```
import numpy as np
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.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.decomposition import PCA

# Load data
iris = load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA for visualization
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Dataset shape:", X.shape)
print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```



```
# Create figure with subplots
fig, axes = plt.subplots(2, 2, figure=figsize=(12, 10))

# Plot 1: Sepal length vs sepal width
colors = ['red', 'green', 'blue']
for i, target_name in enumerate(target_names):
    mask = y_train == i
    axes[0, 0].scatter(X_train[mask, 0], X_train[mask, 1],
                       c=colors[i], label=target_name, alpha=0.7)
axes[0, 0].set_xlabel(feature_names[0])
axes[0, 0].set_ylabel(feature_names[1])
axes[0, 0].set_title('Sepal length vs sepal width')
axes[0, 0].legend()

# Plot 2: Petal length vs petal width
for i, target_name in enumerate(target_names):
    mask = y_train == i
    axes[0, 1].scatter(X_train[mask, 2], X_train[mask, 3],
                       c=colors[i], label=target_name, alpha=0.7)
axes[0, 1].set_xlabel(feature_names[2])
axes[0, 1].set_ylabel(feature_names[3])
axes[0, 1].set_title('Petal length vs petal width')
axes[0, 1].legend()

# Plot 3: PCA visualization
for i, target_name in enumerate(target_names):
    mask = y_train == i
    axes[1, 0].scatter(X_train_pca[mask, 0], X_train_pca[mask, 1],
                       c=colors[i], label=target_name, alpha=0.7)
axes[1, 0].set_xlabel('PC1')
axes[1, 0].set_ylabel('PC2')
axes[1, 0].set_title('PCA visualization')
axes[1, 0].legend()

# Plot 4: Feature distributions
for i, feature_name in enumerate(feature_names):
    axes[1, 1].hist(X_train[:, i], bins=20, alpha=0.7, label=feature_name)
axes[1, 1].set_xlabel('Feature value')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_title('Feature distributions')
axes[1, 1].legend()

plt.tight_layout()
plt.savefig('iris_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

# Print statistics
print("\n=== Dataset Statistics ===")
print("Mean values:")
for i, feature_name in enumerate(feature_names):
    print(f{feature_name}: {X_train[:, i].mean():.2f}")
print("\nStandard deviations:")
for i, feature_name in enumerate(feature_names):
    print(f{feature_name}: {X_train[:, i].std():.2f}")
```



```
# Train Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)

# Predictions
y_pred_dt = dt.predict(X_test_scaled)

# Evaluation
print("\n=== Decision Tree Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("\n Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_dt))
print("\n Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=target_names))

# Feature importance
print("\n Feature Importance:")
for i, feature_name in enumerate(feature_names):
    print(f{feature_name}: {dt.feature_importances_[i]:.2f}")
```

```
=== Decision Tree Results ===
Accuracy: 1.0

 Confusion Matrix:
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]

 Classification Report:
              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

 Feature Importance:
sepal length (cm): 0.00
sepal width (cm): 0.00
petal length (cm): 0.56
petal width (cm): 0.44
```

```
# Cross-validation
cv_scores = cross_validation(dt, X_train_scaled, y_train, cv=5)
print("\n=== Cross-Validation Results ===")
print("CV Scores:", cv_scores)
print("Mean CV Score:", cv_scores.mean())
print("Standard Deviation:", cv_scores.std())

# Hyperparameter tuning
from sklearn.model_selection import GridSearchCV
param_grid = {
    'max_depth': [None, 3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42),
                           param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

print("\n=== Hyperparameter Tuning ===")
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)

# Evaluate tuned model
dt_tuned = grid_search.best_estimator_
y_pred_tuned = dt_tuned.predict(X_test_scaled)
print("\n=== Tuned Model Results ===")
print("Accuracy:", accuracy_score(y_test, y_pred_tuned))
```

```
=== Cross-Validation Results ===
CV Scores: [1. 1. 1. 1. 1.]
Mean CV Score: 1.0
Standard Deviation: 0.0

=== Hyperparameter Tuning ===
Best parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best cross-validation score: 1.0

=== Tuned Model Results ===
Accuracy: 1.0
```

```
# Save the model
import joblib
joblib.dump(dt_tuned, 'iris_decision_tree.pkl')
joblib.dump(scaler, 'iris_scaler.pkl')

# Load and test
dt_loaded = joblib.load('iris_decision_tree.pkl')
scaler_loaded = joblib.load('iris_scaler.pkl')

# Test on new data
sample = np.array([[5.1, 3.5, 1.4, 0.2]])
sample_scaled = scaler_loaded.transform(sample)
prediction = dt_loaded.predict(sample_scaled)
print("\n=== Prediction Test ===")
print("Sample:", sample)
print("Prediction:", prediction[0])
print("Predicted class:", target_names[prediction[0]])
```

```
=== Prediction Test ===
Sample: [[5.1 3.5 1.4 0.2]]
Prediction: 0
Predicted class: setosa
```

```
# Feature importance visualization
plt.figure(figure=(10, 6))
importances = dt.feature_importances_
indices = np.argsort(importances)[::-1]
plt.bar(range(len(feature_names)), importances[indices])
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.xticks(range(len(feature_names)), [feature_names[i] for i in indices], rotation=45)
plt.tight_layout()
plt.savefig('feature_importance.png', dpi=300, bbox_inches='tight')
plt.show
```



```
# Decision boundary visualization (2D)
from matplotlib import ListedColormap
fig, ax = plt.subplots(figure=(8, 6))
# Create mesh
h = .02
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
# Predict on mesh
Z = dt.predict(scaler.inverse_transform(pca.inverse_transform(np.c_[xx.ravel(), yy.ravel()])))
Z = Z.reshape(xx.shape)
# Plot
cmap = ListedColormap(['red', 'green', 'blue'])
ax.contourf(xx, yy, Z, cmap=cmap, alpha=0.3)
for i, target_name in enumerate(target_names):
    mask = y_train == i
    ax.scatter(X_train_pca[mask, 0], X_train_pca[mask, 1],
               c=colors[i], label=target_name, alpha=0.7)
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_title('Decision Boundary (PCA Space)')
ax.legend()
plt.savefig('decision_boundary.png', dpi=300, bbox_inches='tight')
plt.show
```



```
# Model summary
print("\n=== Model Summary ===")
print("Decision Tree Classifier")
print("Number of features:", dt.n_features_in_)
print("Number of classes:", dt.n_classes_)
print("Tree depth:", dt.tree_.max_depth)
print("Number of leaves:", dt.tree_.n_leaves)
```

```
=== Model Summary ===
Decision Tree Classifier
Number of features: 4
Number of classes: 3
Tree depth: 3
Number of leaves: 4
```