Decision Tree Implementation

Types Of Decision Tree

ID3, C4.5, and CART are three popular algorithms for building decision trees. Each has its own approach to selecting splits and handling data. Here's a detailed comparison:

---

1. ID3 (Iterative Dichotomiser 3)

Characteristics:

- **Inventor**: Ross Quinlan in 1986.
- **Type**: Classification only.
- **Splitting Criterion**: **Information Gain** based on **Entropy**.
- **Handling of Data**:
  - Only deals with **categorical data** (both input features and the target variable).
  - **Cannot handle missing data**.

Process:

- **Entropy** measures the level of disorder in a dataset.
- **Information Gain** is calculated as the difference between the entropy of the parent node and the weighted entropy of child nodes.
- ID3 chooses the feature that provides the highest information gain to split the data.

Formula:

- **Entropy (H)**:
  `H(S) = − Σ p(i) ∗ log2(p(i))`
- **Information Gain (IG)**:
  `IG(S, A) = H(S) − Σ (|Sv| / |S|) ∗ H(Sv)`
  Where `S` is the set of samples and `Sv` are the subsets of `S` after splitting on feature `A`.

Advantages:

- Simple and intuitive.
- Good for smaller datasets and easy to implement.

Limitations:

- Tends to overfit on smaller datasets.
- Only works with categorical variables.
- Cannot handle missing values.

2. **C4.5**

Characteristics:

- **Inventor**: Ross Quinlan in 1993 (an improvement on ID3).
- **Type**: Primarily classification, but can be extended to regression.
- **Splitting Criterion**: **Information Gain Ratio**.
- **Handling of Data**:
  - Works with both **categorical and continuous data**.
  - **Can handle missing data** by assigning a probability to each outcome.

Process:

- C4.5 uses **Gain Ratio**, a modification of **Information Gain** that normalizes the information gain by accounting for the size of the split.
- It chooses the attribute that maximizes this gain ratio to split the data.

Formula:

- **Information Gain Ratio (GR)**:
  `GR(S, A) = IG(S, A) / Split Information(A)`
  Where `Split Information` is a measure that penalizes splits that result in many branches with small datasets.

Handling Continuous Attributes:

- Converts continuous attributes into discrete ones by setting a threshold. This threshold splits the data into two partitions: one where the values are below or equal to the threshold, and the other where values are above.

Advantages:

- Works with both categorical and continuous data.
- Handles missing data effectively.
- Deals better with **imbalanced datasets** by using the gain ratio.

Limitations:

- Computationally expensive due to sorting of continuous features.
- Still prone to **overfitting**, although it is less sensitive than ID3.

---

3. **CART (Classification and Regression Trees)**

Characteristics:

- **Inventor**: Breiman et al. in 1984.
- **Type**: Can handle both **classification** and **regression** tasks.
- **Splitting Criterion**:
  - For **classification**: **Gini Impurity**.

    - For **regression**: **Mean Squared Error (MSE)** or **Variance**.
- **Handling of Data**:
    - Works with both **categorical and continuous data**.
    - **Can handle missing data** by surrogate splitting (using another feature to split the data if the primary one is missing).

Process:

- For classification tasks, CART uses the **Gini Index** as a measure of impurity, and splits the data by selecting the feature that minimizes this index.
- For regression tasks, CART uses the **MSE** or **variance** to decide how to split the data.

Formula:

- **Gini Impurity**:

  `Gini(S) = 1 - Σ (p(i)^2)`
  Where `p(i)` is the probability of a particular class in the dataset.

- **Mean Squared Error (MSE)**:

  `MSE = Σ (Actual - Predicted)^2 / n`
  Used for regression tasks.

Advantages:

- Works with both classification and regression tasks.
- Handles continuous and categorical variables effectively.
- Pruning techniques can be applied to reduce overfitting.
- Binary tree (splits each node into exactly two branches), which simplifies decision making.

Limitations:

- Tends to overfit the data if pruning is not used.
- Prone to **instability**, meaning small changes in the data can lead to a different tree structure.

Comparison Summary

Algorithm	Type	Splitting Criterion	Data Handling	Handles Missing Data	Overfitting Handling	Use Cas
ID3	Classification	Information Gain (Entropy)	Categorical Only	No	Prone to overfitting	Basic, sm scale classificat
C4.5	Classification	Information Gain Ratio	Both Categorical	Yes	Less overfitting	Advanced classificat

Algorithm	Type	Splitting Criterion	Data Handling	Handles Missing Data	Overfitting Handling	Use Cas
			& Continuous			large datasets
CART	Classification & Regression	Gini Index (Classification) / MSE (Regression)	Both Categorical & Continuous	Yes	Can prune to reduce overfitting	General-purpose, classificat and regressio

Summary:

- **ID3** is a simple algorithm that works well for smaller datasets but only with categorical data. It tends to overfit without pruning.
- **C4.5** improves on ID3 by handling continuous data and normalizing information gain with gain ratio. It can handle missing values and is more robust.
- **CART** is highly versatile as it supports both classification and regression. It uses Gini impurity for classification and MSE for regression, and it can handle both types of data effectively.

Each algorithm has its strengths and use cases depending on the complexity of the data and the problem being solved.

Lets' start Decision Tree implementation

Import all the required packages

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

Read the dataset

In [3]:
```
dataframe = pd.read_csv('./WineQT.csv')
dataframe.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulp
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	

In [4]:
```
dataframe.columns
```

Out[4]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sug
ar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'dens
ity',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [5]:
```
set(dataframe['quality'])
```

Out[5]:
```
{3, 4, 5, 6, 7, 8}
```

Note: EDA is skipped in this part of the implementation

Splitting the data into X(independent features) and y(target column)

In [6]:
```
X = dataframe.drop(columns='quality', axis=1)
y = dataframe['quality']
```

In [7]:
```
X.columns
```

Out[7]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sug
ar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'dens
ity',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

Split the data into training and testing set

In [8]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

Data Modeling

In [9]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

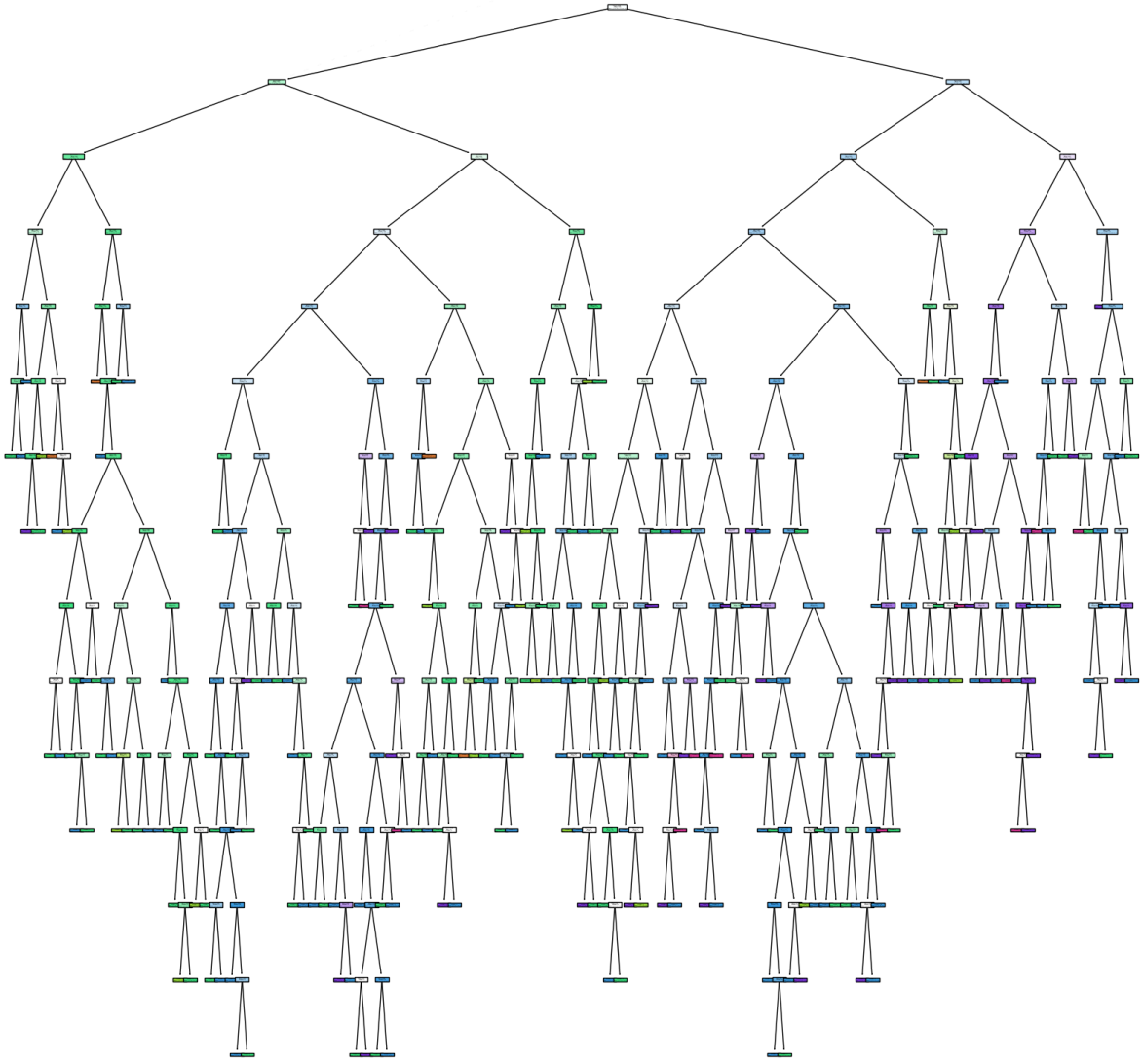
Model Predictions

In [10]:
```
y_pred = model.predict(X_test)
```

In [11]:
```
y_pred
```

Out[11]:
```
array([5, 6, 6, 6, 7, 6, 5, 6, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 6, 6, 6, 6,
       7, 5, 5, 5, 6, 6, 5, 7, 6, 7, 6, 6, 6, 6, 5, 6, 7, 6, 5, 7, 6, 6,
       5, 6, 5, 5, 5, 5, 7, 5, 6, 5, 6, 5, 5, 6, 7, 6, 5, 5, 6, 6, 7, 5,
       5, 6, 5, 6, 6, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 6, 6, 5, 5, 5, 6, 5,
       5, 6, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 4, 7, 6, 5, 5, 6, 6, 5, 5, 5,
       7, 5, 5, 8, 8, 5, 6, 5, 5, 4, 5, 6, 5, 7, 7, 5, 7, 6, 6, 5, 5, 3,
       8, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 7, 6, 6, 6, 6, 7, 6, 6, 5, 6,
       5, 5, 6, 6, 6, 5, 5, 5, 4, 5, 6, 6, 6, 5, 5, 5, 7, 5, 6, 5, 6, 7,
       6, 5, 6, 5, 4, 6, 6, 5, 6, 5, 6, 6, 6, 6, 7, 5, 6, 6, 5, 5, 5, 5,
       6, 7, 5, 5, 7, 6, 6, 5, 6, 6, 6, 5, 5, 6, 7, 6, 6, 5, 5, 5, 6, 5,
       5, 7, 6, 6, 6, 5, 5, 6, 6, 7, 5, 5, 3, 7, 3, 5, 5, 6, 5, 5, 6, 7,
       7, 7, 6, 5, 6, 6, 5, 6, 6, 5, 6, 4, 7, 7, 6, 6, 6, 6, 5, 7, 7, 6,
       5, 5, 5, 5, 5, 5, 7, 6, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 7, 6,
       5])
```

In [21]:
```
from sklearn import tree
plt.figure(figsize=(20, 20))
tree.plot_tree(model, filled=True)
plt.savefig('decistion_tree.png')
```



Top 20 sample data points from the dataframe

In [13]:
```
sampleData = dataframe.head(20)
sampleData
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sul
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	
6	7.9	0.600	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	
7	7.3	0.650	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	
8	7.8	0.580	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	
9	6.7	0.580	0.08	1.8	0.097	15.0	65.0	0.9959	3.28	
10	5.6	0.615	0.00	1.6	0.089	16.0	59.0	0.9943	3.58	
11	7.8	0.610	0.29	1.6	0.114	9.0	29.0	0.9974	3.26	
12	8.5	0.280	0.56	1.8	0.092	35.0	103.0	0.9969	3.30	
13	7.9	0.320	0.51	1.8	0.341	17.0	56.0	0.9969	3.04	
14	7.6	0.390	0.31	2.3	0.082	23.0	71.0	0.9982	3.52	
15	7.9	0.430	0.21	1.6	0.106	10.0	37.0	0.9966	3.17	
16	8.5	0.490	0.11	2.3	0.084	9.0	67.0	0.9968	3.17	
17	6.9	0.400	0.14	2.4	0.085	21.0	40.0	0.9968	3.43	
18	6.3	0.390	0.16	1.4	0.080	11.0	23.0	0.9955	3.34	
19	7.6	0.410	0.24	1.8	0.080	4.0	11.0	0.9962	3.28	

In [22]:
```
sampleData.describe()
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sul dioxide
count	20.00000	20.000000	20.000000	20.00000	20.000000	20.000000	20.0000
mean	7.66500	0.536250	0.164500	1.88000	0.097450	15.050000	47.4000
std	1.07912	0.168342	0.191572	0.35333	0.058655	6.939399	22.5584
min	5.60000	0.280000	0.000000	1.20000	0.065000	4.000000	11.0000
25%	7.37500	0.397500	0.000000	1.60000	0.075750	10.750000	32.7500
50%	7.70000	0.580000	0.095000	1.80000	0.083000	15.000000	47.0000
75%	7.90000	0.652500	0.252500	2.07500	0.093250	17.000000	61.2500
max	11.20000	0.880000	0.560000	2.60000	0.341000	35.000000	103.0000

In [14]:
```
X_sample = sampleData.drop(columns='quality', axis=1)
y_sample = sampleData['quality']
```

In [15]:
```
sampleModel = DecisionTreeClassifier()
sampleModel.fit(X_sample, y_sample)
```

Out[15]:
```
DecisionTreeClassifier()
```

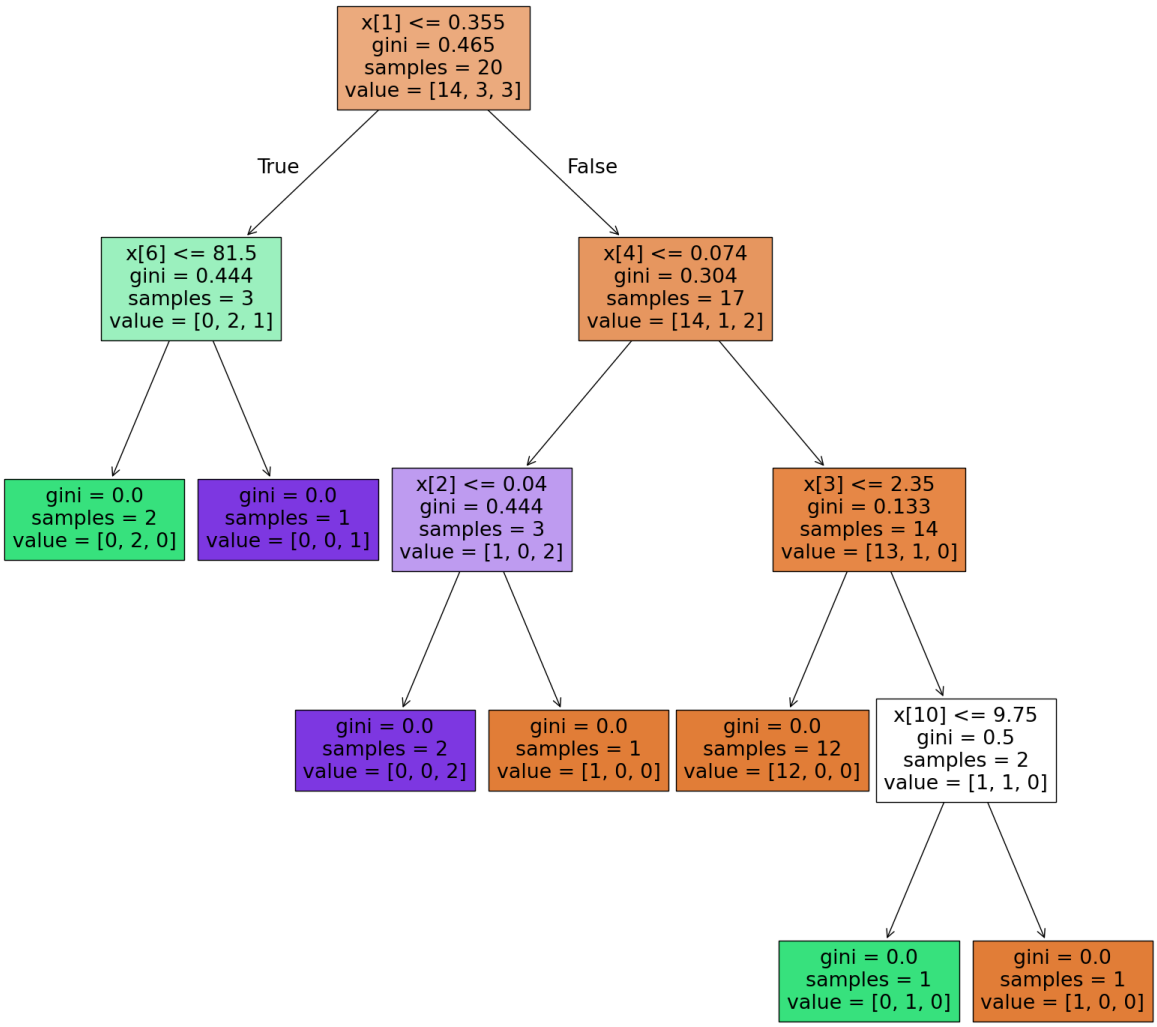
In [16]:
```
sampleData.columns
```

Out[16]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sug
ar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'dens
ity',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [17]:
```
X_sample.columns
```

Out[17]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sug
ar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'dens
ity',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

In [23]:
```
from sklearn import tree
plt.figure(figsize=(20, 20))
tree.plot_tree(sampleModel, filled=True)
plt.savefig("Sample_Decistion_Tree.png")
```



To understand how the decision tree in your image works, let's break down the key components:

1. Feature Splitting: `x[1] <= 0.355`

This represents the decision rule at the root of the decision tree. The tree splits based on this feature value (i.e., the feature at index `1`, denoted as `x[1]`).

- **Interpretation**: The tree checks whether the value of `x[1]` is less than or equal to `0.355`. If `True`, it moves to the left branch; if `False`, it moves to the right branch.

The process of selecting which feature and value to split on is determined by an algorithm (like CART) that tries to maximize information gain or minimize impurity (Gini index or entropy).

2. Gini Index

The **Gini Index** is a measure of impurity or disorder used in decision trees to determine the quality of a split. It tells us how pure or impure the nodes are.

- **Gini Index Formula**:

  `Gini = 1 - ∑(p_i)^2`
  where `p_i` is the probability of each class `i` in the node.

- **Gini Calculation**:
  - A node with a Gini index of `0` represents perfect purity, meaning all elements in that node belong to one class.
  - A Gini index closer to `1` indicates a higher impurity or a more diverse set of classes in that node.

In your tree:

- The root node has a **Gini index of 0.465**. This means the data at the root is somewhat impure and contains a mix of classes.
- As we move down the tree, the Gini index decreases, indicating that the data in the child nodes becomes purer.

3. Samples

- The number of **samples** represents the number of data points that reach a particular node.
  - At the root, we have `20 samples`, meaning the decision tree was trained on 20 data points.
  - As we move down the tree, the samples split across nodes depending on how the data satisfies the feature conditions.

For example:

- The root node has 20 samples.
- After the split, the left node (corresponding to `x[1] <= 0.355`) has `3 samples`, and the right node has `17 samples`.

4. Value

The **value** at each node represents the distribution of classes (target variable) at that point in the tree. The format of the value is usually:

```
value = [class_1_count, class_2_count, ..., class_n_count]
```

In your case, it looks like this tree is dealing with a classification problem involving 3 classes (or labels).

For example:

- The root node has `value = [14, 3, 3]`, meaning there are `14 samples` of class 1, `3 samples` of class 2, and `3 samples` of class 3 at the root.

As we move down the tree, the value becomes more concentrated towards a single class due to the splits, which helps the model classify the data.

Putting It All Together:

- **Root Node** (`x[1] <= 0.355`):
  - **Gini**: `0.465` → Impurity measure at the root, indicating a mix of classes.
  - **Samples**: `20` → Total number of training samples at the root.
  - **Value**: `[14, 3, 3]` → Distribution of classes at the root node (14 of class 1, 3 of class 2, 3 of class 3).
- **Left Child Node** (result of the split `x[1] <= 0.355`):
  - **Gini**: `0.444` → Impurity measure at this node.
  - **Samples**: `3` → Three data points end up in this node.
  - **Value**: `[0, 2, 1]` → Two of class 2 and one of class 3.

Each split tries to reduce the Gini index, improving the tree's ability to make accurate predictions by grouping similar data points together. The decision rules are chosen to maximize the difference in the Gini index between parent and child nodes.

Why Feature `x[1] <= 0.355` is Chosen

The feature `x[1]` with the threshold `0.355` is selected because, at the root, it results in the best split according to the decision tree algorithm. This selection is based on maximizing the **information gain** or minimizing the **impurity** (Gini index, in this case). The algorithm evaluates different features and thresholds and picks the one that best separates the data points according to the target classes.

In [19]:
```
set(sampleData.quality)
```

Out[19]:
```
{5, 6, 7}
```

In [20]:
```
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.12      0.20      0.15         5
           5       0.68      0.71      0.70       126
           6       0.58      0.57      0.57       115
           7       0.45      0.41      0.43        34
           8       0.00      0.00      0.00         3

    accuracy                           0.59       286
   macro avg       0.31      0.32      0.31       286
weighted avg       0.59      0.59      0.59       286
```

```
/Users/vaibhavarde/Desktop/DATASCIENCE/ChaiCode/MLNotes/MLNotes/lib/python
3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetri
cWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/vaibhavarde/Desktop/DATASCIENCE/ChaiCode/MLNotes/MLNotes/lib/python
3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetri
cWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Users/vaibhavarde/Desktop/DATASCIENCE/ChaiCode/MLNotes/MLNotes/lib/python
3.10/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetri
cWarning: Precision is ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```