```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('p_iris.csv')
```

```
X = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].values
y = data['Species_1'].astype(int).values
```

```
def gini_impurity(y):
    if len(y) == 0:
        return 0
    prob = np.bincount(y) / len(y)
    return 1 - np.sum(prob ** 2)

# Define function to find the best split
def best_split(X, y):
    m, n = X.shape
    best_gain = 0
    best_split_value = None
    best_left_indices = None
    best_right_indices = None

    for feature_index in range(n):
        thresholds = np.unique(X[:, feature_index])
        for threshold in thresholds:
            left_indices = np.where(X[:, feature_index] <= threshold)[0]
            right_indices = np.where(X[:, feature_index] > threshold)[0]

            if len(left_indices) == 0 or len(right_indices) == 0:
                continue

            # Calculate Gini impurity for left and right
            left_impurity = gini_impurity(y[left_indices])
            right_impurity = gini_impurity(y[right_indices])

            # Calculate weighted average Gini impurity
            gain = gini_impurity(y) - (len(left_indices) / m * left_impurity + len(right_indices) / m * right_impurity)

            if gain > best_gain:
                best_gain = gain
                best_split_value = (feature_index, threshold)
                best_left_indices = left_indices
                best_right_indices = right_indices

    return best_split_value, best_left_indices, best_right_indices

# Define a Decision Tree Classifier
class DecisionTreeClassifier:
    def __init__(self, depth_limit=None):
        self.depth_limit = depth_limit
        self.tree = None

    def _build_tree(self, X, y, depth=0):
        if len(np.unique(y)) == 1 or (self.depth_limit and depth >= self.depth_limit):
            return np.bincount(y).argmax()

        split, left_indices, right_indices = best_split(X, y)
        if split is None:
            return np.bincount(y).argmax()

        left_tree = self._build_tree(X[left_indices], y[left_indices], depth + 1)
        right_tree = self._build_tree(X[right_indices], y[right_indices], depth + 1)

        return (split, left_tree, right_tree)

    def fit(self, X, y):
        self.tree = self._build_tree(X, y)

    def _predict_single(self, x, tree):
        if not isinstance(tree, tuple):
            return tree

        feature_index, threshold = tree[0]
        if x[feature_index] <= threshold:
            return self._predict_single(x, tree[1])
        else:
            return self._predict_single(x, tree[2])
```

```
    def predict(self, X):
        return np.array([self._predict_single(x, self.tree) for x in X])

# Train the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(depth_limit=5)  # Set depth limit for tree
dt_classifier.fit(X, y)

# Make predictions
y_pred = dt_classifier.predict(X)

# Evaluate accuracy
accuracy = np.mean(y_pred == y) * 100
print(f"Accuracy: {accuracy:.2f}")

# Display the predictions
print("Predictions:", y_pred)
```

```
Accuracy: 100.00
Predictions: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

```
def plot_decision_boundaries(X, y, model):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))

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

    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.xlabel('Sepal Length (cm)')
    plt.ylabel('Sepal Width (cm)')
    plt.title('Decision Boundary of Decision Tree Classifier')
    plt.show()
```

```
plot_decision_boundaries(X, y, dt_classifier)
```