```
In [1]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
In [2]: df = pd.read_excel("Iris.xlsx")
```

```
In [3]: print(df.head())
```

```
   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
```

Explore the data

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

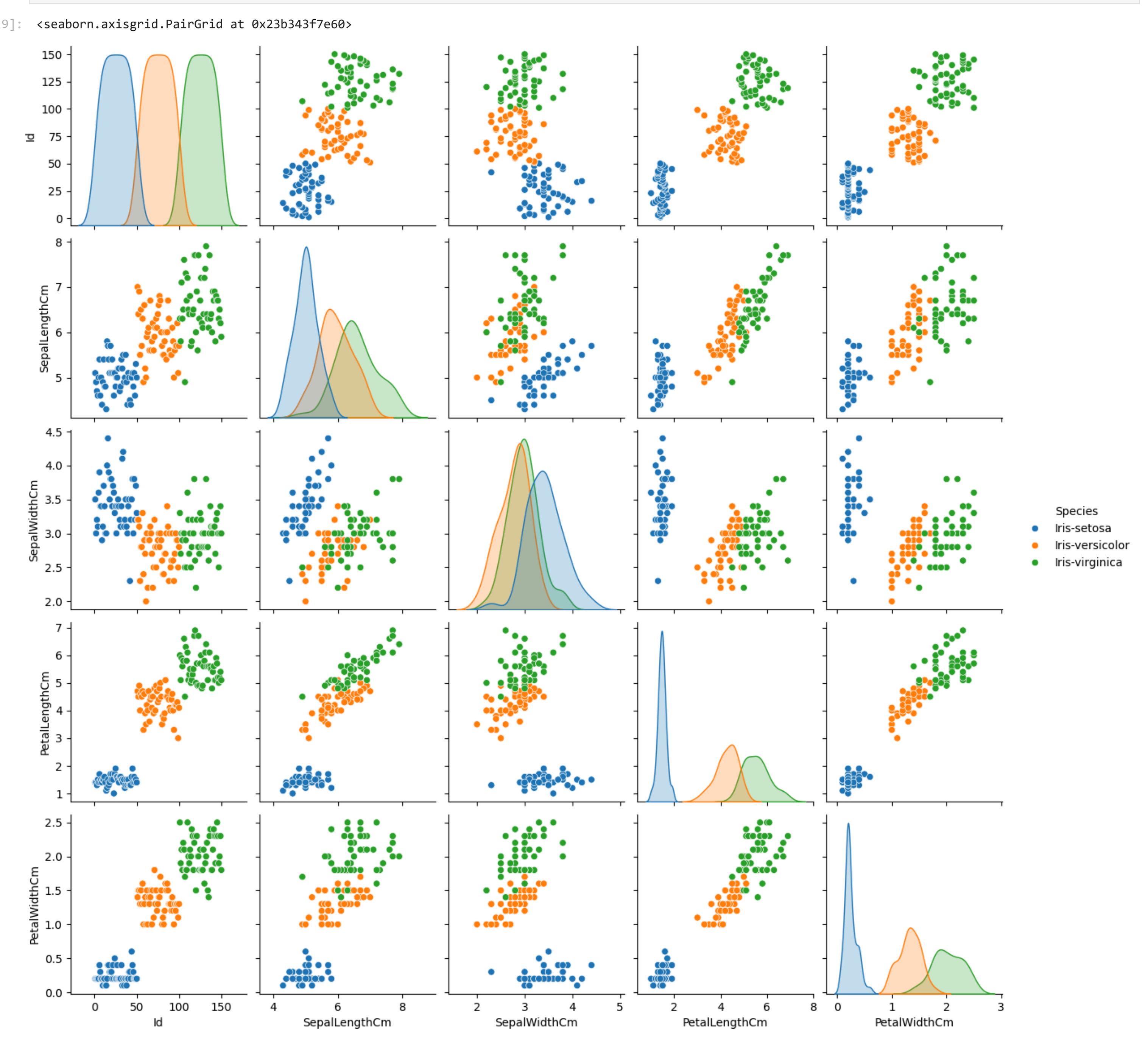
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 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
None
```

```
In [8]: print(df.columns)
```

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

```
In [9]: sns.pairplot(df, hue='Species')
```

```
Out[9]: <seaborn.axisgrid.PairGrid at 0x23b343f7e60>
```



check for missing values

```
In [10]: print("\nMissing values:\n", df.isnull().sum())
```

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

```
In [11]: df = df.drop(columns=['Id'])
```

Encode the 'Species' column to numbers

```
In [13]: le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
```

```
In [14]: 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)
```

Train a machine learning model

```
In [31]: # Define models
models = {
 "Logistic Regression": LogisticRegression(max_iter=200),
 "K-Nearest Neighbors": KNeighborsClassifier(),
 "Decision Tree": DecisionTreeClassifier(),
 "Random Forest": RandomForestClassifier(),
 "Support Vector Machine": SVC()
}
```

```
In [32]: for name, model in models.items():
 model.fit(X_train, y_train)
 y_pred = model.predict(X_test)
```

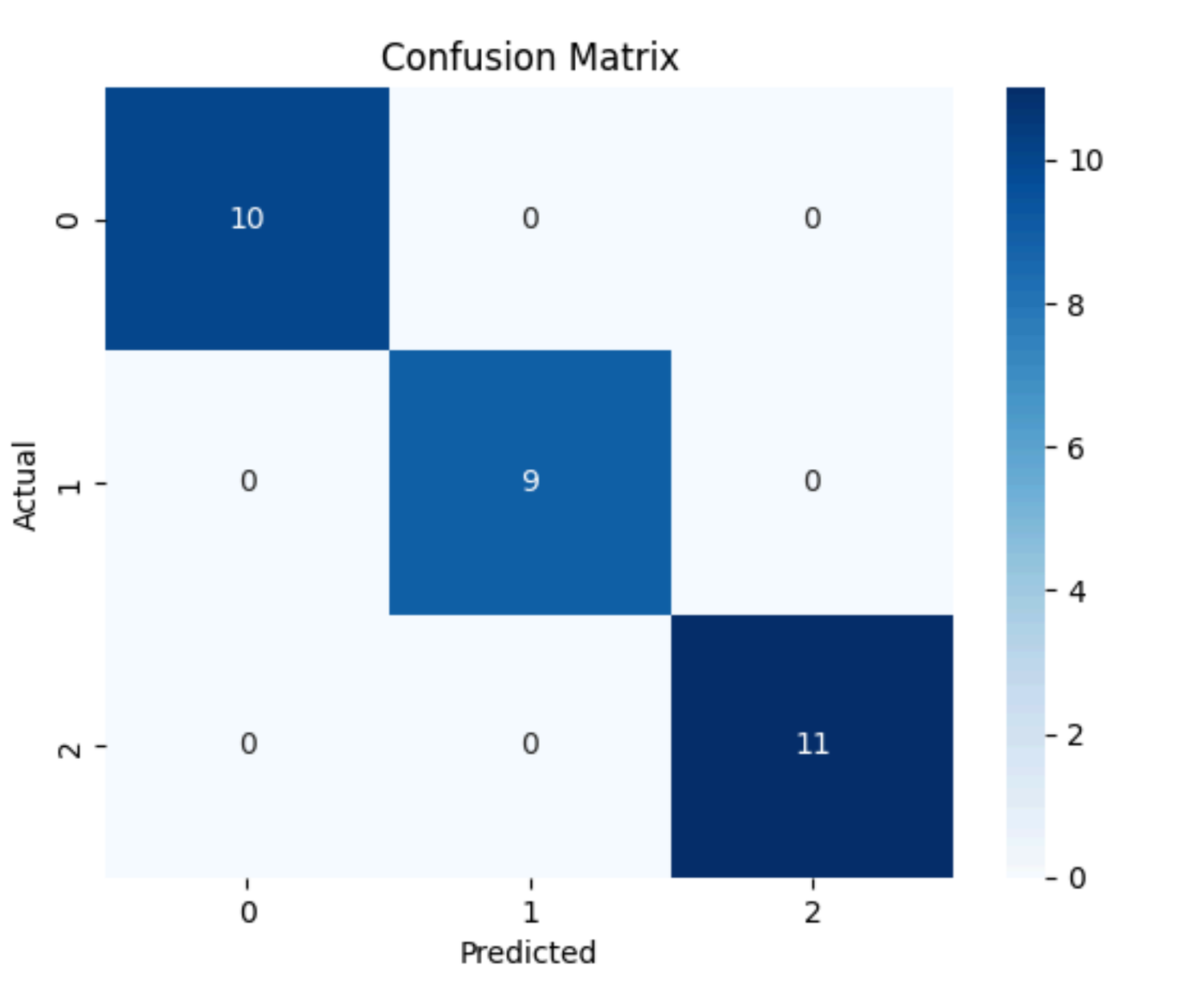
```
In [33]: print(f"\n ◆ {name}")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
◆ Support Vector Machine
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

Make a prediction and evaluate

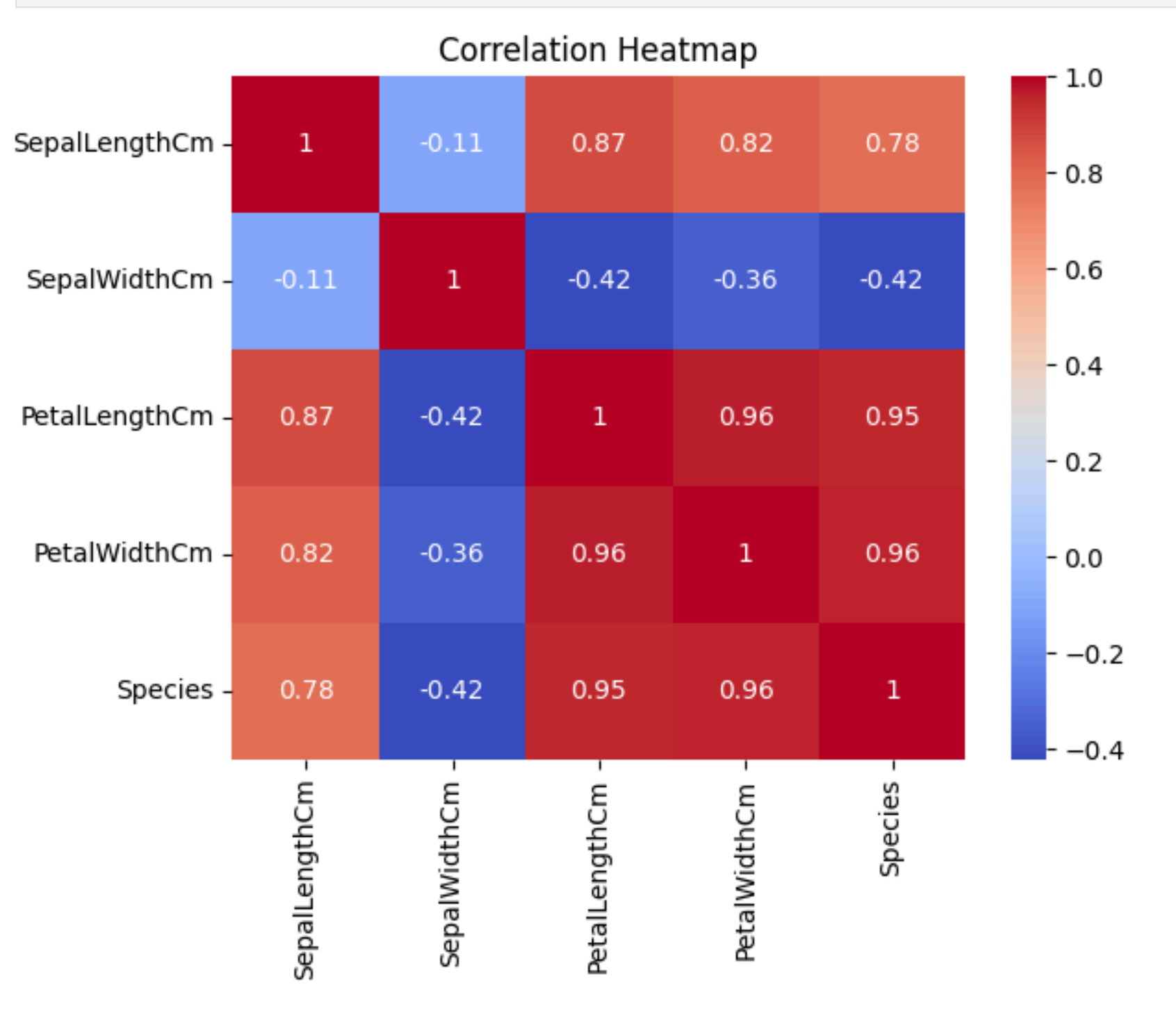
```
In [36]: # Predict using the trained model
y_pred = model.predict(X_test)

# Print evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot Confusion Matrix as Heatmap
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap="Blues", fmt='g')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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



A heatmap is a visual tool that uses colors to represent the strength or intensity of data relationships. In my Iris classification project, I used heatmaps to highlight how strongly different flower measurements are related and to visually analyze the model's performance through a confusion matrix. This made complex data patterns easier to interpret at a glance.

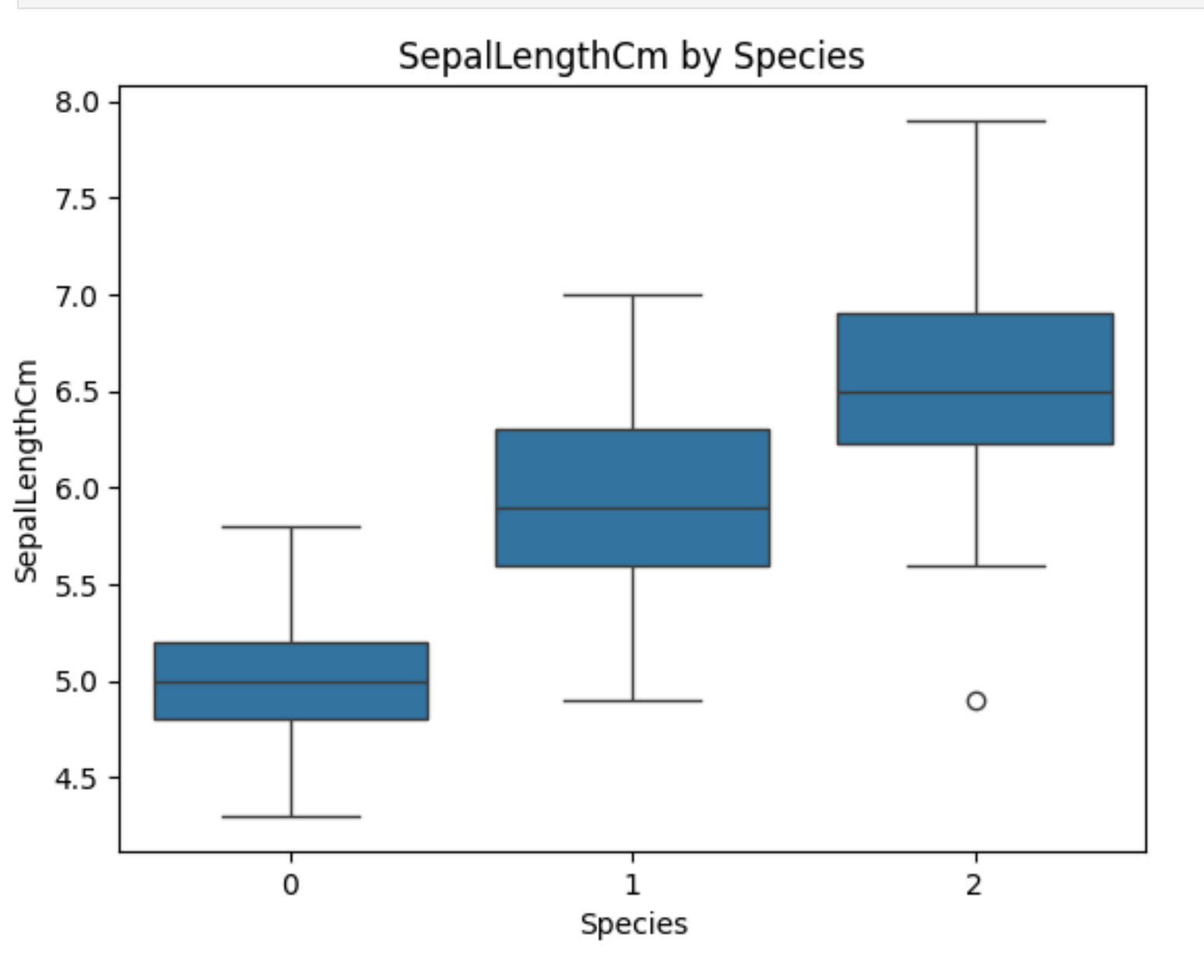
```
In [17]: import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.show()
```

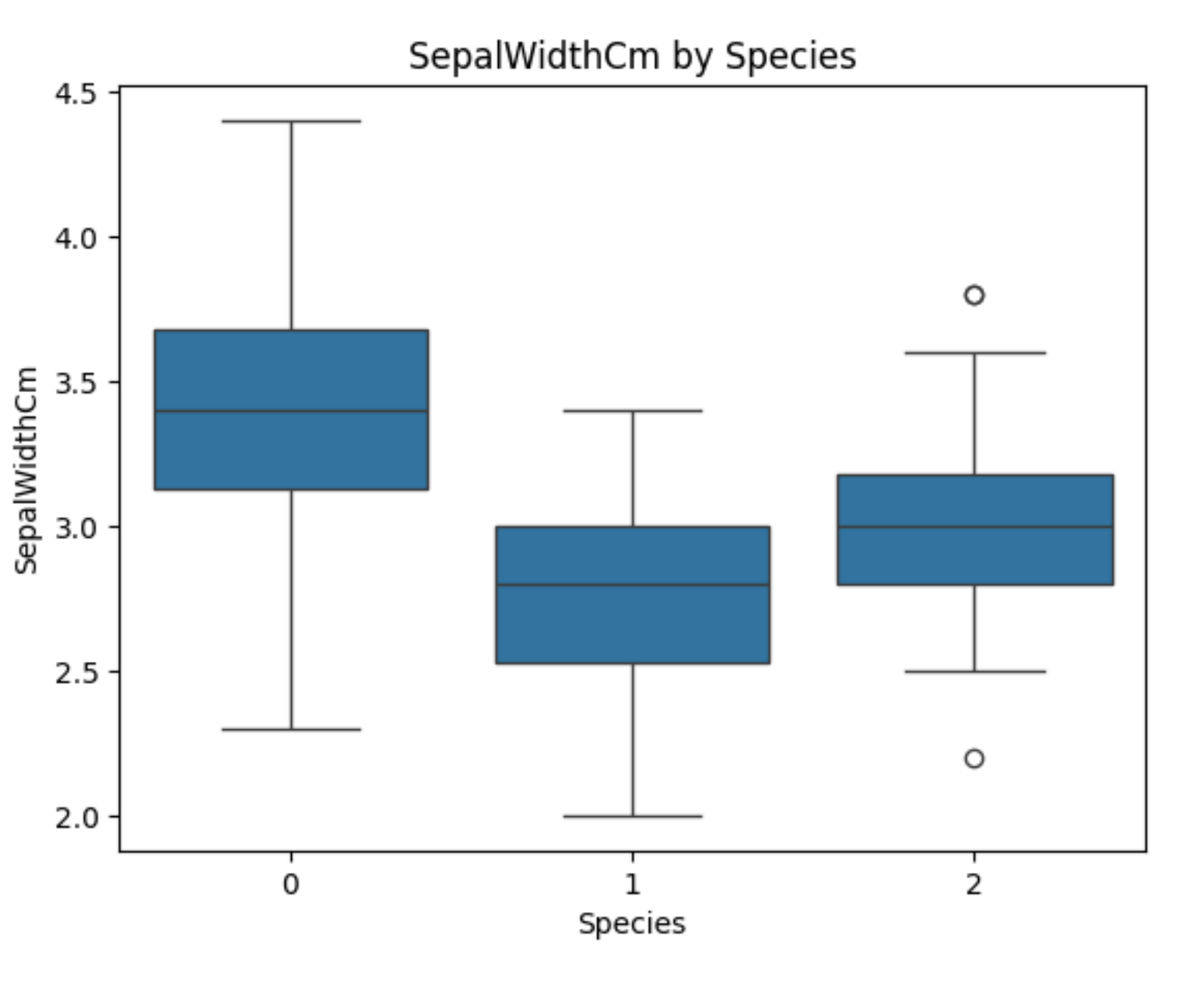


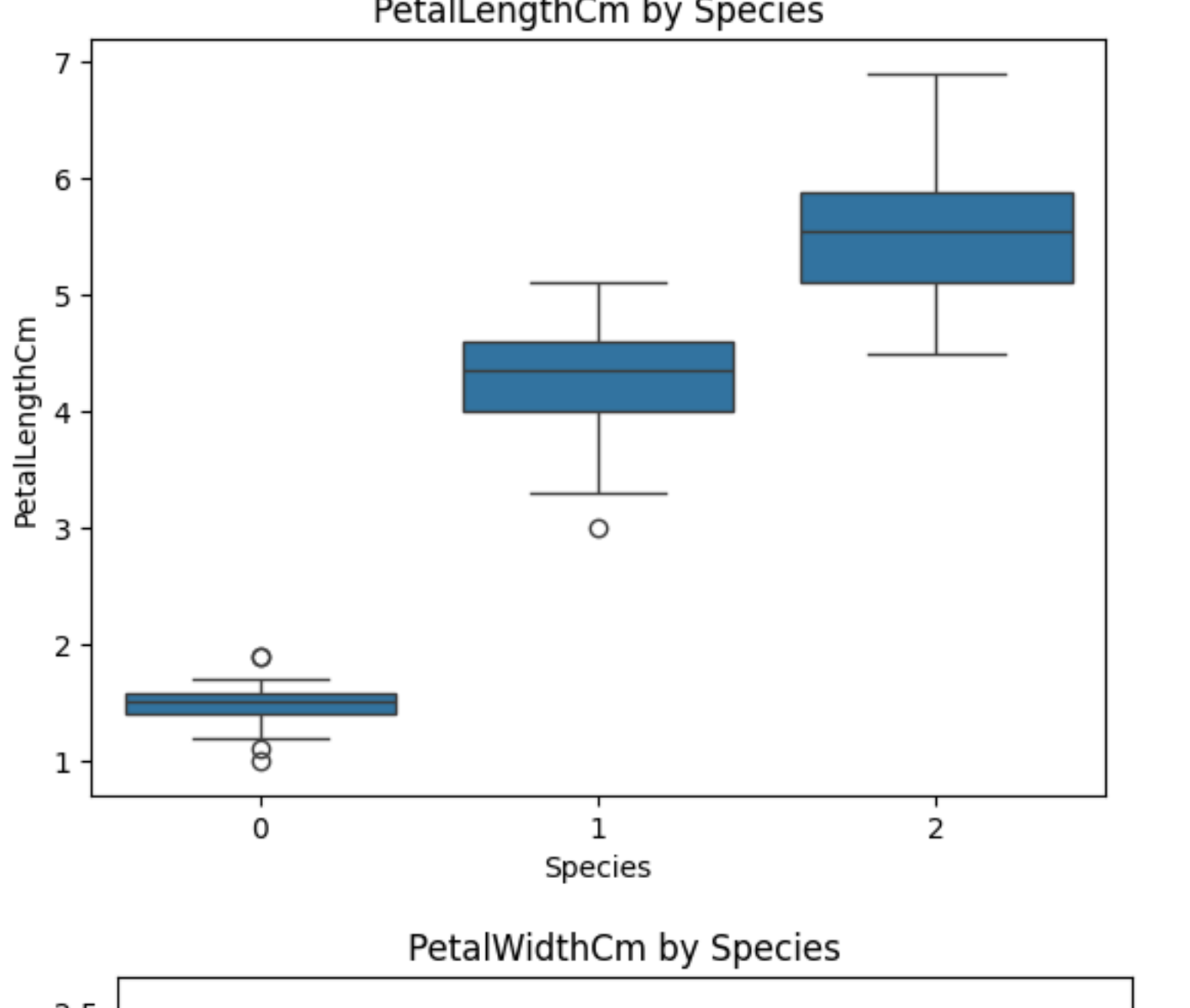
+1 = strong positive relationship (both increase together) 0 = no relationship −1 = strong negative relationship (one increases, the other decreases)

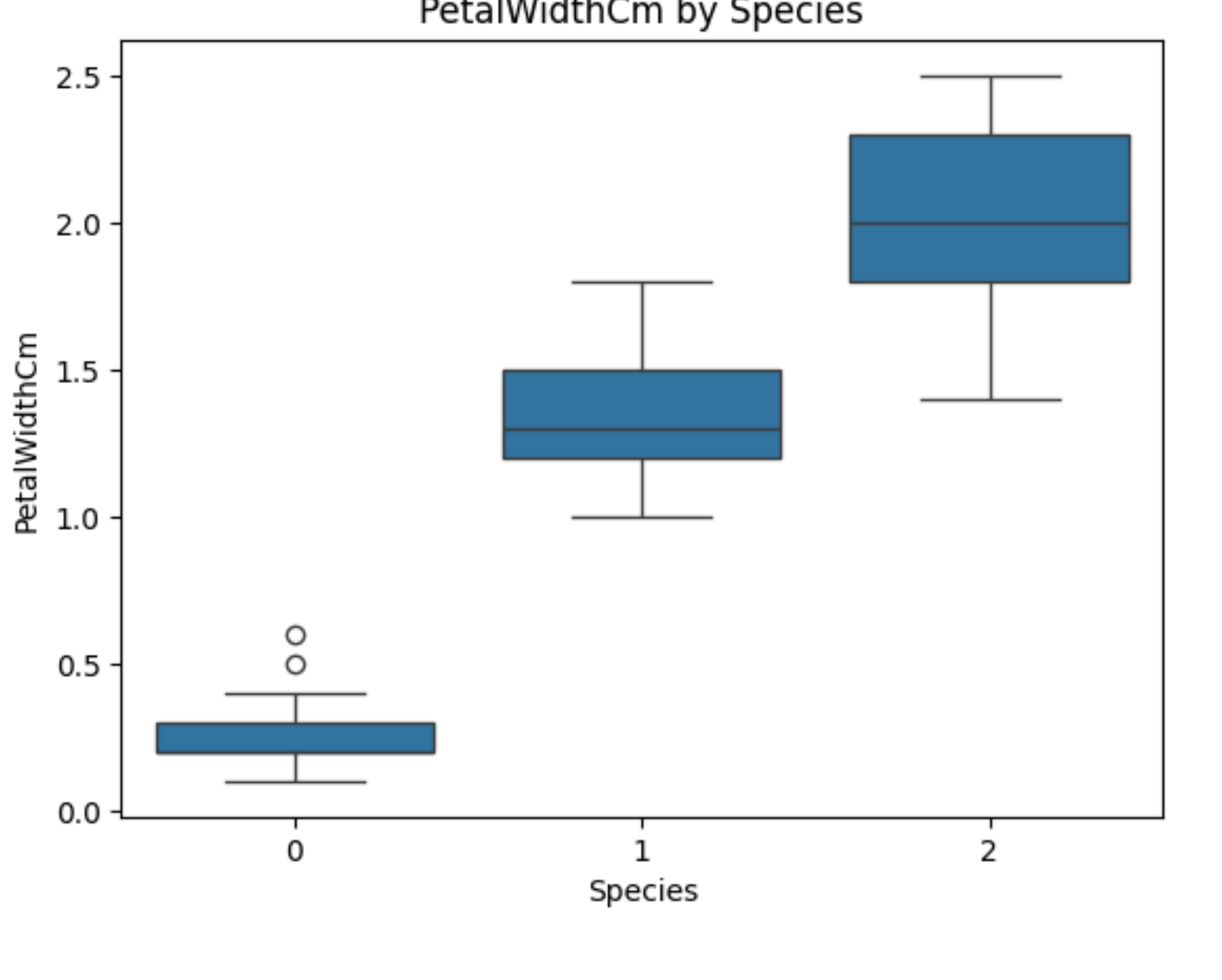
Boxplot

```
In [19]: # Boxplots
for col in df.columns[:-1]:
 sns.boxplot(x='Species', y=col, data=df)
 plt.title(f'{col} by Species')
 plt.show()
```









Exploratory Data Analysis (EDA)

```
In [20]: # import libraries

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
```

```
In [21]: # loading dataset
df = pd.read_excel("Iris.xlsx")
```

```
In [22]: # dropping unneccessary data
df.drop(columns=['Id'], inplace=True)
```

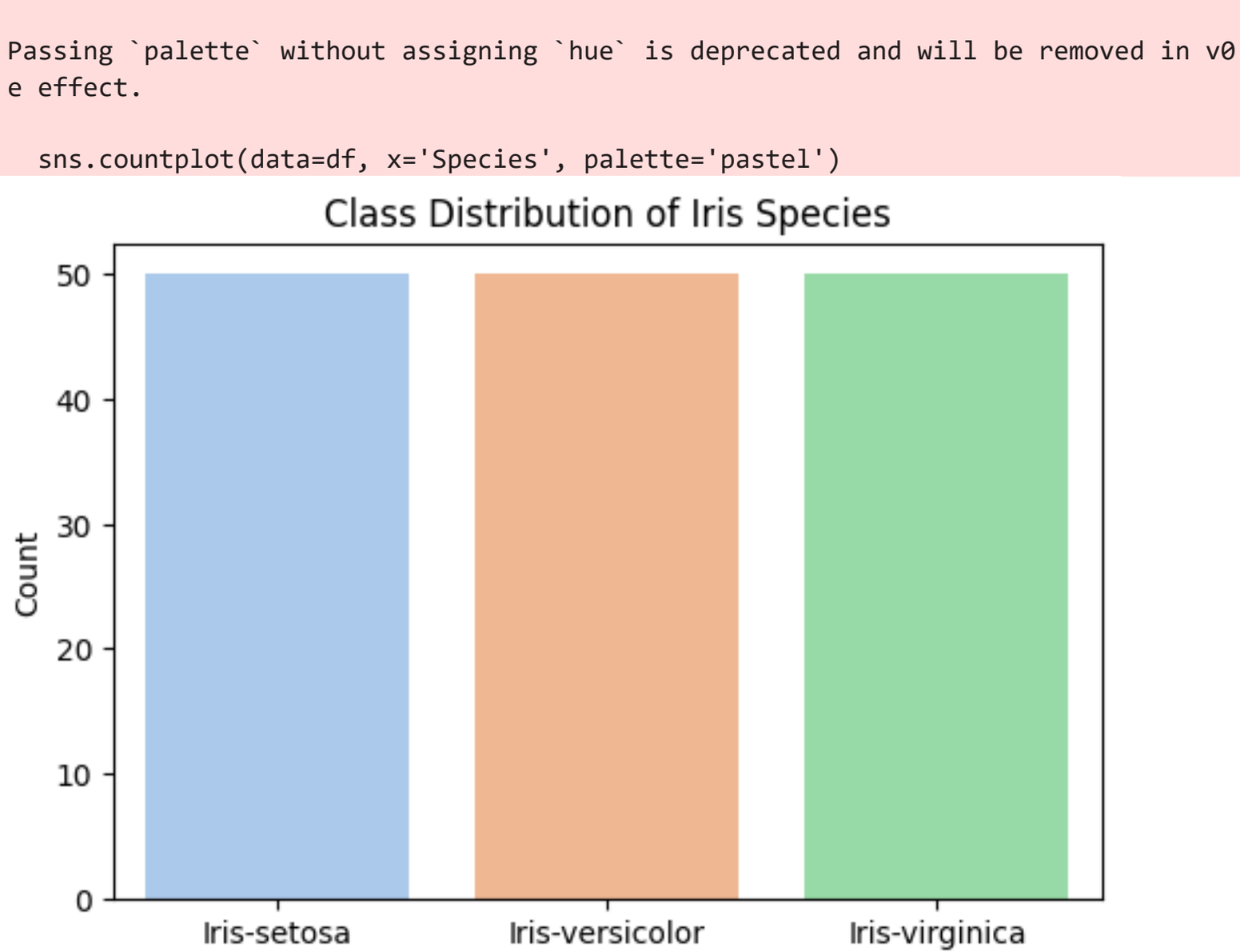
```
In [23]: # Encoding target label
le = LabelEncoder()
df['Species_encoded'] = le.fit_transform(df['Species'])
```

```
In [24]: # class distribution
print(df['Species'].value_counts())
```

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

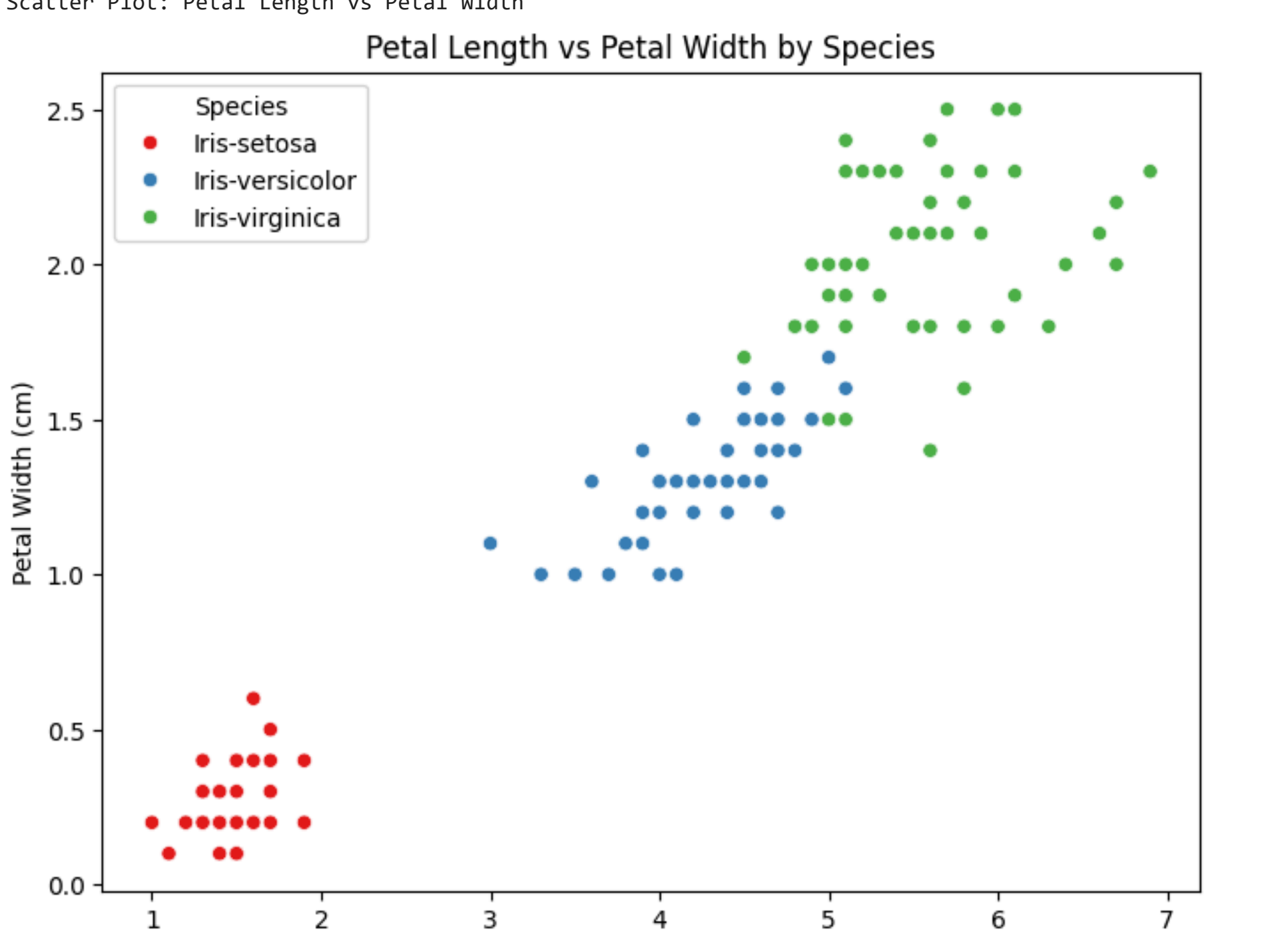
```
In [25]: # Visulize class disyribution
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='Species', palette='pastel')
plt.title("Class Distribution of Iris Species")
plt.xlabel("Species")
plt.ylabel("Count")
plt.show()
```

```
C:\Users\amrozia\AppData\Local\Temp\ipykernel_19396\2384506071.py:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the sam
e effect.

  sns.countplot(data=df, x='Species', palette='pastel')
```




```
In [27]: # Relation between features
print("\nScatter Plot: Petal Length vs Petal Width")
# Visualize separability using a scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='PetalLengthCm', y='PetalWidthCm', hue='Species', palette='Set1')
plt.title("Petal Length vs Petal Width by Species")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.show()
# ------------------------
# 3. Summary Statistics
# ------------------------
print("\nSummary Statistics:\n")
print(df.describe())
```

```
Scatter Plot: Petal Length vs Petal Width
```




```
Summary Statistics:

       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

       Species_encoded
count       150.000000
mean          1.000000
std           0.819232
min           0.000000
25%           0.000000
50%           1.000000
75%           2.000000
max           2.000000
```