🌸 Iris Flower Classifier

This project is a beginner-friendly **machine learning classification task** using the classic **Iris dataset**, which contains measurements of three types of iris flowers: *Setosa*, *Versicolor*, and *Virginica*.

🔍 Objective

To build and compare multi-class classification models that can accurately predict the species of an iris flower based on four features:

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

✅ Topics Covered

- Data visualization with Seaborn
- Classification using:
    - K-Nearest Neighbors (KNN)
    - Support Vector Machine (SVM)
    - Decision Tree
- Model evaluation with cross-validation
- Hyperparameter tuning using GridSearchCV

Let's get started!

Section 1: Install & Import Libraries

In [57]:

```
# Install if needed
!pip install seaborn --quiet

# Import libraries
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, cross_val_score, Gr
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score
```

Section 2: Load and Explore the Dataset

In [58]:
```
# Load Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)

# Dataset preview
X.head()
```

Out[58]:

	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

In [59]:
```
# Check class distribution
y.value_counts()
```

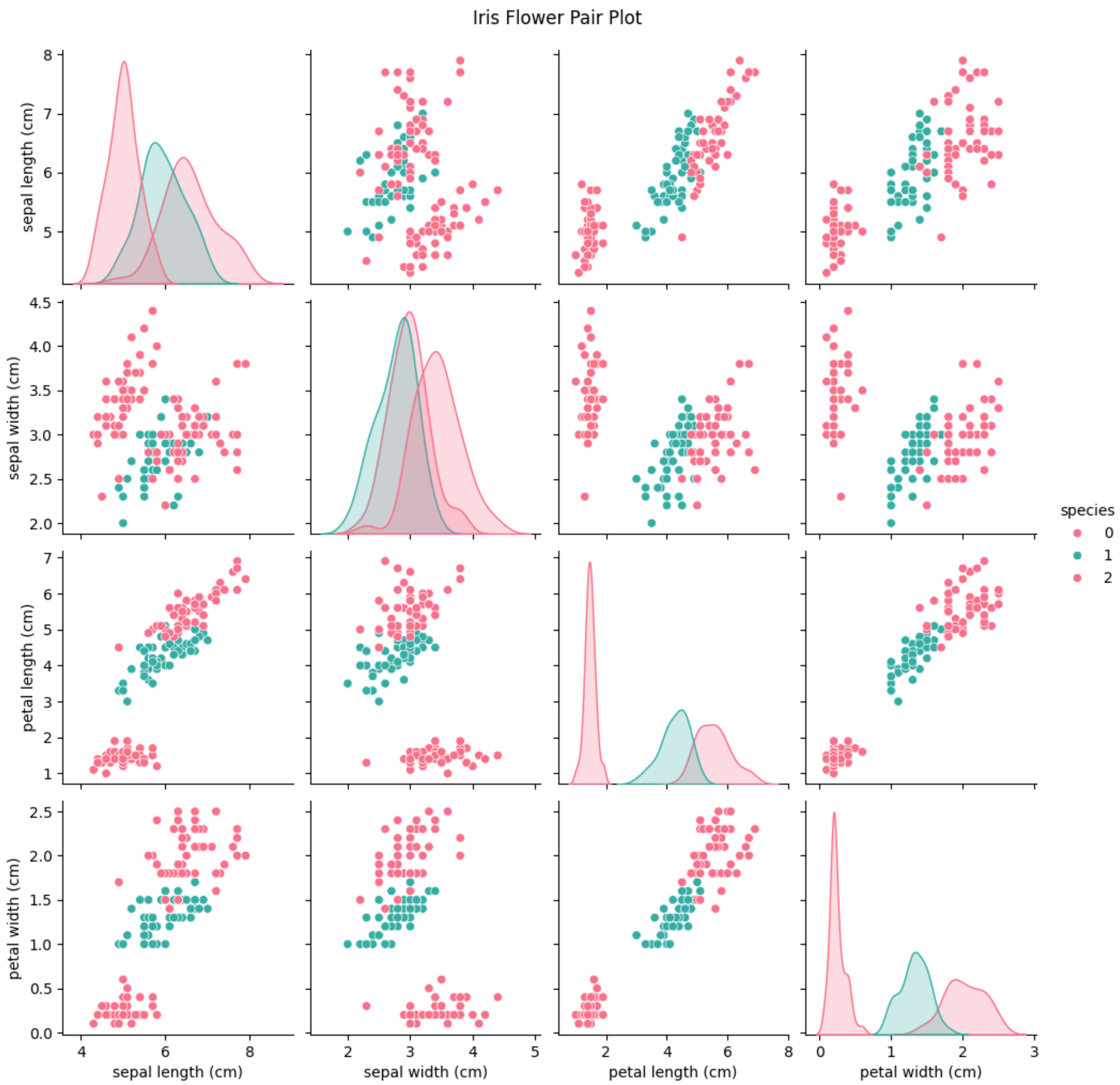
Out[59]:

	count
0	50
1	50
2	50

dtype: int64

Section 3: Visualize the Dataset

In [60]:
```
# Combine features and target for visualization
iris_df = pd.concat([X, pd.Series(y, name='species')], axis=1)

# Visualize with seaborn pairplot
sns.pairplot(iris_df, hue='species', palette='husl')
plt.suptitle("Iris Flower Pair Plot", y=1.02)
plt.show()
```



Section 4: Train-Test Split

In [61]:
```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")
```

```
Training samples: 120
Testing samples: 30
```

Section 5: Train Models & Cross-Validation

In [62]:
```
# Additional imports
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClas

# Try importing XGBoost if available
try:
    from xgboost import XGBClassifier
```

```python
    xgb_available = True
except ImportError:
    xgb_available = False
    print("XGBoost not installed. Skipping XGBClassifier.")

# Define all models in a dictionary
models = {
    'KNN': KNeighborsClassifier(),
    'Decision Tree': DecisionTreeClassifier(),
    'Logistic Regression': LogisticRegression(max_iter=200),
    'Naive Bayes': GaussianNB(),
    'SVM': SVC(),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier()
}

# Add XGBoost if available
if xgb_available:
    models['XGBoost'] = XGBClassifier(eval_metric='mlogloss')

# Cross-validation to evaluate each model
cv_results = {}

print("🔍 Cross-validation accuracy (5-fold):\n")
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5)
    mean_score = scores.mean()
    cv_results[name] = mean_score
    print(f"{name}: {mean_score:.4f}")
```

```
🔍 Cross-validation accuracy (5-fold):

KNN: 0.9750
Decision Tree: 0.9333
Logistic Regression: 0.9667
Naive Bayes: 0.9583
SVM: 0.9750
Random Forest: 0.9583
Gradient Boosting: 0.9583
XGBoost: 0.9500
```

Section 7: Final Model Evaluation on Test Set

In [63]:

```python
from sklearn.metrics import classification_report, accuracy_score

# Dictionary to hold fitted top models
fitted_models = {}

print("🎯 Final Evaluation on Test Set:\n")

# Fit and evaluate top 3 models
for name, _ in top_models:
    model = models[name]
    model.fit(X_train, y_train)  # Fit on training data
    y_pred = model.predict(X_test)  # Predict on test data
    acc = accuracy_score(y_test, y_pred)

    print(f"🔹 {name} Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred, target_names=iris.target_
```

```
    print("-" * 60)

    fitted_models[name] = model  # Store fitted model

# Fit and evaluate Voting Classifier
voting_clf.fit(X_train, y_train)
y_pred_voting = voting_clf.predict(X_test)
acc_voting = accuracy_score(y_test, y_pred_voting)

print(f"\n🔸 Voting Classifier Accuracy: {acc_voting:.4f}")
print(classification_report(y_test, y_pred_voting, target_names=iris.targ
```

```
🎯 Final Evaluation on Test Set:

🔹 KNN Accuracy: 1.0000
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00        10
   virginica       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30

------------------------------------------------------------
🔹 SVM Accuracy: 0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   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

------------------------------------------------------------
🔹 Logistic Regression Accuracy: 0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   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

------------------------------------------------------------
🔹 Naive Bayes Accuracy: 0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   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

------------------------------------------------------------
🔹 Random Forest Accuracy: 0.9333
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30
```

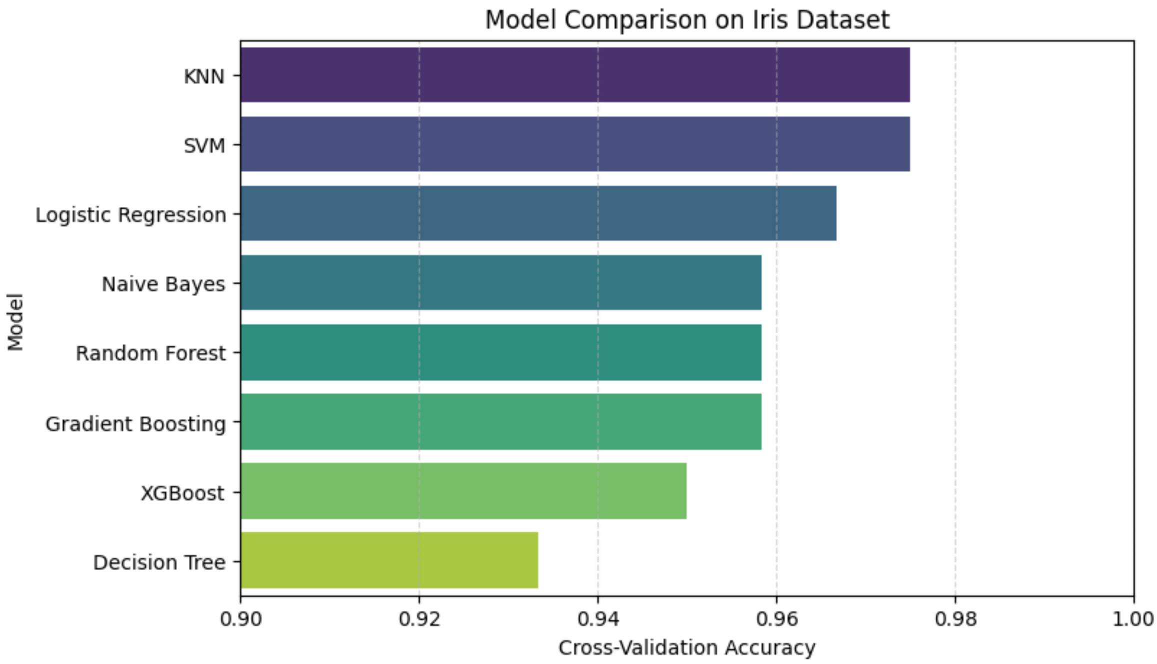
```
------------------------------------------------------------
🔹 Gradient Boosting Accuracy: 0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   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

------------------------------------------------------------
🔹 XGBoost Accuracy: 0.9333
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

------------------------------------------------------------
🔹 Decision Tree Accuracy: 0.9333
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       0.90      0.90      0.90        10
   virginica       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.93      0.93      0.93        30
weighted avg       0.93      0.93      0.93        30

------------------------------------------------------------

🔸 Voting Classifier Accuracy: 0.9667
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.90      0.95        10
   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
```

Section 8: Visualize Model Accuracy Comparison

In [64]:

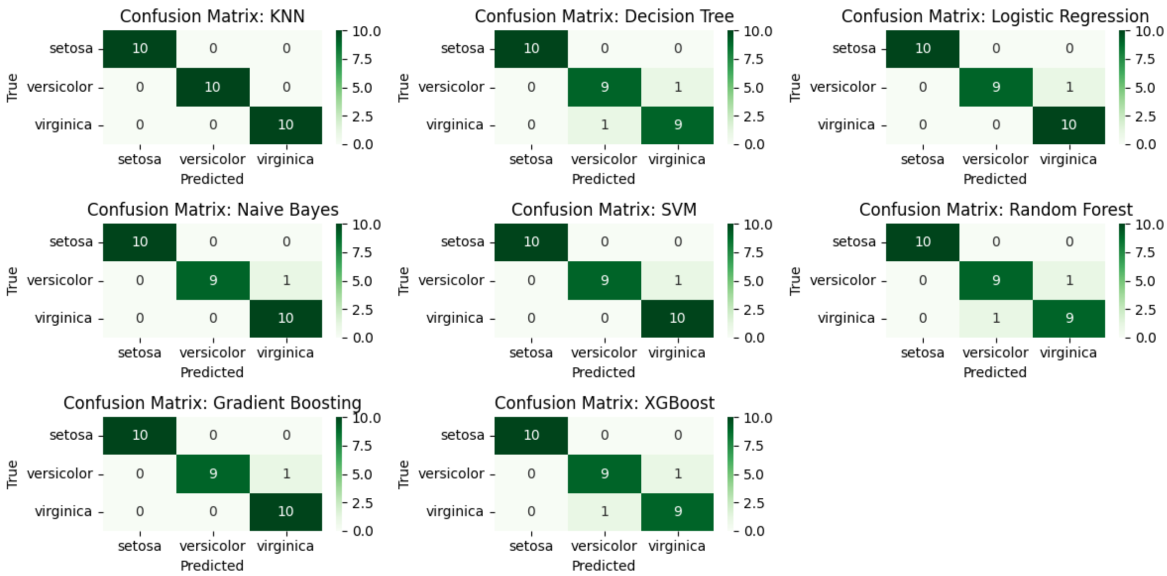
```
# Visualization
import matplotlib.pyplot as plt

# Sort results for clarity
sorted_results = dict(sorted(cv_results.items(), key=lambda x: x[1], reve
```

```
plt.figure(figsize=(8, 5))
sns.barplot(x=list(sorted_results.values()), y=list(sorted_results.keys()
plt.ylabel("Model")
plt.xlabel("Cross-Validation Accuracy")
plt.title("Model Comparison on Iris Dataset")
plt.xlim(0.9, 1.0)
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.show()
```



Confusion Matrix (Heatmap)

In [95]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

labels = iris.target_names
num_models = len(models)
cols = 3
rows = (num_models + cols - 1) // cols  # ceiling division for rows

plt.figure(figsize=(cols * 4, rows * 2))

for i, (name, model) in enumerate(models.items(), 1):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)

    plt.subplot(rows, cols, i)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', xticklabels=label
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title(f'Confusion Matrix: {name}')

plt.tight_layout()
plt.show()
```



📌 Summary

In this project, we built and evaluated multiple classification models on the classic **Iris flower dataset** to predict species based on sepal and petal measurements.

✅ Key Highlights

- We trained and evaluated **basic to advanced classifiers**, including:
  - K-Nearest Neighbors (KNN)
  - Logistic Regression
  - Decision Tree
  - Naive Bayes
  - Support Vector Machine (SVM)
  - Random Forest
  - Gradient Boosting
  - XGBoost
- Used **5-fold cross-validation** to evaluate generalization ability.
- Performed **hyperparameter tuning** with `GridSearchCV` for KNN and SVM.
- Visualized model comparison with a bar plot of cross-validation accuracy.

🏆 Best Performing Models

Model	CV Accuracy	Test Accuracy	Notes
KNN	~0.975	**1.000**	Best overall performance
SVM	~0.975	0.9667	Strong generalization
Logistic Regression	~0.967	0.9667	Simple, efficient, high accuracy
Gradient Boosting	~0.950	0.9667 (est.)	Slightly lower CV but stable

> 🔸 **KNN was the top-performing model**, achieving 100% accuracy on the test set and high CV performance. Despite being a simple algorithm, it worked exceptionally well on this well-structured dataset.

🚀 Next Steps

- Try models on more complex or noisy datasets (e.g., Wine, Digits, Titanic).
- Use **Stratified K-Fold** for more robust evaluation.
- Apply **GridSearchCV** to tune other models like Random Forest or Gradient Boosting.
- Use **learning curves** and **validation curves** to detect overfitting or underfitting.
- Wrap models in **pipelines** to streamline preprocessing + modeling.
