# DATASET - IRIS

## 1. Data Analysis

Load the dataset and perform initial exploratory data analysis (EDA) to understand its structure and characteristics.

In [5]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [6]:
```
# Load the Iris dataset
iris_df = pd.read_csv('IRIS.csv')
iris_df
```

Out[6]:

| | sepal_length | sepal_width | petal_length | petal_width | 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 [7]:
```
# Display basic information about the dataset
print(iris_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

In [8]:
```
# Summary statistics
print(iris_df.describe())
```

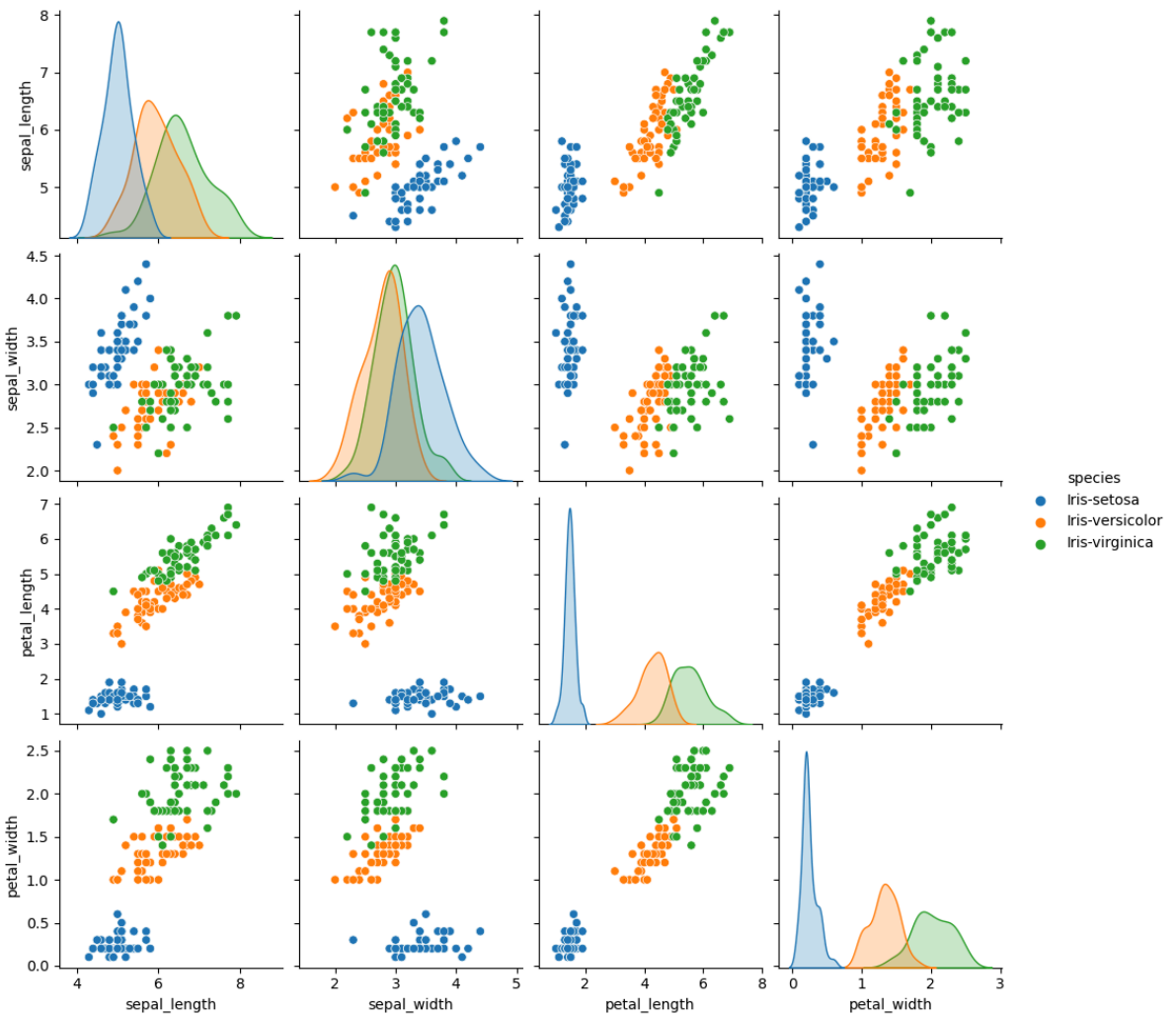
```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
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 [9]:
```
# Class distribution
print(iris_df['species'].value_counts())
```

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

In [10]:
```
# Pairplot for initial visualization
sns.pairplot(iris_df, hue='species')
plt.show()
```

```
C:\Users\Nishita Bala\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWar
ning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




# 2. Preprocessing

Prepare the data for modeling by handling missing values, encoding categorical variables, and splitting into training and testing sets.

In [11]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

In [12]:
```
# Encode categorical target variable 'species'
le = LabelEncoder()
iris_df['species_encoded'] = le.fit_transform(iris_df['species'])
```

In [13]:
```
# Split dataset into features (X) and target (y)
X = iris_df.drop(['species', 'species_encoded'], axis=1)
y = iris_df['species_encoded']
```

In [14]:
```
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

# 3. Feature Engineering

While the Iris dataset is relatively clean, you might want to scale the features for some models or engineer new features

In [15]:
```
from sklearn.preprocessing import StandardScaler

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

# 4. Machine Learning Modeling Techniques

In [16]:
```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
```

In [17]:
```
# Initialize SVM classifier
svm_clf = SVC(kernel='rbf', random_state=42)
```

In [18]:
```
# Train the model
svm_clf.fit(X_train_scaled, y_train)
```

Out[18]:
```
SVC(random_state=42)
```

In [19]:
```
# Predictions
y_pred = svm_clf.predict(X_test_scaled)
```

In [20]:
```
# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
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
```

In [21]:

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=le.classes_, yticklа
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```