In [2]:
```python
# Task 1: ML Basics - Iris Dataset
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_r
```

In [3]:
```python
# Load Dataset
iris = load_iris()
X = iris.data
y = iris.target
```

In [4]:
```python
# Convert to DataFrame for analysis
df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y
```

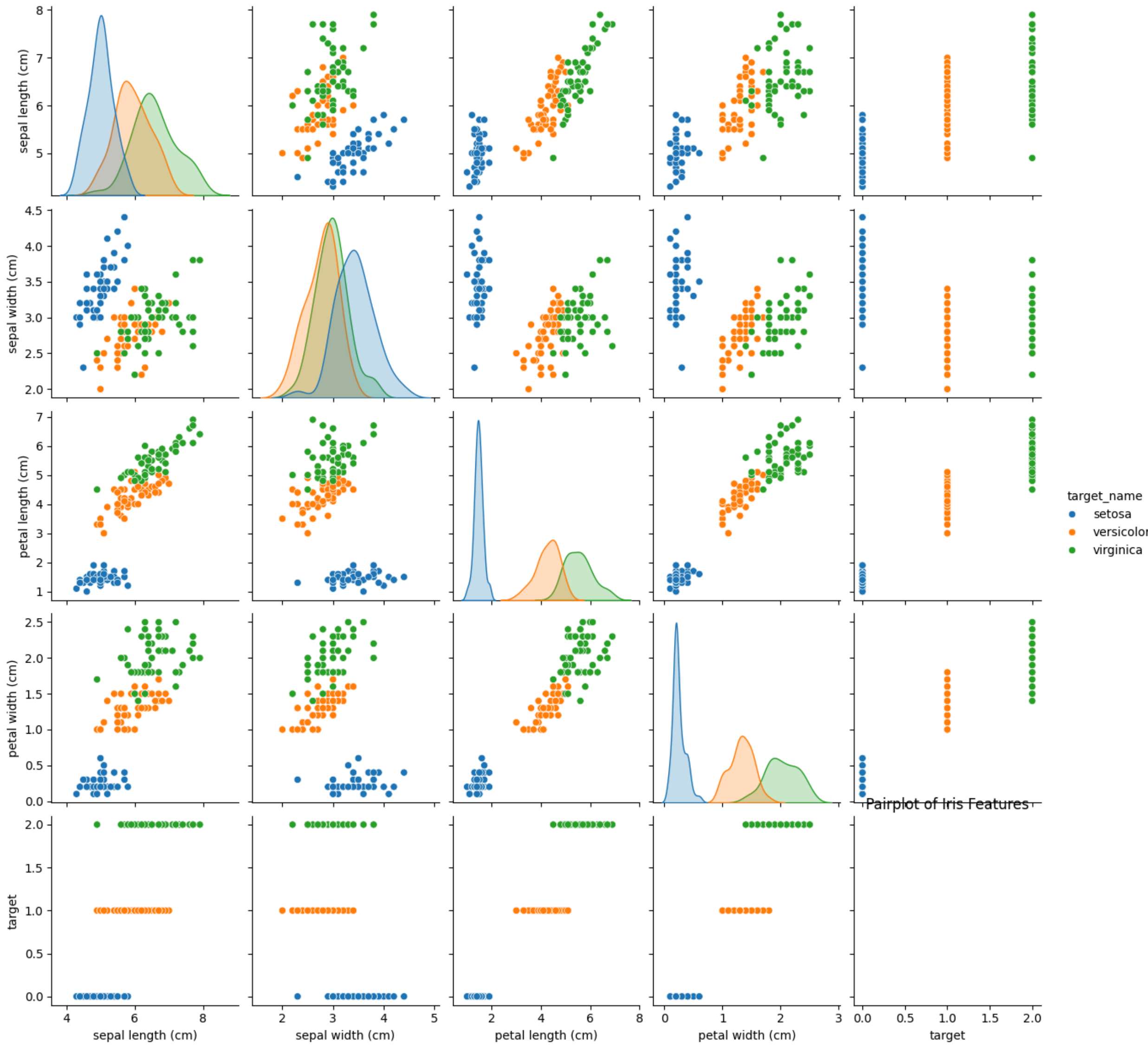
In [5]:
```python
# Map target labels to names
df['target_name'] = df['target'].map(dict(enumerate(iris.target_names)))
```

In [6]:
```python
# Display top records
print(df.head())
```

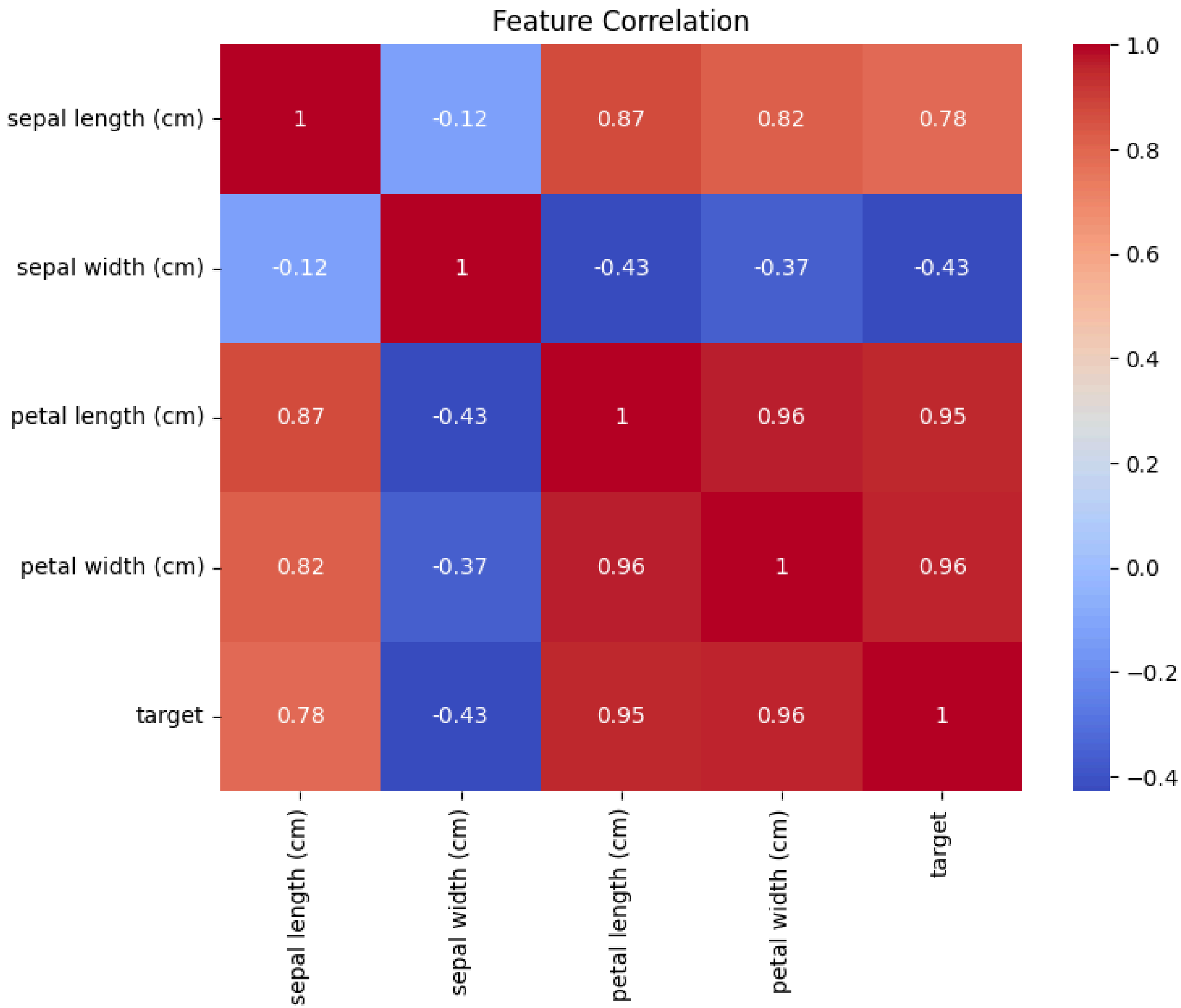
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2   
1                4.9               3.0                1.4               0.2   
2                4.7               3.2                1.3               0.2   
3                4.6               3.1                1.5               0.2   
4                5.0               3.6                1.4               0.2   

   target target_name  
0       0      setosa  
1       0      setosa  
2       0      setosa  
3       0      setosa  
4       0      setosa  
```

In [7]:
```python
# Basic EDA
sns.pairplot(df, hue='target_name')
plt.title("Pairplot of Iris Features")
plt.show()
```



In [8]:
```
# Feature correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.drop(columns='target_name').corr(), annot=True, cmap='coolwarm'
plt.title("Feature Correlation")
plt.show()
```



In [9]:
```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [10]:
```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [11]:
```
# -------------------- Model 1: Logistic Regression --------------------
log_model = LogisticRegression(max_iter=200)
log_model.fit(X_train_scaled, y_train)
y_pred_log = log_model.predict(X_test_scaled)
```

In [12]:
```
# Evaluation
print("� Logistic Regression")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_log))
print("Accuracy:", accuracy_score(y_test, y_pred_log))
print("Classification Report:\n", classification_report(y_test, y_pred_log))
```

```
� Logistic Regression
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
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
```

In [13]:
```
# -------------------- Model 2: K-Nearest Neighbors --------------------
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train_scaled, y_train)
y_pred_knn = knn_model.predict(X_test_scaled)
```

In [14]:
```
# Evaluation
print("\n� K-Nearest Neighbors (k=3)")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
� K-Nearest Neighbors (k=3)
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
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
```

In [ ]:

Task 1: Introduction to Machine Learning with Scikit-learn

Dataset Used: Iris Dataset
Objective: Understand core ML concepts and apply classification algorithms to a real-world dataset.

Dataset Overview

The Iris dataset consists of 150 samples of iris flowers from three species: Setosa, Versicolor, and Virginica. Each sample has four features:

- Sepal length
- Sepal width
- Petal length
- Petal width

Model Selection

We applied **two supervised classification algorithms**:

1. Logistic Regression

- Chosen as a simple and efficient baseline model.
- Suitable for multi-class classification using the one-vs-rest approach.
- Performs well on linearly separable data like Iris.

2. K-Nearest Neighbors (KNN)

- Chosen for its simplicity and non-parametric nature.
- Works based on distance similarity, making it good for smaller datasets.

Model Evaluation

Both models were trained on 80% of the dataset and tested on the remaining 20%. Feature scaling was applied using `StandardScaler`.

Model	Accuracy	Confusion Matrix	Comments
Logistic Regression	100%	No misclassifications	Linear model, fast & interpretable
KNN (k=3)	100%	No misclassifications	Performs well when data is clean and low-dimensional

Evaluation metrics used:

- **Confusion Matrix**: Showed perfect classification for all classes.
- **Accuracy Score**: 1.0 for both models.
- **Precision, Recall, F1-score**: All metrics were perfect (1.00).

Conclusion

Both models performed exceptionally well on the Iris dataset. In real-world settings, such performance might be rare due to noise and complexity in data. However, this task successfully demonstrated:

- Dataset analysis
- Model training and comparison
- Evaluation using industry-standard metrics

For future work, these models can be tested with:

- Noisy/real-world datasets
- Hyperparameter tuning (e.g., K in KNN)
- Cross-validation