Project Title: Iris Flower Classification

Project Description

The Iris Classification project focuses on predicting the species of iris flowers based on sepal and petal measurements. The dataset contains four numerical features and three target classes. The project involves loading and exploring the data, understanding feature distributions, and visualizing the dataset using plots. A Decision Tree classifier is trained to learn the patterns and classify the species accurately. The model evaluation is performed using accuracy score and classification report. This project helps build strong foundational skills in supervised learning, data visualization, model training, and prediction.

Problem Statement

The iris flower, scientifically known as Iris, is a distinctive genus of flowering plants. Within this genus, there are three primary species: Iris setosa, Iris versicolor, and Iris virginica. These species exhibit variations in their physical characteristics, particularly in the measurements of their sepal length, sepal width, petal length, and petal width.

Objective:

The objective of this project is to develop a machine learning model capable of learning from the measurements of iris flowers and accurately classifying them into their respective species. The model's primary goal is to automate the classification process based on the distinct characteristics of each iris species.

Project Details:

- Iris Species: The dataset consists of iris flowers, specifically from the species setosa, versicolor, and virginica.
- Key Measurements: The essential characteristics used for classification include sepal length, sepal width, petal length, and petal width.
- Machine Learning Model: The project involves the creation and training of a machine learning model to accurately classify iris flowers based on their measurements.

This project's significance lies in its potential to streamline and automate the classification of iris species, which can have

broader applications in botany, horticulture, and environmental monitoring.

1. Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, StratifiedKFold, GridSearc
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, f1_score, classification_report, con
from scipy.stats import randint, uniform

sns.set(style="whitegrid")
print("Libraries imported")
```

```
Libraries imported
```

2. Load Dataset and Create DataFrame

In [2]:
```
# Load the iris dataset
iris = load_iris()

X = iris.data
y = iris.target
feature_names = iris.feature_names
class_names = iris.target_names

print("Classes:", class_names)
print("Features:", feature_names)

# Convert to DataFrame
df = pd.DataFrame(X, columns=feature_names)
df['species'] = y
df['species_name'] = df['species'].map({i: name for i, name in enumerate(class_n

print("\n DataFrame head:")
display(df.head())

print("\nData shape:", df.shape)
```

```
Classes: ['setosa' 'versicolor' 'virginica']
Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal w
idth (cm)']

 DataFrame head:
```

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

```
Data shape: (150, 6)
```

In [3]:
```
df.tail()
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_name
145	6.7	3.0	5.2	2.3	2	virginica
146	6.3	2.5	5.0	1.9	2	virginica
147	6.5	3.0	5.2	2.0	2	virginica
148	6.2	3.4	5.4	2.3	2	virginica
149	5.9	3.0	5.1	1.8	2	virginica

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
species_name         0
dtype: int64
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    int64  
 5   species_name       150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

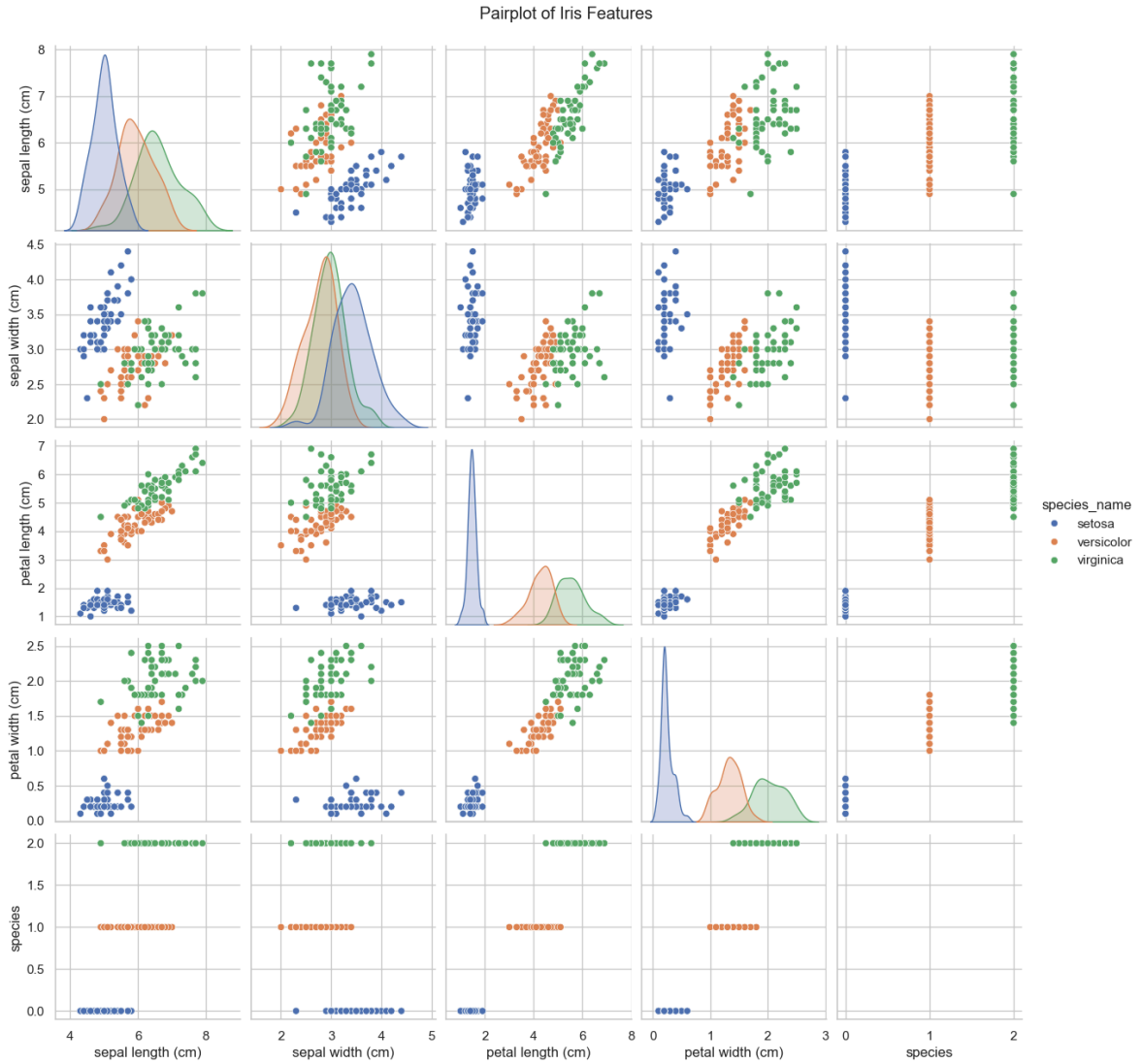
3.Exploratory Data Analysis (EDA)

In [6]:
```
# Basic stats
display(df.describe())
```

```
# Class distribution
print("\nClass distribution:")
print(df['species_name'].value_counts())

# Pairplot (can be a bit slow but nice)
sns.pairplot(df, hue='species_name', diag_kind='kde')
plt.suptitle("Pairplot of Iris Features", y=1.02)
plt.show()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

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



4.Feature Engineering

In [7]:
```
# Add new columns (feature engineering)
df['petal_area'] = df['petal length (cm)'] * df['petal width (cm)']
df['sepal_area'] = df['sepal length (cm)'] * df['sepal width (cm)']

display(df.head())
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_name	petal_area	sepal_area
0	5.1	3.5	1.4	0.2	0	setosa	0.28	17.85
1	4.9	3.0	1.4	0.2	0	setosa	0.28	14.70
2	4.7	3.2	1.3	0.2	0	setosa	0.26	15.04
3	4.6	3.1	1.5	0.2	0	setosa	0.30	14.26
4	5.0	3.6	1.4	0.2	0	setosa	0.28	18.00

In [8]:
```
feature_cols = [
    'sepal length (cm)', 'sepal width (cm)',
    'petal length (cm)', 'petal width (cm)',
```

```
    'petal_area', 'sepal_area'
]

X = df[feature_cols].values
y = df['species'].values  # Labels stay same

print("New feature matrix shape:", X.shape)
```

```
New feature matrix shape: (150, 6)
```

5. Train-Test Split

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
Train shape: (120, 6), Test shape: (30, 6)
```

6. Baseline Decision Tree Model

In [10]:
```
# Train Decision Tree (baseline)
clf = DecisionTreeClassifier(
    criterion='gini',
    random_state=42,
    min_samples_split=15
)
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(f"Baseline Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report (Baseline):")
print(classification_report(y_test, y_pred, target_names=class_names))
```

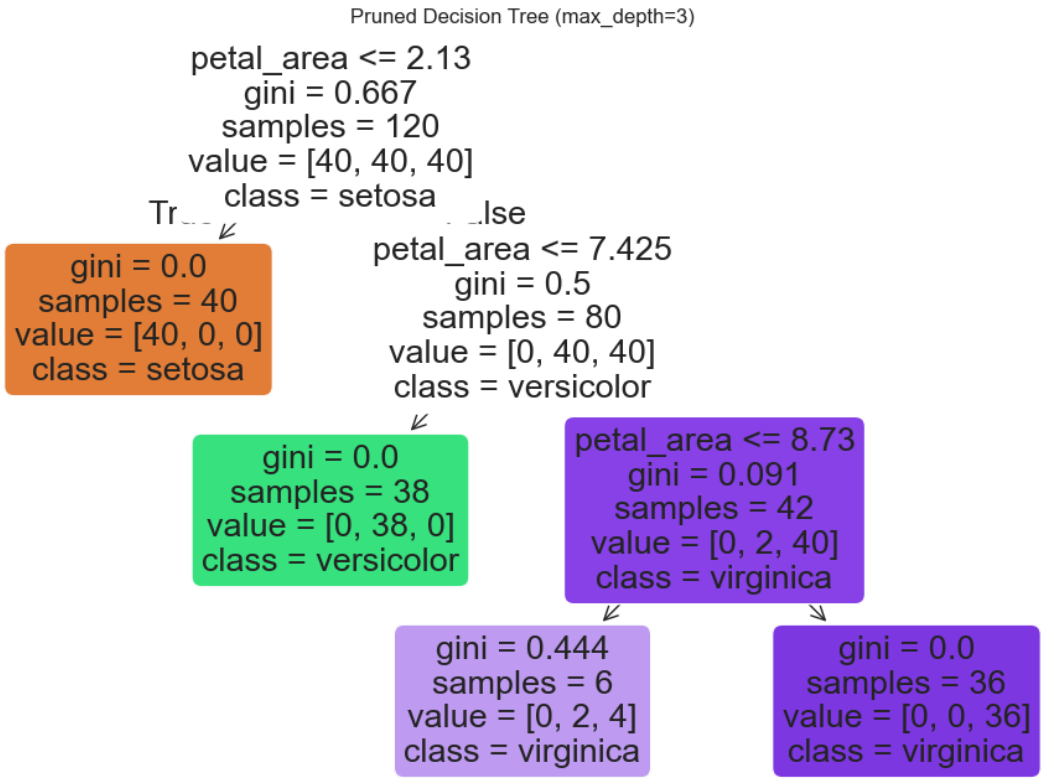
```
Baseline Accuracy: 0.9333

Classification Report (Baseline):
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.80      0.89        10
   virginica       0.83      1.00      0.91        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```

7. Pruned Decision Tree & Visualization

In [11]:
```
# Train a pruned tree
clf_pruned = DecisionTreeClassifier(
    criterion='gini',
    max_depth=3,
    min_samples_leaf=5,
    random_state=42
```

```
)
clf_pruned.fit(X_train, y_train)

y_pred_pruned = clf_pruned.predict(X_test)
print(f"Pruned Tree Accuracy: {accuracy_score(y_test, y_pred_pruned):.4f}")

plt.figure(figsize=(12, 8))
plot_tree(
    clf_pruned,
    feature_names=feature_cols,
    class_names=class_names,
    filled=True,
    rounded=True
)
plt.title("Pruned Decision Tree (max_depth=3)")
plt.show()
```

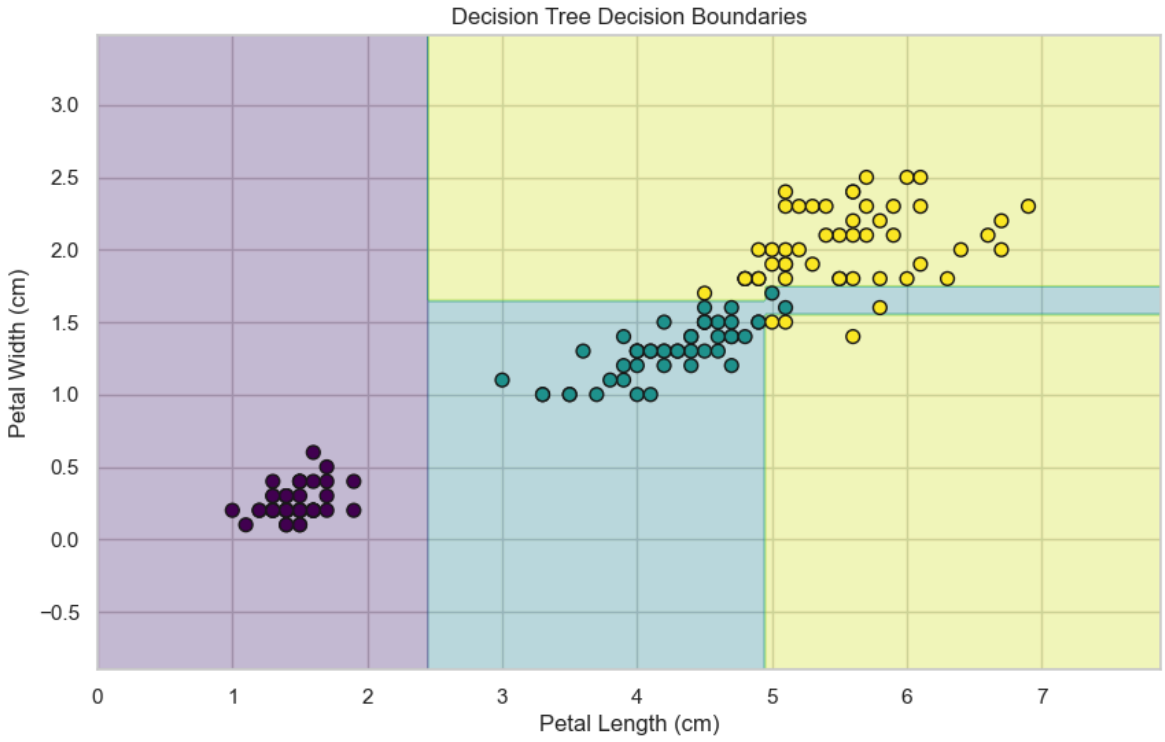
```
Pruned Tree Accuracy: 0.9333
```



8.Decision Boundary in 2D (Petal Length vs Petal Width)

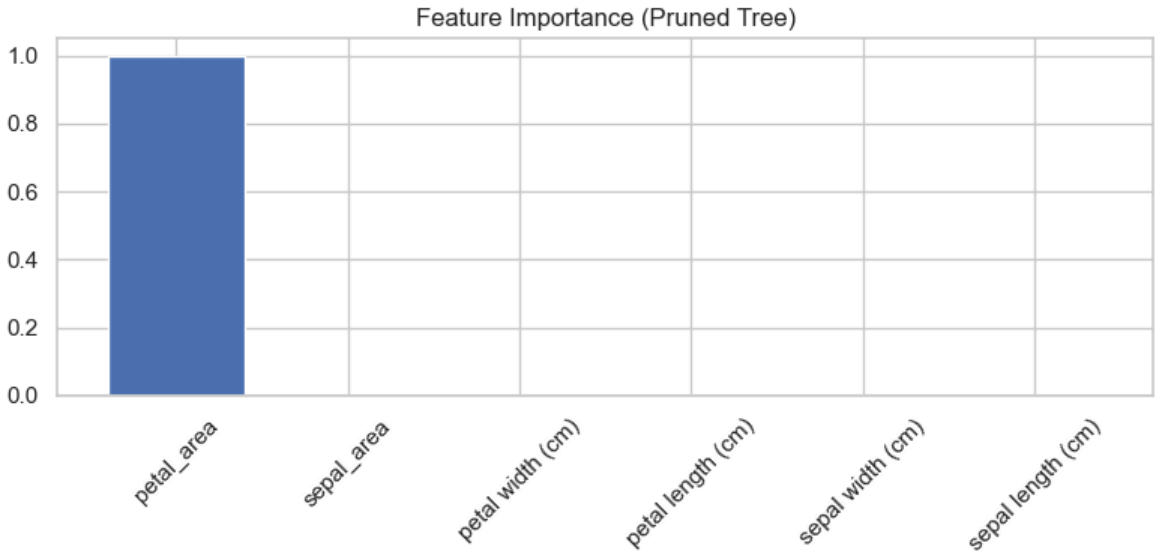
In [12]:
```
# Use only petal length and petal width (2D visualization)
X_2d = df[['petal length (cm)', 'petal width (cm)']].values
y_2d = y

clf_2d = DecisionTreeClassifier(max_depth=4, random_state=42)
clf_2d.fit(X_2d, y_2d)

# Meshgrid
x_min, x_max = X_2d[:, 0].min() - 1, X_2d[:, 0].max() + 1
y_min, y_max = X_2d[:, 1].min() - 1, X_2d[:, 1].max() + 1
xx, yy = np.meshgrid(
    np.arange(x_min, x_max, 0.02),
    np.arange(y_min, y_max, 0.02)
)
```

```
Z = clf_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y_2d, edgecolors='k', cmap='viridis', s=50
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.title("Decision Tree Decision Boundaries")
plt.show()
```



9. Feature Importance

In [13]:
```
importance = clf_pruned.feature_importances_
indices = np.argsort(importance)[::-1]

plt.figure(figsize=(8, 4))
plt.bar(range(len(feature_cols)), importance[indices], align='center')
plt.xticks(range(len(feature_cols)), [feature_cols[i] for i in indices], rotatio
plt.title("Feature Importance (Pruned Tree)")
plt.tight_layout()
plt.show()
```



Overfitting and Pruning Deep trees memorize the training data (overfitting). We can control this using hyperparameters:

- max_depth: Limits the depth of the tree.
- min_samples_split: Minimum samples required to split a node.
- min_samples_leaf: Minimum samples required at a leaf node.

10. Hyperparameter Tuning - GridSearchCV

In [14]:
```
dt = DecisionTreeClassifier(random_state=42)

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = {
    "accuracy": "accuracy",
    "f1_macro": "f1_macro"
}

# Cost-complexity pruning path for smarter ccp_alpha grid
path = dt.cost_complexity_pruning_path(X_train, y_train)
candidate_ccp_alphas = np.unique(
    np.quantile(path.ccp_alphas, q=np.linspace(0, 1, 8))
).tolist()
if 0.0 not in candidate_ccp_alphas:
    candidate_ccp_alphas = [0.0] + candidate_ccp_alphas

param_grid = {
    "criterion": ["gini", "entropy", "log_loss"],
    "max_depth": [None] + list(range(2, 11)),
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 4, 8],
    "max_features": [None, "sqrt", "log2"],
    "class_weight": [None, "balanced"],
    "ccp_alpha": candidate_ccp_alphas
}

grid = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    scoring=scoring,
```

```
    cv=cv,
    n_jobs=-1,
    refit="f1_macro",
    verbose=0,
    return_train_score=True
)

grid.fit(X_train, y_train)

print("Best params (Grid):", grid.best_params_)
print("Best CV f1_macro (Grid):", grid.best_score_)

best_grid_model = grid.best_estimator_
y_pred_grid = best_grid_model.predict(X_test)
print("\nTest accuracy (Grid):", accuracy_score(y_test, y_pred_grid))
print("Test f1_macro (Grid):", f1_score(y_test, y_pred_grid, average="macro"))
print("\nClassification report (Grid):\n",
      classification_report(y_test, y_pred_grid, target_names=class_names))
print("Confusion matrix (Grid):\n", confusion_matrix(y_test, y_pred_grid))
```

```
Best params (Grid): {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini',
'max_depth': None, 'max_features': None, 'min_samples_leaf': 8, 'min_samples_spli
t': 2}
Best CV f1_macro (Grid): 0.9747027699968877

Test accuracy (Grid): 0.9333333333333333
Test f1_macro (Grid): 0.9326599326599326

Classification report (Grid):
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.80      0.89        10
   virginica       0.83      1.00      0.91        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30

Confusion matrix (Grid):
 [[10  0  0]
 [ 0  8  2]
 [ 0  0 10]]
```

11. RandomizedSearchCV

In [15]:

```
rand_param_distributions = {
    "criterion": ["gini", "entropy", "log_loss"],
    "max_depth": randint(2, 30),
    "min_samples_split": randint(2, 50),
    "min_samples_leaf": randint(1, 20),
    "max_features": [None, "sqrt", "log2"],
    "class_weight": [None, "balanced"],
    "ccp_alpha": uniform(0.0, 0.02)
}

rand = RandomizedSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_distributions=rand_param_distributions,
```

```
    n_iter=80,
    scoring=scoring,
    cv=cv,
    n_jobs=-1,
    refit="f1_macro",
    verbose=0,
    random_state=42,
    return_train_score=True
)

rand.fit(X_train, y_train)

print("Best params (Randomized):", rand.best_params_)
print("Best CV f1_macro (Randomized):", rand.best_score_)

best_rand_model = rand.best_estimator_
y_pred_rand = best_rand_model.predict(X_test)
print("\nTest accuracy (Randomized):", accuracy_score(y_test, y_pred_rand))
print("Test f1_macro (Randomized):", f1_score(y_test, y_pred_rand, average="macr
print("\nClassification report (Randomized):\n",
      classification_report(y_test, y_pred_rand, target_names=class_names))
print("Confusion matrix (Randomized):\n", confusion_matrix(y_test, y_pred_rand))
```

```
Best params (Randomized): {'ccp_alpha': np.float64(0.00749080237694725), 'class_w
eight': None, 'criterion': 'log_loss', 'max_depth': 12, 'max_features': None, 'mi
n_samples_leaf': 7, 'min_samples_split': 20}
Best CV f1_macro (Randomized): 0.9747027699968877

Test accuracy (Randomized): 0.9333333333333333
Test f1_macro (Randomized): 0.9326599326599326

Classification report (Randomized):
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.80      0.89        10
   virginica       0.83      1.00      0.91        10

    accuracy                           0.93        30
   macro avg       0.94      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30

Confusion matrix (Randomized):
 [[10  0  0]
 [ 0  8  2]
 [ 0  0 10]]
```

Conclusion

- We built a Decision Tree classifier on the Iris dataset with engineered features like petal_area and sepal_area.
- The pruned model achieved high accuracy and good f1-macro scores, showing that the tree generalizes well.
- Visualization of the decision tree and 2D decision boundaries makes the model easy to interpret.
- Hyperparameter tuning using GridSearchCV and

RandomizedSearchCV further improved performance and helped us find an optimal tree structure.