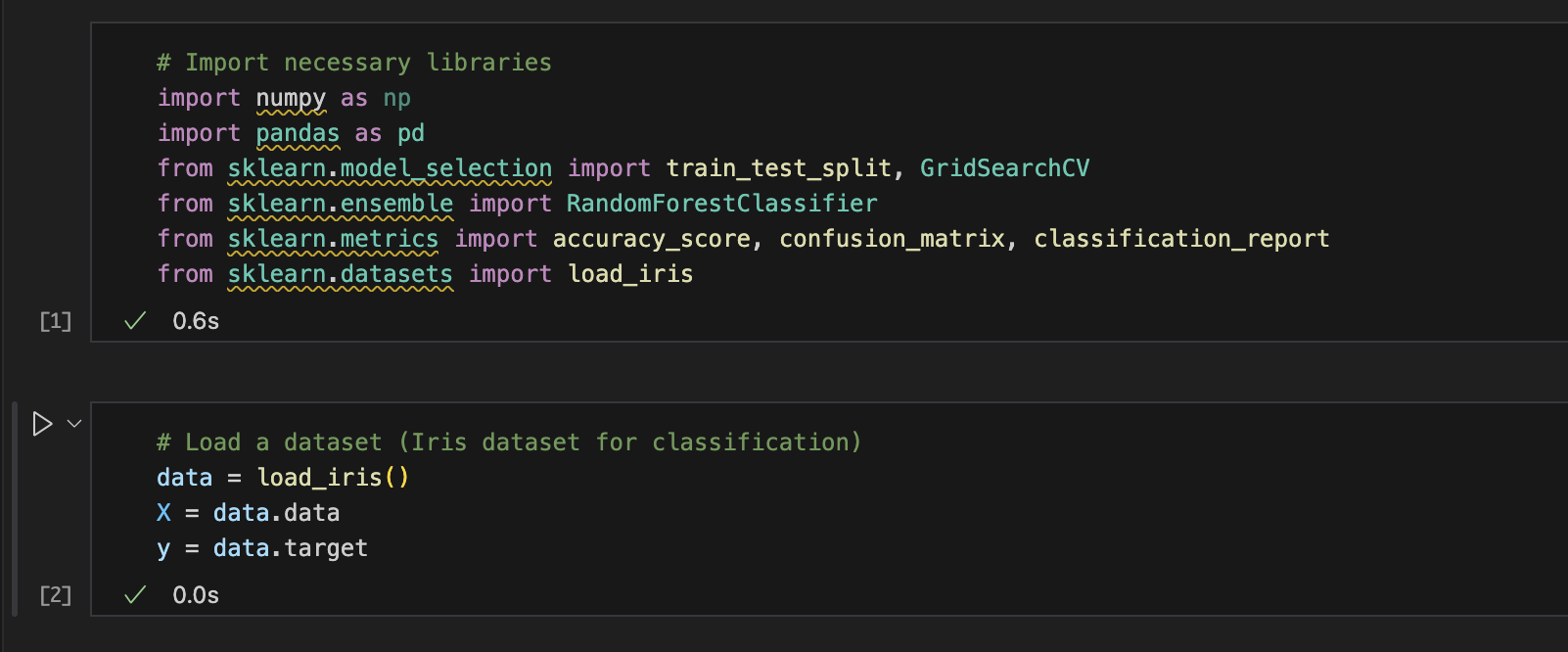
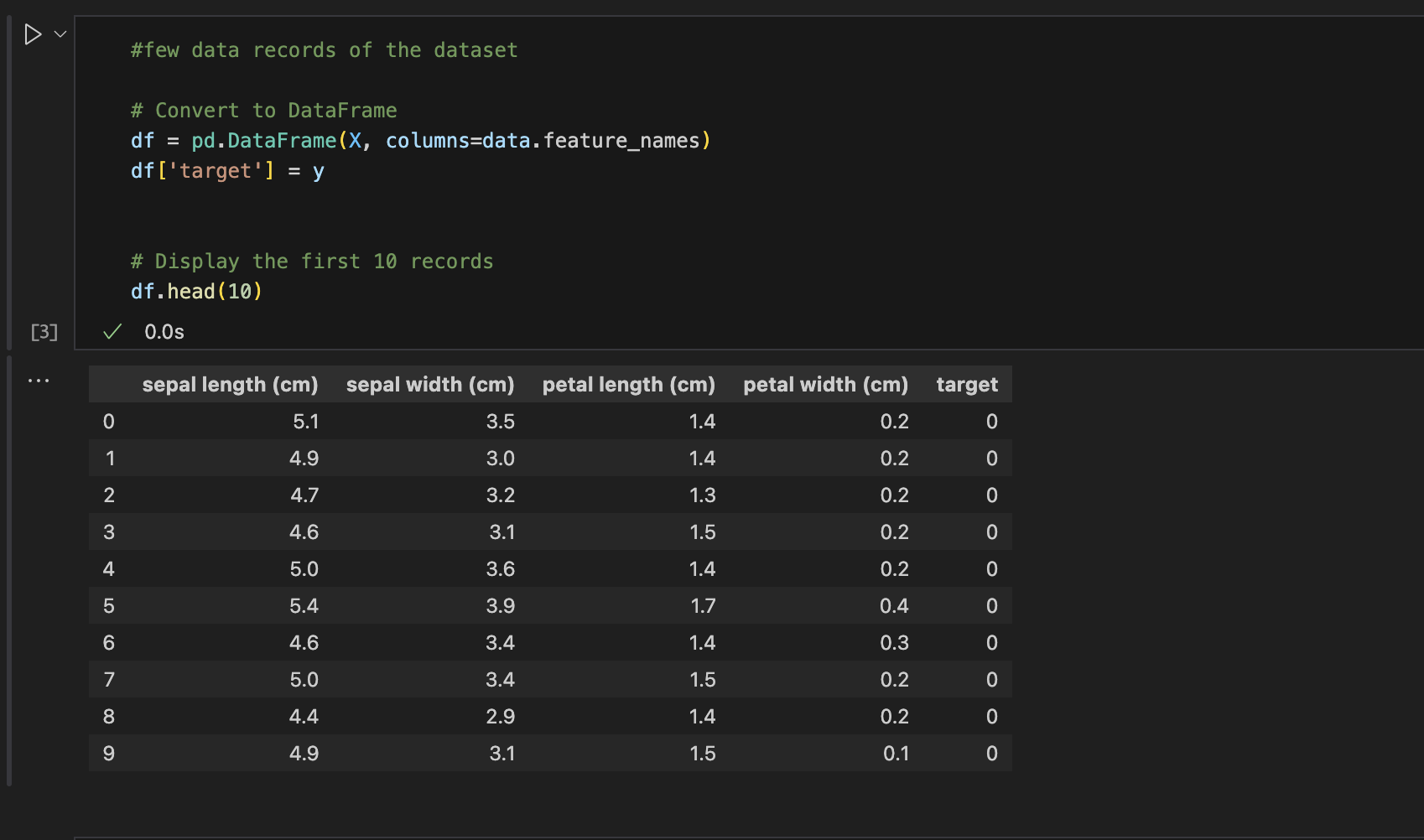
**Q2. Design and implement a Random Forest Classifier or Regressor model and explore the impact of hyperparameter tuning on model performance using scikit-learn, and evaluate its performance using various performance measures for a specific machine learning task.**

**Code:**

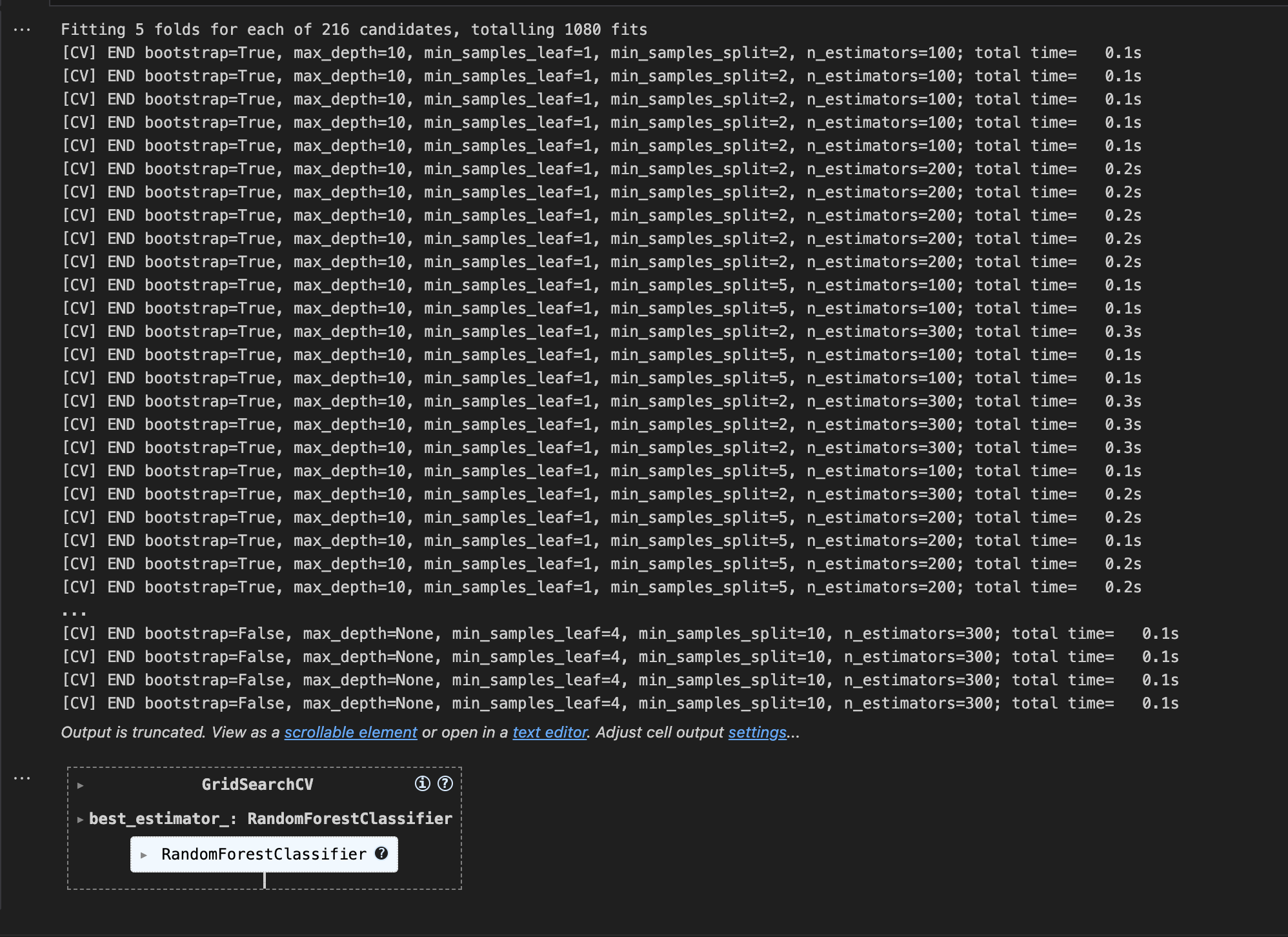
****

****

**A screen shot of a computer program

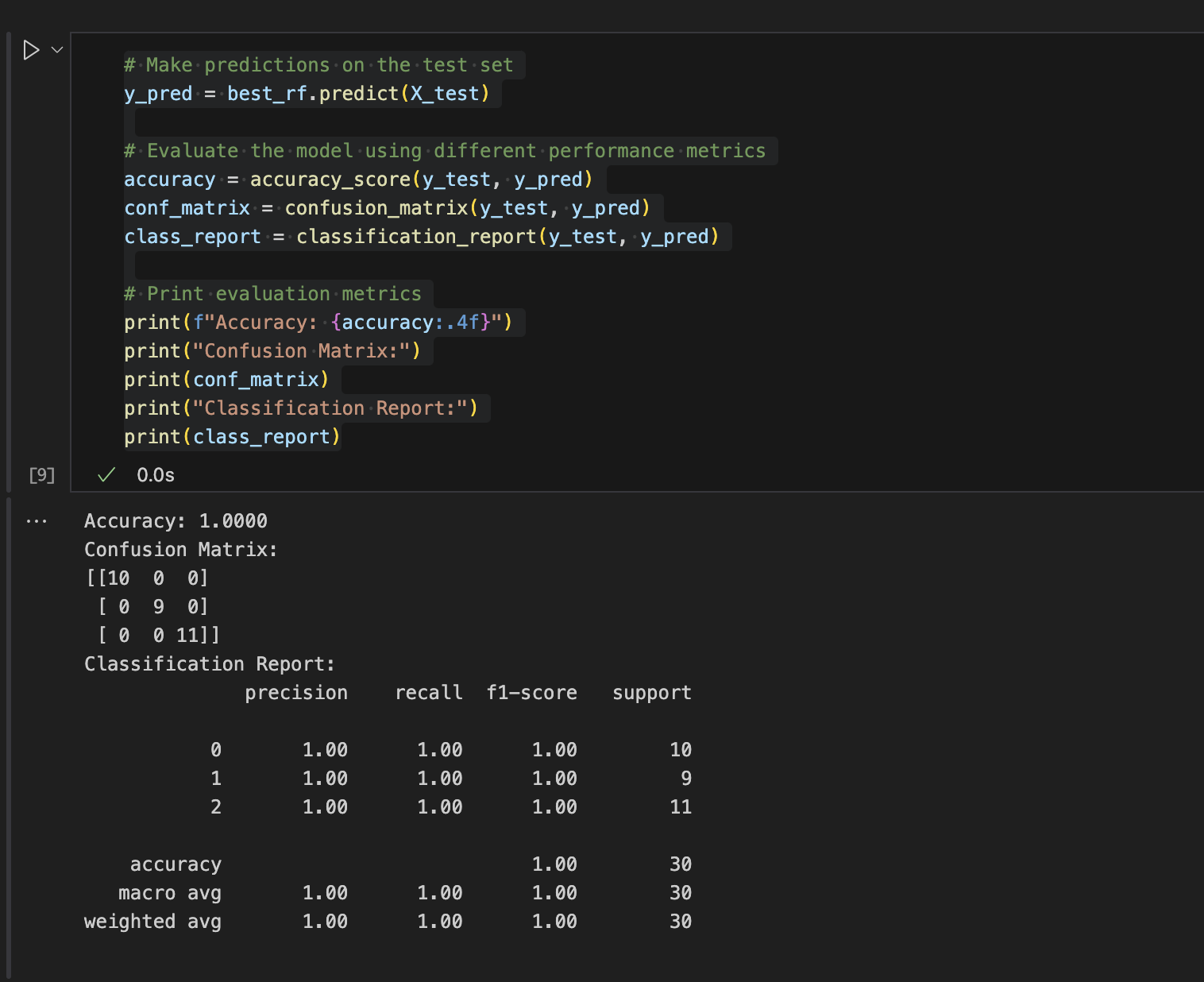
Description automatically generated**

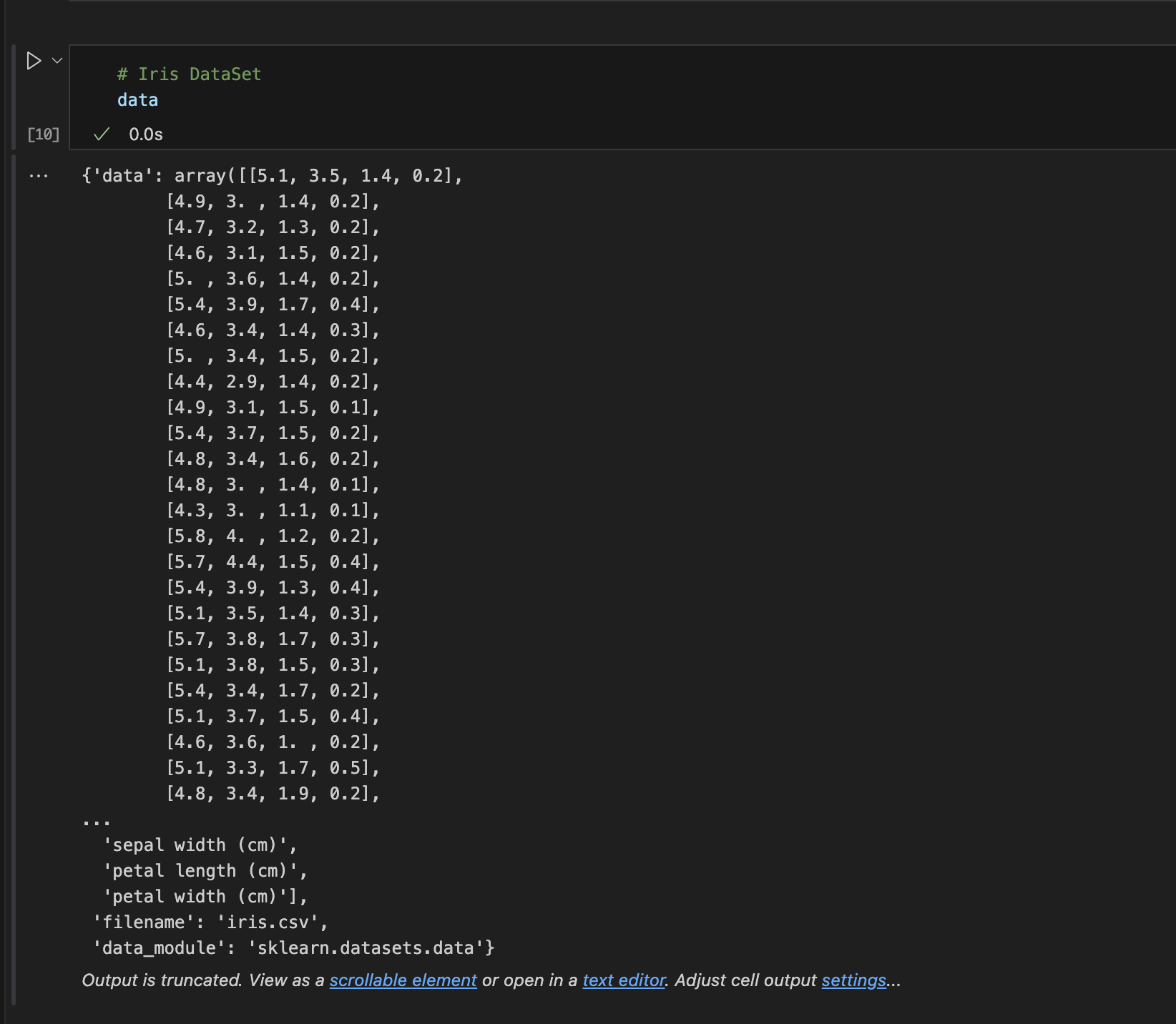
****



A screenshot of a computer

Description automatically generated





**Code As Text:**

**# Import necessary libraries**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.datasets import load\_iris

**# Load a dataset (Iris dataset for classification)**

data = load\_iris()

X = data.data

y = data.target

**#few data records of the dataset**

**# Convert to DataFrame**

df = pd.DataFrame(X, columns=data.feature\_names)

df['target'] = y

**# Display the first 10 records**

df.head(10)

**# Split the data into training and testing sets (80% train, 20% test)**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**# Initialize the Random Forest Classifier**

rf = RandomForestClassifier(random\_state=42)

**# Set up hyperparameters grid for tuning**

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [10, 20, 30, None],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'bootstrap': [True, False]

}

**# Set up the GridSearchCV to find the best hyperparameters**

grid\_search = GridSearchCV(estimator=rf, param\_grid=param\_grid, cv=5, verbose=2, n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

**# Display the best parameters found by GridSearchCV**

print("Best Hyperparameters:", grid\_search.best\_params\_)

**# Get the best estimator (model with tuned hyperparameters)**

best\_rf = grid\_search.best\_estimator\_

**# Train the model on the training data**

best\_rf.fit(X\_train, y\_train)

**# Make predictions on the test set**

y\_pred = best\_rf.predict(X\_test)

**# Evaluate the model using different performance metrics**

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

**# Print evaluation metrics**

print(f"Accuracy: {accuracy:.4f}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

