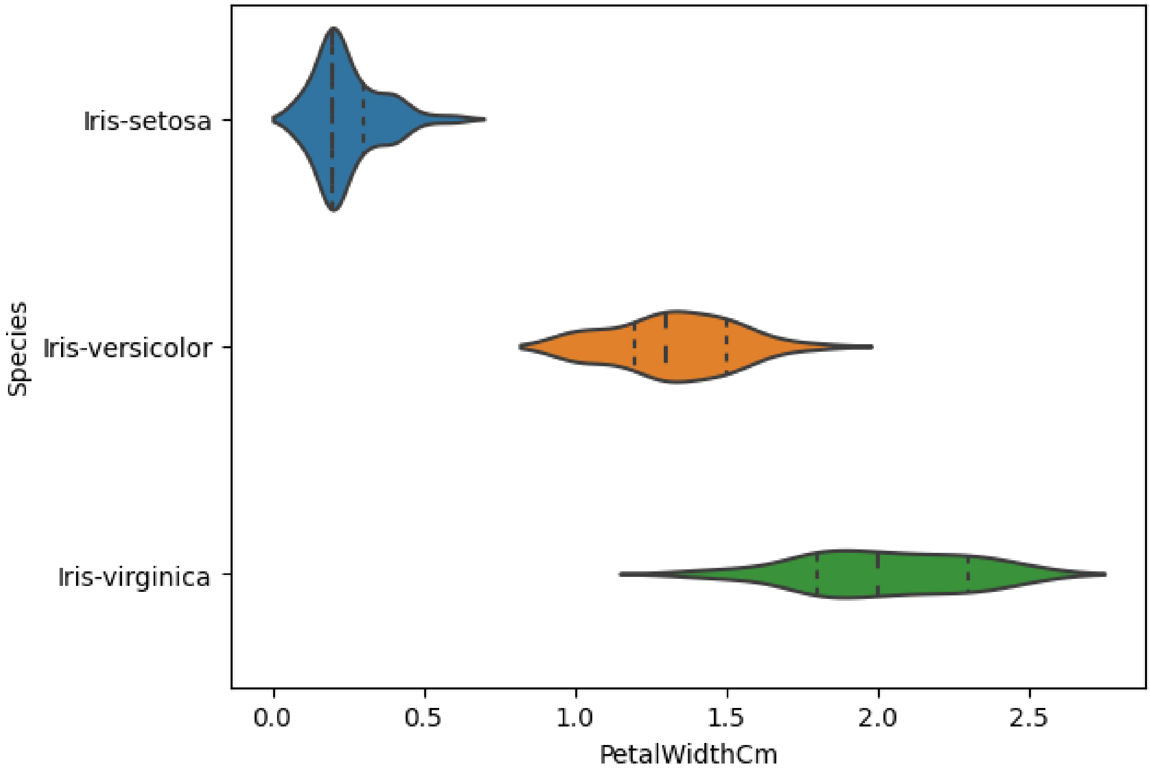
In [22]:
```
# Show violin plots to visualize the distribution of sepal_length for each class
g = sns.violinplot(y='Species', x='SepalLengthCm', data=df, inner='quartile')
plt.show()
g = sns.violinplot(y='Species', x='SepalWidthCm', data=df, inner='quartile')
plt.show()
g = sns.violinplot(y='Species', x='PetalLengthCm', data=df, inner='quartile')
```

```
plt.show()
g = sns.violinplot(y='Species', x='PetalWidthCm', data=df, inner='quartile')
plt.show()
```





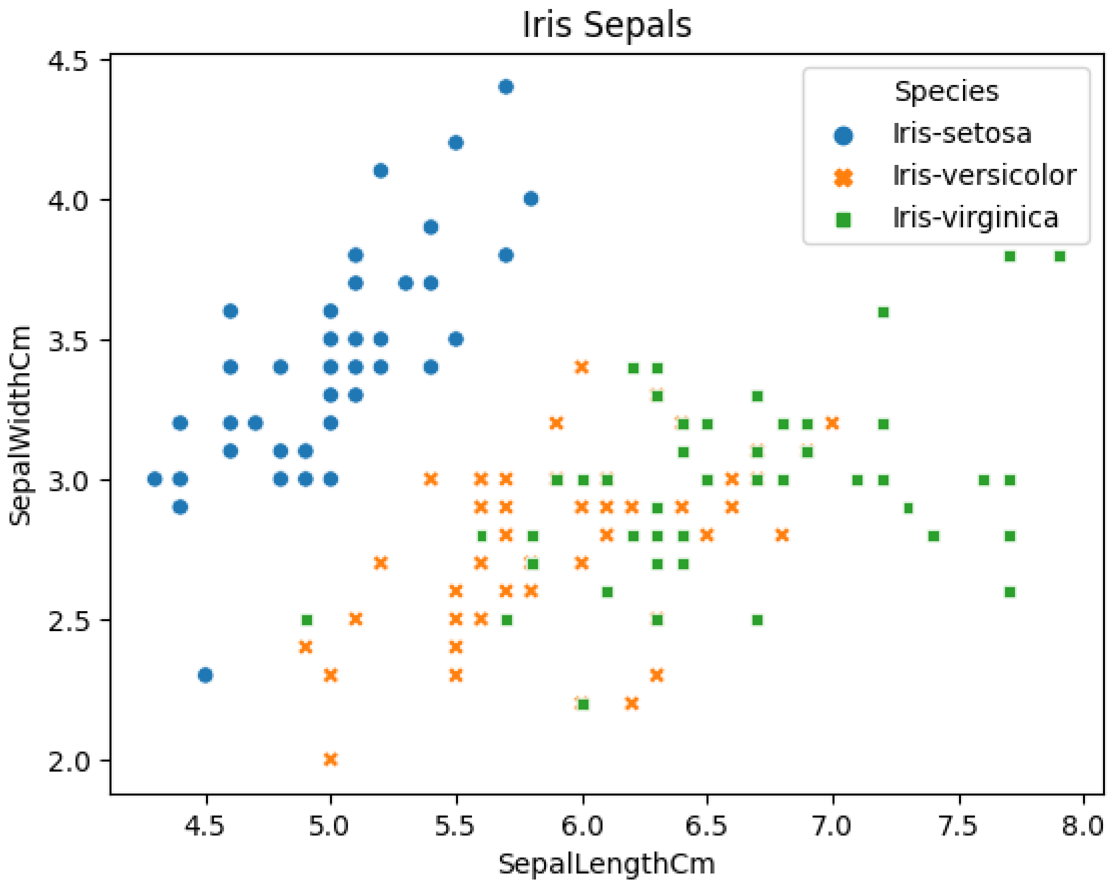




In [23]:
```
df.columns
```

Out[23]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

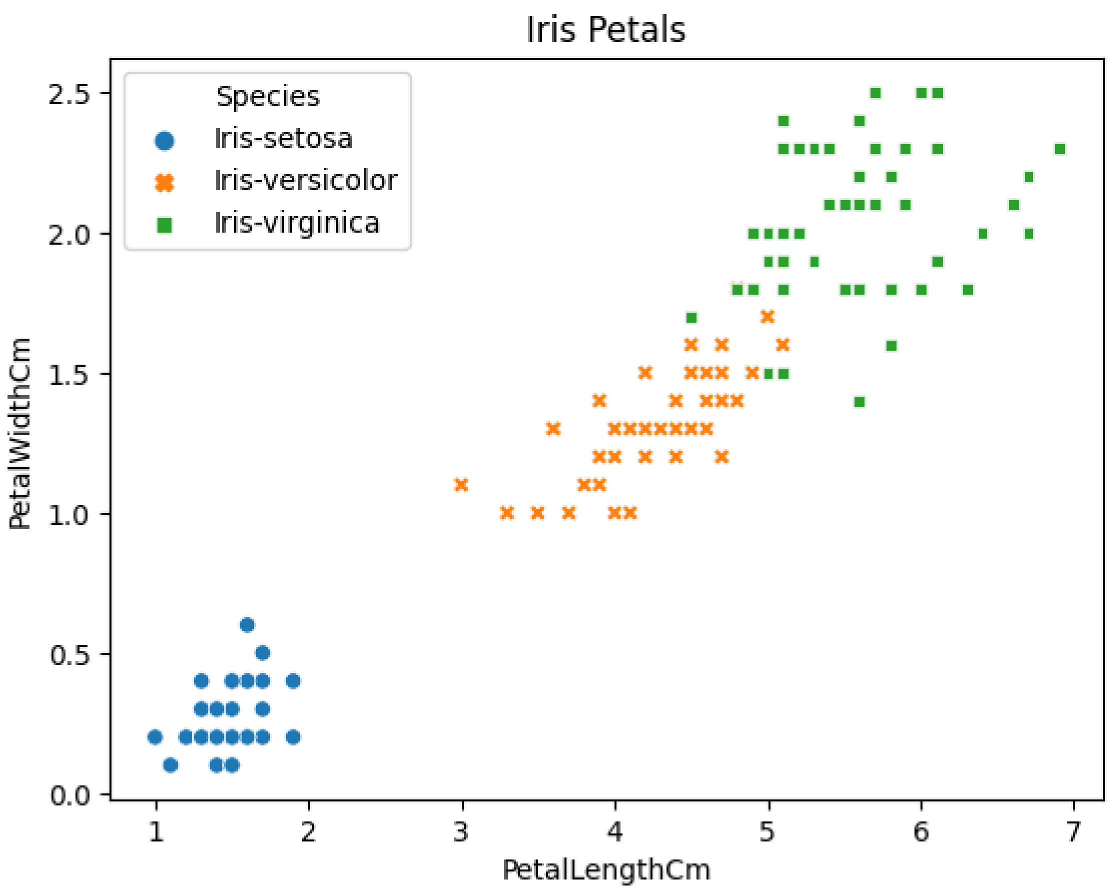
In [24]:
```
# Show scatter plots to visualize the relationship between sepal_length and sepa
sns.scatterplot(data=df, x='SepalLengthCm', y='SepalWidthCm', hue='Species', sty
plt.title('Iris Sepals')
plt.show()
```



In [25]:
```
df.columns
```

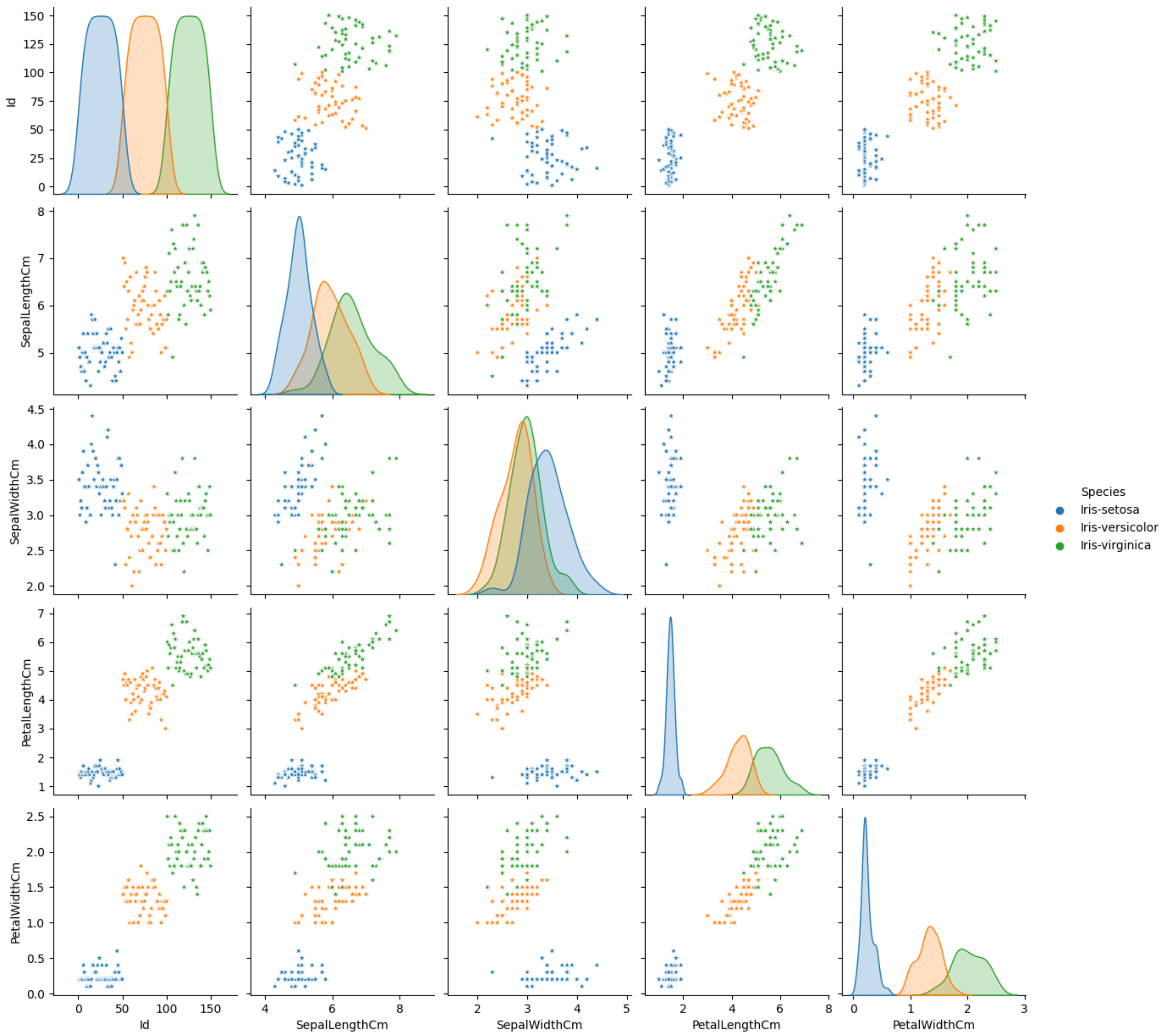
Out[25]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [26]:
```
# Show scatter plots to visualize the relationship between petal_length and peta
sns.scatterplot(data=df, x='PetalLengthCm', y='PetalWidthCm', hue='Species', sty
plt.title('Iris Petals')
plt.show()
```



In [27]:
```
# Show pair plot to visualize the relationships between all numerical columns wi
sns.pairplot(df, hue='Species', markers='*')
```

Out[27]: <seaborn.axisgrid.PairGrid at 0x28955318350>



In [30]:
```
# Data Encoding
# %matplotlib inline
sns.set_palette('Set1')
```

In [31]:
```
# Encode the 'class' column using LabelEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
```

In [32]:
```
# Separate features and target variable
x = df.drop(['Species'], axis=1)
y = df['Species']
```

In [33]:
```
# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_
```

In [34]:
```
# Model Training and Evaluation
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
```

In [35]:
```
# Support Vector Machine (SVM) classifier
from sklearn.svm import SVC
classifier = SVC()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("\nSVM Classifier:")
print(classification_report(y_test, y_pred))
```

```
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy_score(y_pred, y_test))
```

```
SVM Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         7

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

[[10  0  0]
 [ 0 13  0]
 [ 0  0  7]]
Accuracy: 1.0
```

In [36]:

```
# Gaussian Naive Bayes classifier
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("\nGaussian Naive Bayes Classifier:")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy_score(y_pred, y_test))
```

```
Gaussian Naive Bayes Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         7

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

[[10  0  0]
 [ 0 13  0]
 [ 0  0  7]]
Accuracy: 1.0
```

In [37]:

```
# Multinomial Naive Bayes classifier
from sklearn.naive_bayes import MultinomialNB
classifier = MultinomialNB()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("\nMultinomial Naive Bayes Classifier:")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy_score(y_pred, y_test))
```

```
Multinomial Naive Bayes Classifier:
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        10
           1       0.75      0.69      0.72        13
           2       0.56      0.71      0.63         7

    accuracy                           0.77        30
   macro avg       0.77      0.77      0.76        30
weighted avg       0.79      0.77      0.77        30

[[9 1 0]
 [0 9 4]
 [0 2 5]]
Accuracy: 0.7666666666666667
```

In [38]:

```
# Bernoulli Naive Bayes classifier
from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB()
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("\nBernoulli Naive Bayes Classifier:")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy_score(y_pred, y_test))
```

```
Bernoulli Naive Bayes Classifier:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        10
           1       0.00      0.00      0.00        13
           2       0.23      1.00      0.38         7

    accuracy                           0.23        30
   macro avg       0.08      0.33      0.13        30
weighted avg       0.05      0.23      0.09        30

[[ 0  0 10]
 [ 0  0 13]
 [ 0  0  7]]
Accuracy: 0.23333333333333334
```

```
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [39]:

```
# Complement Naive Bayes classifier
from sklearn.naive_bayes import ComplementNB
classifier = ComplementNB()
```

```
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
print("\nComplement Naive Bayes Classifier:")
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
print('Accuracy:', accuracy_score(y_pred, y_test))
```

```
Complement Naive Bayes Classifier:
              precision    recall  f1-score   support

           0       0.71      1.00      0.83        10
           1       0.00      0.00      0.00        13
           2       0.44      1.00      0.61         7

    accuracy                           0.57        30
   macro avg       0.38      0.67      0.48        30
weighted avg       0.34      0.57      0.42        30

[[10  0  0]
 [ 4  0  9]
 [ 0  0  7]]
Accuracy: 0.5666666666666667
```

```
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\kushw\AppData\Roaming\Python\Python311\site-packages\sklearn\metrics\_cl
assification.py:1469: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [ ]:

```
for clf in classifiers:
    clf.fit(x_train, x_train['sepal_length'])
    name = clf.__class__.__name__

    train_predictions = clf.predict(x_test)
    acc = accuracy_score(x_test['sepal_length'], train_predictions)

    log_entry = pd.DataFrame([[name, acc * 100, 11]], columns=log_cols)
    log = log.append(log_entry)

# Visualize the accuracy of different classifiers using a bar plot
sns.set_color_codes("muted")
sns.barplot(x='Accuracy', y='Classifier', data=log, color="b")
plt.xlabel('Accuracy %')
plt.title('Classifier Accuracy')
plt.show()

# Pie chart to show the distribution of 'sepal_length'
plt.figure(figsize=(8, 8))
sepal_lengths = df['sepal_length']
unique_values = sepal_lengths.unique()
value_counts = sepal_lengths.value_counts()
```

```python
plt.pie(value_counts, labels=unique_values, autopct='%.2f%%', startangle=90)
plt.title('Distribution of Sepal Length')
plt.show()
```

In [45]:
```python
Text = '''In this project, we performed data analysis and classification on the 

We started by loading the dataset and performing data visualization to gain insi

After the data analysis, we encoded the target variable 'Species' using LabelEnc

Next, we split the data into training and testing sets and trained several class

The results of the classification showed that different classifiers achieved var

In conclusion, the Iris dataset is a classic and well-known dataset that serves 

print(Text)
```

```
In this project, we performed data analysis and classification on the Iris datase
t using various machine learning models. The Iris dataset contains samples of iri
s flowers, each belonging to one of three species: Setosa, Versicolor, or Virgini
ca. Our goal was to classify the flowers into their respective species based on t
heir sepal and petal dimensions.

We started by loading the dataset and performing data visualization to gain insig
hts into the distribution and relationships of the features. We used scatter plot
s, box plots, violin plots, and pair plots to visualize the relationships between
the features and the target classes. These visualizations helped us understand th
e characteristics of each species and detect any potential outliers.

After the data analysis, we encoded the target variable 'Species' using LabelEnco
der to convert the categorical class labels into numerical format.

Next, we split the data into training and testing sets and trained several classi
fiers on the training data. We evaluated the performance of each classifier using
metrics such as precision, recall, F1-score, and accuracy. The classifiers we use
d were Support Vector Machine (SVM), Gaussian Naive Bayes, Multinomial Naive Baye
s, Bernoulli Naive Bayes, and Complement Naive Bayes.

The results of the classification showed that different classifiers achieved vary
ing levels of accuracy and performance on the Iris dataset. The SVM classifier pe
rformed well, achieving high accuracy in predicting the species. Gaussian Naive B
ayes also showed good performance on this dataset, demonstrating the usefulness o
f probabilistic models for classification tasks.

In conclusion, the Iris dataset is a classic and well-known dataset that serves a
s a great starting point for exploring data analysis and machine learning classif
ication techniques. By applying various classifiers and visualizing the data, we
gained valuable insights into the relationships between features and the classes,
and we successfully predicted the species of iris flowers based on their sepal an
d petal dimensions.
```

In [ ]: