Step 1 — Importing Libraries

Purpose of Each Import:

- **warnings** → Ignore unnecessary warnings for cleaner output.
- **numpy, pandas** → Handle numerical and tabular data.
- **matplotlib, seaborn** → For data visualization and plots.
- **train_test_spilt** → For dividing the dataset into 85/15 ratio

---

In [97]:
```
import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

#Builtin Model
from sklearn.svm import LinearSVC
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
```

Step 2 — Load Iris dataset

Goal:

- Load the Iris dataset, inspect it.
- Make sure that the dataset is fully loaded and ready for processing.

In [98]:
```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/master/iris.csv"
df = pd.read_csv(url)

# Basic info and sanity check
print("Dataset Shape:", df.shape)
print("\nFirst 5 rows:\n", df.head())
print("\nClass distribution:\n", df['species'].value_counts())
```

```
Dataset Shape: (150, 5)

First 5 rows:
    sepal_length  sepal_width  petal_length  petal_width species
0            5.1          3.5           1.4          0.2  setosa
1            4.9          3.0           1.4          0.2  setosa
2            4.7          3.2           1.3          0.2  setosa
3            4.6          3.1           1.5          0.2  setosa
4            5.0          3.6           1.4          0.2  setosa

Class distribution:
 species
setosa        50
versicolor    50
virginica     50
Name: count, dtype: int64
```

Step 3 — Create train/test split (15% test)

Goal:

- Split the dataset into training and testing sets using **stratified sampling** so each class is equally represented.
- We will use **15% of the data for testing** and `random_state=42` for reproducibility.

In this step we will:

- Perform a 85/15 train-test split
- Inspect shapes and class distribution

In [99]:
```
# Features and target
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
target_name = 'species'

X = df[feature_names]
y = df[target_name]

# Stratified train-test split (15% test)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.15,strati


# Output shapes
print("\n=== Train/Test Split Information ===")
print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)

# Class distribution in train and test sets
```

```
print("\nClass distribution in Train:")
print(y_train.value_counts())

print("\nClass distribution in Test:")
print(y_test.value_counts())
```

```
=== Train/Test Split Information ===
Train shape: (127, 4)
Test shape: (23, 4)

Class distribution in Train:
species
setosa        43
virginica     42
versicolor    42
Name: count, dtype: int64

Class distribution in Test:
species
versicolor    8
virginica     8
setosa        7
Name: count, dtype: int64
```

Step 4 — Label Encoding

We need to convert the categorical `species` labels into numeric form because SVM only works with numerical labels.

This step performs:

- Create a LabelEncoder
- Fit on full species column

In [100…

```
# Create encoder
label_encoder = LabelEncoder()

# Encode full y labels
y_encoded = label_encoder.fit_transform(y)

# Encode train and test labels separately
y_train_enc = label_encoder.transform(y_train)
y_test_enc  = label_encoder.transform(y_test)


print("Classes found:", label_encoder.classes_)
print("\nSample encoded labels from training set:\n", y_train_enc[:10])
```

```
Classes found: ['setosa' 'versicolor' 'virginica']

Sample encoded labels from training set:
 [2 0 2 0 1 0 0 0 0 1]
```

Step 5 — Train Linear SVM (One-vs-One) with Cross-Validation

Now that labels are encoded, we train a **Linear Support Vector Machine** using the **One-vs-One (OvO)** strategy.

We will perform:

- GridSearchCV with 5-fold cross-validation
- Tune parameter C across: `[0.1, 1, 5, 10, 50, 100]`
- Select the best model based on highest accuracy

In [101…

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Linear SVM + OvO strategy
svm = SVC(kernel="linear", decision_function_shape="ovo")


# Define grid for parameter tuning
param_grid = {
    "C": [0.01, 0.1, 0.5, 1, 2, 5, 10, 20, 50, 100]
}

# Grid search with cross-validation
grid = GridSearchCV(estimator=svm, param_grid=param_grid ,scoring="accuracy" ,

# Train using training data
grid.fit(X_train, y_train_enc)

print("Best C value:", grid.best_params_["C"])
print("Best cross-validation accuracy:", grid.best_score_)

# Save best model for next steps
best_svm = grid.best_estimator_
```

```
Best C value: 1
Best cross-validation accuracy: 0.9846153846153847
```

Step 6 — Predict on Test Data

In this step, we will:

- Train the best SVM (`best_svm`) on the **full training data**
- Use it to **predict labels** on the test dataset

In [102…

```
# Train the best SVM on full training data
best_svm.fit(X_train, y_train_enc)

# Predict labels on test data
y_pred = best_svm.predict(X_test)

# Show first 10 predictions vs true labels
print("First 10 predictions:", y_pred[:10])
print("First 10 true labels:", y_test_enc[:10])

# Compute accuracy
accuracy = np.mean(y_pred == y_test_enc)
print(f"\nTest Accuracy: {accuracy*100:.2f}%")
```

```
First 10 predictions: [1 2 1 1 0 1 0 0 2 1]
First 10 true labels: [1 2 1 1 0 1 0 0 2 1]

Test Accuracy: 100.00%
```

Step 7 — Visualize Results and Report Metrics

Now that we have predictions, we will:

- Plot the **confusion matrix** using seaborn
- Report **precision, recall, and F1-score** for each class

This gives a clear evaluation of the SVM classifier's performance.

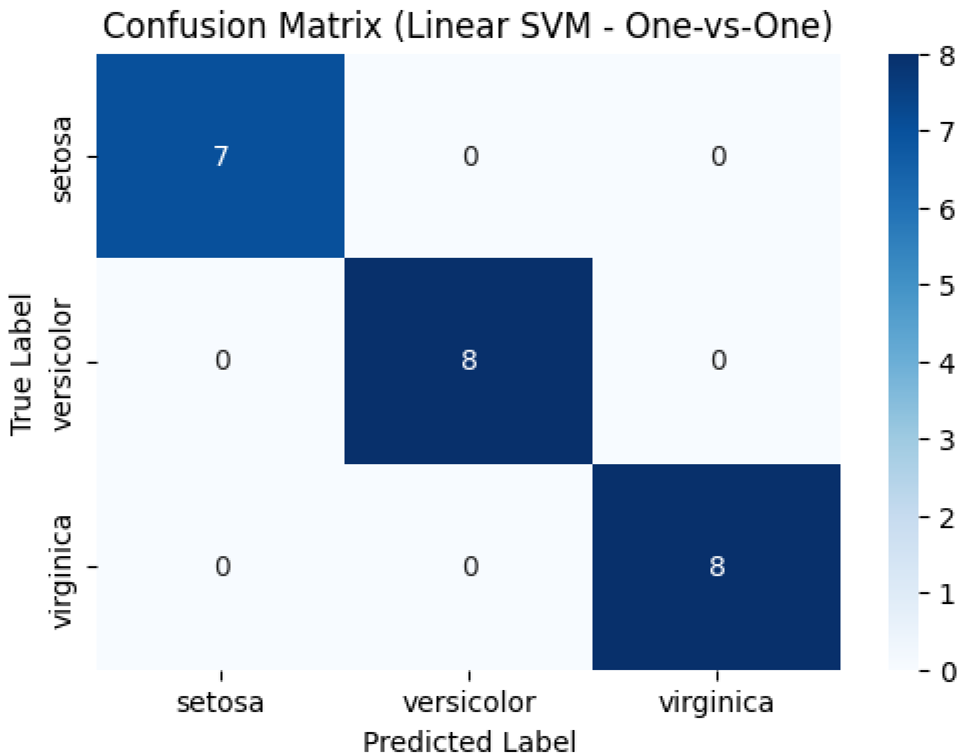
In [103…

```
# Confusion Matrix
cm = confusion_matrix(y_test_enc, y_pred)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d",
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix (Linear SVM - One-vs-One)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
```

```
plt.show()

# Precision, Recall, F1-score
print("\nClassification Report:")
print(classification_report(y_test_enc, y_pred, target_names=label_encoder.cla
```




```
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00         7
  versicolor       1.00      1.00      1.00         8
   virginica       1.00      1.00      1.00         8

    accuracy                           1.00        23
   macro avg       1.00      1.00      1.00        23
weighted avg       1.00      1.00      1.00        23
```