In [1]:
```
pip install yellowbrick
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: yellowbrick in c:\users\abhim\appdata\roaming\python\python311\site-packages (1.5)
Requirement already satisfied: matplotlib!=3.0.0,>=2.0.2 in c:\programdata\anaconda3\lib\site-packages (from yellowbri
ck) (3.7.1)
Requirement already satisfied: scipy>=1.0.0 in c:\programdata\anaconda3\lib\site-packages (from yellowbrick) (1.10.1)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\programdata\anaconda3\lib\site-packages (from yellowbrick) (1.
3.0)
Requirement already satisfied: numpy>=1.16.0 in c:\programdata\anaconda3\lib\site-packages (from yellowbrick) (1.24.
3)
Requirement already satisfied: cycler>=0.10.0 in c:\programdata\anaconda3\lib\site-packages (from yellowbrick) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib!=3.0.
0,>=2.0.2->yellowbrick) (1.0.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib!=3.0.
0,>=2.0.2->yellowbrick) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib!=3.0.
0,>=2.0.2->yellowbrick) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\abhim\appdata\roaming\python\python311\site-packages
(from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (23.2)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib!=3.0.0,>
=2.0.2->yellowbrick) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib!=3.0.
0,>=2.0.2->yellowbrick) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\abhim\appdata\roaming\python\python311\site-packa
ges (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.2)
Requirement already satisfied: joblib>=1.1.1 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=1.0.0-
>yellowbrick) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>
=1.0.0->yellowbrick) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from pyt
hon-dateutil>=2.7->matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip install mlxtend
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting mlxtend
  Obtaining dependency information for mlxtend from https://files.pythonhosted.org/packages/1c/07/512f6a780239ad6ce06ce2aa7b4067583f5ddcfc7703a964a082c706a070/mlxtend-0.23.1-py3-none-any.whl.metadata
  Downloading mlxtend-0.23.1-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: scipy>=1.2.1 in c:\programdata\anaconda3\lib\site-packages (from mlxtend) (1.10.1)
Requirement already satisfied: numpy>=1.16.2 in c:\programdata\anaconda3\lib\site-packages (from mlxtend) (1.24.3)
Requirement already satisfied: pandas>=0.24.2 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from mlxtend) (2.2.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\programdata\anaconda3\lib\site-packages (from mlxtend) (1.3.0)
Requirement already satisfied: matplotlib>=3.0.0 in c:\programdata\anaconda3\lib\site-packages (from mlxtend) (3.7.1)
Requirement already satisfied: joblib>=0.13.2 in c:\programdata\anaconda3\lib\site-packages (from mlxtend) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (23.2)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\programdata\anaconda3\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-packages (from pandas>=0.24.2->mlxtend) (2022.7)
Requirement already satisfied: tzdata>=2022.7 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from pandas>=0.24.2->mlxtend) (2024.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\abhim\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Downloading mlxtend-0.23.1-py3-none-any.whl (1.4 MB)
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   ---------------------------------------- 0.0/1.4 MB ? eta -:--:--
   - -------------------------------------- 0.0/1.4 MB 667.8 kB/s eta 0:00:03
   --- ------------------------------------ 0.1/1.4 MB 1.0 MB/s eta 0:00:02
   ----- ---------------------------------- 0.2/1.4 MB 1.4 MB/s eta 0:00:01
   ------- -------------------------------- 0.3/1.4 MB 1.5 MB/s eta 0:00:01
   ---------- ----------------------------- 0.4/1.4 MB 1.5 MB/s eta 0:00:01
   ------------ --------------------------- 0.5/1.4 MB 1.6 MB/s eta 0:00:01
   --------------- ------------------------ 0.6/1.4 MB 1.6 MB/s eta 0:00:01
   ------------------ --------------------- 0.7/1.4 MB 1.7 MB/s eta 0:00:01
   -------------------- ------------------- 0.8/1.4 MB 1.7 MB/s eta 0:00:01
   ----------------------- ---------------- 0.8/1.4 MB 1.7 MB/s eta 0:00:01
   ------------------------- -------------- 0.9/1.4 MB 1.8 MB/s eta 0:00:01
   ---------------------------- ----------- 1.0/1.4 MB 1.8 MB/s eta 0:00:01
   ------------------------------ --------- 1.1/1.4 MB 1.8 MB/s eta 0:00:01
   --------------------------------- ------ 1.2/1.4 MB 1.8 MB/s eta 0:00:01
   ----------------------------------- ---- 1.3/1.4 MB 1.8 MB/s eta 0:00:01
   ------------------------------------- -- 1.4/1.4 MB 1.8 MB/s eta 0:00:01
   ---------------------------------------- 1.4/1.4 MB 1.8 MB/s eta 0:00:00
Installing collected packages: mlxtend
Successfully installed mlxtend-0.23.1
Note: you may need to restart the kernel to use updated packages.
```

In [6]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import statsmodels.api as sm
from sklearn.cluster import KMeans
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import classification_report, roc_curve, auc, confusion_matrix, cohen_kappa_score, matthews_corr
from sklearn.preprocessing import StandardScaler, label_binarize, LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, learning_curve
from sklearn import model_selection
from sklearn import metrics
from yellowbrick.classifier import ClassificationReport
from mlxtend.plotting import plot_decision_regions
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
import warnings
warnings.filterwarnings('ignore')
```

In [7]:
```
df= pd.read_csv(r"C:\Users\abhim\Downloads\Iris.csv")
```

In [8]:
```
df
```

Out[8]:

	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 [11]:
```
data  = df.drop('Id',axis=1)
```

In [12]:
```
data
```

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

150 rows × 5 columns

In [13]:
```
data.head()
```

Out[13]:

	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 [14]:
```
data.describe()
```

Out[14]:

	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

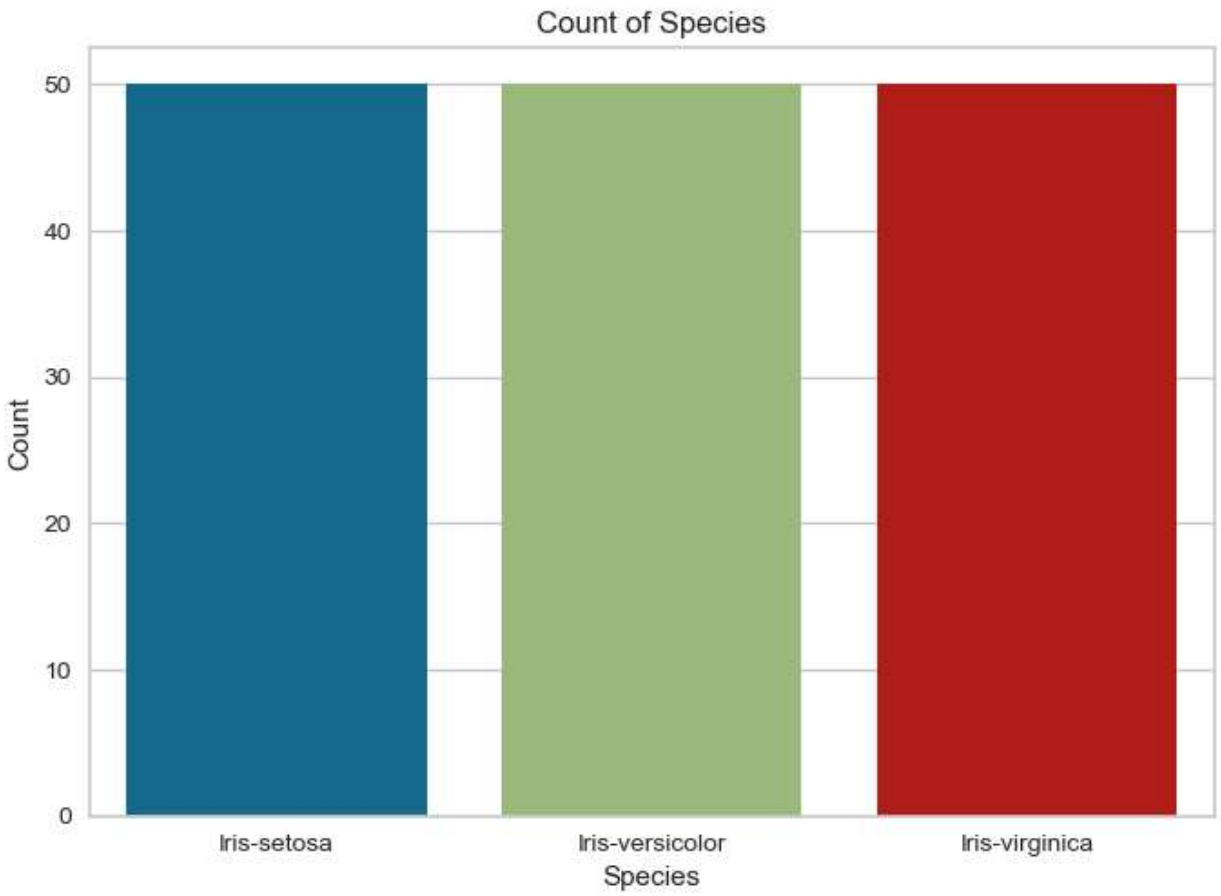
In [17]:
```
data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].mul(10)
```

Out[17]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	51.0	35.0	14.0	2.0
1	49.0	30.0	14.0	2.0
2	47.0	32.0	13.0	2.0
3	46.0	31.0	15.0	2.0
4	50.0	36.0	14.0	2.0
...	...	...	...	...
145	67.0	30.0	52.0	23.0
146	63.0	25.0	50.0	19.0
147	65.0	30.0	52.0	20.0
148	62.0	34.0	54.0	23.0
149	59.0	30.0	51.0	18.0

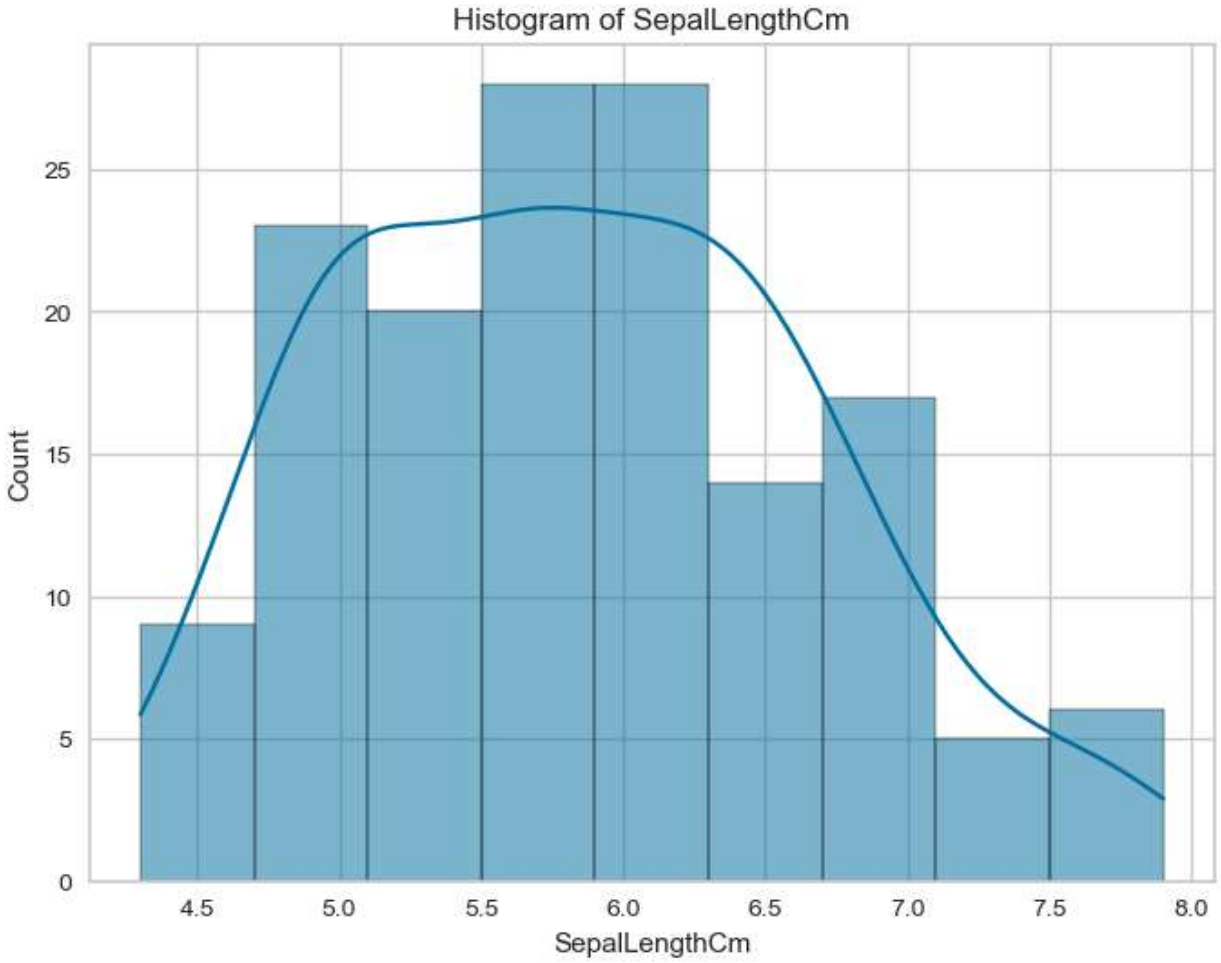
150 rows × 4 columns

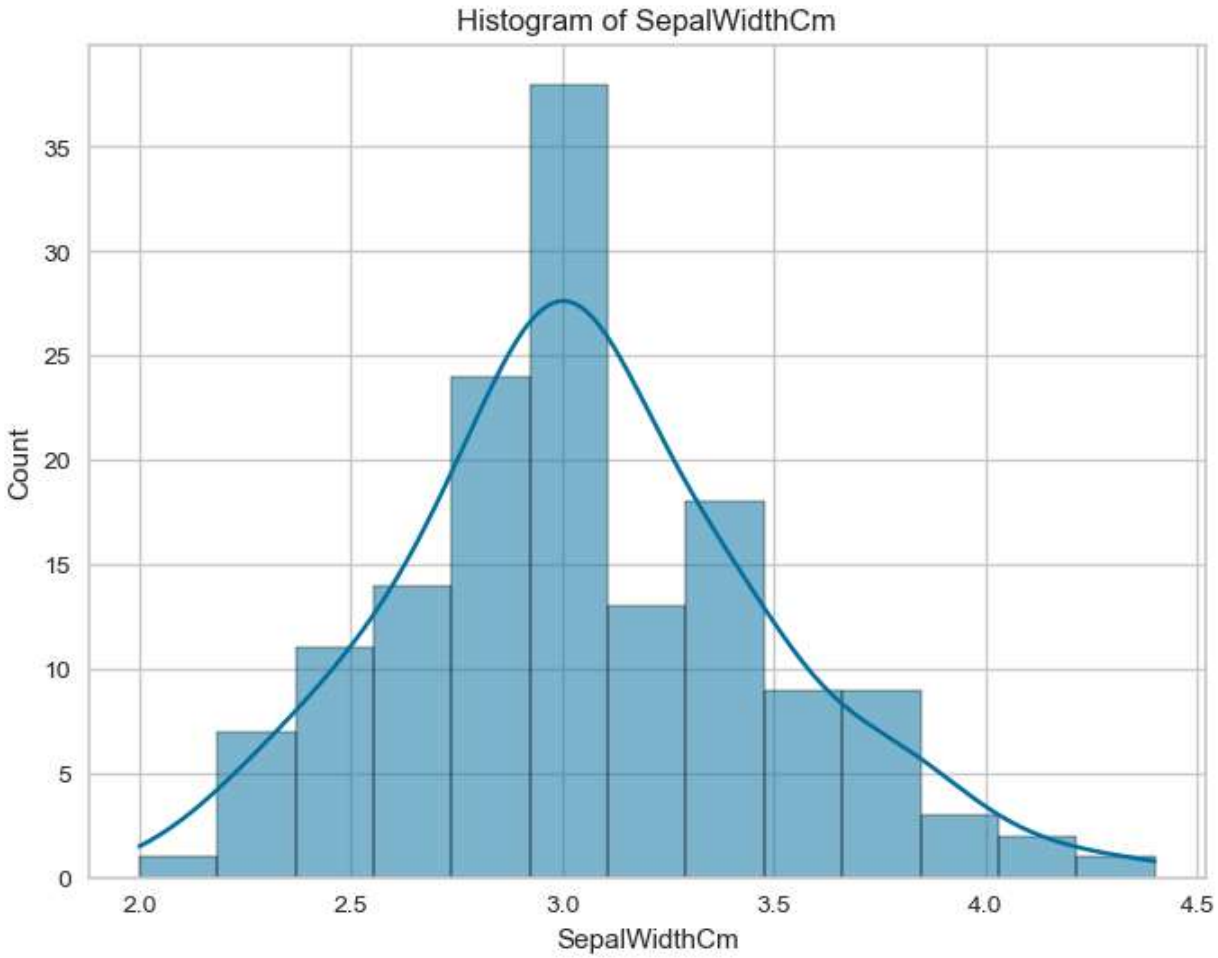
In [18]:
```
sns.countplot(data=data, x='Species')
plt.xlabel('Species')
plt.ylabel('Count')
plt.title('Count of Species')
plt.show()
```

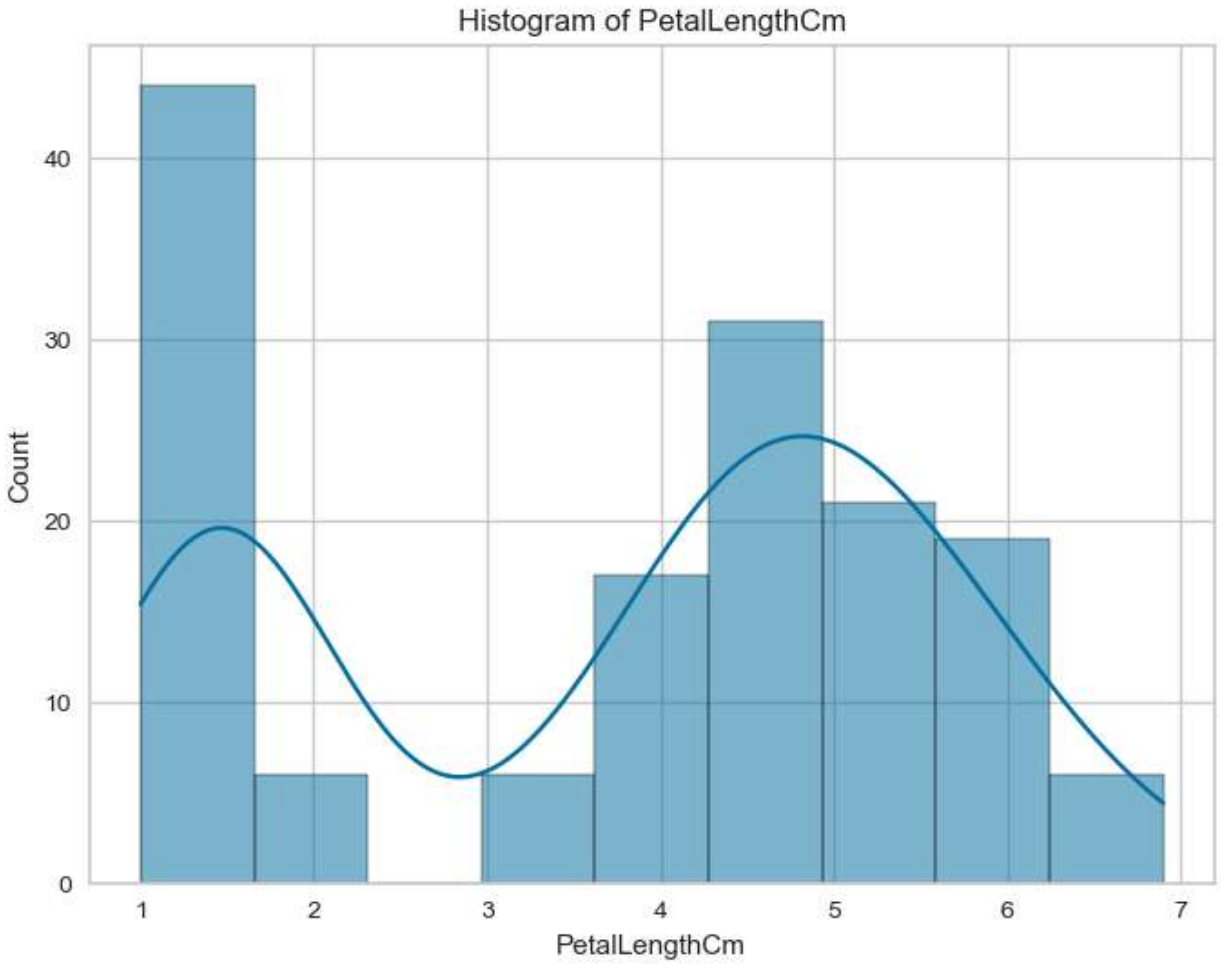


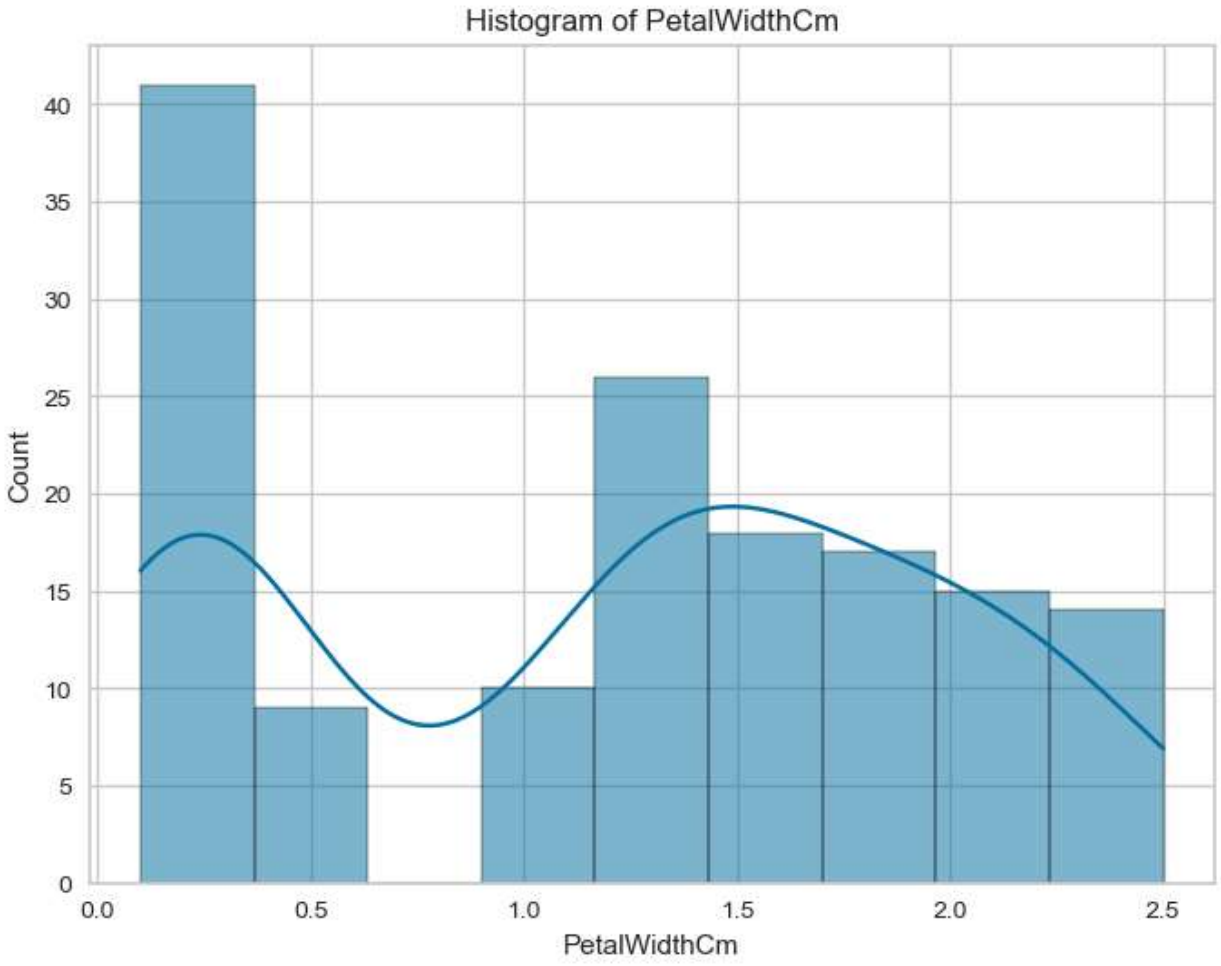
In [20]:
```
features = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']

for feature in features:
    plt.figure(figsize=(8, 6))
    sns.histplot(data=data, x=feature, kde=True)
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.title(f'Histogram of  {feature}')
    plt.show()
```

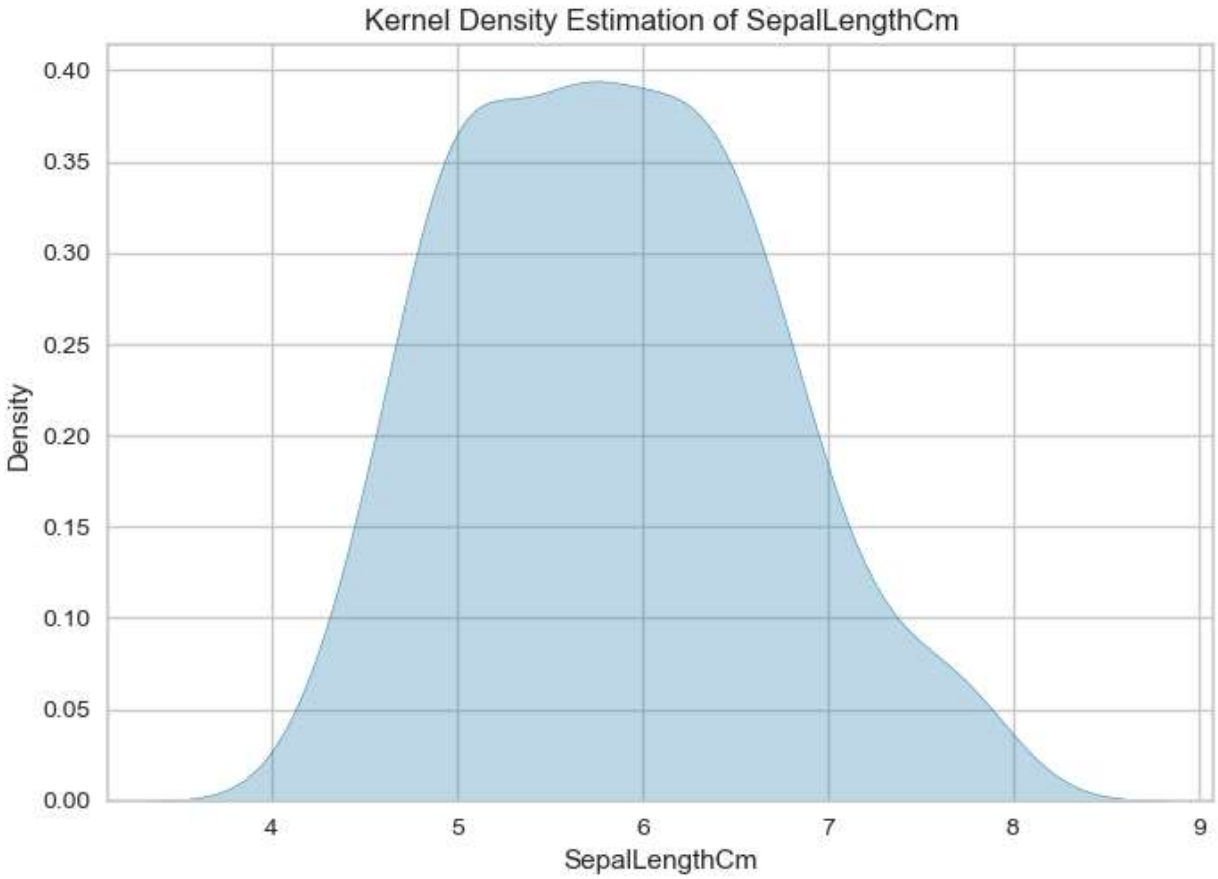


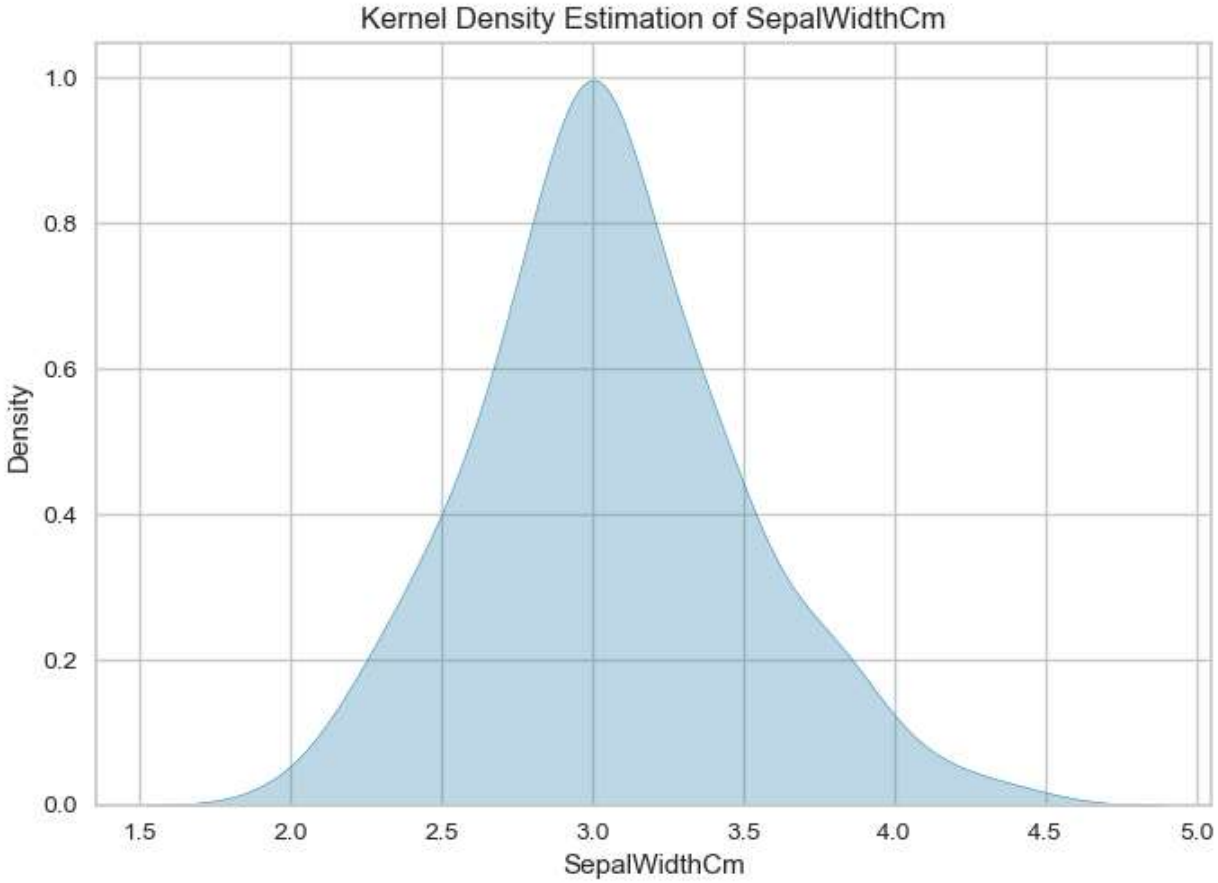

Histogram of SepalWidthCm
Count
SepalWidthCm
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5

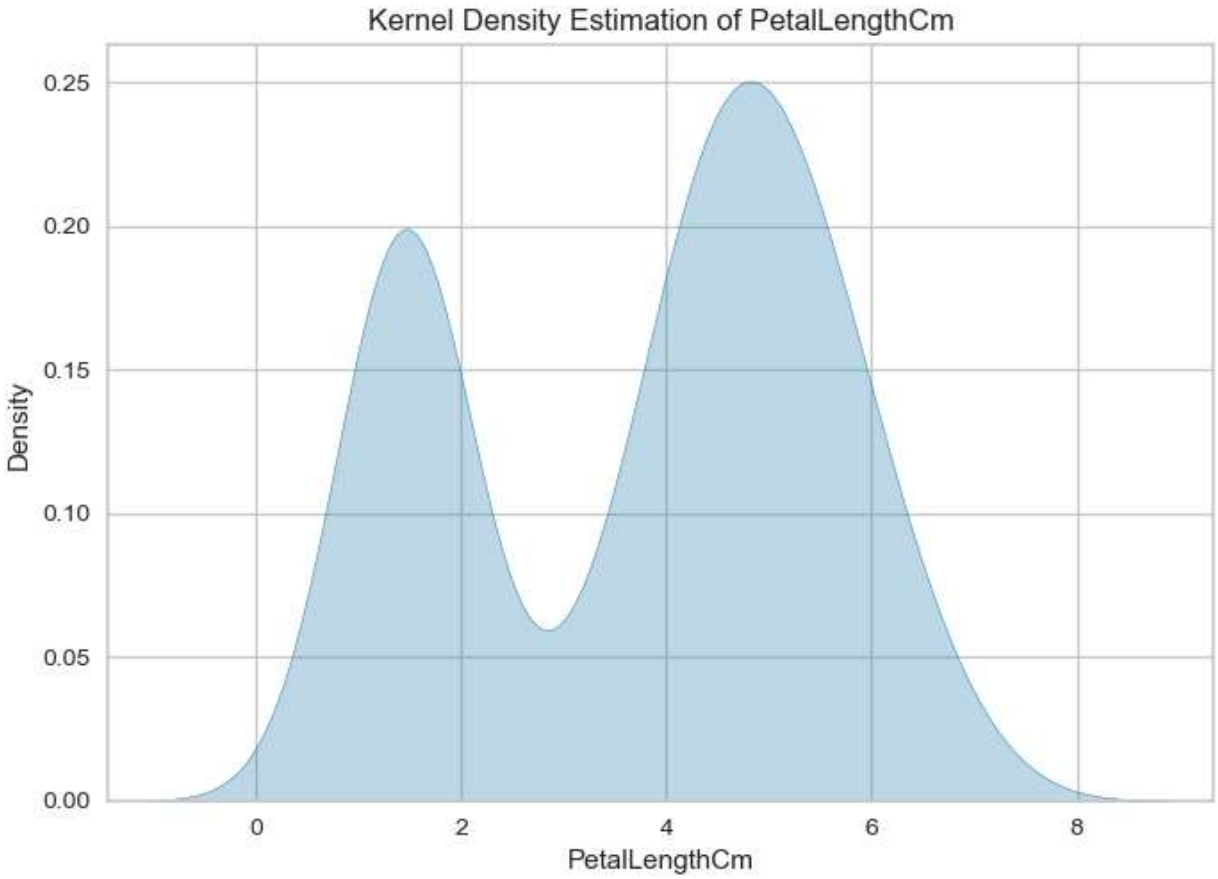

Histogram of PetalLengthCm
Count
PetalLengthCm
0
10
20
30
40
1
2
3
4
5
6
7

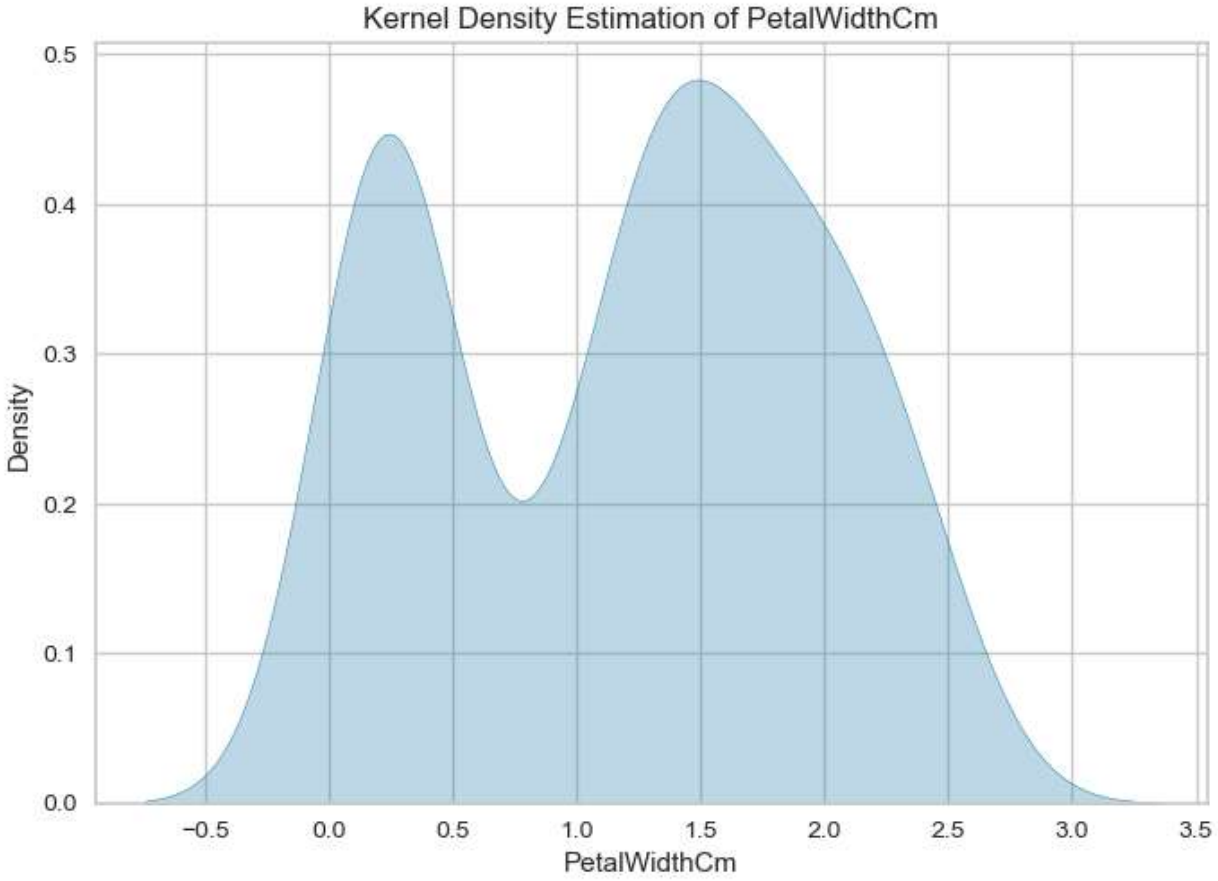


In [19]:
```
for feature in data.columns[:-1]:
    sns.kdeplot(data=data, x=feature, fill=True)
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.title(f'Kernel Density Estimation of {feature}')
    plt.show()
```


Kernel Density Estimation of SepalLengthCm
Density
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
4
5
6
7
8
9
SepalLengthCm


Kernel Density Estimation of SepalWidthCm
Density
1.0
0.8
0.6
0.4
0.2
0.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
SepalWidthCm





- Petal Length:

The histogram shows two distinct groups in the petal length data, which suggests that this feature might be a good indicator to differentiate between certain Iris species. One group is centered around a petal length of 1-2 cm, and the other is more spread out, ranging from 3 to 7 cm. This could potentially indicate a difference between a certain Iris species with shorter petals and the other species with longer petals.

- Petal Width:

Similar to petal length, petal width also shows two distinct groups. One group has a petal width of less than 1 cm, and the other ranges from 1 to 2.5 cm. This feature could also be useful in differentiating between Iris species.

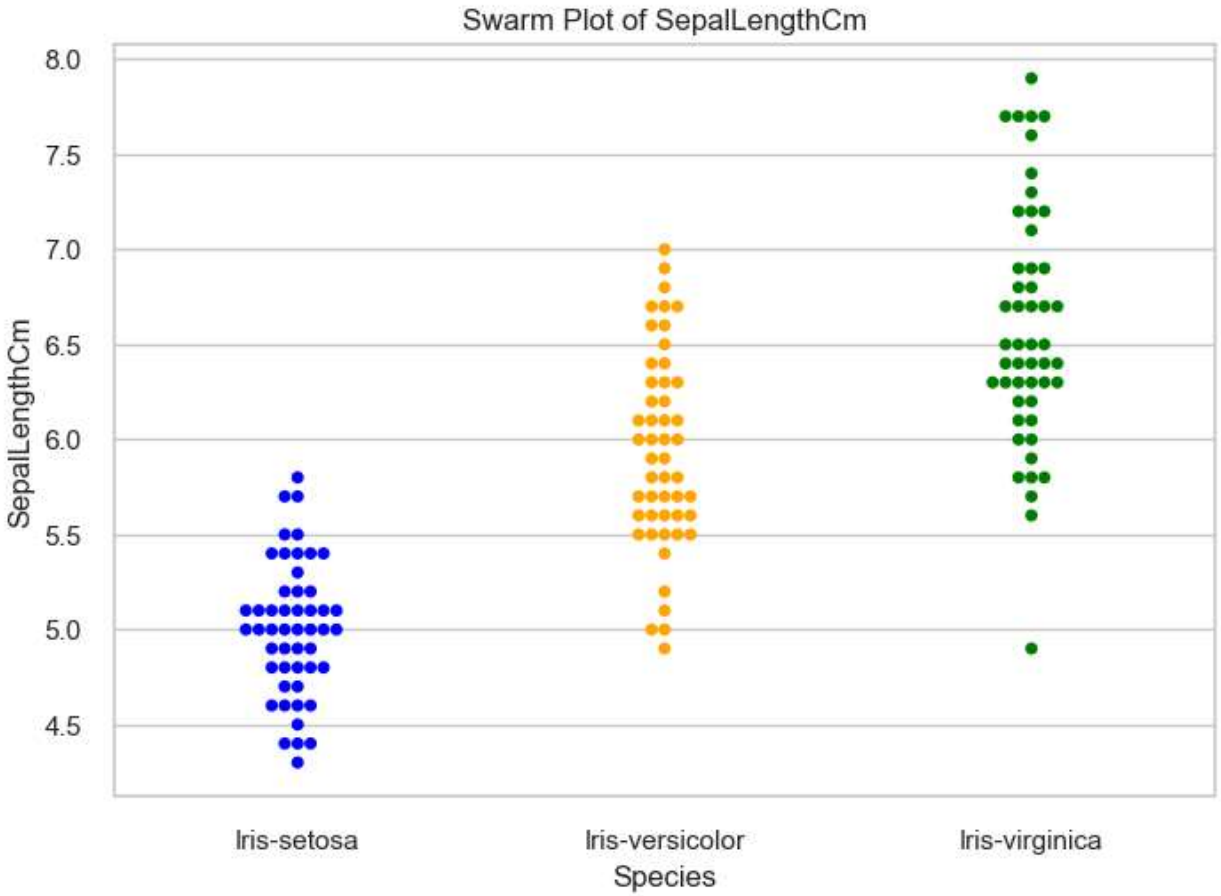
- Sepal Length:

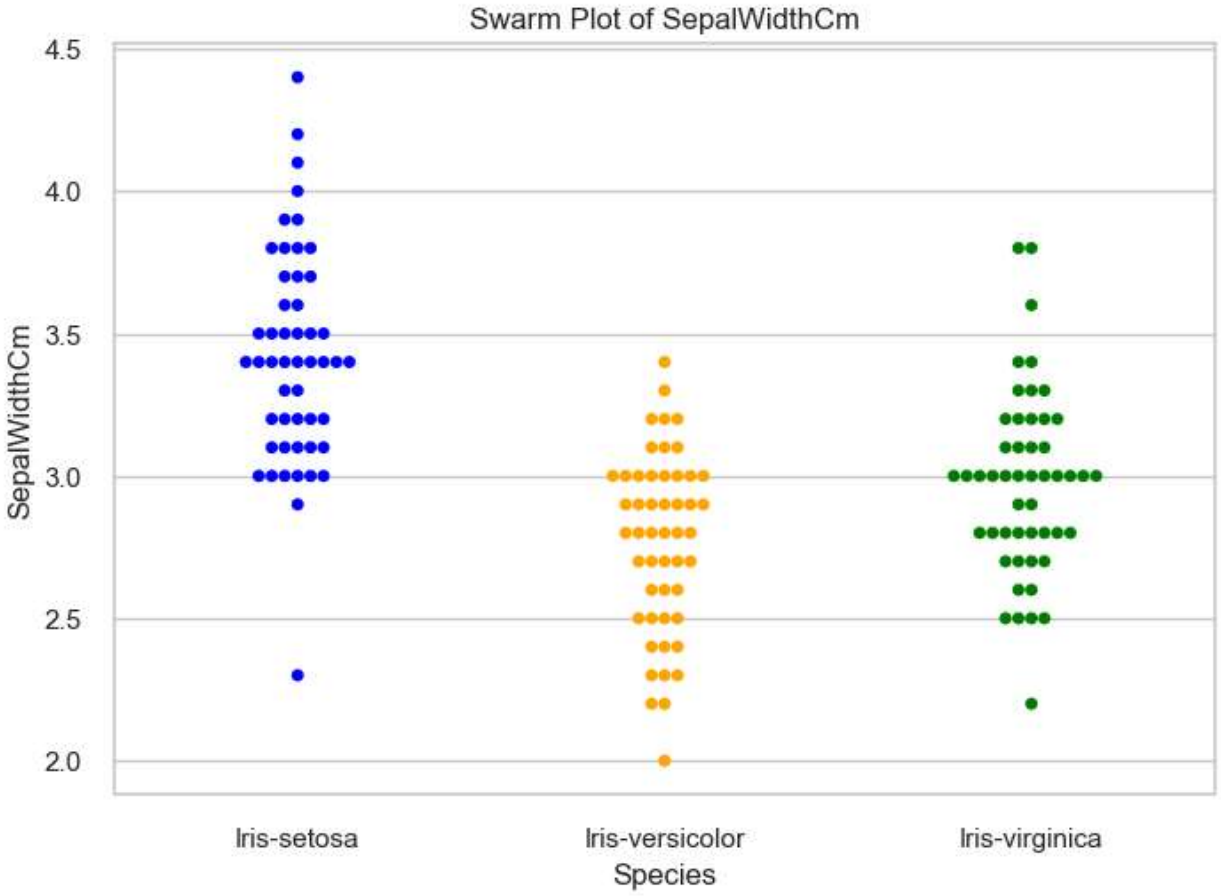
The sepal length data appears to follow a normal distribution, with most of the lengths centered around 5-6 cm. There doesn't seem to be a clear separation or grouping in the sepal length data, which suggests that sepal length alone might not be a good feature to differentiate between the species.

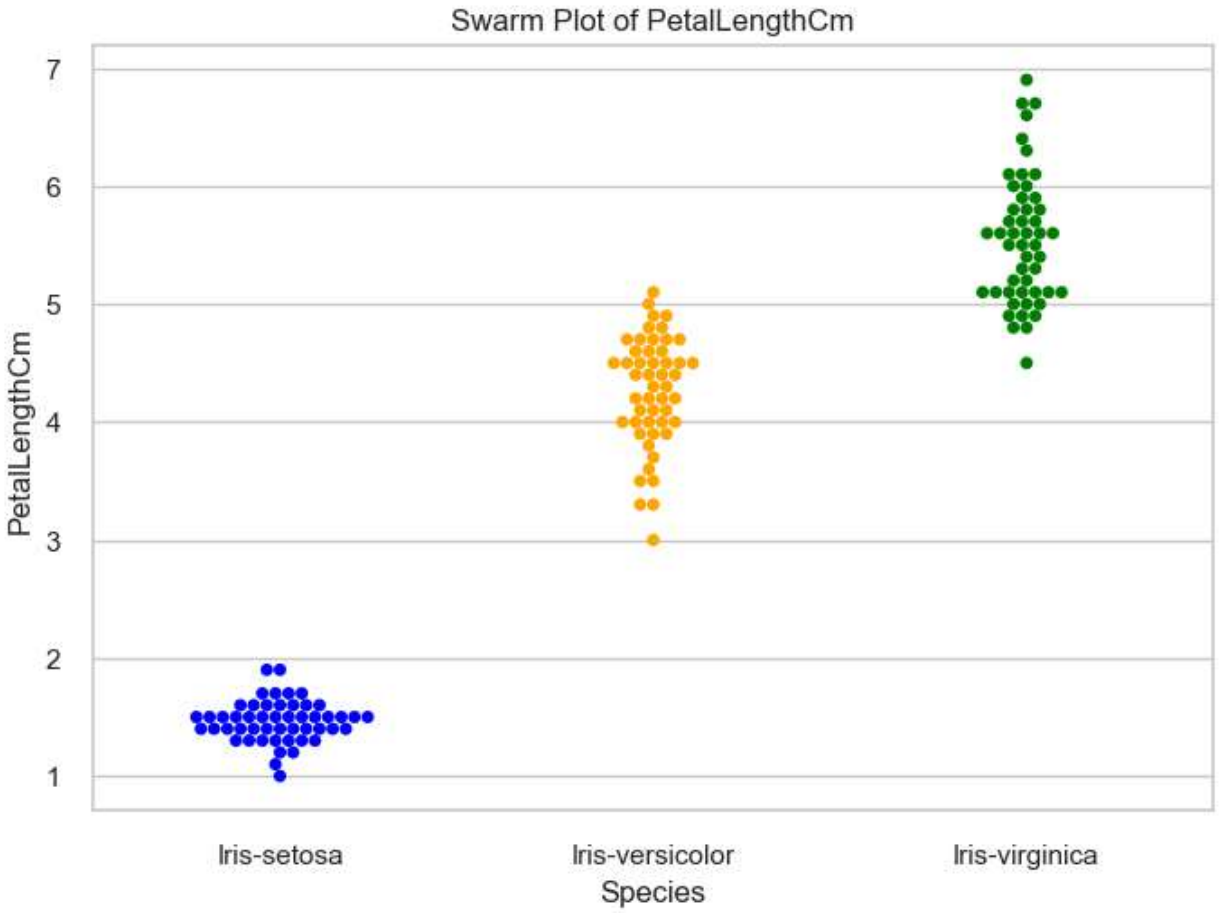
- Sepal Width:

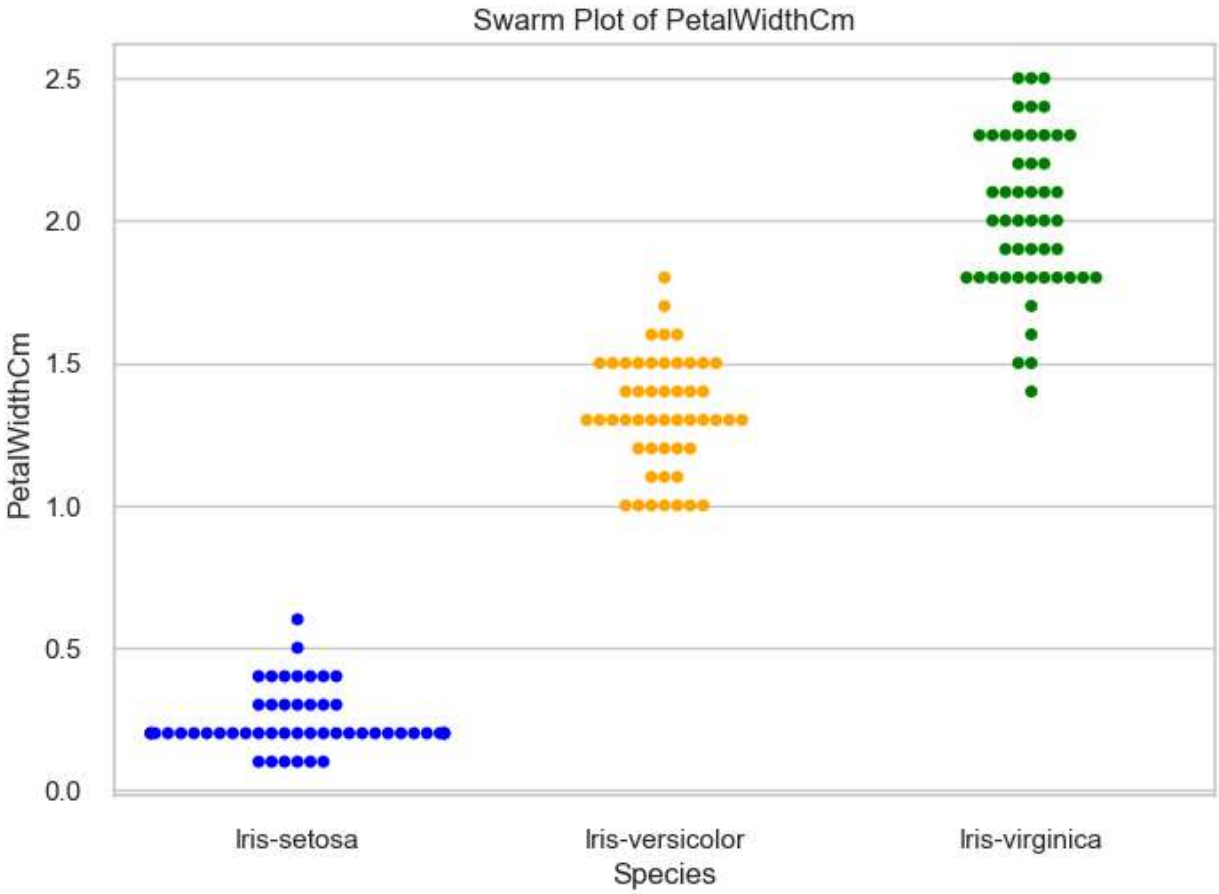
The sepal width data also appears to follow a normal distribution, with most of the widths centered around 3 cm. Like sepal length, sepal width also doesn't show clear separations or groups, suggesting that it might not be as useful for differentiating between the species as petal length or petal width.

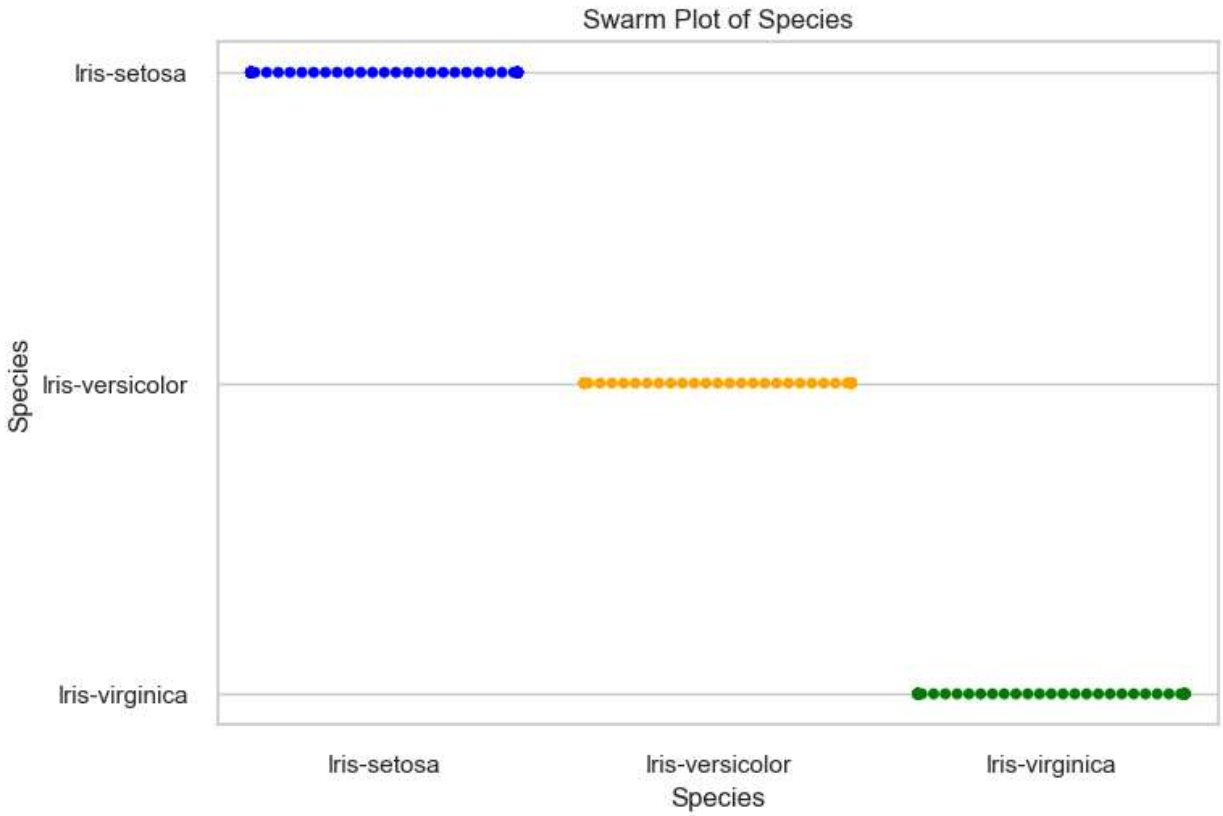
In [88]:
```
def swarm_plots(data):
    features = data.columns[:-1]
    species_colors = {'setosa': 'blue', 'versicolor': 'orange', 'virginica': 'green'}
    for feature in features:
        plt.figure()
        sns.swarmplot(data=data, x='Species', y=feature, palette=species_colors.values())
        plt.xlabel('Species')
        plt.ylabel(feature)
        plt.title(f'Swarm Plot of  {feature}')
        plt.show()
swarm_plots(data)
```


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


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


Swarm Plot of PetalLengthCm
PetalLengthCm
7
6
5
4
3
2
1
Iris-setosa
Iris-versicolor
Iris-virginica
Species


Swarm Plot of PetalWidthCm
PetalWidthCm
2.5
2.0
1.5
1.0
0.5
0.0
Iris-setosa
Iris-versicolor
Iris-virginica
Species



Analysis of Iris Flower Characteristics

Sepal Length:

- **Setosa**: Setosa species generally exhibits shorter sepals compared to Versicolor and Virginica.
- **Versicolor and Virginica**: These species have more overlap in sepal length, but Virginica tends to have slightly longer sepals than Versicolor.

Sepal Width:

- **Setosa**: Setosa species tends to have wider sepals compared to Versicolor and Virginica.
- **Versicolor and Virginica**: There is some overlap in sepal width between these species, with Versicolor generally having slightly narrower sepals compared to Virginica.

Petal Length:

- **Setosa**: Setosa species has significantly shorter petals compared to Versicolor and Virginica.
- **Versicolor and Virginica**: Versicolor and Virginica show differentiation in petal length, with Virginica generally having longer petals compared to Versicolor.

Petal Width:

- **Setosa**: Setosa species has narrower petals compared to Versicolor and Virginica.
- **Versicolor**: Versicolor has intermediate petal width.

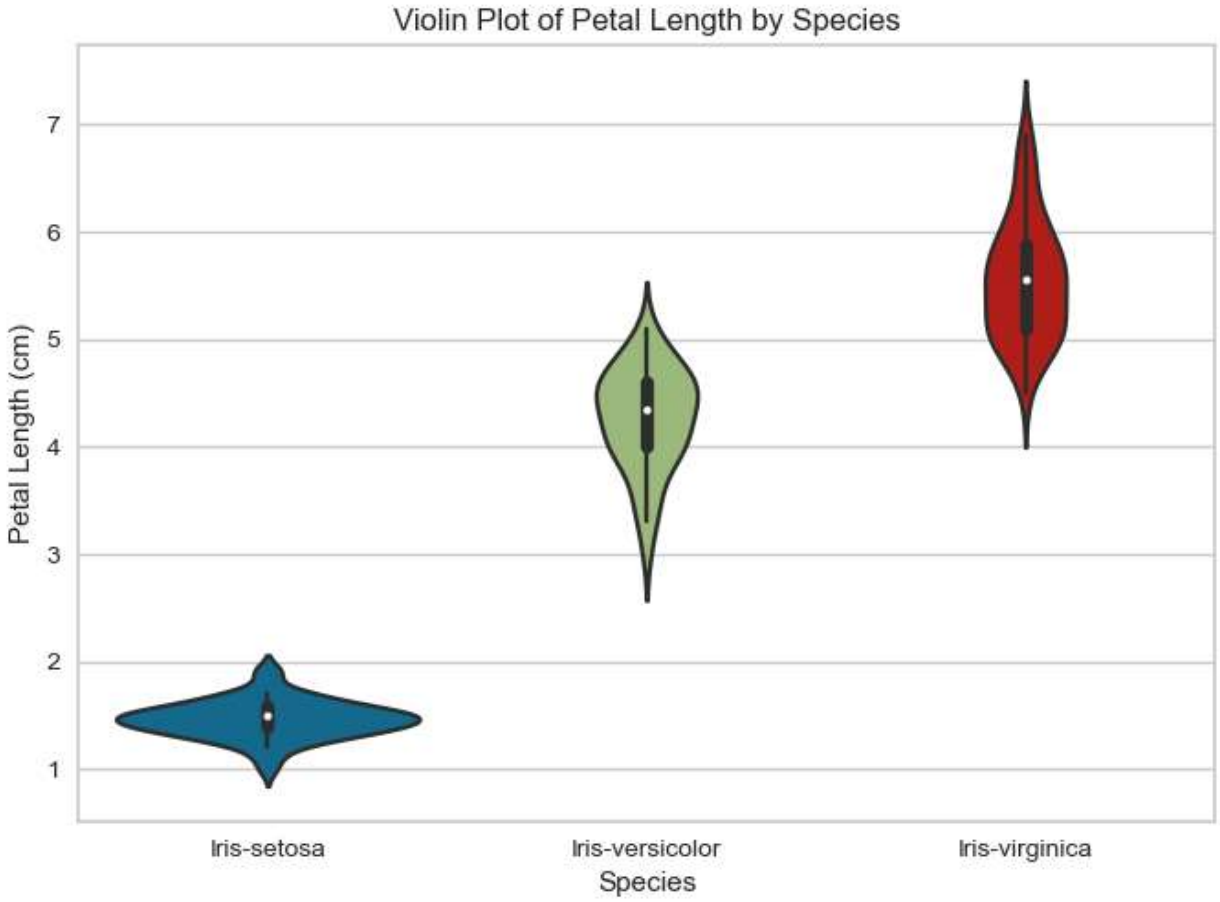
- **Virginica**: Virginica species has the widest petals among the three.

Summary:

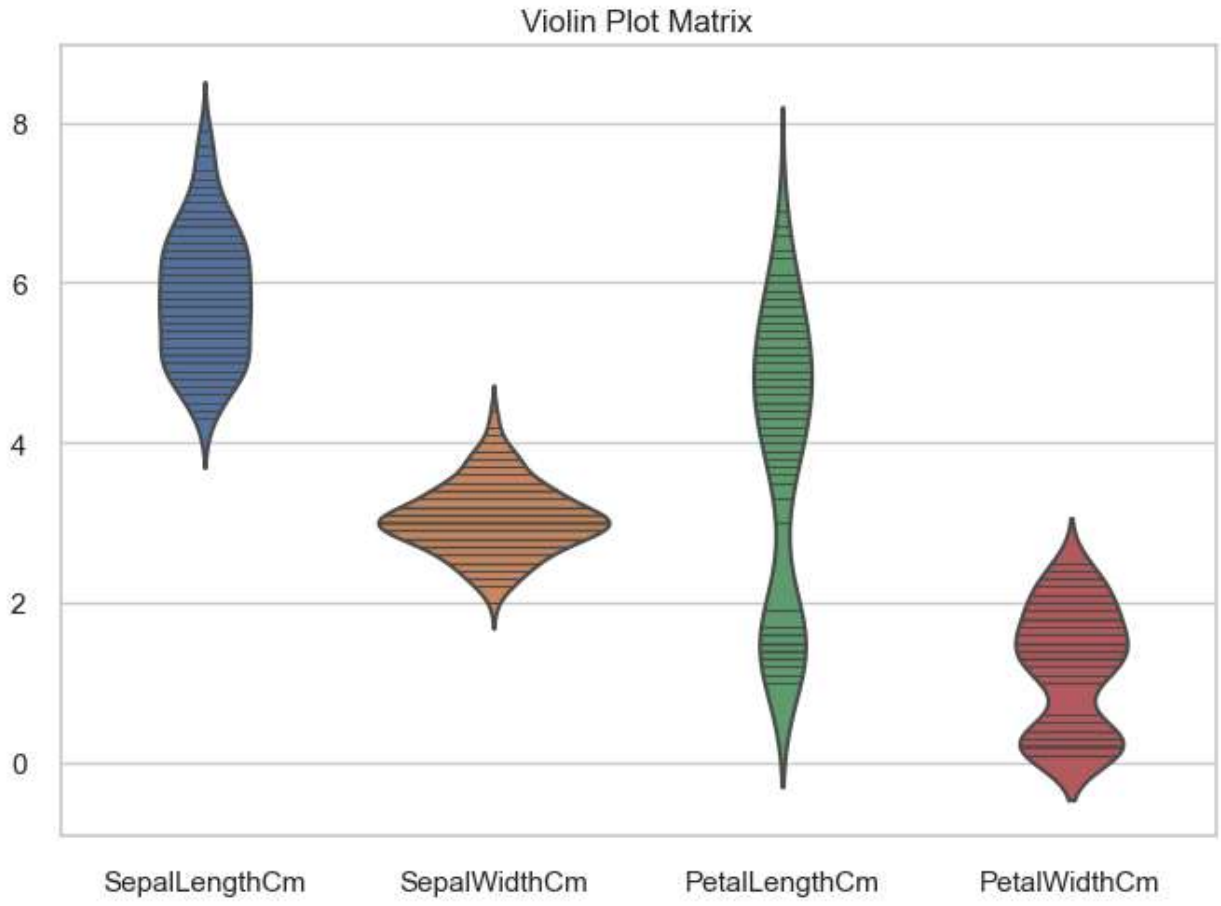
- **Sepal Characteristics**: Setosa is distinct in sepal width, while Virginica tends to have slightly longer sepals. Sepal length shows clearer distinctions between Setosa and the others.
- **Petal Characteristics**: Petal length and width are significant in distinguishing Setosa from Versicolor and Virginica, while Versicolor and Virginica differ mainly in petal length and width.

This analysis provides insights into the distinct characteristics of each Iris species based on sepal and petal measurements, highlighting how these features can be used to differentiate between them.

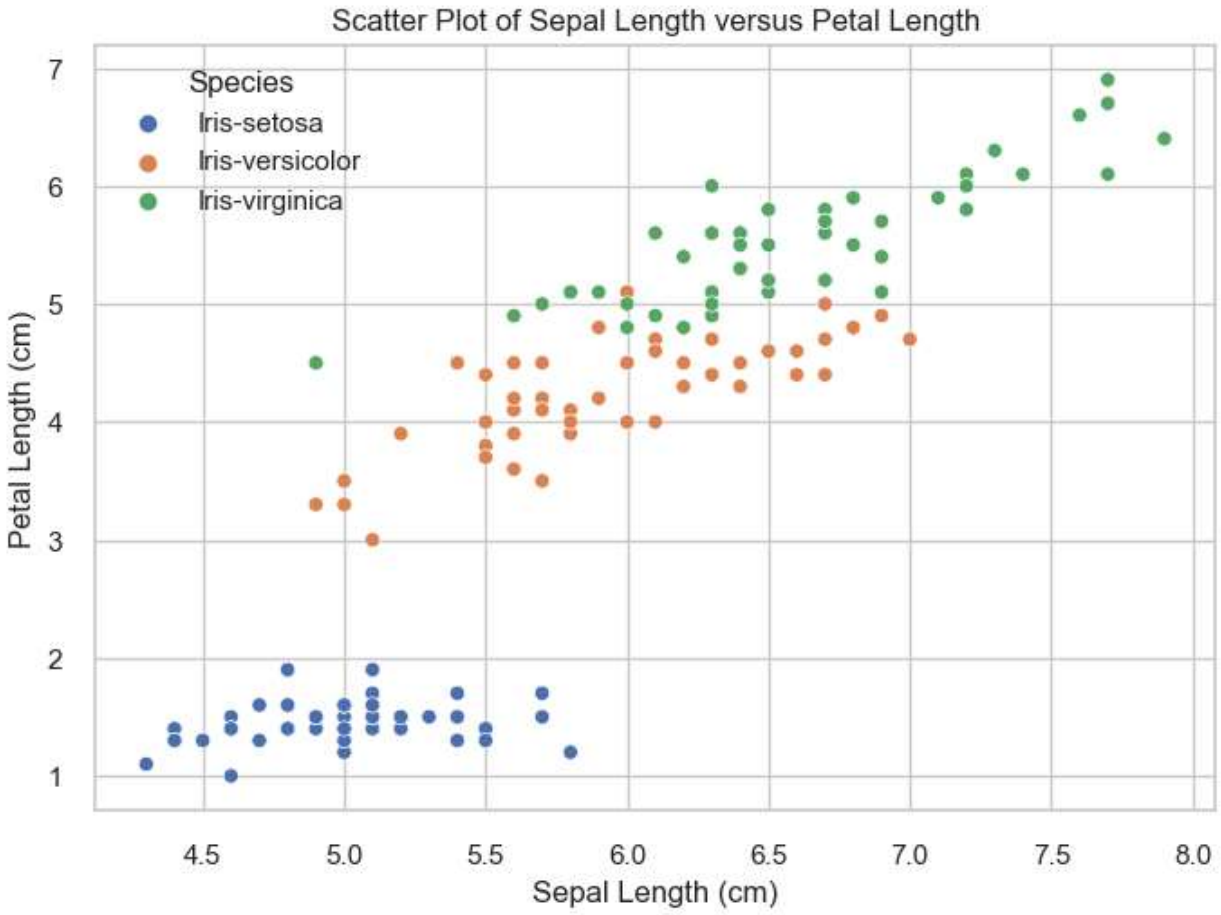
In [22]:
```
sns.violinplot(data=data, x='Species', y='PetalLengthCm')
plt.xlabel('Species')
plt.ylabel('Petal Length (cm)')
plt.title('Violin Plot of  Petal Length by Species')
plt.show()
```



In [23]:
```
sns.set(style="whitegrid")
sns.violinplot(data=data, inner='stick')
plt.title('Violin Plot Matrix')
plt.show()
```

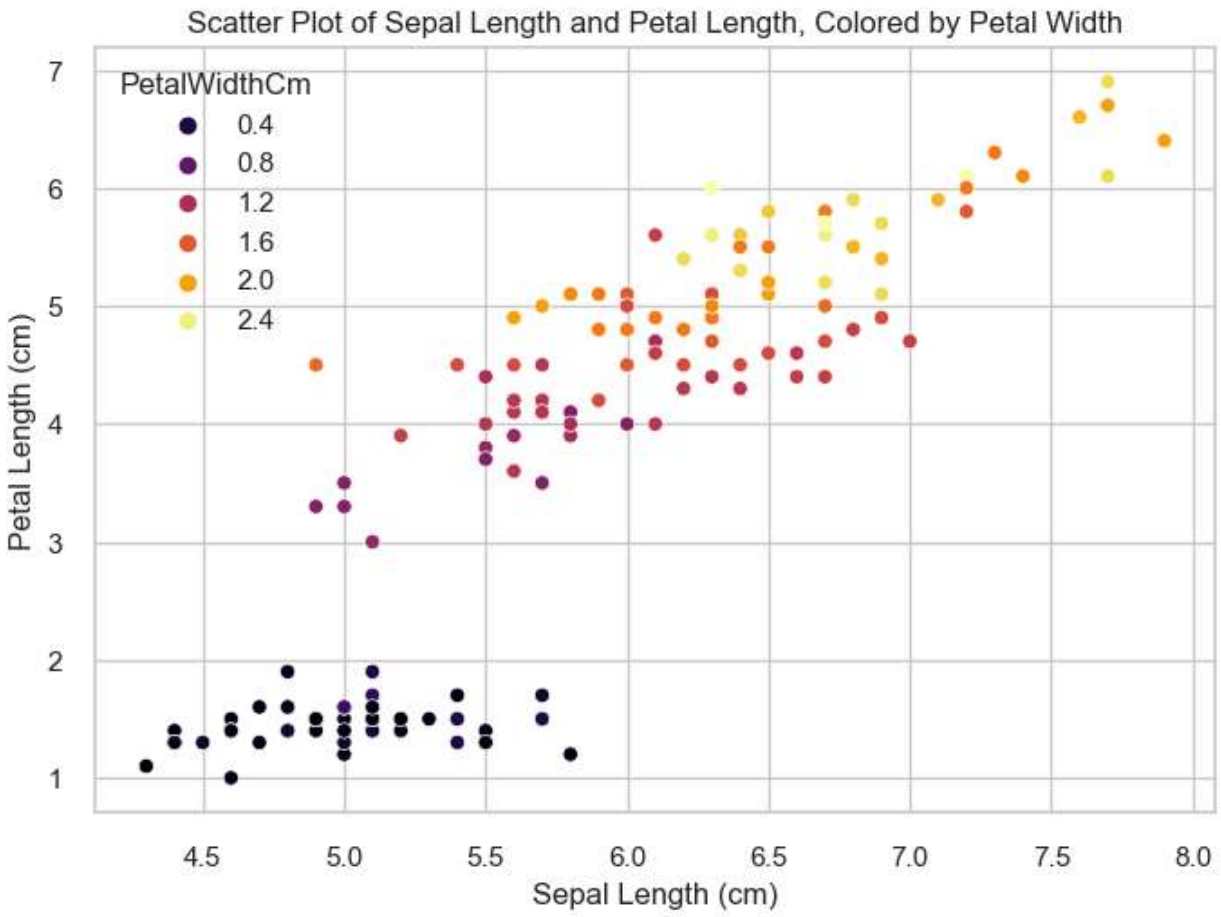


In [24]:
```
sns.scatterplot(data=data, x='SepalLengthCm', y='PetalLengthCm', hue='Species')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Petal Length (cm)')
plt.title('Scatter Plot of Sepal Length versus Petal Length')
plt.show()
```

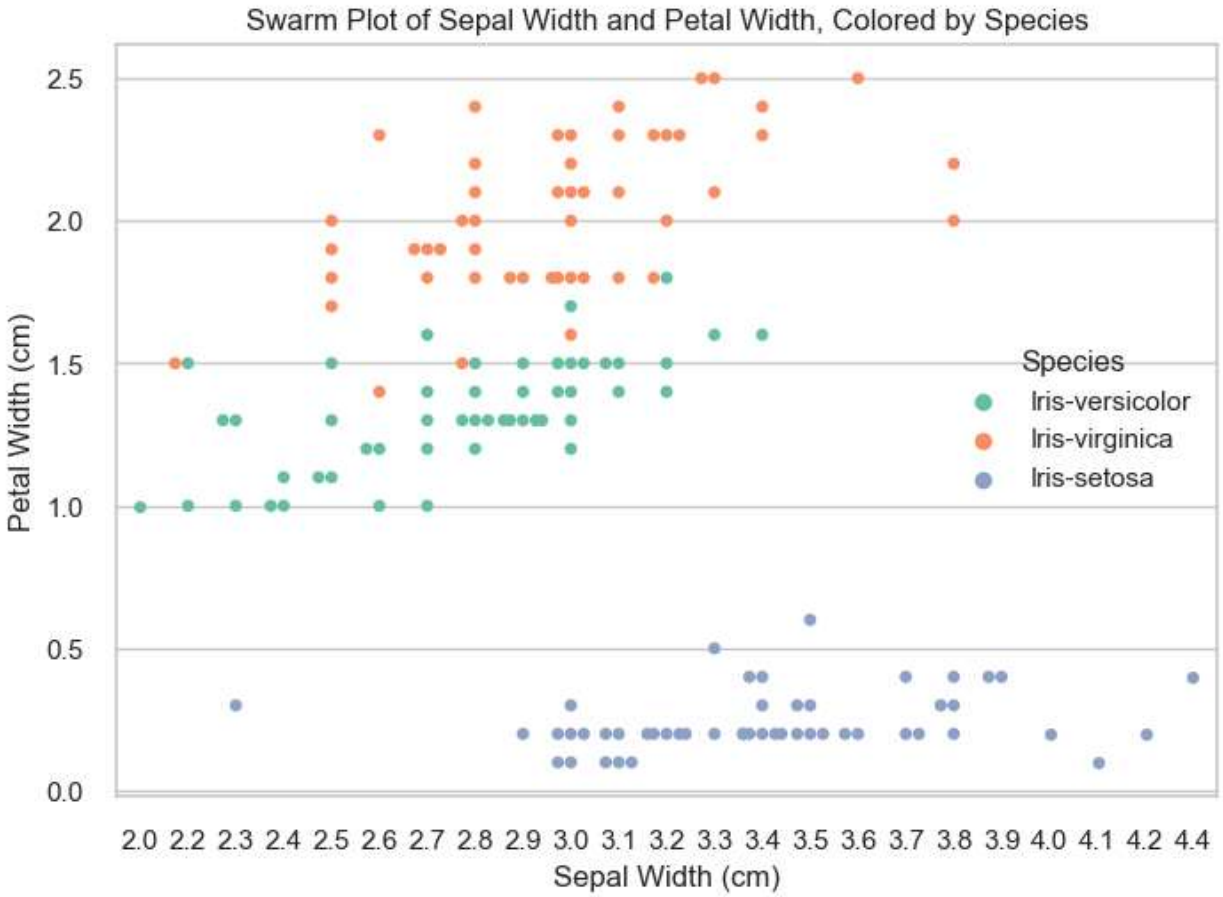


In [ ]:

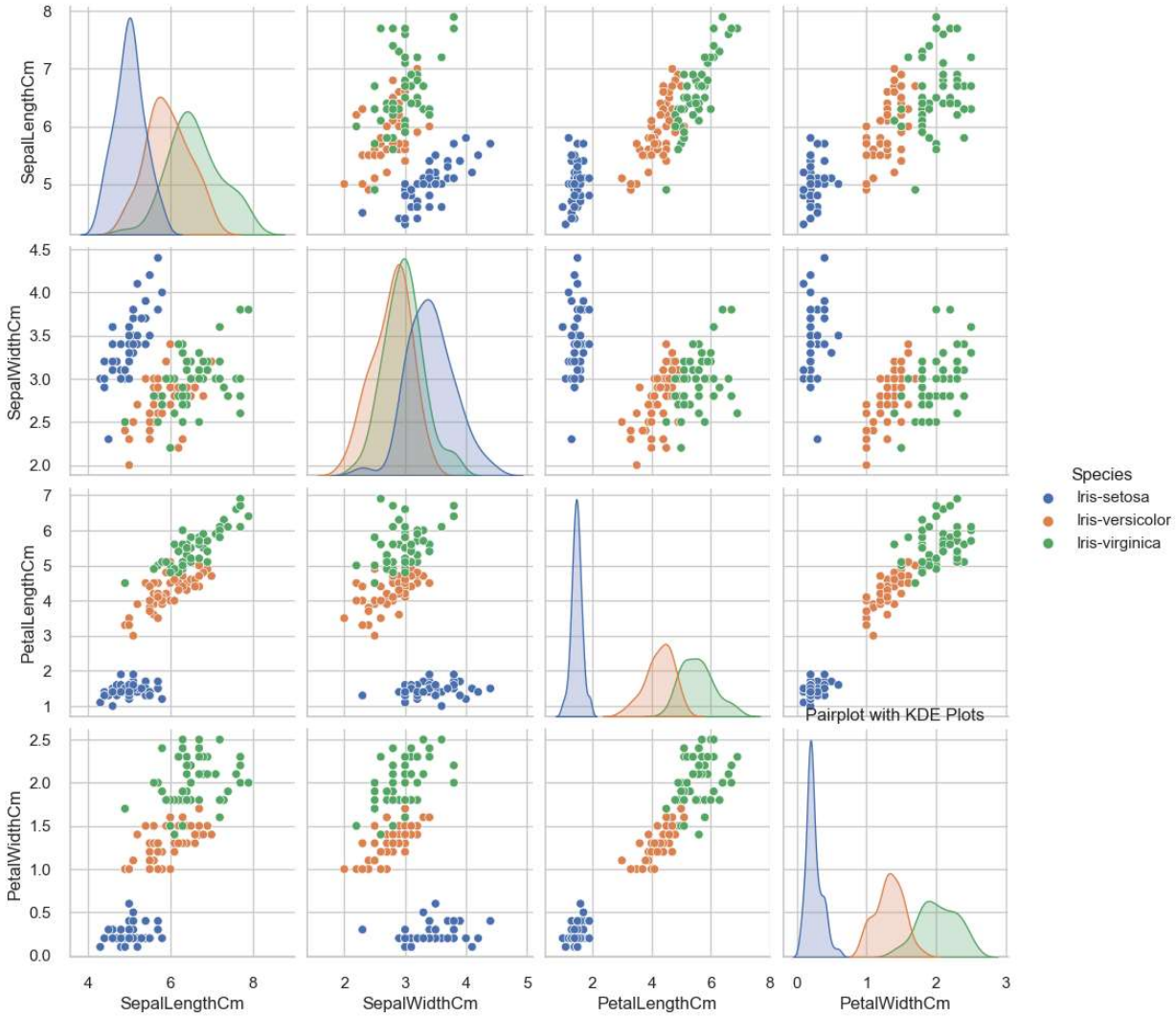
In [87]:
```
sns.scatterplot(data=data, x='SepalLengthCm', y='PetalLengthCm', hue='PetalWidthCm', palette='inferno')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Petal Length (cm)')
plt.title('Scatter Plot of Sepal Length and Petal Length, Colored by Petal Width')
plt.show()
```



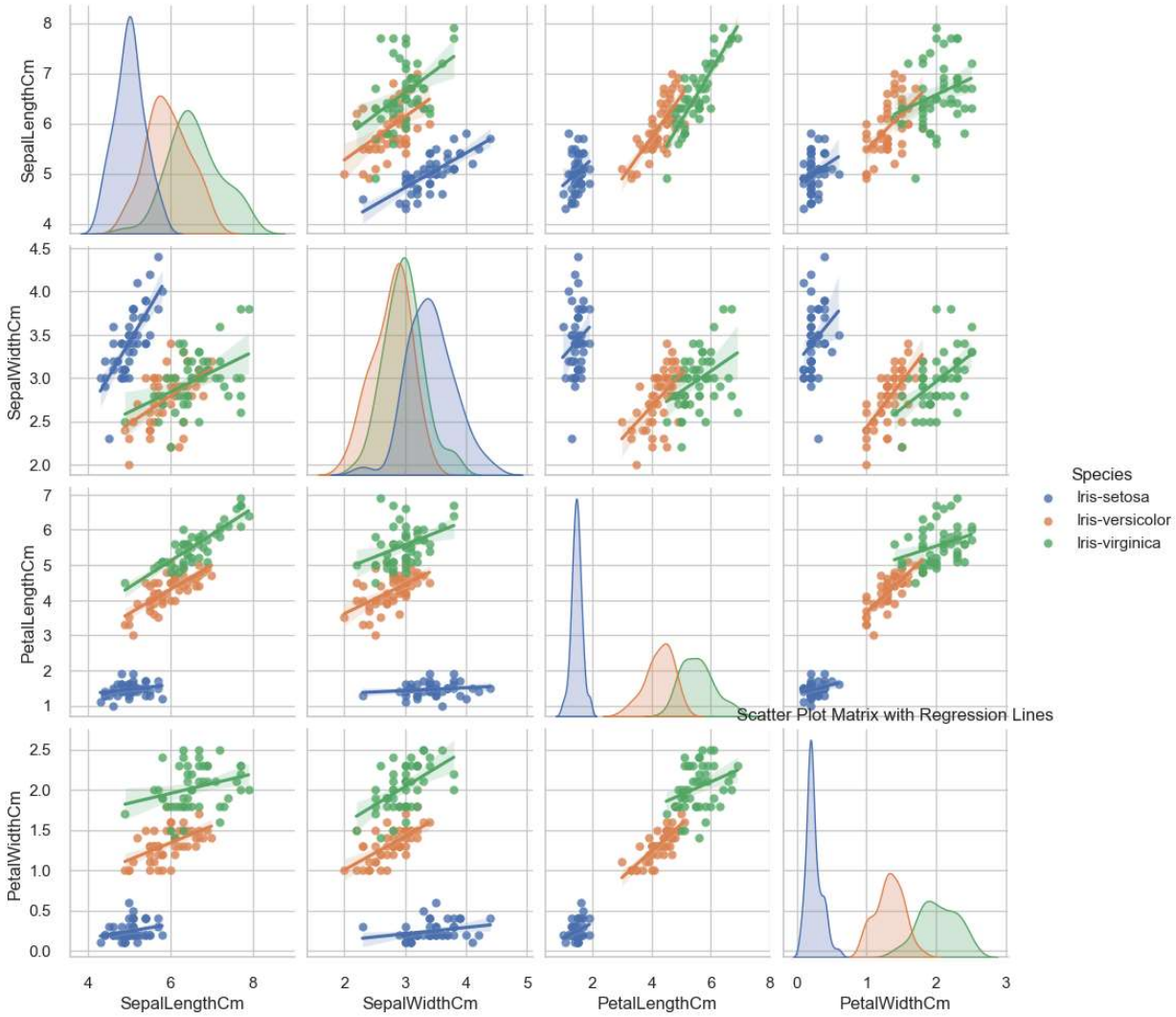
In [26]:
```
sns.swarmplot(data=data, x='SepalWidthCm', y='PetalWidthCm', hue='Species', palette='Set2')
plt.xlabel('Sepal Width (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('Swarm Plot of Sepal Width and Petal Width, Colored by Species')
plt.show()
```



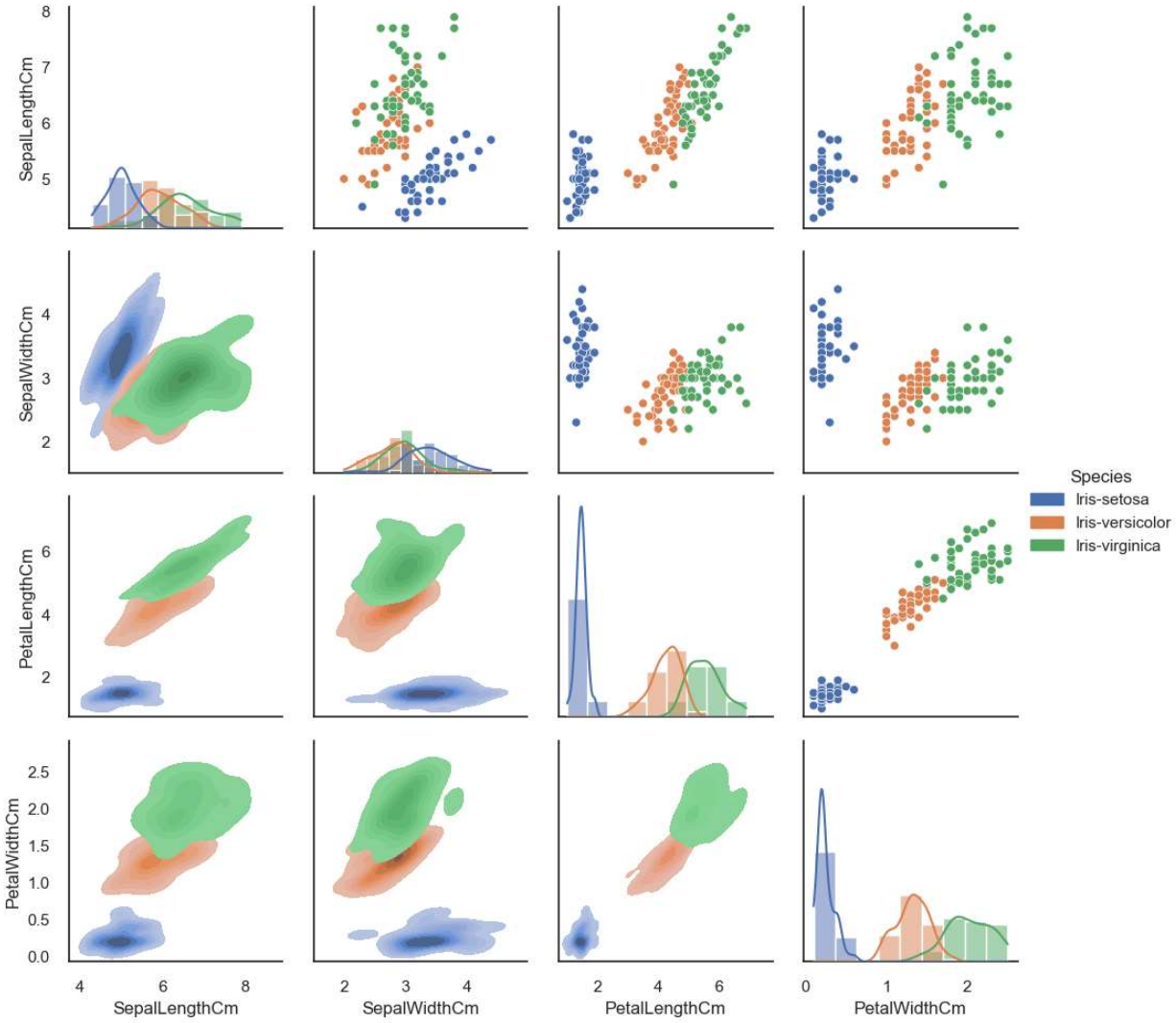
In [27]:
```
sns.pairplot(data=data, hue='Species', diag_kind='kde')
plt.title('Pairplot with KDE Plots')
plt.show()
```



In [28]:
```
sns.pairplot(data=data, kind='reg', hue='Species')
plt.title('Scatter Plot Matrix with Regression Lines')
plt.show()
```



In [29]:
```
sns.set(style="white")
g = sns.PairGrid(data, hue="Species")
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot, fill=True)
g.map_diag(sns.histplot, kde=True)
g.add_legend()
plt.show()
```



Analysis of Iris Flower Characteristics

Sepal Length vs. Sepal Width:

- **Setosa**: Setosa generally exhibits shorter and wider sepals compared to Versicolor and Virginica.
- **Versicolor and Virginica**: There is some overlap between these species, but Versicolor tends to have slightly narrower sepals compared to Virginica.

Petal Length vs. Petal Width:

- **Setosa**: Setosa species has shorter and narrower petals compared to Versicolor and Virginica.
- **Versicolor**: Versicolor has medium-length and medium-width petals.
- **Virginica**: Virginica species has the longest and widest petals among the three.

Sepal Length vs. Petal Length:

- **Setosa**: Setosa has shorter sepals and petals compared to Versicolor and Virginica.

- **Versicolor**: Versicolor has intermediate length petals compared to Setosa and Virginica.
- **Virginica**: Virginica has the longest petals among the three species.

Sepal Length vs. Petal Width:

- **Setosa**: Setosa has shorter sepals and narrower petals compared to Versicolor and Virginica.
- **Versicolor**: Versicolor shows medium petal width compared to Setosa and Virginica.
- **Virginica**: Virginica has the widest petals among the three species.

Sepal Width vs. Petal Length:

- **Setosa**: Setosa has wider sepals and shorter petals compared to Versicolor and Virginica.
- **Versicolor**: Versicolor shows medium petal length compared to Setosa and Virginica.
- **Virginica**: Virginica has longer petals compared to Setosa and Versicolor.

Sepal Width vs. Petal Width:

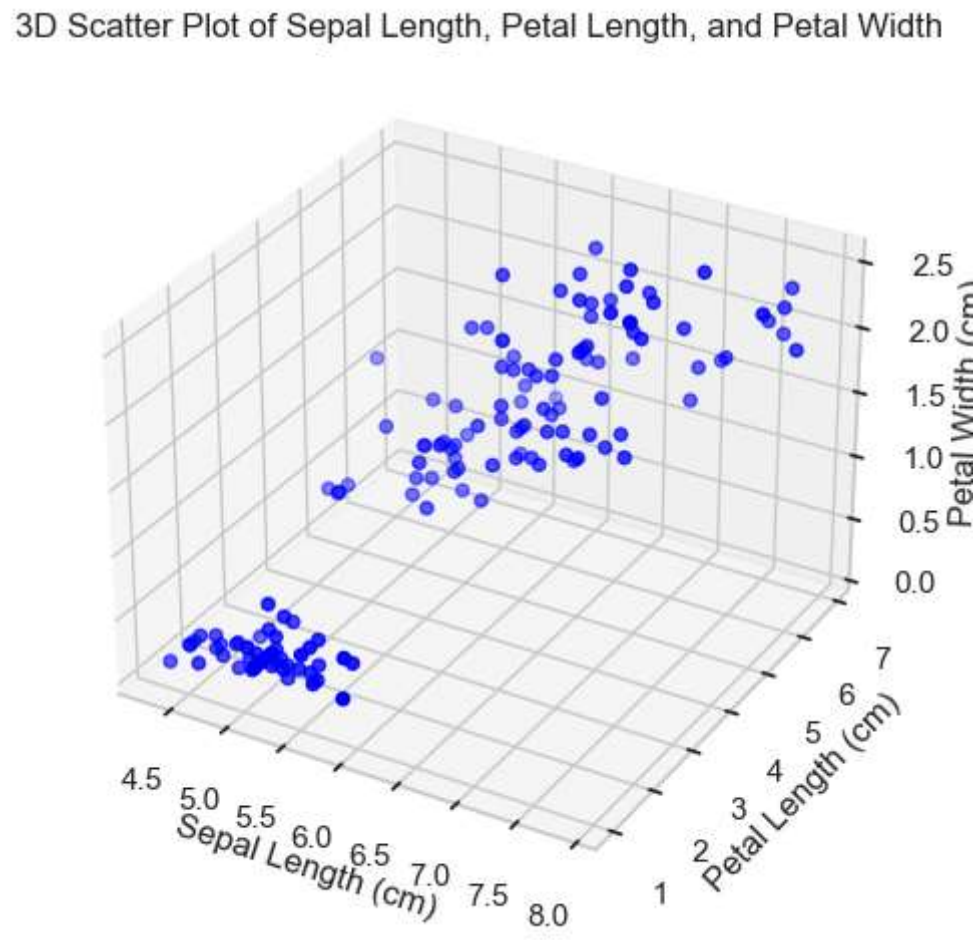
- **Setosa**: Setosa has wider sepals and narrower petals compared to Versicolor and Virginica.
- **Versicolor**: Versicolor shows medium petal width compared to Setosa and Virginica.
- **Virginica**: Virginica has wider petals compared to Setosa and Versicolor.

Summary:

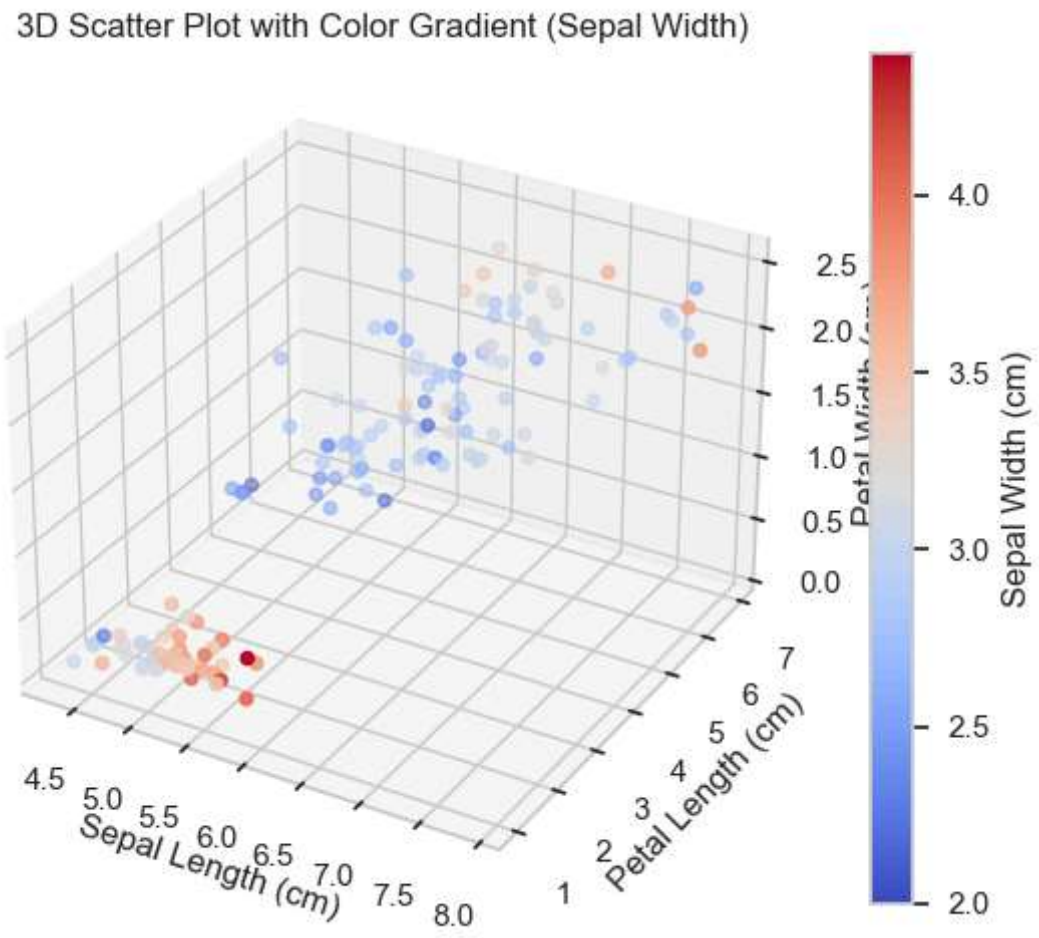
- **Sepal Characteristics**: Setosa stands out with shorter and wider sepals, while Versicolor and Virginica show more overlap but differ slightly in width.
- **Petal Characteristics**: Setosa has distinctively shorter and narrower petals, while Versicolor has medium-sized petals and Virginica has the longest and widest petals.
- **Relationships between Features**: The relationships between sepal length/width and petal length/width clearly differentiate the species, with Setosa consistently exhibiting shorter and wider features, Versicolor showing intermediate characteristics, and Virginica having longer and wider features.

This analysis provides a comprehensive understanding of how different measurements of sepals and petals can be used to distinguish between the Setosa, Versicolor, and Virginica species of Iris flowers.

In [85]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data['SepalLengthCm'], data['PetalLengthCm'], data['PetalWidthCm'], c='blue')
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Petal Length (cm)')
ax.set_zlabel('Petal Width (cm)')
ax.set_title('3D Scatter Plot of Sepal Length, Petal Length, and Petal Width')
plt.show()
```

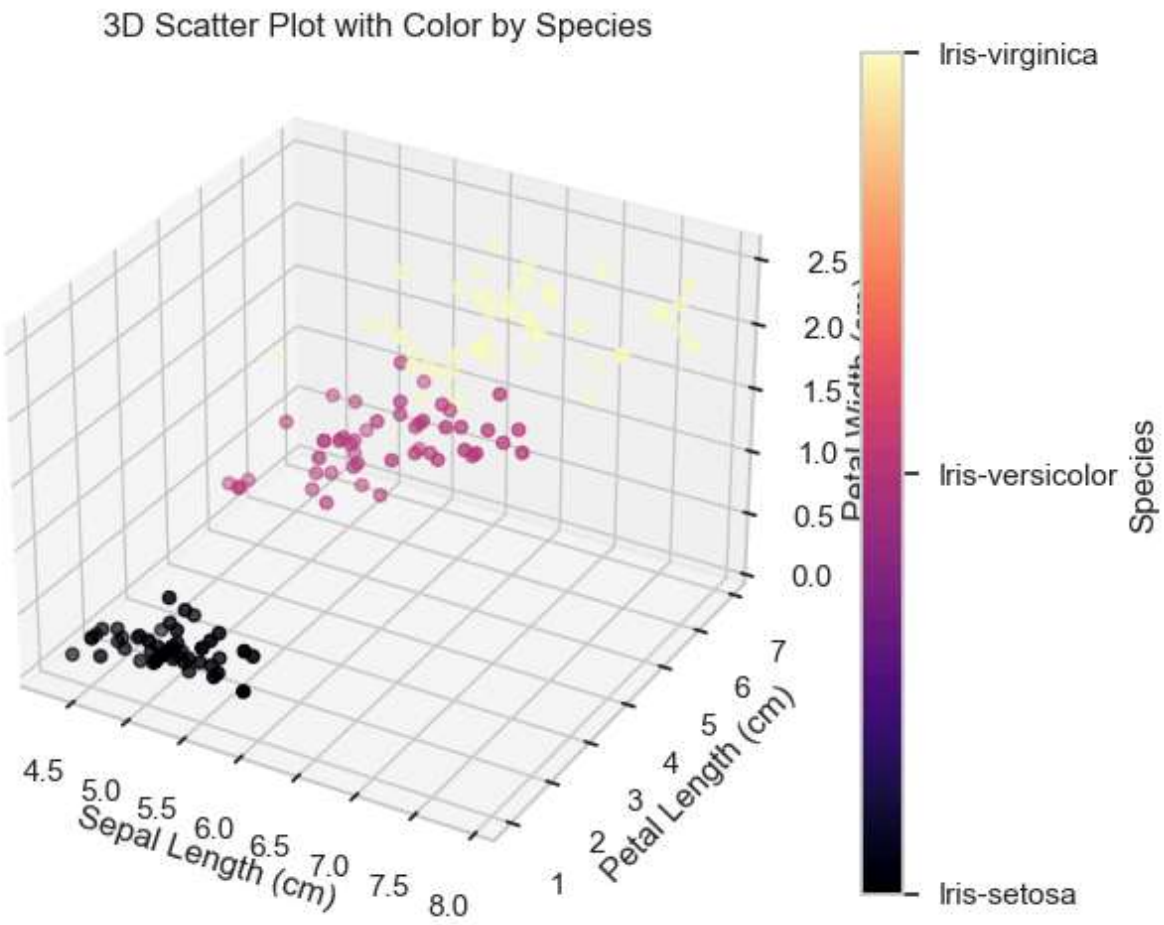


In [82]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(data['SepalLengthCm'], data['PetalLengthCm'], data['PetalWidthCm'], c=data['SepalWidthCm'], cmap='coo
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Petal Length (cm)')
ax.set_zlabel('Petal Width (cm)')
ax.set_title('3D Scatter Plot with Color Gradient (Sepal Width)')
cbar = plt.colorbar(sc)
cbar.set_label('Sepal Width (cm)')
plt.show()
```



In [76]:
```
species_mapping = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
data['SpeciesNumeric'] = data['Species'].map(species_mapping)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(data['SepalLengthCm'], data['PetalLengthCm'], data['PetalWidthCm'], c=data['SpeciesNumeric'], cmap='ma
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Petal Length (cm)')
ax.set_zlabel('Petal Width (cm)')
ax.set_title('3D Scatter Plot with Color by Species')
cbar = plt.colorbar(sc)
cbar.set_ticks([0, 1, 2])
cbar.set_ticklabels(species_mapping.keys())
cbar.set_label('Species')
plt.show()
```



- The **Iris-setosa** species (blue points) is clearly separated from the other two species. This indicates that setosa has distinct feature measurements compared to versicolor and virginica, which supports our previous observations from the univariate, bivariate, and multivariate analyses.
- The **Iris-versicolor** and **Iris-virginica** species (purple and off-white points, respectively) exhibit some overlap but also show noticeable separation. This suggests that while these two species share similarities in their feature measurements, there are still discernible differences that allow for their distinction.

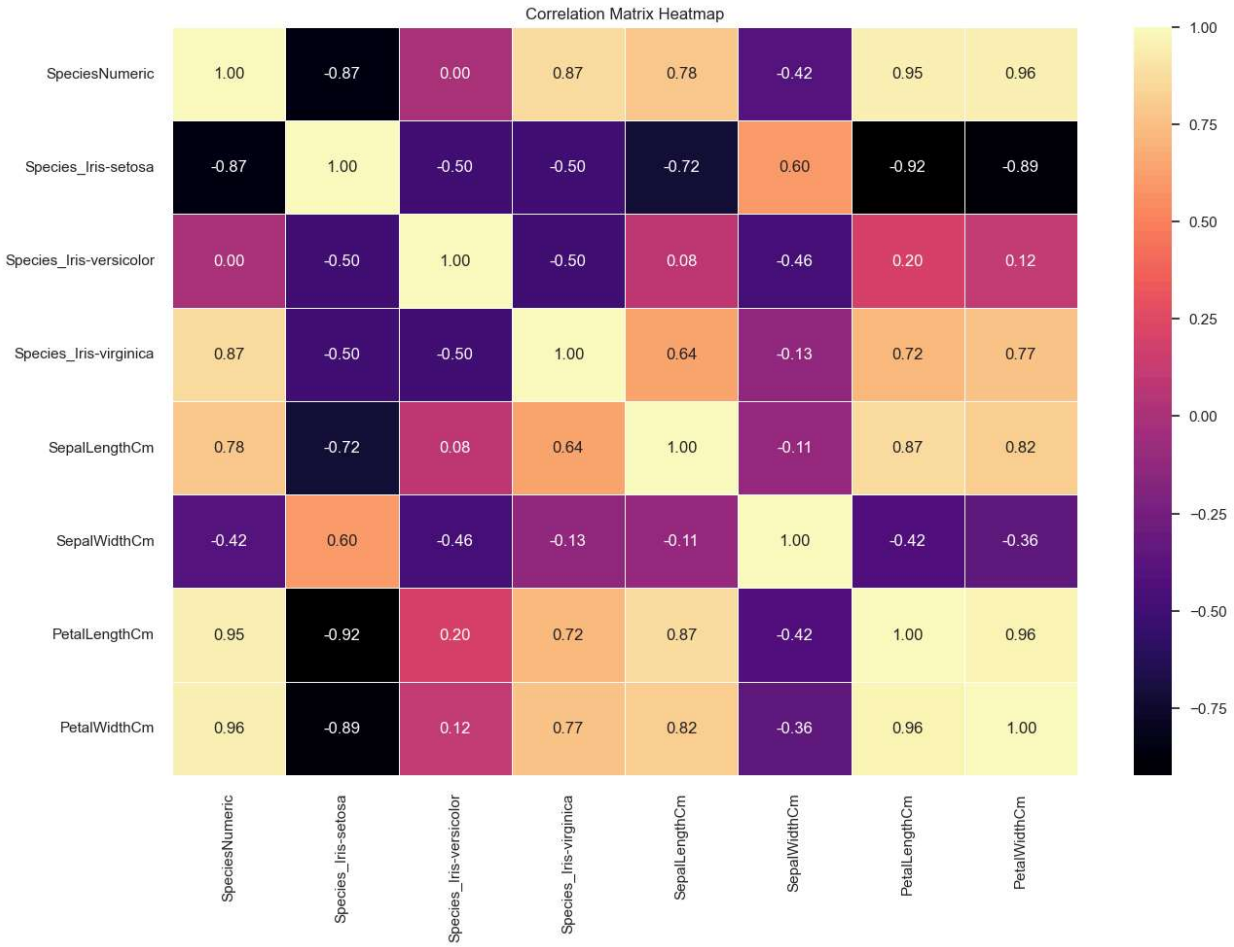
In [34]:
```
categorical_columns = ['Species']
numerical_columns = ['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
dummy_variables = pd.get_dummies(data, columns=categorical_columns, drop_first=False)
```

In [86]:
```
scaler = StandardScaler()
scaled_numerical = scaler.fit_transform(data[numerical_columns])
scaled_numerical_df = pd.DataFrame(scaled_numerical, columns=numerical_columns)
```

In [36]:
```
dummy_variables = dummy_variables.drop(numerical_columns, axis=1)
processed_df = pd.concat([dummy_variables, scaled_numerical_df], axis=1)
```

In [40]:
```
correlation_matrix = processed_df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='magma', linewidths=0.5, fmt='.2f')
```

```
plt.title("Correlation Matrix Heatmap")
plt.show()
```



Correlation Analysis of Iris Dataset Features

Species_Iris-setosa:

- **PetalLengthCm and PetalWidthCm**: Strongly negatively correlated with values of -0.92 and -0.88 respectively. This indicates that as petal length and width increase, it becomes less likely for the sample to belong to the setosa species. 📉

Species_Iris-versicolor:

- **PetalLengthCm and PetalWidthCm**: Weak positive correlations with values of 0.20 and 0.12 respectively. This suggests that petal dimensions do not significantly contribute to identifying the versicolor species. ➖

Species_Iris-virginica:

- **PetalLengthCm and PetalWidthCm**: Fairly strong positive correlations with values of 0.72 and 0.77 respectively. As petal length and width increase, it becomes more likely for the sample to belong to the virginica species. 📈

SepalLengthCm:

- **PetalLengthCm and PetalWidthCm**: Strong positive correlations with values of 0.87 and 0.81 respectively. This implies that as sepal length increases, petal length and width also tend to increase. 📈

SepalWidthCm:

- **PetalLengthCm and PetalWidthCm**: Negatively correlated with values of -0.42 and -0.35 respectively. As sepal width increases, there is a tendency for petal length and width to decrease, although this correlation is not very strong. 📉

PetalLengthCm and PetalWidthCm:

- **Correlation Coefficient**: Very strong correlation of 0.96. This indicates that petal length and width tend to increase together. 📈

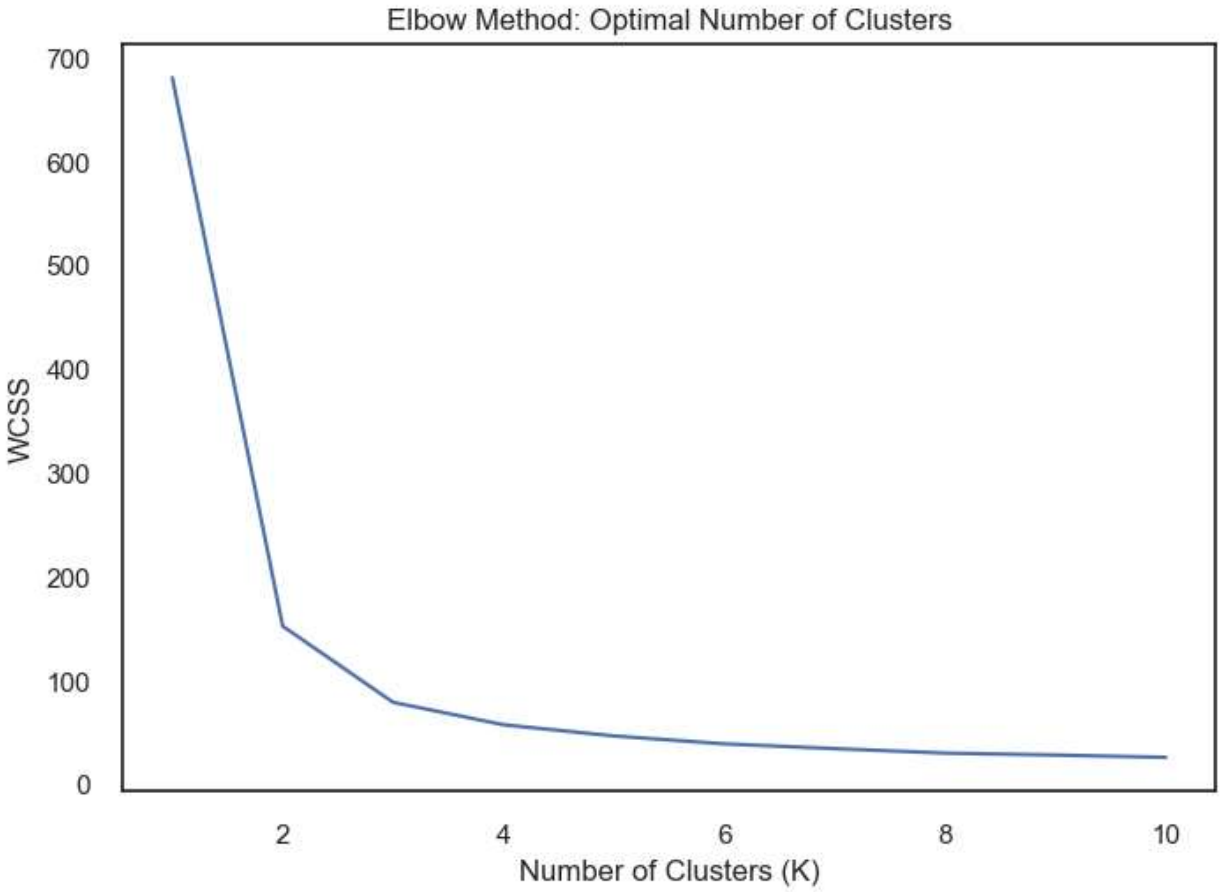
Note:

- **Causation Disclaimer**: It's important to note that correlation does not imply causation. This analysis shows the relationships between variables but does not indicate that one variable causes changes in another. For instance, the strong correlation between petal length and width does not necessarily mean that increasing petal length causes the width to increase; there could be other factors influencing both variables simultaneously.

In [41]:
```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]

wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.title('Elbow Method: Optimal Number of Clusters')
plt.show()
```



Analysis of Optimal Number of Clusters using WCSS

WCSS (Within Cluster Sum of Squares) Analysis:

The WCSS metric tends to decrease as the number of clusters increases in a K-means clustering algorithm. This behavior is expected because with more clusters, each data point tends to be closer to the centroid of its assigned cluster, resulting in a reduction of the sum of squares.

Rate of Decrease of WCSS:

Initially, as the number of clusters increases, the WCSS decreases steeply. This rapid decrease indicates that clustering is effectively reducing the variance within clusters and improving cluster homogeneity.

Identification of the "Elbow" Point:

Beyond a certain number of clusters, typically observed around 2 clusters, the rate of decrease in WCSS begins to diminish noticeably. This point, often referred to as the "elbow" in the WCSS plot, signifies where the rate of improvement in clustering effectiveness slows down significantly.

Optimal Number of Clusters:

The "elbow" point in the WCSS plot is commonly considered as the optimal number of clusters. It represents the balance between maximizing the reduction in WCSS and minimizing the complexity of the model. Choosing this point helps in achieving a meaningful clustering result without overfitting to noise or unnecessary complexity.

Conclusion:

Understanding the behavior of WCSS and identifying the "elbow" point provides valuable insights into determining the optimal number of clusters in K-means clustering. This approach ensures that the clustering solution is both interpretable and effective in capturing the underlying structure of the data.

Considerations

Elbow Method and Optimal Clusters

The elbow plot analysis suggests that the optimal number of clusters for the Iris dataset is around 2 clusters. However, it's important to note that the actual dataset contains 3 distinct species of Iris flowers.

Discrepancy Explanation

The discrepancy between the elbow method's suggestion and the actual number of species (3) could be attributed to the observed overlap between the Versicolor and Virginica species in previous analyses. This overlap diminishes the distinct clustering patterns that the elbow method relies on to determine the optimal number of clusters.

Limitations of the Elbow Method

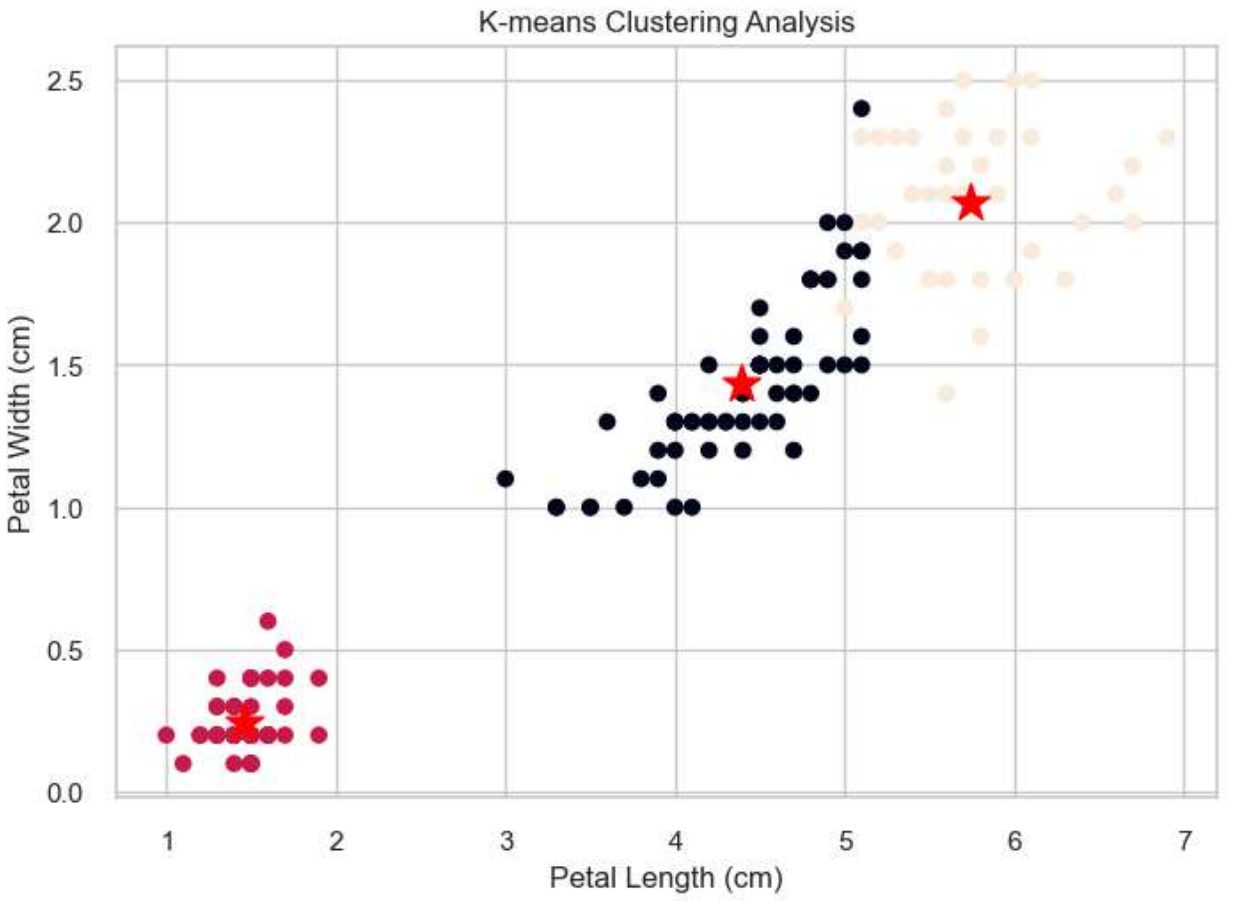
The elbow method serves as a heuristic for identifying the optimal number of clusters by analyzing the rate of decrease in within-cluster sum of squares (WCSS). However, its effectiveness can be limited in cases where there's significant overlap or ambiguity between clusters, as seen in the Iris dataset.

Conclusion

While the elbow method provides valuable guidance, it's important to interpret its results cautiously, especially in complex datasets with overlapping clusters. Understanding the underlying data structure and considering domain knowledge are crucial for determining the appropriate number of clusters that best represent the dataset's characteristics.

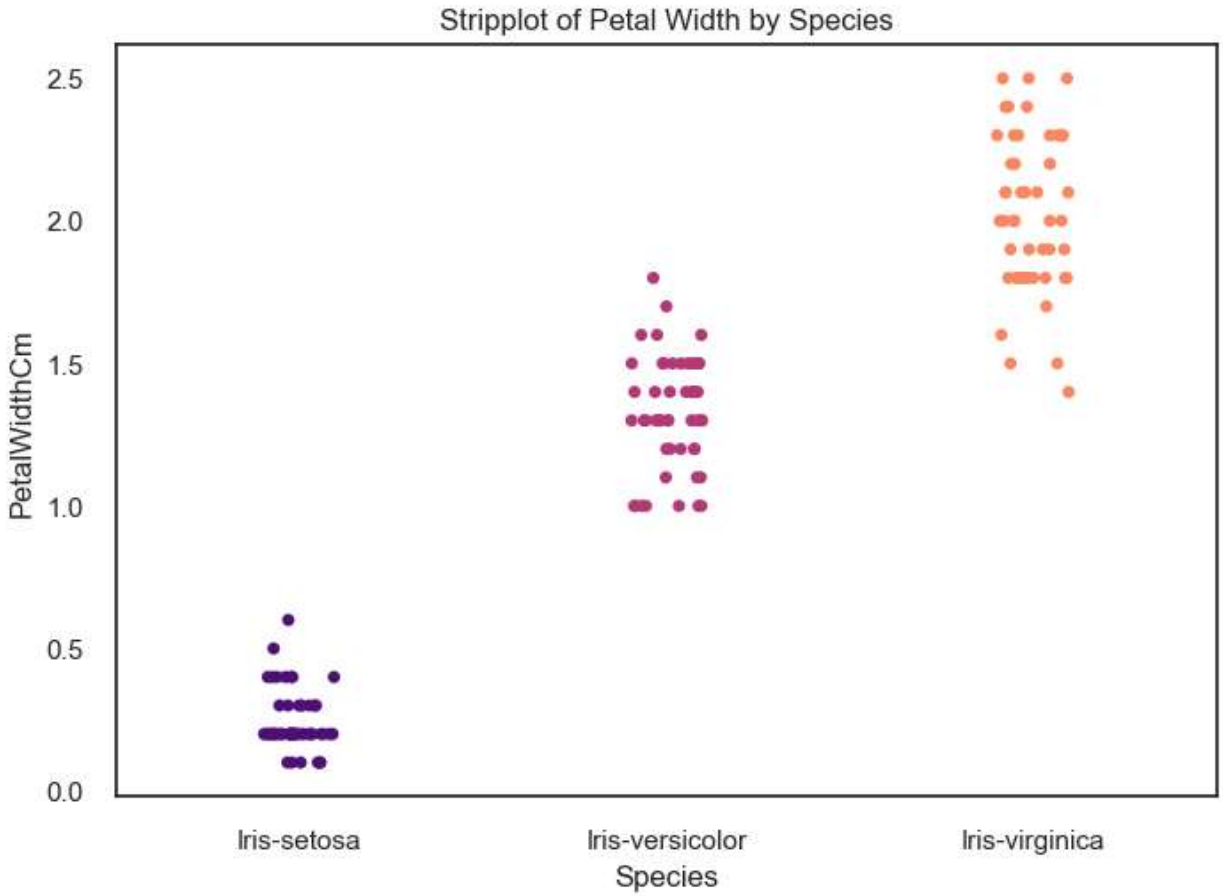
In [89]:
```
k = 3
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

```
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'], c=labels)
plt.scatter(centroids[:, 2], centroids[:, 3], marker='*', color='red', s=250)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('K-means Clustering Analysis')
plt.show()
```

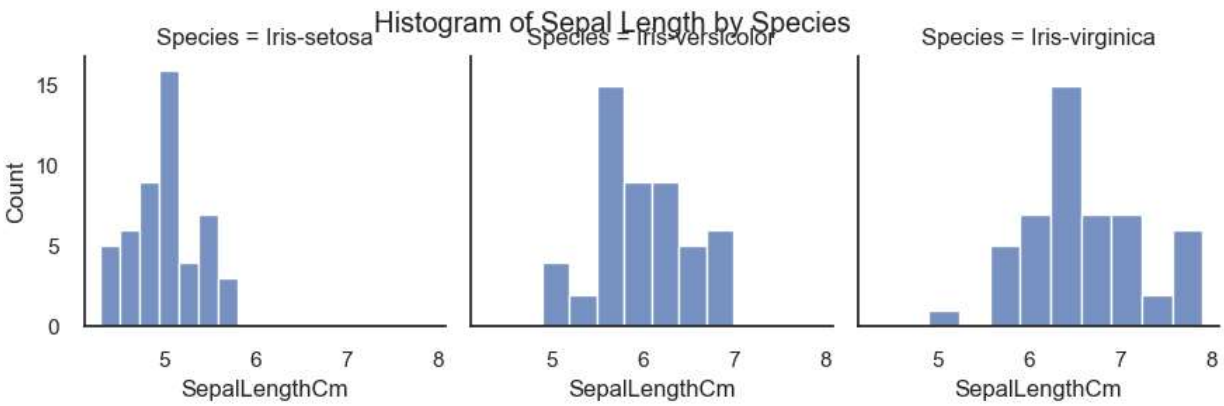


In [54]:

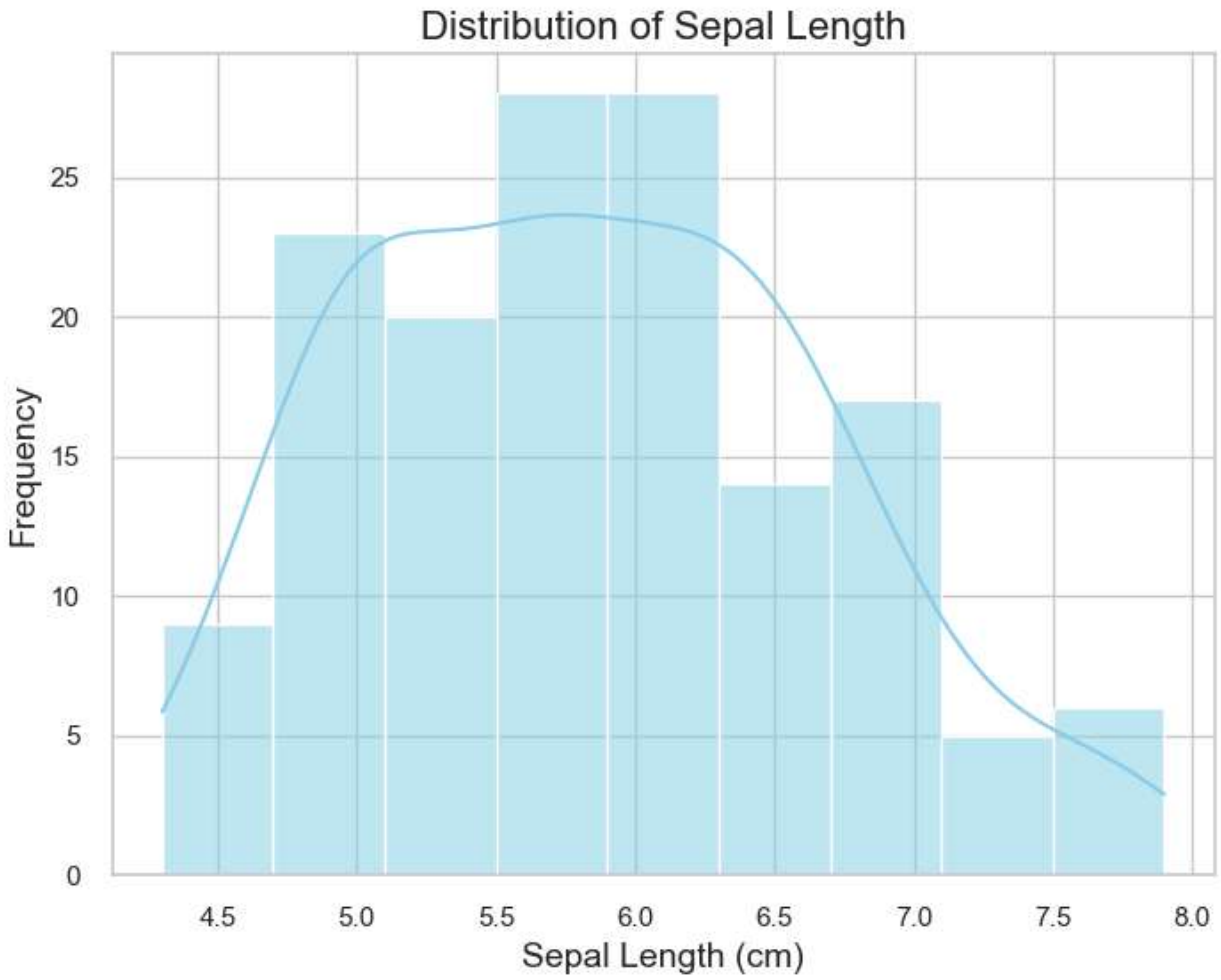
```
sns.stripplot(x='Species', y='PetalWidthCm', data=data, palette='magma')
plt.title('Stripplot of  Petal Width by Species')
plt.show()
```



In [61]:
```
g = sns.FacetGrid(df, col='Species',palette='crest')
g.map(sns.histplot, 'SepalLengthCm')
plt.suptitle('Histogram of Sepal Length by Species')
plt.show()
```

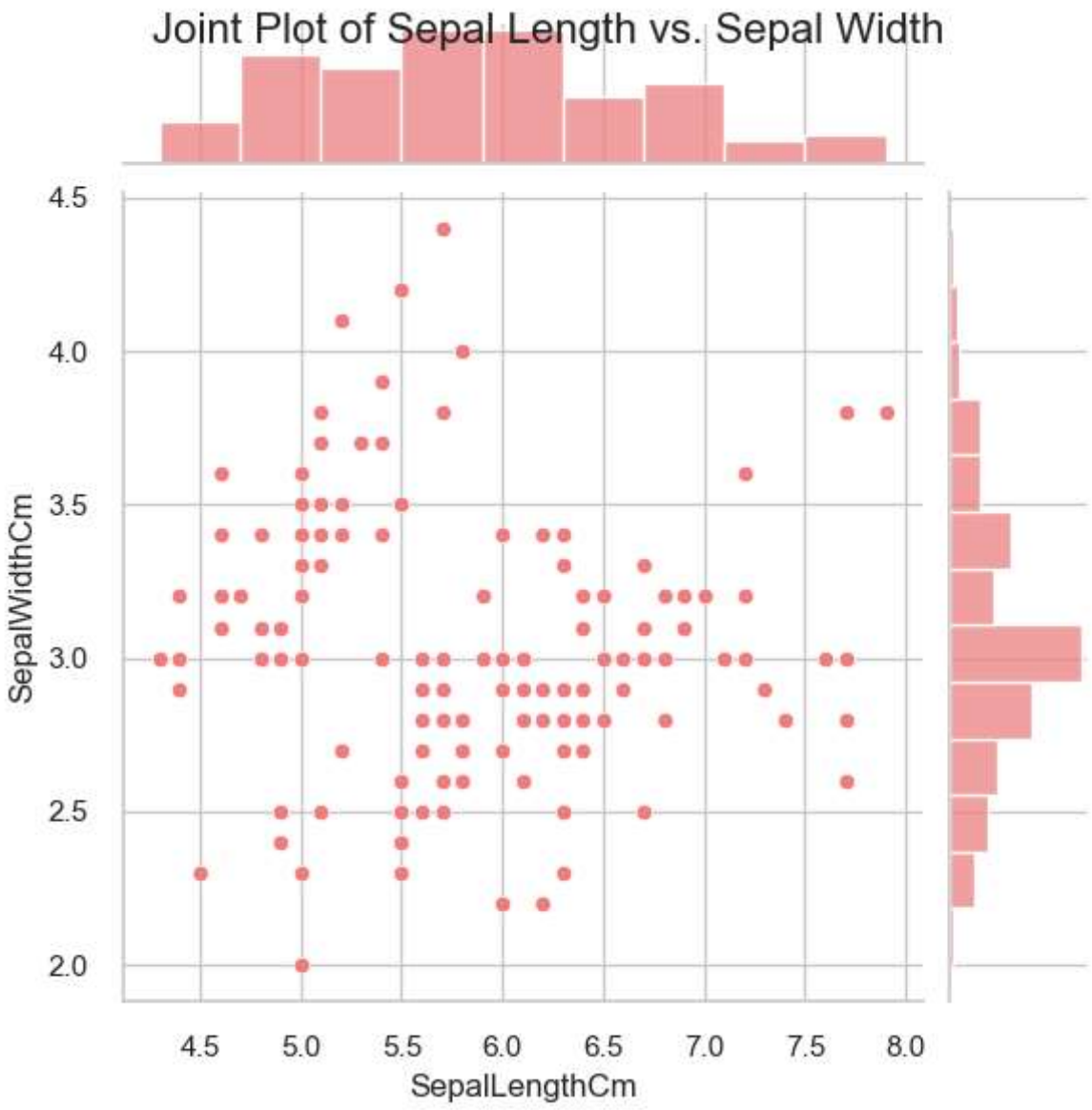


In [71]:
```
sns.set_style("whitegrid")
plt.figure(figsize=(8, 6))
sns.histplot(df['SepalLengthCm'], kde=True, color='skyblue')
plt.title('Distribution of Sepal Length', fontsize=16)
plt.xlabel('Sepal Length (cm)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```

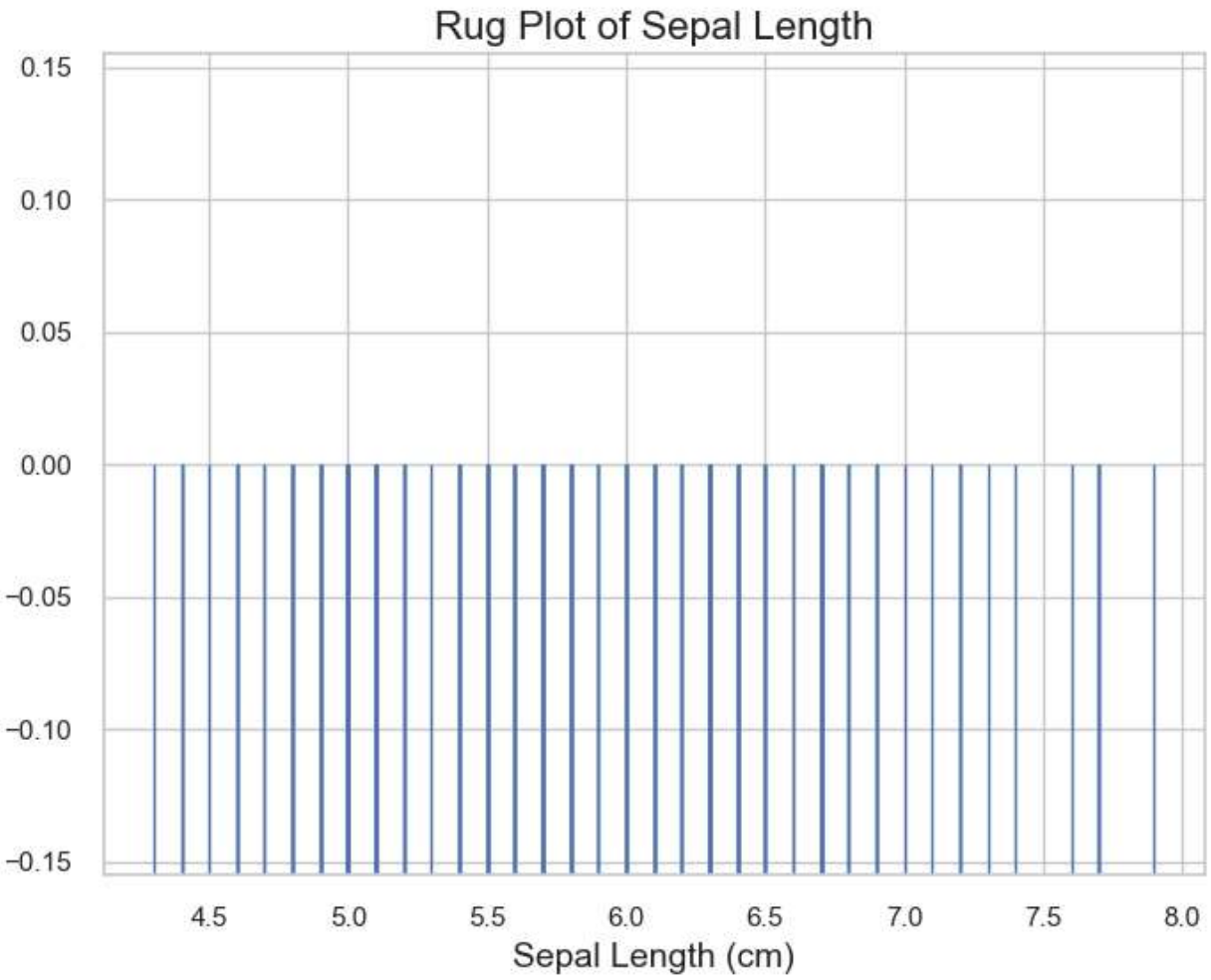


In [72]:
```
plt.figure(figsize=(8, 6))
sns.jointplot(x='SepalLengthCm', y='SepalWidthCm', data=df, kind='scatter', color='lightcoral')
plt.suptitle('Joint Plot of Sepal Length vs. Sepal Width', fontsize=16)
plt.show()
```

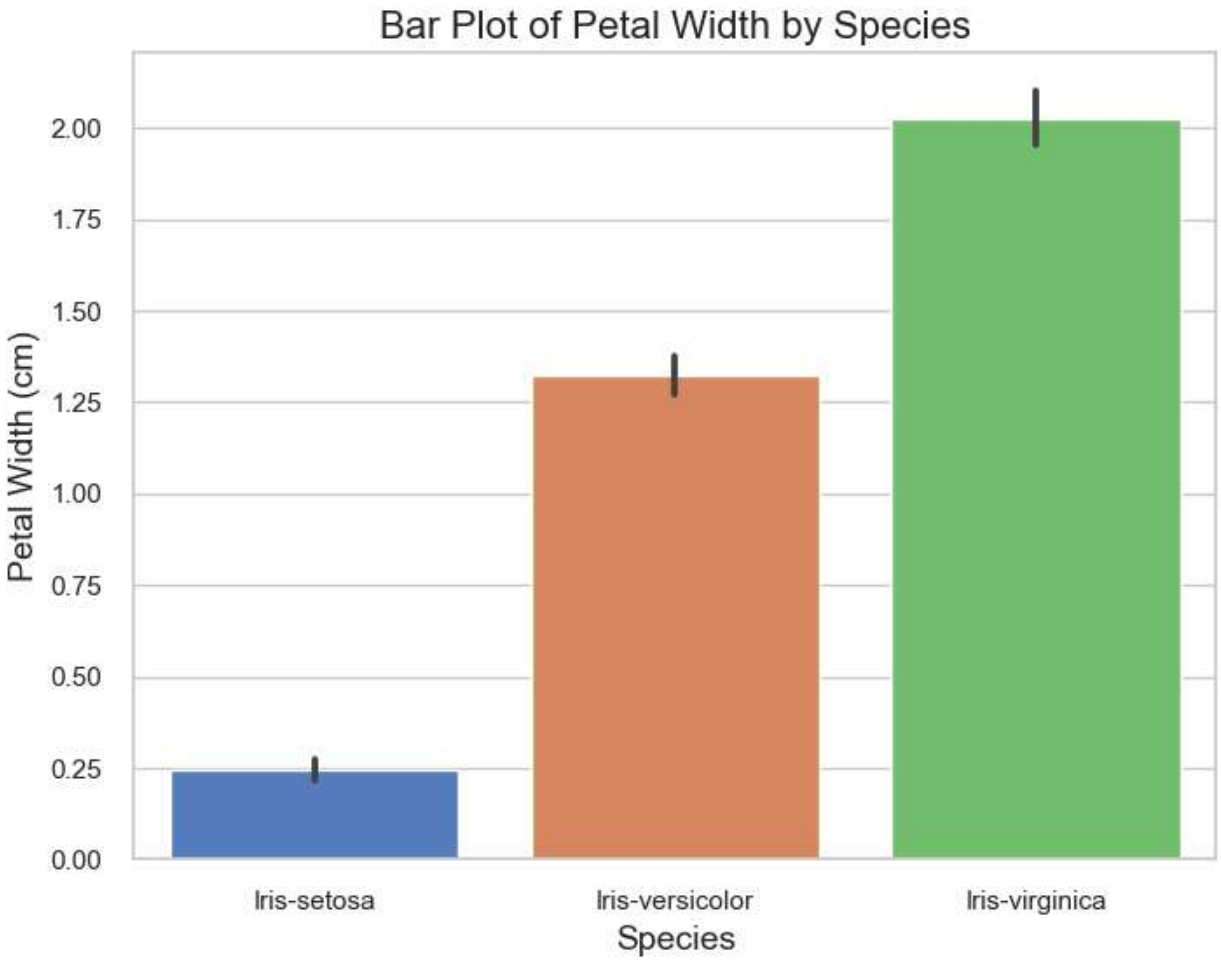
<Figure size 800x600 with 0 Axes>



In [73]:
```
plt.figure(figsize=(8, 6))
sns.rugplot(x='SepalLengthCm', data=df, height=0.5, palette='pastel')
plt.title('Rug Plot of Sepal Length', fontsize=16)
plt.xlabel('Sepal Length (cm)', fontsize=14)
plt.show()
```



In [74]:
```
plt.figure(figsize=(8, 6))
sns.barplot(x='Species', y='PetalWidthCm', data=df, palette='muted')
plt.title('Bar Plot of  Petal Width by Species', fontsize=16)
plt.xlabel('Species', fontsize=14)
plt.ylabel('Petal Width (cm)', fontsize=14)
plt.show()
```



In [75]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(x='Species', y='PetalLengthCm', data=df, palette='pastel')
plt.title('Box Plot of Petal Length by Species', fontsize=16)
plt.xlabel('Species', fontsize=14)
plt.ylabel('Petal Length (cm)', fontsize=14)
plt.show()
```


Box Plot of Petal Length by Species
Petal Length (cm)
7
6
5
4
3
2
1
Iris-setosa
Iris-versicolor
Iris-virginica
Species

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: