```
!pip install anytree
```

```
Collecting anytree
  Downloading anytree-2.12.1-py3-none-any.whl (44 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.9/44.9 kB 1.3 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from anytree) (1.16.0)
Installing collected packages: anytree
Successfully installed anytree-2.12.1
```

```
!pip install pandas numpy typing
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: typing in /usr/local/lib/python3.10/dist-packages (3.7.4.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas)
```

```
import pandas as pd
import math
from collections import Counter
from anytree import NodeMixin, RenderTree
from typing import Dict, List
from anytree import RenderTree
from graphviz import Digraph
from graphviz import Source
```

```
class DecTree():

    def __init__(self, data: pd.DataFrame, target_attribute: str):
        self.data = data
        self.target_attr = target_attribute
        self.target_attr_vals = data[target_attribute].unique()
        self.root_node = None

    def pmf_target(self, df: pd.DataFrame) -> Dict[str, float]:
        target_counts = df[self.target_attr].value_counts(normalize=True)
        return dict(target_counts)

    def entropy(self, pmf: Dict[str, float]) -> float:
        entropy_val = -sum(p * math.log2(p) for p in pmf.values())
        return entropy_val

    def cal_entropy_df(self, df: pd.DataFrame) -> float:
        pmf = self.pmf_target(df)
        entropy_val = self.entropy(pmf)
        return entropy_val

    def info_gain_attribute(self, df: pd.DataFrame, attribute: str) -> float:
        total_entropy = self.cal_entropy_df(df)
        attribute_groups = df.groupby(attribute)
        weighted_entropy = sum(len(subgroup) / len(df) * self.cal_entropy_df(subgroup) for _, subgroup in attribute_gro
        information_gain = total_entropy - weighted_entropy
        return information_gain

    def max_info_gain_attribute(self, df: pd.DataFrame, attributes: List[int]) -> str:
        info_gain_dict = {attr: self.info_gain_attribute(df, attr) for attr in attributes}
        max_attr = max(info_gain_dict, key=info_gain_dict.get)
        print("max_info_gain_attribute = ", max_attr, "INFG-Value = ", info_gain_dict[max_attr])
        return max_attr

    def build_tree_infgain(self, df: pd.DataFrame, attr_list: List[str], start_node: 'DecTreeNode'):

      if len(df[self.target_attr].unique()) == 1:
          start_node.name = df[self.target_attr].iloc[0]
          return

      if not attr_list:
          start_node.name = df[self.target_attr].value_counts().idxmax()
          return

      max_attr = self.max_info_gain_attribute(df, attr_list)
      start_node.attribute = max_attr

      remaining_attrs = [attr for attr in attr_list if attr != max_attr]  # Exclude the selected attribute
      for value in df[max_attr].unique():
        value_subset = df[df[max_attr] == value]
        if value_subset.empty:
            leaf_node = DecTreeNode(name=df[self.target_attr].mode().values[0], attribute=max_attr, parent=start_node)
        else:
            child_node = DecTreeNode(name=value, attribute=max_attr, parent=start_node)
            self.build_tree_infgain(value_subset, attr_list[:], child_node)

    def generate_tree(self):
        attributes = self.data.columns.to_list()
        attributes.remove(self.target_attr)

        start_node = DecTreeNode("start", "start")
        self.build_tree_infgain(self.data, attributes, start_node)

        self.root_node = start_node

    def print_tree(self):
        for pre, _, node in RenderTree(self.root_node):
            print(f"{pre} {node.attribute}={node.name}")


    def predict(self, X: pd.DataFrame) -> List[str]:
        predictions = []
        for _, row in X.iterrows():
            node = self.root_node
            while node.children:
                attr_value = row[node.attribute.split('=')[0]]
                next_child = next((child for child in node.children if child.name == attr_value), None)
                if next_child:
                    node = next_child
                else:
                    break
            predictions.append(node.name)
```

```
        return predictions


class DecTreeNode(NodeMixin):
    def __init__(self, name: str, attribute: str, parent: 'DecTreeNode' = None):
        super(DecTreeNode, self).__init__()
        self.name = name
        self.attribute = attribute
        self.parent = parent
        self.attr value = f"{attribute}={name}"
```

```
!pwd
```

```
/content
```

```
data  = pd.read_csv("data.csv")
data
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes
4	rainy	cool	normal	False	yes
5	rainy	cool	normal	True	no
6	overcast	cool	normal	True	yes
7	sunny	mild	high	False	no
8	sunny	cool	normal	False	yes
9	rainy	mild	normal	False	yes
10	sunny	mild	normal	True	yes
11	overcast	mild	high	True	yes
12	overcast	hot	normal	False	yes
13	rainy	mild	high	True	no

```
tree = DecTree(data, 'play')
tree.generate_tree()
tree.print_tree()


dot = Digraph()
for pre, _, node in RenderTree(tree.root_node):
    dot.node(node.attr_value)
    if node.parent:
        dot.edge(node.parent.attr_value, node.attr_value)
dot.render("decision_tree_visualization", format="png", view=True)
Source(dot.source)
```

```
max info gain attribute =  outlook INFG Value =  0.2467498197743933
```

```
dt_train = pd.read_csv("cars_train.csv")
dt_test = pd.read_csv("cars_test.csv")

dt_test.head()
```

	buying_price	maintenance_cost	num_doors	num_persons	size_luggage	safet
0	vhigh	med	2	2	small	lo
1	high	high	3	4	big	me
2	low	med	4	2	big	lo
3	vhigh	high	5more	4	small	lo
4	med	med	4	2	med	hig

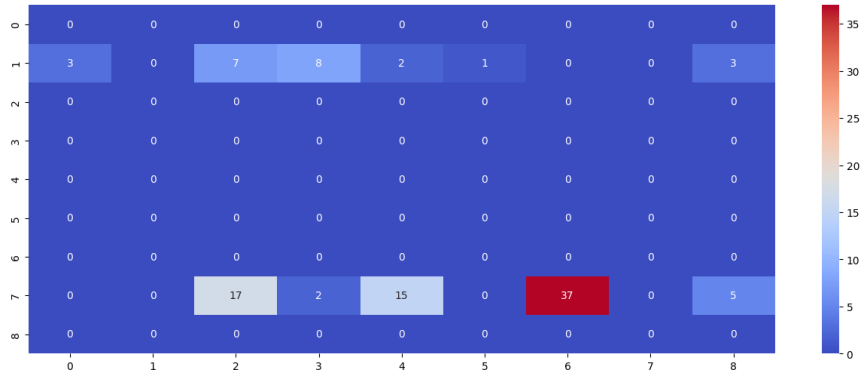
(outlook=sunny) (outlook=overcast) (outlook=rainy

```
# Train
dec_tree = DecTree(dt_train, "decision")
dec_tree.generate_tree()
dec_tree.print_tree()
```

```
        │   │   │   ├──  size_luggage=acc
        │   │   │   └──  size_luggage=unacc
        │   │   ├──  buying_price=acc
        │   │   └──  num_doors=med
        │   │       ├──  num_doors=unacc
        │   │       └──  num_doors=acc
        │   └──  num_doors=med
        │       ├──  buying_price=4
        │       │   ├──  buying_price=acc
        │       │   ├──  buying_price=acc
        │       │   └──  size_luggage=high
        │       │       ├──  size_luggage=acc
        │       │       └──  size_luggage=unacc
        │       ├──  num_doors=acc
        │       ├──  num_doors=acc
        │       └──  num_doors=unacc
        ├──  num_persons=unacc
        └──  buying_price=4
            ├──  maintenance_cost=low
            │   ├──  maintenance_cost=acc
            │   ├──  maintenance_cost=acc
            │   ├──  size_luggage=vhigh
            │   │   ├──  size_luggage=unacc
            │   │   ├──  size_luggage=unacc
            │   │   └──  size_luggage=acc
            │   └──  maintenance_cost=acc
            ├──  maintenance_cost=vhigh
            │   ├──  size_luggage=low
            │   │   ├──  size_luggage=acc
            │   │   └──  size_luggage=unacc
            │   ├──  maintenance_cost=unacc
            │   ├──  size_luggage=med
            │   │   ├──  size_luggage=unacc
            │   │   ├──  size_luggage=acc
            │   │   └──  size_luggage=unacc
            │   └──  maintenance_cost=unacc
            ├──  maintenance_cost=med
            │   ├──  size_luggage=high
            │   │   ├──  size_luggage=acc
            │   │   └──  size_luggage=unacc
            │   ├──  maintenance_cost=acc
            │   ├──  num_doors=vhigh
            │   │   ├──  num_doors=acc
            │   │   ├──  size_luggage=5more
            │   │   │   ├──  size_luggage=unacc
            │   │   │   └──  size_luggage=acc
            │   │   └──  num_doors=unacc
            │   └──  maintenance_cost=acc
            └──  maintenance_cost=high
                ├──  num_doors=low
                │   ├──  num_doors=unacc
                │   └──  num_doors=acc
                ├──  maintenance_cost=unacc
                ├──  size_luggage=med
                │   ├──  size_luggage=acc
                │   ├──  size_luggage=acc
                │   └──  size_luggage=unacc
                └──  maintenance_cost=unacc
```

```
# Test
dt_test_x = dt_test.drop(columns="decision")
dt_test_y = dt_test["decision"].to_list()

preds = dec_tree .predict(dt_test_x)
```

```
# Evaluate results
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt

import seaborn as sns

acc = accuracy_score(dt_test_y, preds)
cm = confusion_matrix(dt_test_y, preds)
# print(f"Accuracy: {acc}")
# print(cm)

plt.figure(figsize=(16, 6))
fig8= sns.heatmap(cm, annot=True, cmap='coolwarm')
fig8.figure.savefig('Confusion_Metrix.png',  bbox_inches='tight')
```




```
# Create a graphviz Digraph object
dot = Digraph()

# Add nodes and edges to the graphviz object based on your tree structure
for pre, _, node in RenderTree(dec_tree.root_node):
    dot.node(node.attr_value)

    if node.parent:
        dot.edge(node.parent.attr_value, node.attr_value)

# Visualize the decision tree
dot.render("decision_tree_visualization", format="png", view=True)
Source(dot.source)
```