```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import datasets
```

```
iris=datasets.load_iris()
X=iris.data
Y=iris.target
iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
```

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=4)
```

```
hyperParameters={
    'criterion':['gini','entropy'],
    'min_samples_split':[2,10,20],
    'min_samples_leaf':[1,5,10],
    'max_depth':[None,5,10,15]
}
```

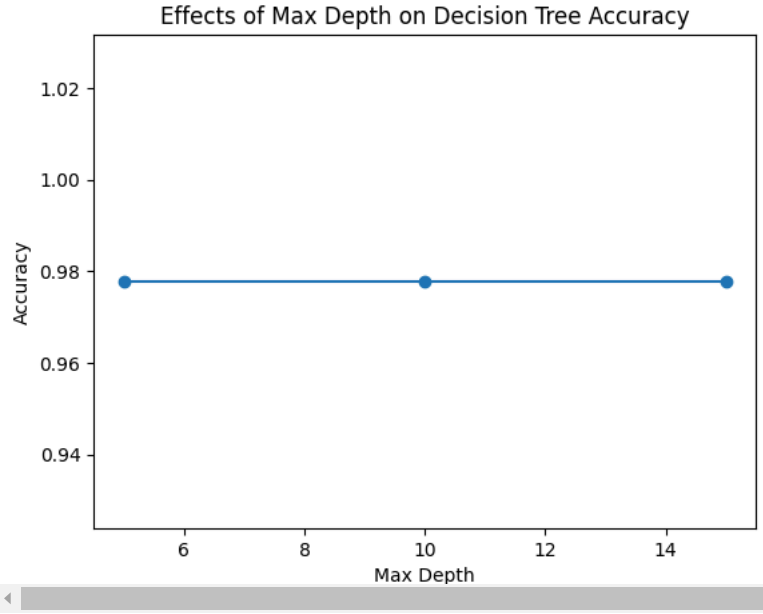
```
def evaluate_decision_tree(criterion,min_samples_split,min_samples_leaf,max_depth):
    clf=DecisionTreeClassifier(criterion=criterion,min_samples_split=min_samples_split,min_samples_leaf=min_samples_leaf,max_depth=ma
```

```
    clf.fit(X_train,Y_train)
    y_pred=clf.predict(X_test)
    accuracy=accuracy_score(Y_test,y_pred)
    return accuracy
```

```
results = []

for criterion in hyperParameters['criterion']:
    for min_samples_split in hyperParameters['min_samples_split']:
        for min_samples_leaf in hyperParameters['min_samples_leaf']:
            for max_depth in hyperParameters['max_depth']:
                accuracy = evaluate_decision_tree(criterion, min_samples_split, min_samples_leaf, max_depth)
                results.append((criterion, min_samples_split, min_samples_leaf, max_depth, accuracy))

results_df = pd.DataFrame(results, columns=['criterion', 'min_samples_split', 'min_samples_leaf', 'max_depth', 'accuracy'])

print(results_df.sort_values(by='accuracy', ascending=False).head())

```

```
    criterion  min_samples_split  min_samples_leaf  max_depth  accuracy
0        gini                  2                 1        NaN  0.977778
1        gini                  2                 1        5.0  0.977778
52    entropy                 10                 5        NaN  0.977778
51    entropy                 10                 1       15.0  0.977778
50    entropy                 10                 1       10.0  0.977778
```

```
max_depth_values=[None,5,10,15]
accuracies=[]

for max_depth in max_depth_values:
    accuracy=evaluate_decision_tree('gini',2,1,max_depth)
    accuracies.append(accuracy)

plt.plot(max_depth_values,accuracies,marker='o')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.title("Effects of Max Depth on Decision Tree Accuracy")
plt.show()
```




```
# Decision Tree Without using Sklearn
```

```
import numpy as np
import pandas as pd
from math import log2
import matplotlib.pyplot as plt
from sklearn import datasets


```

```
iris=datasets.load_iris()
x=iris.data
y=iris.target
iris
```

```
def entropy(y):
    unique_classes=np.unique(y)
    total_samples=len(y)
    entropy_value=0
    for label in unique_classes:
        p=np.sum(y==label)/total_samples
        entropy_value -= p*log2(p) if p>0 else 0
    return entropy_value

def information_gain(x,y,feature_index):
    total_entropy=entropy(y)
    values,counts=np.unique(X[:,feature_index],return_counts=True)
    weighted_entropy=0

    for value,count in zip(values,counts):
        subset= y[x[:,feature_index]==value]
        weighted_entropy+=(count/len(x))*entropy(subset)
    return total_entropy - weighted_entropy

def best_split(x,y):
    best_gain=-1
    best_feature=-1
    for feature_index in range(X.shape[1]):
        gain=information_gain(x,y,feature_index)
        if(gain>best_gain):
            best_gain=gain
            best_feature=feature_index
    return best_feature

def build_tree(X, y, depth=0, max_depth=None):
    """Recursively builds a decision tree."""
    if len(np.unique(y)) == 1:
        return np.unique(y)[0]
    if max_depth is not None and depth >= max_depth:
        return np.random.choice(np.unique(y))

    best_feature = best_split(X, y)
    tree = {best_feature: {}}

    values = np.unique(X[:, best_feature])
    for value in values:
        subset_X = X[X[:, best_feature] == value]
        subset_y = y[X[:, best_feature] == value]
        tree[best_feature][value] = build_tree(subset_X, subset_y, depth + 1, max_depth)

    return tree

def predict_tree(tree, sample):
    if not isinstance(tree, dict):
        return tree  # Leaf node
    feature = list(tree.keys())[0]
    feature_value = sample[feature]
    return predict_tree(tree[feature].get(feature_value), sample)

def accuracy_tree(tree, x, y):
    predictions = [predict_tree(tree, sample) for sample in x]
    return np.mean(predictions == y)

```

```
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
y = iris.target

df = pd.DataFrame(x, columns=iris.feature_names)
df['target'] = y

tree = build_tree(x, y, max_depth=3)

accuracy = accuracy_tree(tree, x, y)
print(f"Decision Tree Accuracy (from scratch): {accuracy * 100:.2f}%")
```

```
Decision Tree Accuracy (from scratch): 100.00%
```

```
Start coding or generate with AI.
```