Project Title	iris classification
Tools	Jupyter Notebook and VS code
Technologies	Machine learning
Domain	Data Analytics
Project Difficulties level	Beginner

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

Project Overview

The Iris Classification project involves creating a machine learning model to classify iris flowers into three species (Setosa, Versicolour, and Virginica) based on the length and width of their petals and sepals. This is a classic problem in machine learning and is often used as an introductory example for classification algorithms.

Problem Statement:

- The model should achieve a high level of accuracy in classifying iris species.
- The model's predictions should be consistent and reliable, as measured by cross-validation.
- The final report should provide clear and comprehensive documentation of the project, including all code, visualizations, and findings.

By achieving these objectives, the project will demonstrate the ability to apply machine learning techniques to a classic classification problem, providing insights into the characteristics of different iris species and the effectiveness of various algorithms for this task.

Project Steps

Understanding the Problem

- The goal is to classify iris flowers into one of three species based on four features: sepal length, sepal width, petal length, and petal width.

Dataset Preparation

- **Dataset**: The Iris dataset is available in the UCI Machine Learning Repository and is also included in many machine learning libraries, such as scikit-learn.
- **Features**: Sepal length, sepal width, petal length, petal width.
- **Labels**: Iris species (Setosa, Versicolour, Virginica).

Data Exploration and Visualization

- Load the dataset and explore it using descriptive statistics and visualization techniques.
- Use libraries like Pandas for data manipulation and Matplotlib/Seaborn for visualization.
- Example visualizations include scatter plots, pair plots, and histograms to understand the distribution and relationships between features.

Data Preprocessing

- Handle missing values (if any).
- Standardize or normalize the features if necessary to ensure they are on a similar scale.
- Split the dataset into training and testing sets (commonly 80% training and 20% testing).

Model Selection and Training

- Choose a classification algorithm. Common choices for this problem include:
  - K-Nearest Neighbors (KNN)
  - Decision Trees
  - Random Forest
  - Support Vector Machine (SVM)
  - Logistic Regression
- Train the model using the training data.

Model Evaluation

- Evaluate the model using the testing data.
- Use metrics like accuracy, precision, recall, and F1-score to assess the model's performance.

- Visualize the confusion matrix to understand the classification results in detail.

Hyperparameter Tuning

- Use techniques like Grid Search or Random Search to find the optimal hyperparameters for the chosen model.
- Cross-validation can also be employed to ensure the model generalizes well to unseen data.

Model Interpretation and Insights

- Interpret the model results and understand which features are most important for the classification.
- Visualize decision boundaries if using models like Decision Trees or SVM.

Deployment (Optional)

- Deploy the model using a web framework like Flask or Django to create a simple web application where users can input flower measurements and get predictions.

Documentation and Reporting

- Document the entire process, including data exploration, preprocessing, model training, evaluation, and tuning.
- Create a final report or presentation summarizing the project, results, and insights.

Content

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

The columns in this dataset are:

Id

SepalLengthCm

SepalWidthCm

PetalLengthCm

PetalWidthCm

Species

Sepal Width vs. Sepal Length

Example: You can get the basic idea how you can create a project from here

Here’s a basic example using Python and scikit-learn to classify iris species:

```
# Import necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Explore the dataset
print(df.head())
print(df.describe())
sns.pairplot(df, hue='species')
plt.show()

# Split the data into training and testing sets
X = df.drop('species', axis=1)
y = df['species']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the model
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Plot confusion matrix
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

This code demonstrates loading the Iris dataset, splitting it into training and testing sets, standardizing the features, training a KNN classifier, and evaluating its performance.

Additional Tips

- Experiment with different algorithms to see which one performs best for your dataset.
- Use dimensionality reduction techniques like PCA (Principal Component Analysis) to visualize high-dimensional data.
- Perform feature engineering if you believe additional features could improve model performance.

Sample Code And Output

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from PIL import Image

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing
Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory
(/kaggle/working/) that gets preserved as output when you create
a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they
```

```
won't be saved outside of the current session
```

```
/kaggle/input/iriscsv/Iris.csv
/kaggle/input/iris-classification/Iris.csv
```

In [2]:

```
df = pd.read_csv('../input/iris-classification/Iris.csv')
df.head()
```

Out[2]:

	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

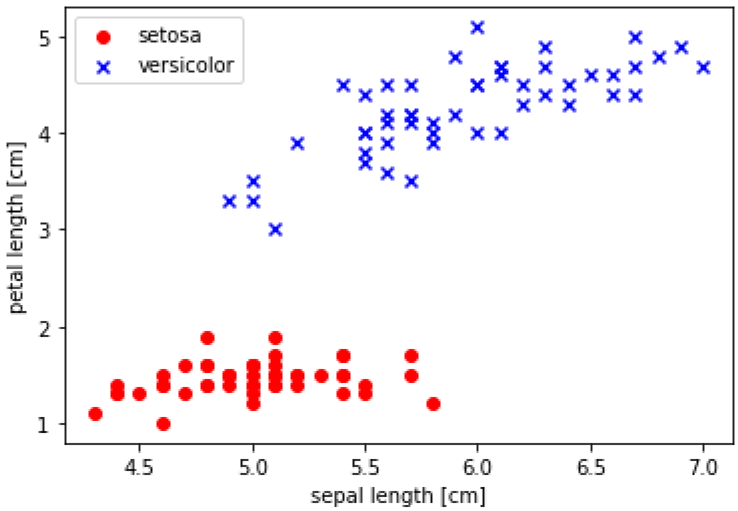
In [3]:

```
y = df.iloc[0:100, 4].values
y = np.where(y == 'Iris-setosa', -1, 1)
X = df.iloc[0:100, [0, 2]].values
```

In [4]:

```
plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o',
label='setosa')
plt.scatter(X[50:100,0], X[50:100,1], color='blue', marker='x',
label='versicolor')
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
```

```
plt.legend(loc='upper left')

plt.show()
```



In [5]:

```
class Perceptron(object):
    """Perceptron Classifier.


       Parameters

       ----------

       eta: float

         Learning rate (between 0.0 and 1.0)

       n_iter: int

         Passes over the training set

       random_state: int

         Random number generator seed for random weight
```

```
initialization.

       Attributes
       ----------
       w_ : 1_d array
         Weights after fitting.
       errors_ : list
         Number of misclassifications (updates) in each epoch.

    """

    def __init__(self, eta = 0.01, n_iter = 50, random_state =
1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def fit(self, X, y):
        """fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_examples, n_features]
          Training vectors, where n_examples is the number of
          examples and n_features is the number of features.
```

```
        y : array-like, shape = [n_examples]
          Target values.

        Returns
        -------
        self : object

        """
        rgen = np.random.RandomState(self.random_state)
        self.w_ = rgen.normal(loc = 0.0, scale = 0.01, size = 1
+ X.shape[1])
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                update = self.eta * (target - self.predict(xi))
                self.w_[1:] += update * xi
                self.w_[0] += update
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self
    def net_input(self, X):
            """Calculate net input"""
            return np.dot(X, self.w_[1:]) + self.w_[0]
```

```
    def predict(self, X):
        """return class label after unit setup"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

In [6]:

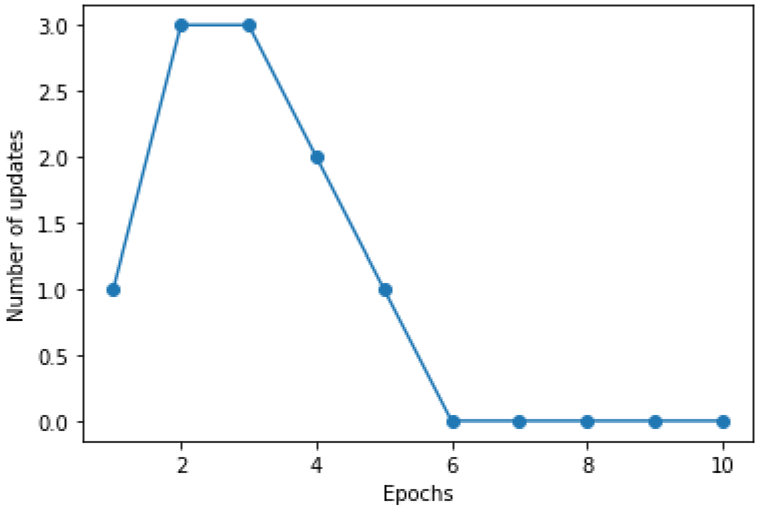
```
ppn = Perceptron(eta=0.1, n_iter=10)
```

In [7]:

```
ppn.fit(X, y);
```

In [8]:

```
plt.plot(range(1, len(ppn.errors_) + 1),ppn.errors_,
marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of updates')
plt.show()
```



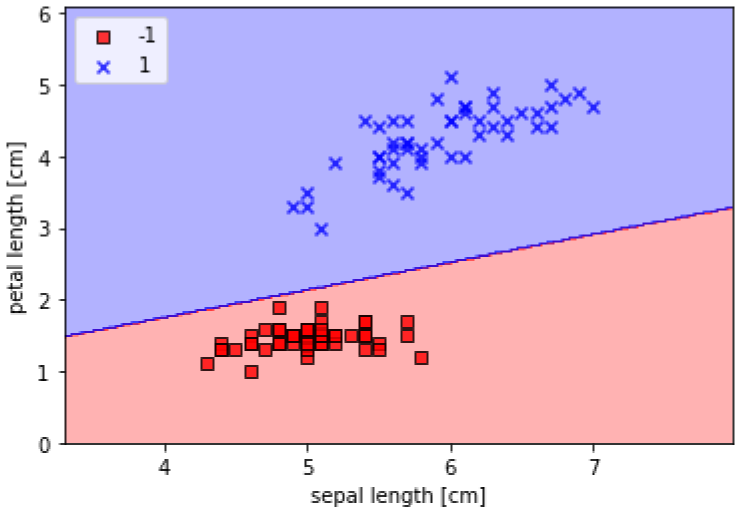
In [9]:

```
from matplotlib.colors import ListedColormap
def plot_decision_regions(X, y, classifier, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max,
resolution),
                           np.arange(x2_min, x2_max,
resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
```

```
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # plot class examples
    for idx, cl in enumerate(np.unique(y)):
         plt.scatter(x=X[y == cl, 0],
                     y=X[y == cl, 1],
                     alpha=0.8,
                     c=colors[idx],
                     marker=markers[idx],
                     label=cl,
                     edgecolor='black')
```

In [10]:

```
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('sepal length [cm]')
plt.ylabel('petal length [cm]')
plt.legend(loc='upper left')
plt.show();
```



In [11]:

```
df.shape
df.info()
df.describe()
```

```
<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
```

Out[11]:

	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

In [12]:

```
df.isnull().sum()
```

Out[12]:

```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
```

```
dtype: int64
```

In [13]:

```
data = df.drop_duplicates(subset ="Species",)
data
```

Out[13]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
50	7.0	3.2	4.7	1.4	Iris-versicolor
100	6.3	3.3	6.0	2.5	Iris-virginica

In [14]:

```
df.value_counts("Species")
```

Out[14]:

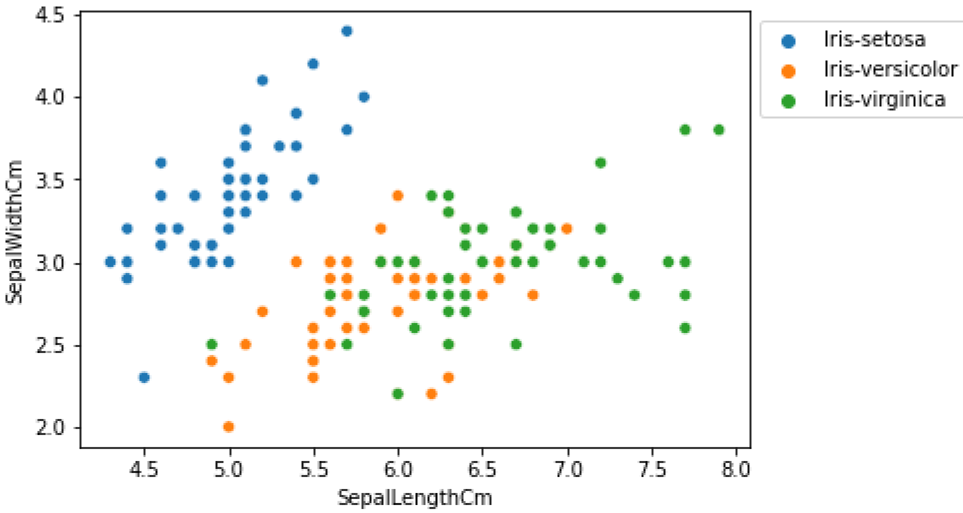
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50

dtype: int64
```

In [15]:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt



sns.countplot(x='Species', data=df, )
plt.show()
```

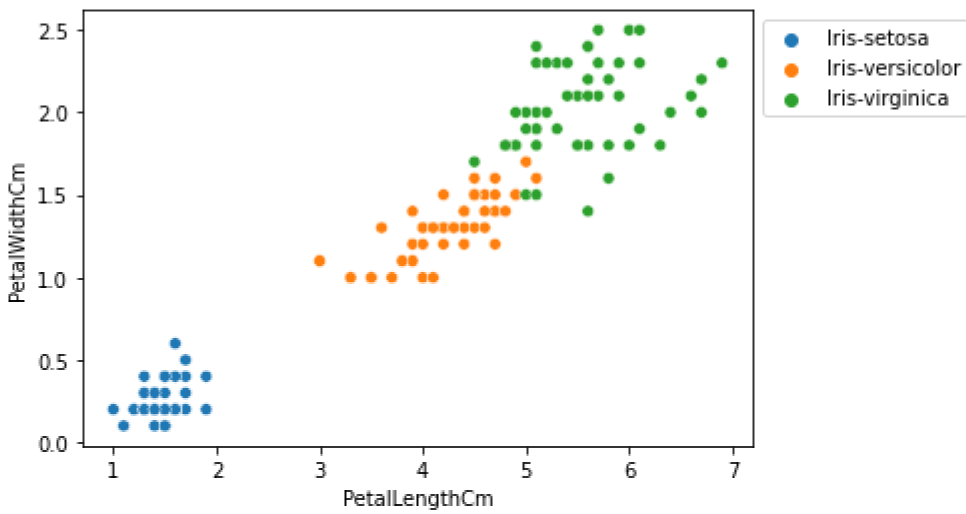
In [16]:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt



sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm',
                  hue='Species', data=df, )


# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)


plt.show()
```



In [17]:

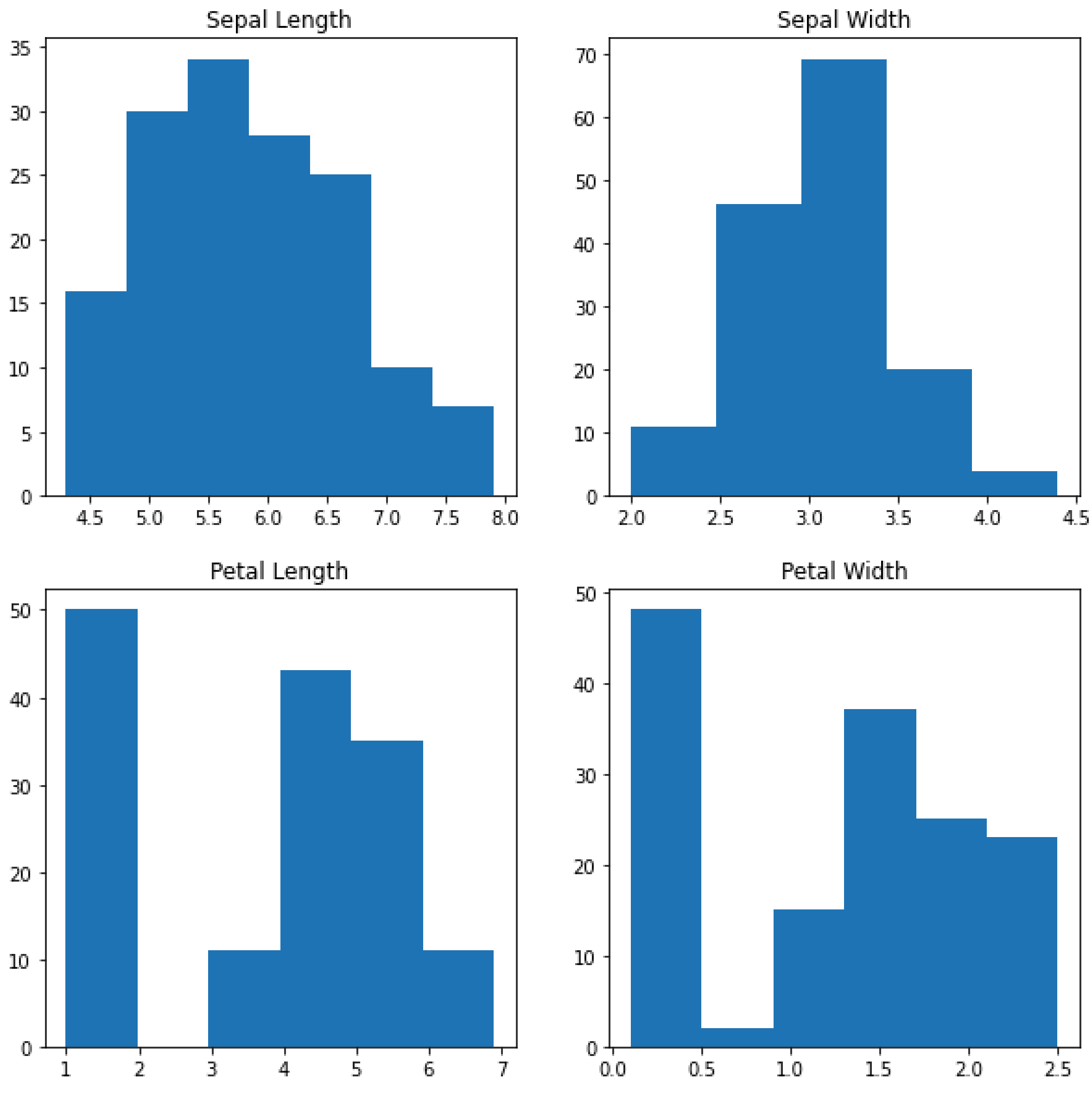
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt



sns.scatterplot(x='PetalLengthCm', y='PetalWidthCm',
                 hue='Species', data=df, )


# Placing Legend outside the Figure
plt.legend(bbox_to_anchor=(1, 1), loc=2)


plt.show()
```



In [18]:

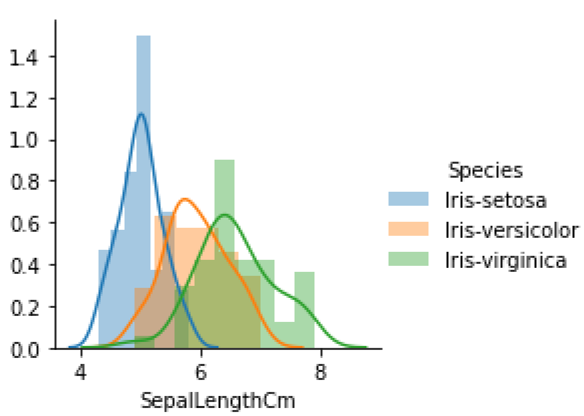
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt



fig, axes = plt.subplots(2, 2, figsize=(10,10))


axes[0,0].set_title("Sepal Length")
axes[0,0].hist(df['SepalLengthCm'], bins=7)


axes[0,1].set_title("Sepal Width")
axes[0,1].hist(df['SepalWidthCm'], bins=5);


axes[1,0].set_title("Petal Length")
axes[1,0].hist(df['PetalLengthCm'], bins=6);
```

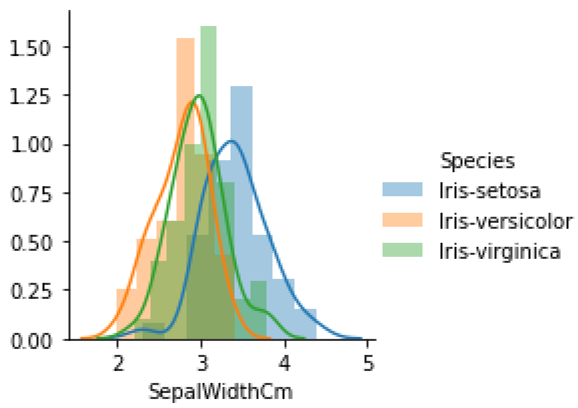
```
axes[1,1].set_title("Petal Width")
axes[1,1].hist(df['PetalWidthCm'], bins=6);
```

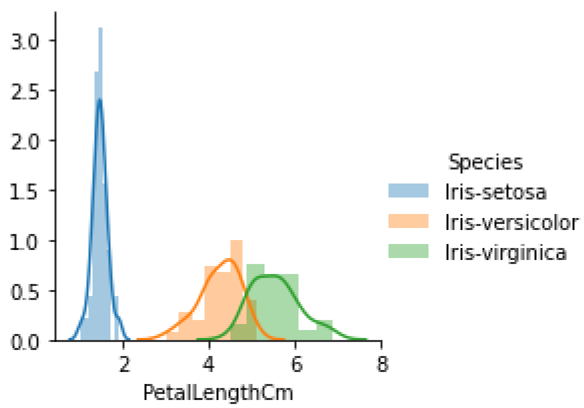


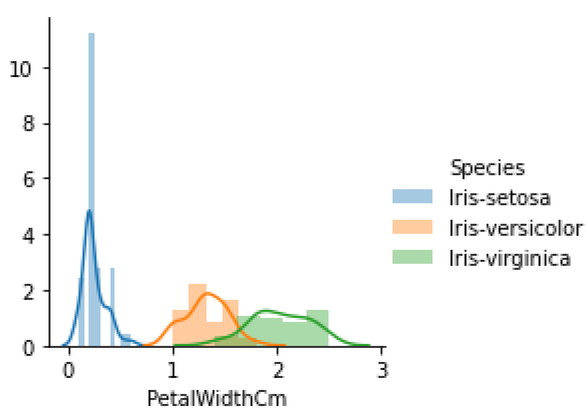

```
In [19]:
# importing packages
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "SepalLengthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "SepalWidthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "PetalLengthCm").add_legend()

plot = sns.FacetGrid(df, hue="Species")
plot.map(sns.distplot, "PetalWidthCm").add_legend()

plt.show()
```









In [20]:

```
data.corr(method='pearson')
```

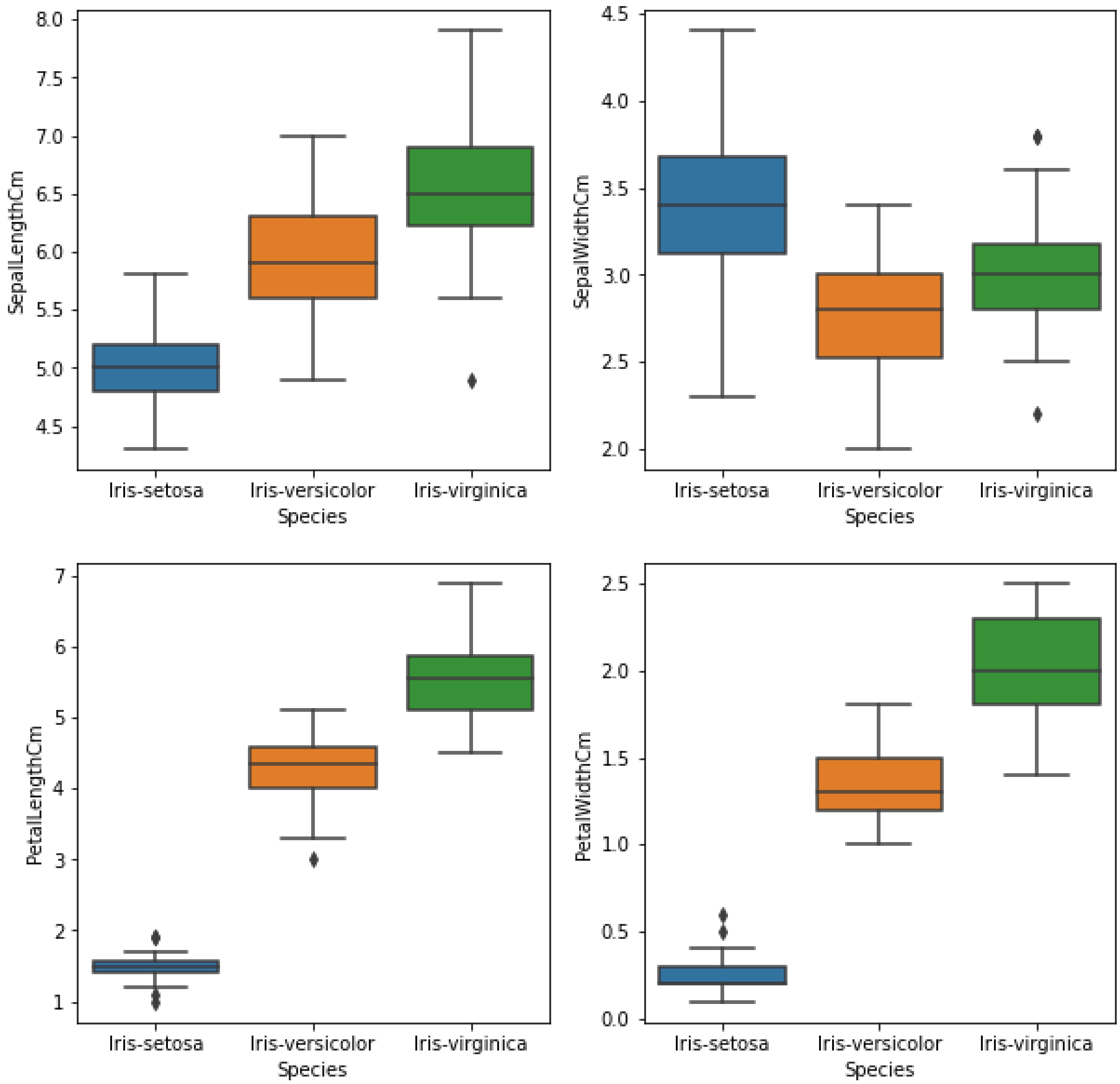
Out[20]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.999226	0.795795	0.643817
SepalWidthCm	-0.999226	1.000000	-0.818999	-0.673417
PetalLengthCm	0.795795	-0.818999	1.000000	0.975713
PetalWidthCm	0.643817	-0.673417	0.975713	1.000000

In [21]:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
```

```
def graph(y):
    sns.boxplot(x="Species", y=y, data=df)

plt.figure(figsize=(10,10))

# Adding the subplot at the specified
# grid position
plt.subplot(221)
graph('SepalLengthCm')

plt.subplot(222)
graph('SepalWidthCm')

plt.subplot(223)
graph('PetalLengthCm')

plt.subplot(224)
graph('PetalWidthCm')

plt.show()
```



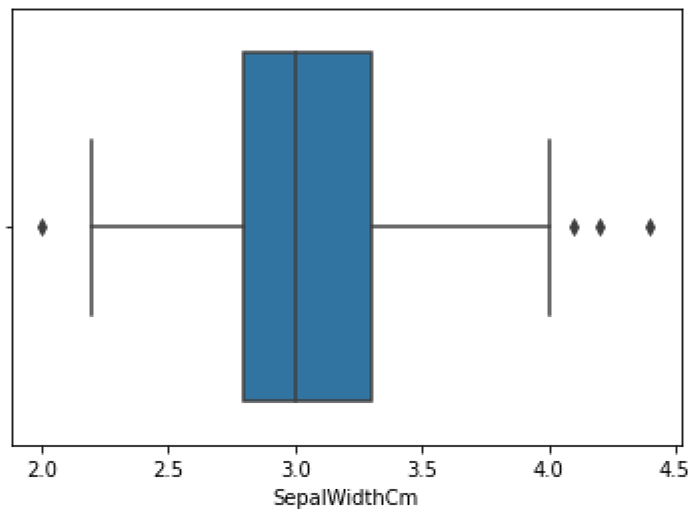
In [22]:

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt


# Load the dataset
df = pd.read_csv('../input/iris-classification/Iris.csv')
```

```
sns.boxplot(x='SepalWidthCm', data=df)
```

Out[22]:

```
<AxesSubplot:xlabel='SepalWidthCm'>
```



In [23]:

linkcode

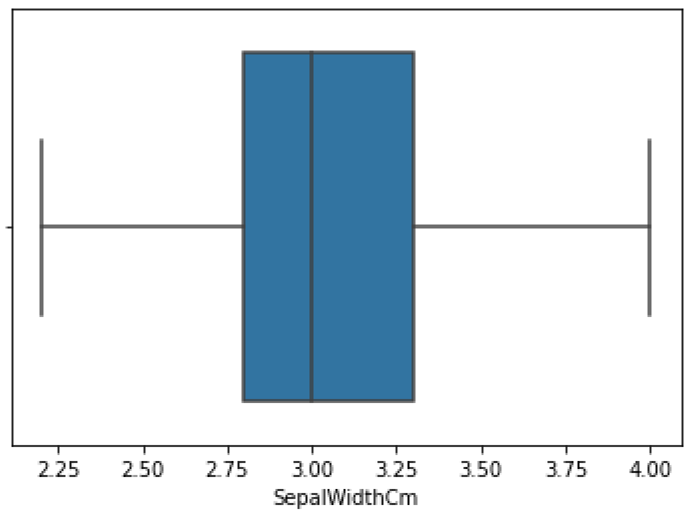
```
# Importing
import sklearn
from sklearn.datasets import load_boston
import pandas as pd
import seaborn as sns


# Load the dataset
```

```python
df = pd.read_csv('../input/iris-classification/Iris.csv')

# IQR
Q1 = np.percentile(df['SepalWidthCm'], 25,
                   interpolation = 'midpoint')

Q3 = np.percentile(df['SepalWidthCm'], 75,
                   interpolation = 'midpoint')
IQR = Q3 - Q1

print("Old Shape: ", df.shape)

# Upper bound
upper = np.where(df['SepalWidthCm'] >= (Q3+1.5*IQR))

# Lower bound
lower = np.where(df['SepalWidthCm'] <= (Q1-1.5*IQR))

# Removing the Outliers
df.drop(upper[0], inplace = True)
df.drop(lower[0], inplace = True)

print("New Shape: ", df.shape)

sns.boxplot(x='SepalWidthCm', data=df)
```

```
Old Shape:  (150, 5)
New Shape:  (146, 5)
```

Out[23]:

```
<AxesSubplot:xlabel='SepalWidthCm'>
```



[Reference link]