Iris Flower Classification

The Iris flower dataset consists of three species: Setosa, Versicolor, and Virginica. These species can be distinguished based on their measurements.

Iris dataset to develop a model that can classify iris flowers into different species based on their sepal and petal measurements.

Let's Import the required Libraries

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Overview of the DataFrame

In [2]:
```
df = pd.read_csv("Iris.csv")
```

In [3]:
```
df
```

Out[3]:

	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

149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [4]:
```
df.describe()
```

Out[4]:

	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 [5]:
```
df.columns
```

Out[5]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [6]:
```
df.head()
```

Out[6]:

	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 [7]:
```
df.tail()
```

Out[7]:

In [7]:
```
df.tail()
```

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
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

Let's Check the Null Values in the Dataset

In [8]:
```
df.isna().sum()
```

Out[8]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

No Null Values Present in the Dataset

In [9]:
```
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
```

jupyter Iris Flower Classification Last Checkpoint: 6 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [10]:
```
df.shape
```

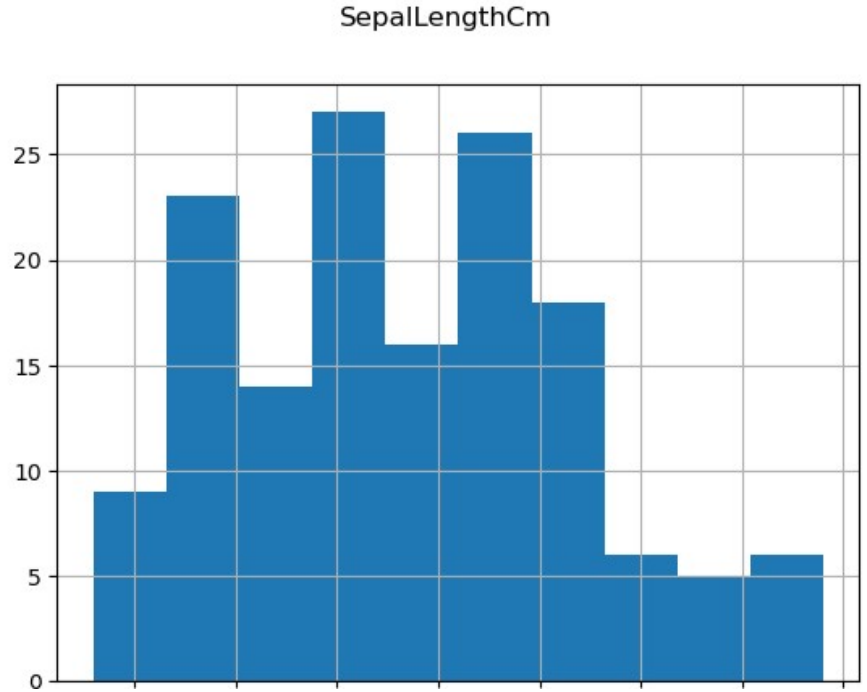
Out[10]: (150, 6)

In [11]:
```
df.sample(5)
```

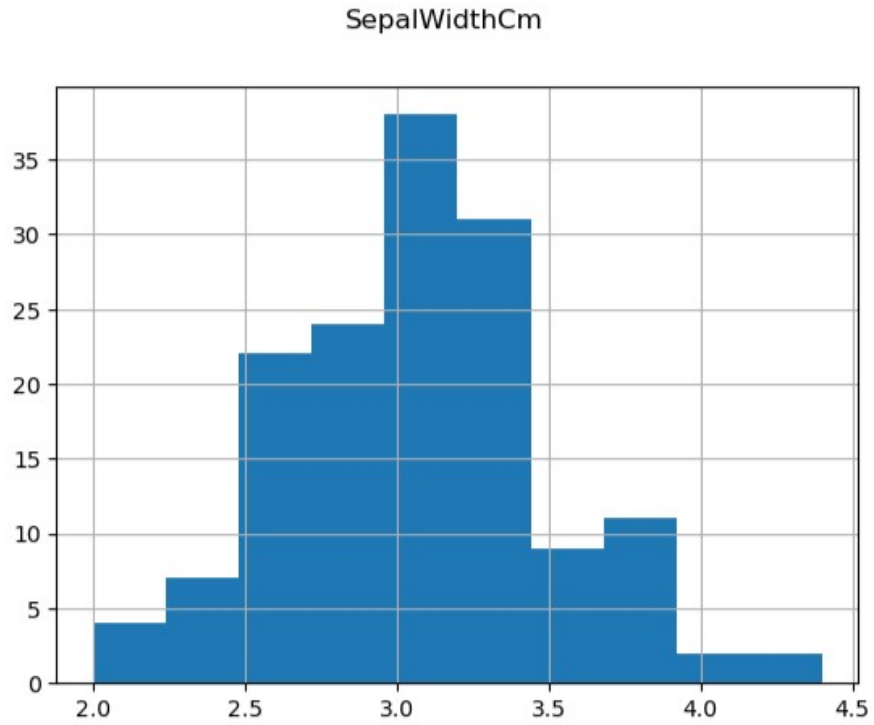
Out[11]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
127	128	6.1	3.0	4.9	1.8	Iris-virginica
104	105	6.5	3.0	5.8	2.2	Iris-virginica
55	56	5.7	2.8	4.5	1.3	Iris-versicolor
84	85	5.4	3.0	4.5	1.5	Iris-versicolor
44	45	5.1	3.8	1.9	0.4	Iris-setosa

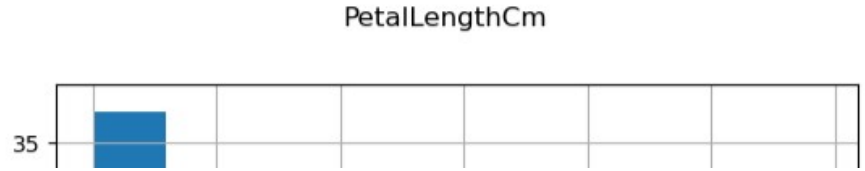
In [12]:
```
col1 = 'SepalLengthCm'
df[col1].hist()
plt.suptitle(col1)
plt.show()
```

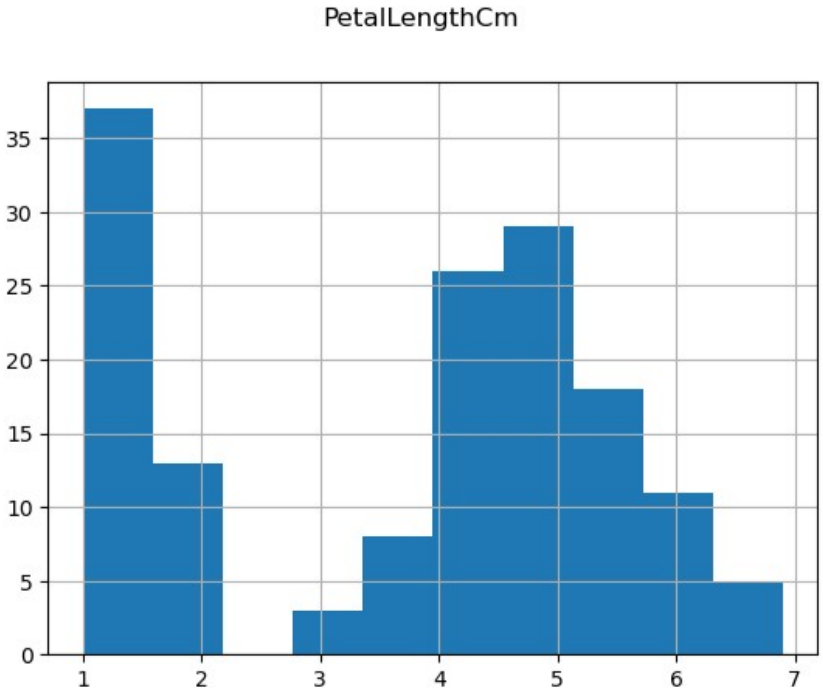


In [13]:
```
col2 = 'SepalWidthCm'
df[col2].hist()
plt.suptitle(col2)
plt.show()
```

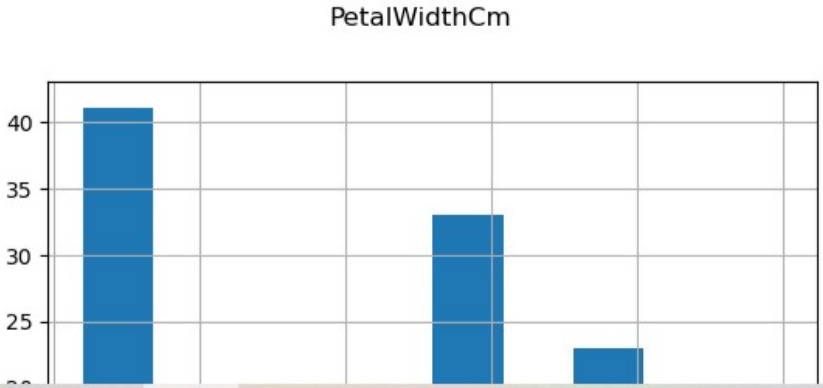


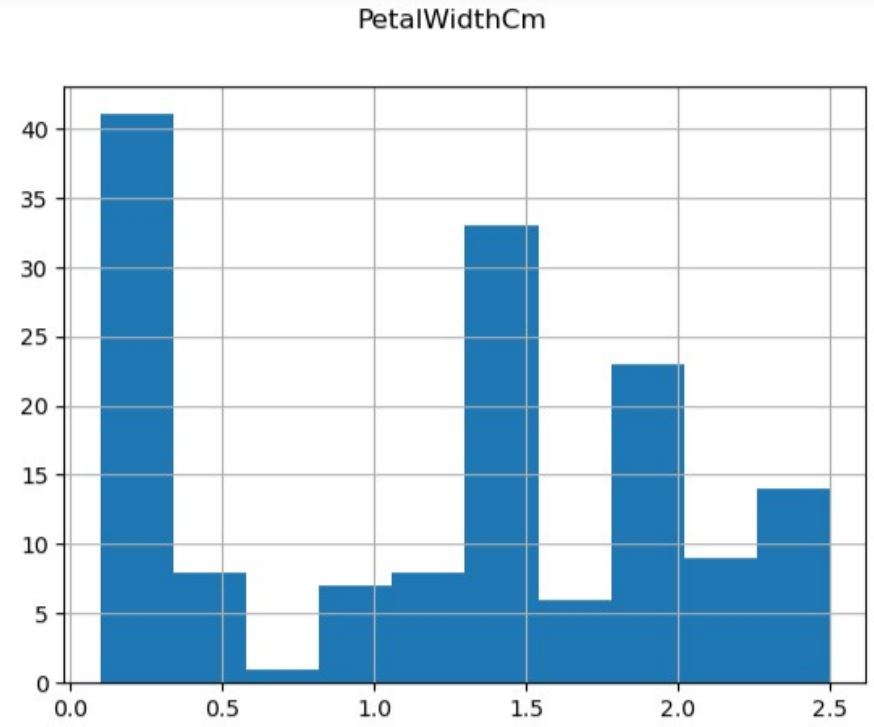
In [14]:
```
col3 = 'PetalLengthCm'
df[col3].hist()
plt.suptitle(col3)
plt.show()
```





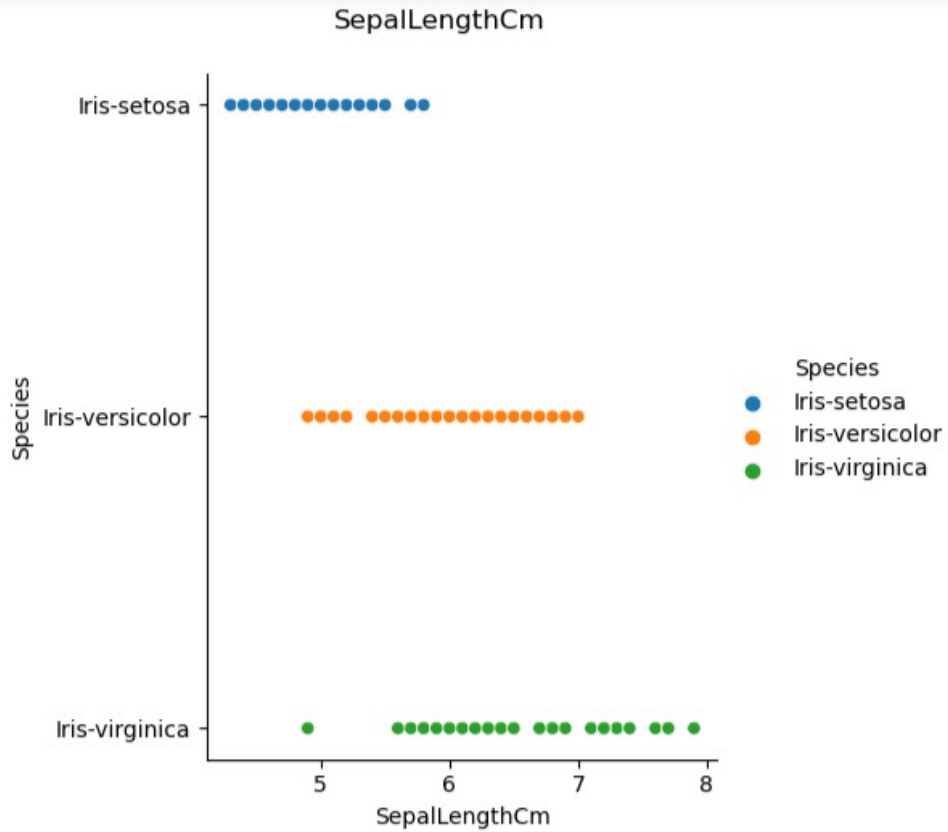
In [15]:
```
col4 = 'PetalWidthCm'
df[col4].hist()
plt.suptitle(col4)
plt.show()
```



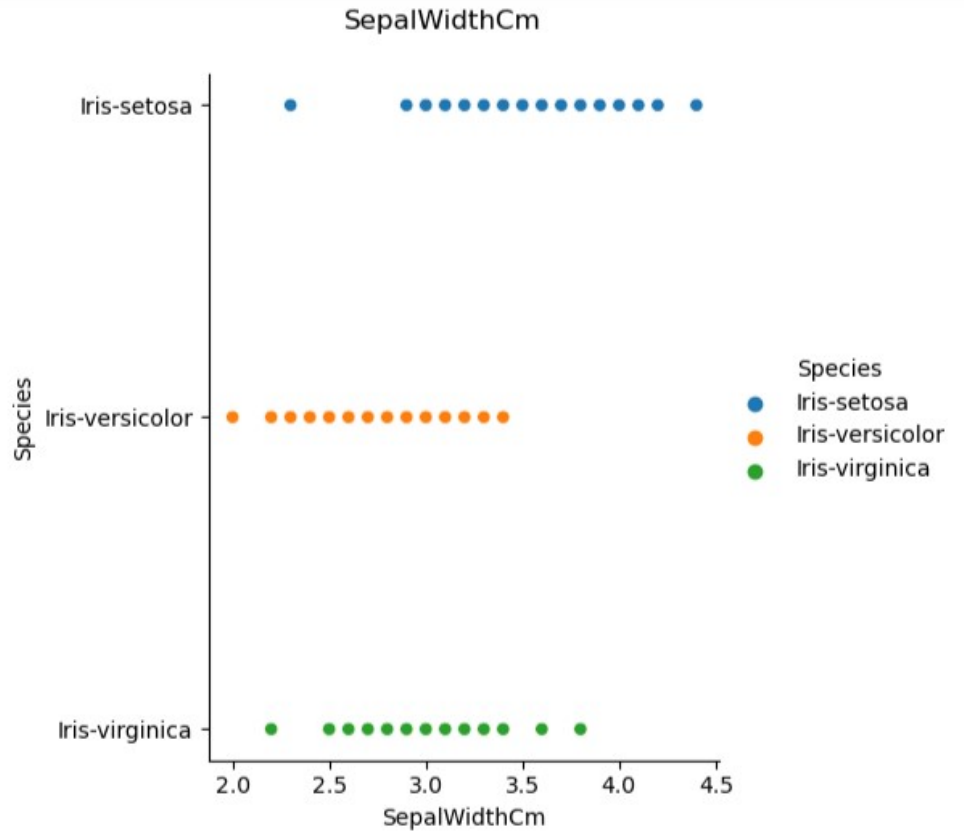


Relationship Between Column and Species

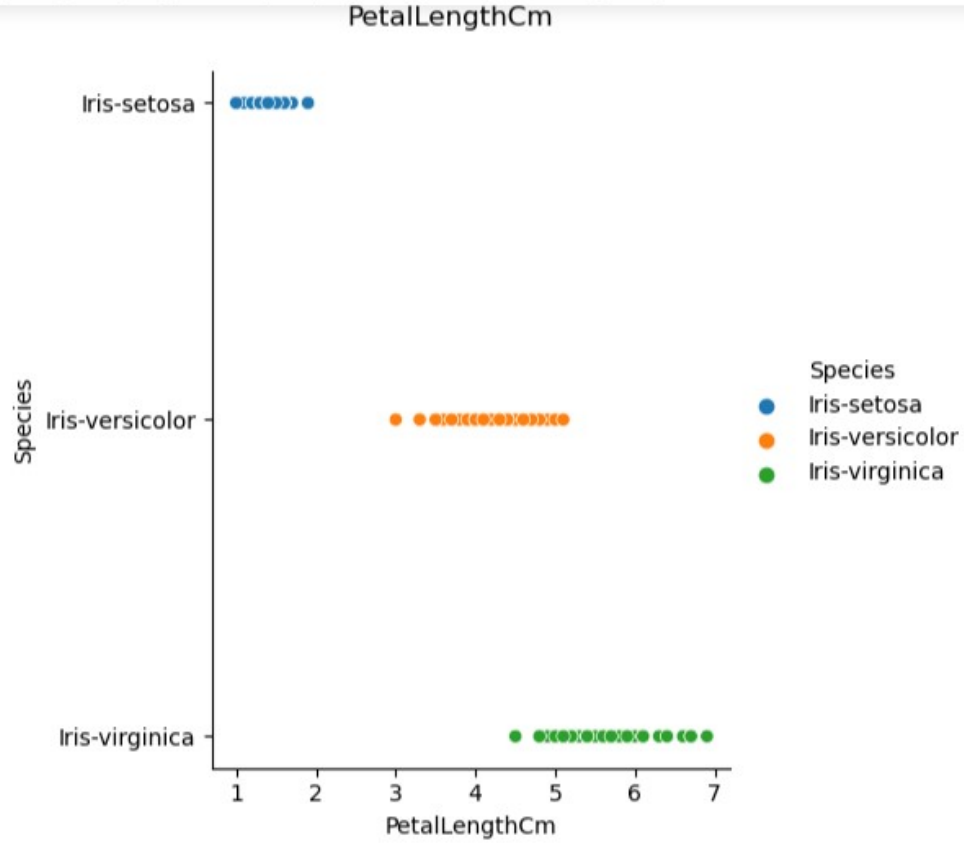
In [16]:
```
col1 = 'SepalLengthCm'
sns.relplot(x=col1, y='Species', hue = 'Species', data = df)
plt.suptitle(col1, y=1.05)
plt.show()
```



In [17]:
```
col2 = 'SepalWidthCm'
sns.relplot(x=col2, y='Species', hue = 'Species', data = df)
plt.suptitle(col2, y=1.05)
plt.show()
```



In [18]:
```
col3 = 'PetalLengthCm'
sns.relplot(x=col3, y='Species', hue = 'Species', data = df)
plt.suptitle(col3, y=1.05)
plt.show()
```

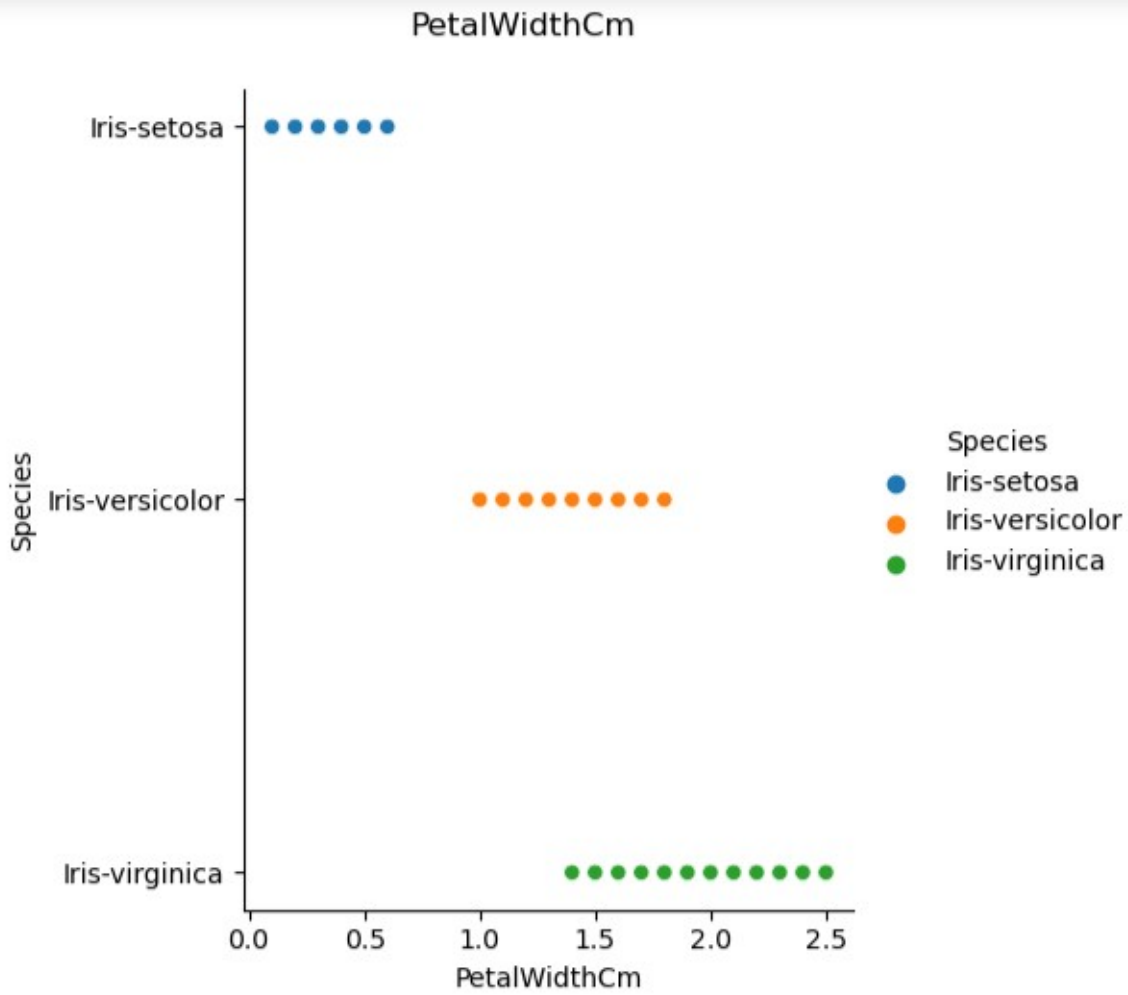


In [19]:
```
col3 = 'PetalWidthCm'
sns.relplot(x=col3, y='Species', hue = 'Species', data = df)
plt.suptitle(col3, y=1.05)
plt.show()
```

jupyter Iris Flower Classification Last Checkpoint: 6 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code



From here It is clear that we can predict the Species of Setosa based on Petal Width or Petal Lenght as it do not overlap with other two species

jupyter Iris Flower Classification Last Checkpoint: 6 hours ago (autosaved)

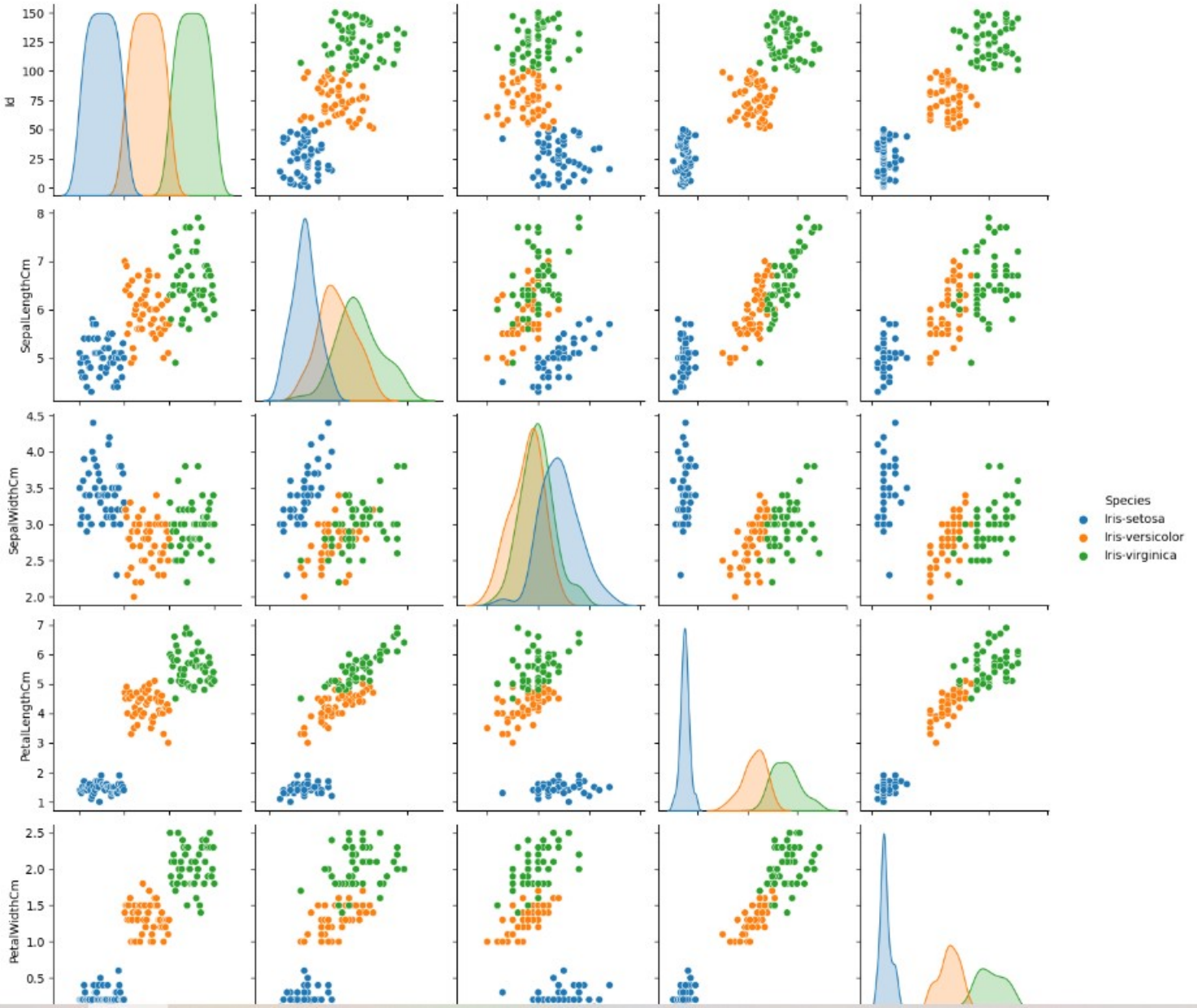
Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [20]:
```
sns.pairplot(df, hue='Species')
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x1e5ceb91520>



Train Test Split

In [21]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

In [22]:
```
df['Species'] = le.fit_transform(df['Species'])
df.head(90)
```

Out[22]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
85	86	6.0	3.4	4.5	1.6	1
86	87	6.7	3.1	4.7	1.5	1
87	88	6.3	2.3	4.4	1.3	1
88	89	5.6	3.0	4.1	1.3	1
89	90	5.5	2.5	4.0	1.3	1

90 rows × 6 columns

In [23]:
```
df['Species'].unique()
```

Out[23]:
```
array([0, 1, 2])
```

In [24]:
```
from sklearn.model_selection import train_test_split
```

In [25]:
```
df_train, df_test =train_test_split(df, test_size = 0.25)
```

In [26]:
```
df_train.shape
```

Out[26]:
```
(112, 6)
```

In [27]:
```
df_test.shape
```

Out[27]:
```
(38, 6)
```

In [28]:
```
df_test.head(10)
```

Out[28]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
139	140	6.9	3.1	5.4	2.1	2
4	5	5.0	3.6	1.4	0.2	0
134	135	6.1	2.6	5.6	1.4	2
59	60	5.2	2.7	3.9	1.4	1
60	61	5.0	2.0	3.5	1.0	1
52	53	6.9	3.1	4.9	1.5	1
16	17	5.4	3.9	1.3	0.4	0
38	39	4.4	3.0	1.3	0.2	0
8	9	4.4	2.9	1.4	0.2	0
127	128	6.1	3.0	4.9	1.8	2

In [29]:
```
df_train.head()
```

Out[29]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
143	144	6.8	3.2	5.9	2.3	2
47	48	4.6	3.2	1.4	0.2	0
64	65	5.6	2.9	3.6	1.3	1
132	133	6.4	2.8	5.6	2.2	2
121	122	5.6	2.8	4.9	2.0	2

Prepare Data For Modelling

In [30]:
```
X_train=df_train.drop(columns=["Species","Id"]).values
```

In [31]:
```
X_train.shape
```

Out[31]:
```
(112, 4)
```

In [32]:
```
y_train=df_train["Species"].values
y_train
```

Out[32]:
```
array([2, 0, 1, 2, 2, 2, 0, 0, 2, 0, 0, 1, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1
```

Out[32]:
```
array([2, 0, 1, 2, 2, 2, 0, 0, 2, 0, 0, 1, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1,
       0, 2, 1, 0, 0, 2, 1, 1, 1, 0, 0, 0, 2, 1, 2, 0, 0, 2, 2, 2, 2, 2,
       1, 1, 2, 0, 0, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 0, 2, 0, 2, 0,
       1, 0, 0, 1, 2, 2, 1, 0, 2, 0, 0, 2, 0, 1, 1, 2, 0, 1, 0, 0, 0, 2,
       2, 1, 2, 1, 0, 0, 1, 2, 0, 2, 1, 0, 2, 1, 2, 0, 2, 1, 2, 0, 1, 1,
       2, 1])
```

In [33]:
```
train, test = train_test_split(df, test_size = 0.25)
print(train.shape)
print(test.shape)
```

```
(112, 6)
(38, 6)
```

In [34]:
```
X_test=df_test.drop(columns=['Species','Id']).values
y_test=df_test['Species'].values
```

In [35]:
```
X_test.shape
```

Out[35]:
```
(38, 4)
```

Manual Modelling based on pairplot

In [36]:
```
df['Species']
```

Out[36]:
```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int32
```

In [37]:
```
def single_feature_prediction(petal_length):
    if petal_length < 2.7:
        return 0
    elif  petal_length < 4.9:
        return 1
    else:
        return 2
```

jupyter Iris Flower Classification Last Checkpoint: 6 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [38]:
```
df_train.columns
```

Out[38]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [39]:
```
X_train[:,3]
```

Out[39]:
```
array([2.3, 0.2, 1.3, 2.2, 2. , 2.4, 0.2, 0.1, 1.8, 0.1, 0.2, 1.5, 1.3,
       0.3, 1.2, 1.1, 1.8, 1. , 1.8, 1.4, 2.3, 1.5, 0.2, 1.9, 1.3, 0.2,
       0.1, 1.8, 1.4, 1.2, 1.6, 0.4, 0.1, 0.1, 1.8, 1.2, 2.3, 0.5, 0.2,
       1.8, 1.8, 1.5, 2. , 1.9, 1.5, 1.3, 2.5, 0.2, 0.2, 1.6, 1.5, 2.3,
       1.4, 1. , 1.8, 1. , 1.4, 2.5, 2.4, 1.7, 1.9, 0.2, 2. , 0.4, 2. ,
       0.2, 1. , 0.4, 0.2, 1.4, 1.8, 2.1, 1.3, 0.4, 2. , 0.3, 0.2, 2.3,
       0.3, 1.2, 1.3, 2.3, 0.2, 1.3, 0.1, 0.2, 0.2, 1.8, 1.6, 1.1, 1.7,
       1.3, 0.4, 0.2, 1.5, 1.5, 0.2, 2.1, 1.5, 0.3, 2.4, 1.3, 2.3, 0.3,
       1.8, 1.1, 2.1, 0.2, 1.2, 1.5, 1.9, 1.3])
```

In [40]:
```
manual_y_prediction =[single_feature_prediction(val) for val in X_train[:,3]]
```

In [41]:
```
y_train
```

Out[41]:
```
array([2, 0, 1, 2, 2, 2, 0, 0, 2, 0, 0, 1, 1, 0, 1, 1, 1, 1, 2, 1, 2, 1,
       0, 2, 1, 0, 0, 2, 1, 1, 1, 0, 0, 0, 2, 1, 2, 0, 0, 2, 2, 2, 2, 2,
       1, 1, 2, 0, 0, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 0, 2, 0, 2, 0,
       1, 0, 0, 1, 2, 2, 1, 0, 2, 0, 0, 2, 0, 1, 1, 2, 0, 1, 0, 0, 0, 2,
       2, 1, 2, 1, 0, 0, 1, 2, 0, 2, 1, 0, 2, 1, 2, 0, 2, 1, 2, 0, 1, 1,
       2, 1])
```

In [42]:
```
manual_y_prediction == y_train
```

Out[42]:
```
array([False,  True, False, False, False, False,  True,  True, False,
        True,  True, False, False,  True, False, False, False, False,
       False, False, False, False,  True, False, False,  True,  True,
       False, False, False, False,  True,  True,  True, False, False,
       False,  True,  True, False, False, False, False, False, False,
       False, False,  True,  True, False, False, False, False, False,
       False, False, False, False, False, False, False,  True, False,
        True, False,  True, False,  True,  True, False, False, False,
       False,  True, False,  True,  True, False,  True, False, False,
       False,  True, False,  True,  True,  True, False, False, False,
       False, False,  True,  True, False, False,  True, False, False,
        True, False, False, False,  True, False, False, False,  True,
       False, False, False, False])
```

In [43]:
```
manual_model_accuracy = np.mean(manual_y_prediction == y_train)
```

In [44]:
```
manual_model_accuracy
```

Out[44]:
```
0.3125
```

This is the manual Model Accuracy result

Modelling

In [45]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [46]:
```
model = LogisticRegression(max_iter=1000)
```

In [47]:
```
model.fit(X_train,y_train)
```

Out[47]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [48]:
```
prediction = model.predict(X_test)
print('Accuracy:',accuracy_score(prediction,y_test))
```

```
Accuracy: 0.9736842105263158
```

Using Confusion Matrix

In [49]:
```
from sklearn.metrics import confusion_matrix,classification_report
confusion_mat = confusion_matrix(y_test,prediction)
print("Confusion matrix: \n",confusion_mat)
print(classification_report(y_test,prediction))
```

```
Confusion matrix:
 [[15  0  0]
 [ 0 11  1]
```

```
Confusion matrix:
 [[15  0  0]
 [ 0 11  1]
 [ 0  0 11]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       1.00      0.92      0.96        12
           2       0.92      1.00      0.96        11

    accuracy                           0.97        38
   macro avg       0.97      0.97      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Using KNN Neighbors

In [50]:
```
from sklearn.neighbors import KNeighborsClassifier
model2 = KNeighborsClassifier(n_neighbors=5)
model2.fit(X_train,y_train)
y_pred2 = model2.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(y_test,y_pred2))
```

```
Accuracy Score: 0.9736842105263158
```

Using Decision Tree

In [51]:
```
from sklearn import tree

dt_model = tree.DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
```

Out[51]:
```
DecisionTreeClassifier()
```

In [52]:
```
from sklearn.metrics import accuracy_score

prediction_dt = dt_model.predict(X_test)
accuracy_dt = accuracy_score(y_test, prediction_dt)
```

jupyter Iris Flower Classification Last Checkpoint: 6 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

In [52]:
```
from sklearn.metrics import accuracy_score

prediction_dt = dt_model.predict(X_test)
accuracy_dt = accuracy_score(y_test, prediction_dt)
```

In [53]:
```
accuracy_dt
```

Out[53]:
```
0.9736842105263158
```

In [64]:
```
results = pd.DataFrame({
    'Model': ['Logistic Regression','KNN','Decision Tree' ],
    'Score': [0.97,0.97,0.97]})

result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

Out[64]:

Score	Model
0.97	Logistic Regression
0.97	KNN
0.97	Decision Tree

All three are giving 97% results.

In [55]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(dt_model,X_train, y_train, scoring='neg_mean_squared_error', cv=10)
rmse_scores = np.sqrt(-scores)
rmse_scores
```

Out[55]:
```
array([0.        , 0.28867513, 0.        , 0.42640143, 0.        ,
       0.        , 0.30151134, 0.30151134, 0.30151134, 0.        ])
```

In [56]:
```
y_test
```

Out[56]:
```
array([2, 0, 2, 1, 1, 1, 0, 0, 0, 2, 1, 2, 1, 0, 0, 0, 2, 0, 0, 1, 1, 2,
       1, 0, 0, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 1, 2, 0])
```

In [57]:
```
prediction_dt
```

Out[57]:
```
array([2, 0, 2, 1, 1, 1, 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 2, 0, 0, 1, 1, 2,
```

Out[57]:
```
array([2, 0, 2, 1, 1, 1, 0, 0, 0, 2, 2, 2, 1, 0, 0, 0, 2, 0, 0, 1, 1, 2,
       1, 0, 0, 1, 0, 2, 2, 0, 1, 1, 0, 2, 2, 1, 2, 0])
```

Creating Category

Let's first enter the data manually

In [58]:
```
category=['Setosa','Versicolor','Virginica']
```

In [59]:
```
data = 5.7,3,4.2,1.1
```

In [60]:
```
data_array = np.array([data])
data_array
```

Out[60]:
```
array([[5.7, 3. , 4.2, 1.1]])
```

In [61]:
```
predic = dt_model.predict(data_array)
```

In [62]:
```
print(category[int(predic[0])])
```

```
Versicolor
```

Let's take the input from the user

In [65]:
```
sepal_length = float(input("Enter Sepal Length (cm): "))
sepal_width = float(input("Enter Sepal Width (cm): "))
petal_length = float(input("Enter Petal Length (cm): "))
petal_width = float(input("Enter Petal Width (cm): "))

# Convert the user input into a NumPy array
input_data = np.array([[sepal_length, sepal_width, petal_length, petal_width]])

# Use the trained model to predict the species of the flower
predicted_species = dt_model.predict(input_data)

# Display the predicted species to the user
print("Predicted Species:", predicted_species[0])
```

```
Enter Sepal Length (cm): 5.5
Enter Sepal Width (cm): 3.9
Enter Petal Length (cm): 1.6
Enter Petal Width (cm): 0.4
Predicted Species: 0
```

It is clear that the Species is Setosa.

As 0 indicates to Setosa

1 indicates to Versicolor

2 indicates Virginica

Conclusion

So, we were given different features of the flowers and have to make a model to predict the species of the flower based on random values of each features.

The features (attributes) of the flowers are as follows:

Sepal Length (in centimeters) Sepal Width (in centimeters) Petal Length (in centimeters) Petal Width (in centimeters)

Throughout our analysis, we observed that the different species of Iris flowers exhibit distinct characteristics in terms of sepal and petal measurements. By using machine learning algorithms such as logistic regression, decision trees, KNN classification, we were able to build models that accurately classify Iris flowers into their respective species based on these measurements.