Iris Dataset: End-to-End Machine Learning Workflow

This notebook demonstrates a complete machine learning workflow on the classic Iris dataset, including data loading, preprocessing, model training, evaluation, and prediction.

---

1. Imports & Setup

In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
```

2. Load the Iris Dataset

We load the dataset from a CSV file and assign column names.

In [3]:
```python
# Load the dataset
df = pd.read_csv('../data/iris.csv', header=None)
df.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
df.head()
```

Out[3]:

	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

3. Exploratory Data Analysis (EDA)

Let's explore the data with some visualizations.

In [4]:
```python
# Basic statistics
df.describe()
```

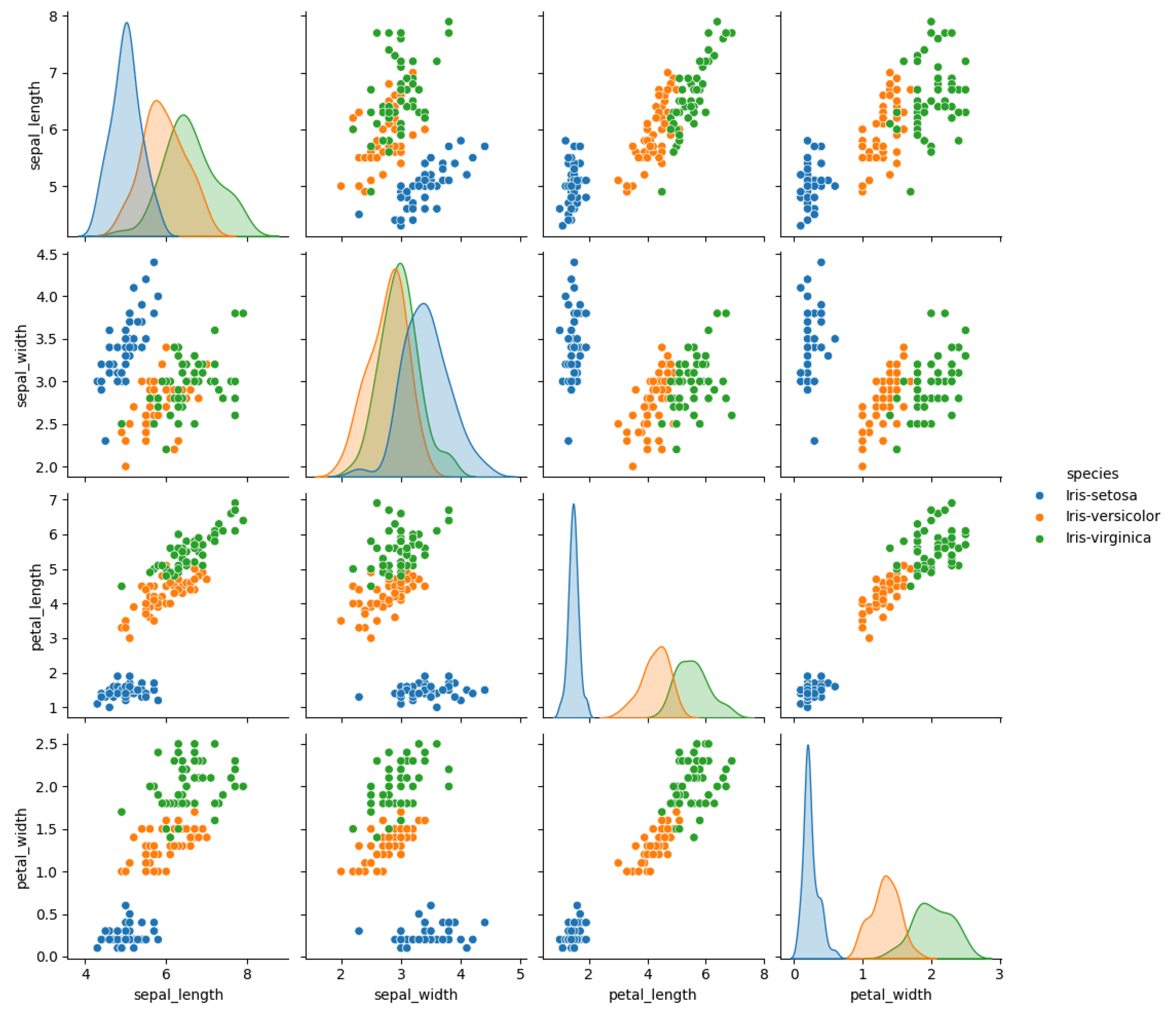
Out[4]:

	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

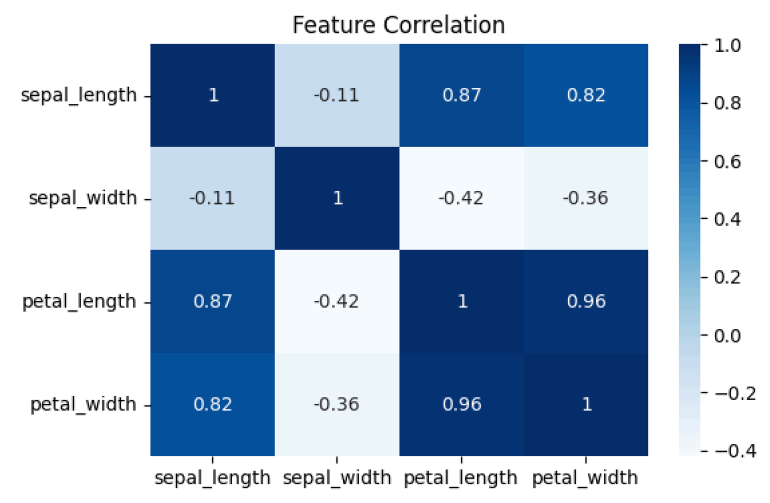
In [5]:
```python
# Class distribution
df['species'].value_counts()
```

Out[5]:
```
species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: count, dtype: int64
```

In [6]:
```python
# Pairplot for feature relationships
sns.pairplot(df, hue='species')
plt.show()
```



In [8]:
```python
# Correlation heatmap (numeric features only)
plt.figure(figsize=(6,4))
sns.heatmap(df.select_dtypes(include='number').corr(), annot=True, cmap='Blues')
plt.title('Feature Correlation')
plt.show()
```



4. Data Preprocessing

We encode the target labels and split the data into training and test sets.

In [9]:
```python
# Encode species labels
le = LabelEncoder()
df['species_encoded'] = le.fit_transform(df['species'])

# Features and target
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species_encoded']

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
print("Label classes:", le.classes_)
```

```
Train shape: (120, 4)
Test shape: (30, 4)
Label classes: ['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

5. Model Training

We train a Logistic Regression classifier on the training data.

In [10]:
```python
# Train the model
clf = LogisticRegression(max_iter=200)
clf.fit(X_train, y_train)
```

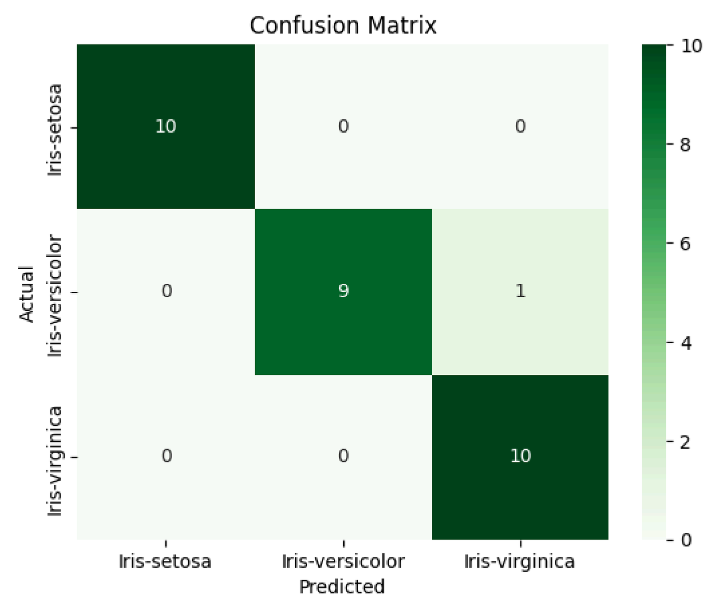
Out[10]:
```
▼ LogisticRegression
▶ Parameters
```

6. Model Evaluation

We evaluate the model's performance on the test set.

In [11]:
```python
# Predict on test set
y_pred = clf.predict(X_test)

# Accuracy
print("Accuracy:", accuracy_score(y_test, y_pred))

# Classification report
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9666666666666667
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      0.90      0.95        10
 Iris-virginica       0.91      1.00      0.95        10

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.97        30
   weighted avg       0.97      0.97      0.97        30
```



7. Save the Model

We save the trained model and label encoder for future use.

In [12]:
```python
joblib.dump({'model': clf, 'label_encoder': le}, '../models/iris_model.pkl')
print("Model saved to ../models/iris_model.pkl")
```

```
Model saved to ../models/iris_model.pkl
```

8. Make Predictions

Let's load the saved model and make a prediction on a new sample.

In [13]:
```python
# Load model and label encoder
saved = joblib.load('../models/iris_model.pkl')
clf_loaded = saved['model']
le_loaded = saved['label_encoder']

# Example sample: [sepal_length, sepal_width, petal_length, petal_width]
sample = np.array([[5.1, 3.5, 1.4, 0.2]])
pred = clf_loaded.predict(sample)
species = le_loaded.inverse_transform(pred)
print(f"Predicted species: {species[0]}")
```

```
Predicted species: Iris-setosa
/home/kashaf/Desktop/iris_python_ml_model/venv/lib/python3.10/site-packages/sklearn/utils/validation.py:2749: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```

Conclusion

We have completed an end-to-end machine learning workflow on the Iris dataset, including data exploration, preprocessing, model training, evaluation, and prediction. You can now extend this notebook with more models, hyperparameter tuning, or advanced visualizations!