Deccision Tree Classifier to predict the safety of Car

In [23]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns # statistical data visualization
%matplotlib inline
```

In [24]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [25]:
```
import warnings

warnings.filterwarnings('ignore')
```

Import Dataset

In [26]:
```
data = 'car_evaluation.csv'
df = pd.read_csv(data, header=None)
```

Exploratory data analysis ¶

In [27]:
```
# view dimensions of dataset

df.shape
```

Out[27]: (1728, 7)

In [28]:
```
#preview the dataset

df.head()
```

Out[28]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [29]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class
```

```
df.columns = col_names

col_names
```

Out[29]: ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']

In [30]:
```
# let's again preview the dataset

df.head()
```

Out[30]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

View summary of the dataset

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   buying    1728 non-null   object
 1   maint     1728 non-null   object
 2   doors     1728 non-null   object
 3   persons   1728 non-null   object
 4   lug_boot  1728 non-null   object
 5   safety    1728 non-null   object
 6   class     1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

Frequency distribution of values in variables

In [32]:
```
col_names = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class

for col in col_names:

    print(df[col].value_counts())
```

```
buying
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
maint
vhigh    432
high     432
med      432
low      432
Name: count, dtype: int64
doors
2        432
3        432
4        432
5more    432
Name: count, dtype: int64
persons
2       576
4       576
more    576
Name: count, dtype: int64
lug_boot
small    576
med      576
big      576
Name: count, dtype: int64
safety
low     576
med     576
high    576
Name: count, dtype: int64
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

In [33]:
```
df['class'].value_counts()
```

Out[33]:
```
class
unacc    1210
acc       384
good       69
vgood      65
Name: count, dtype: int64
```

In [34]:
```
# check missing values in variables

df.isnull().sum()
```

Out[34]:
```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```

Declare feature vector and target variable ¶

In [35]:
```
X = df.drop(['class'], axis=1)

y = df['class']
```

Split data into separate training and test set ¶

In [36]:
```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rand
```

In [37]:
```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rand
```

In [38]:
```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

Out[38]:
```
((1157, 6), (571, 6))
```

Feature Engineering

In [39]:
```
# check data types in X_train

X_train.dtypes
```

Out[39]:
```
buying      object
maint       object
doors       object
persons     object
lug_boot    object
safety      object
dtype: object
```

Encode categorical variables

In [40]:
```
X_train.head()
```

Out[40]:

	buying	maint	doors	persons	lug_boot	safety
48	vhigh	vhigh	3	more	med	low
468	high	vhigh	3	4	small	low
155	vhigh	high	3	more	small	high
1721	low	low	5more	more	small	high
1208	med	low	2	more	small	high

In [41]:
```
# import category encoders

import category_encoders as ce
```

In [42]:
```
from sklearn.preprocessing import OrdinalEncoder

encoder = OrdinalEncoder()

# fit on training categorical columns only
X_train[["buying", "maint", "doors", "persons", "lug_boot", "safety"]] = \
    encoder.fit_transform(X_train[["buying", "maint", "doors", "persons", "lug_b

X_test[["buying", "maint", "doors", "persons", "lug_boot", "safety"]] = \
    encoder.transform(X_test[["buying", "maint", "doors", "persons", "lug_boot",
```

In [43]:
```
from sklearn.preprocessing import OrdinalEncoder

# define categorical columns
cols = ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']

# initialize sklearn OrdinalEncoder
encoder = OrdinalEncoder()

# fit on train and transform
X_train[cols] = encoder.fit_transform(X_train[cols])

# transform only test (no fit here)
X_test[cols] = encoder.transform(X_test[cols])

# check first few rows
X_train.head()
```

Out[43]:

	buying	maint	doors	persons	lug_boot	safety
48	3.0	3.0	1.0	2.0	1.0	1.0
468	0.0	3.0	1.0	1.0	2.0	1.0
155	3.0	0.0	1.0	2.0	2.0	0.0
1721	1.0	1.0	3.0	2.0	2.0	0.0
1208	2.0	1.0	0.0	2.0	2.0	0.0

In [44]:
```
X_test.head()
```

Out[44]:

	buying	maint	doors	persons	lug_boot	safety
599	0.0	0.0	2.0	0.0	1.0	0.0
1201	2.0	1.0	0.0	1.0	1.0	2.0
628	0.0	0.0	3.0	0.0	0.0	2.0
1498	1.0	0.0	3.0	1.0	1.0	2.0
1263	2.0	1.0	2.0	2.0	1.0	1.0

Decision Tree Classifier with criterion gini index ¶

In [45]:
```
#import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
```

In [46]:
```
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)


# fit the model
clf_gini.fit(X_train, y_train)
```

Out[46]:

DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [47]:
```
y_pred_gini = clf_gini.predict(X_test)
```

In [48]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accurac
```

```
Model accuracy score with criterion gini index: 0.7653
```

Compare the train-set and test-set accuracy¶

In [49]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[49]:
```
array(['unacc', 'unacc', 'acc', ..., 'acc', 'unacc', 'unacc'],
      dtype=object)
```

In [50]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_
```

```
Training-set accuracy score: 0.7744
```

Check for overfitting and underfitting¶

In [51]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

```
Training set score: 0.7744
Test set score: 0.7653
```

Visualise decision trees

In [52]:
```
plt.figure(figsize=(12,8))
from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[52]:
```
[Text(0.375, 0.875, 'x[3] <= 0.5\ngini = 0.455\nsamples = 1157\nvalue = [255, 4
9, 813, 40]'),
 Text(0.25, 0.625, 'gini = 0.0\nsamples = 375\nvalue = [0, 0, 375, 0]'),
 Text(0.3125, 0.75, 'True  '),
 Text(0.5, 0.625, 'x[5] <= 0.5\ngini = 0.573\nsamples = 782\nvalue = [255, 49,
438, 40]'),
 Text(0.4375, 0.75, '  False'),
 Text(0.25, 0.375, 'x[1] <= 2.5\ngini = 0.634\nsamples = 263\nvalue = [139, 21,
63, 40]'),
 Text(0.125, 0.125, 'gini = 0.619\nsamples = 194\nvalue = [108.0, 21.0, 25.0, 4
0.0]'),
 Text(0.375, 0.125, 'gini = 0.495\nsamples = 69\nvalue = [31, 0, 38, 0]'),
 Text(0.75, 0.375, 'x[5] <= 1.5\ngini = 0.425\nsamples = 519\nvalue = [116, 28,
375, 0]'),
 Text(0.625, 0.125, 'gini = 0.0\nsamples = 257\nvalue = [0, 0, 257, 0]'),
 Text(0.875, 0.125, 'gini = 0.59\nsamples = 262\nvalue = [116, 28, 118, 0]')]
```

Visualize decision-trees with graphviz¶

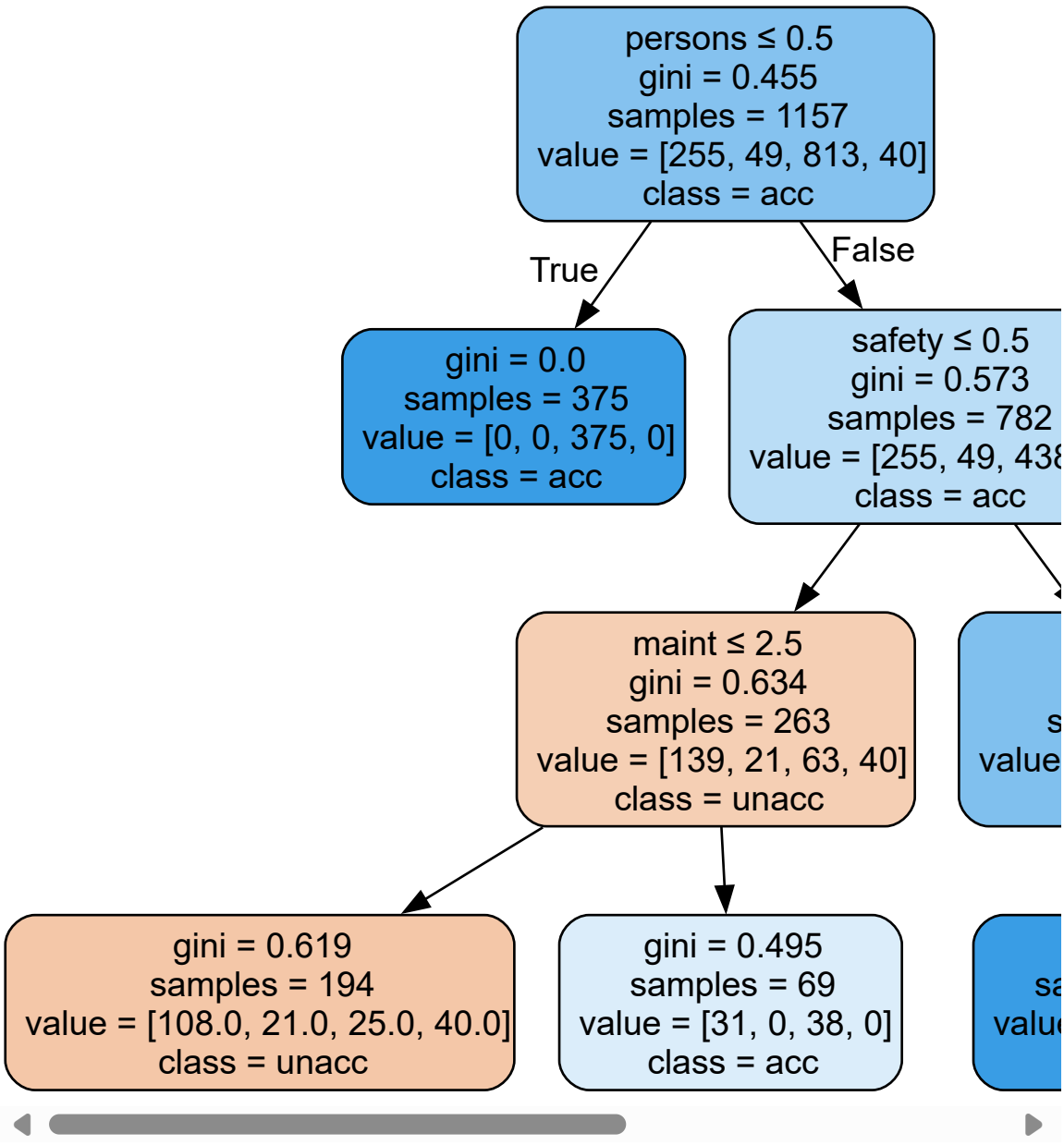
In [53]:
```
import graphviz
from sklearn import tree

dot_data = tree.export_graphviz(
    clf_gini,
    out_file=None,
    feature_names=X_train.columns,
    class_names=[str(cls) for cls in y_train.unique()],  # FIX here
    filled=True,
    rounded=True,
```

```
    special_characters=True
)

graph = graphviz.Source(dot_data)
graph
```

Out[53]:



Decision Tree Classifier with criterion entropy ¶

In [55]:
```
#instantiate the DecisionTreeClassifier model with criterion entropy

clf_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0


# fit the model
clf_en.fit(X_train, y_train)
```

Out[55]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=
0)
```

Predict the Test set results with criterion entropy¶

In [56]:
```
y_pred_en = clf_en.predict(X_test)
```

Check accuracy score with criterion entropy¶

In [57]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_s
```

```
Model accuracy score with criterion entropy: 0.7653
```

Compare the train-set and test-set accuracy¶

In [58]:
```
y_pred_train_en = clf_en.predict(X_train)

y_pred_train_en
```

Out[58]:
```
array(['unacc', 'unacc', 'acc', ..., 'acc', 'unacc', 'unacc'],
      dtype=object)
```

In [59]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_
```

```
Training-set accuracy score: 0.7744
```

Check for overfitting and underfitting¶

In [60]:
```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

```
Training set score: 0.7744
Test set score: 0.7653
```

Visualize decision-trees¶

In [61]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[61]:
```
[Text(0.375, 0.875, 'x[3] <= 0.5\nentropy = 1.2\nsamples = 1157\nvalue = [255,
49, 813, 40]'),
 Text(0.25, 0.625, 'entropy = 0.0\nsamples = 375\nvalue = [0, 0, 375, 0]'),
 Text(0.3125, 0.75, 'True  '),
 Text(0.5, 0.625, 'x[5] <= 0.5\nentropy = 1.465\nsamples = 782\nvalue = [255, 4
9, 438, 40]'),
 Text(0.4375, 0.75, '  False'),
 Text(0.25, 0.375, 'x[1] <= 2.5\nentropy = 1.684\nsamples = 263\nvalue = [139,
21, 63, 40]'),
 Text(0.125, 0.125, 'entropy = 1.668\nsamples = 194\nvalue = [108.0, 21.0, 25.
0, 40.0]'),
 Text(0.375, 0.125, 'entropy = 0.993\nsamples = 69\nvalue = [31, 0, 38, 0]'),
 Text(0.75, 0.375, 'x[5] <= 1.5\nentropy = 1.049\nsamples = 519\nvalue = [116,
28, 375, 0]'),
 Text(0.625, 0.125, 'entropy = 0.0\nsamples = 257\nvalue = [0, 0, 257, 0]'),
 Text(0.875, 0.125, 'entropy = 1.383\nsamples = 262\nvalue = [116, 28, 118,
0]')]
```

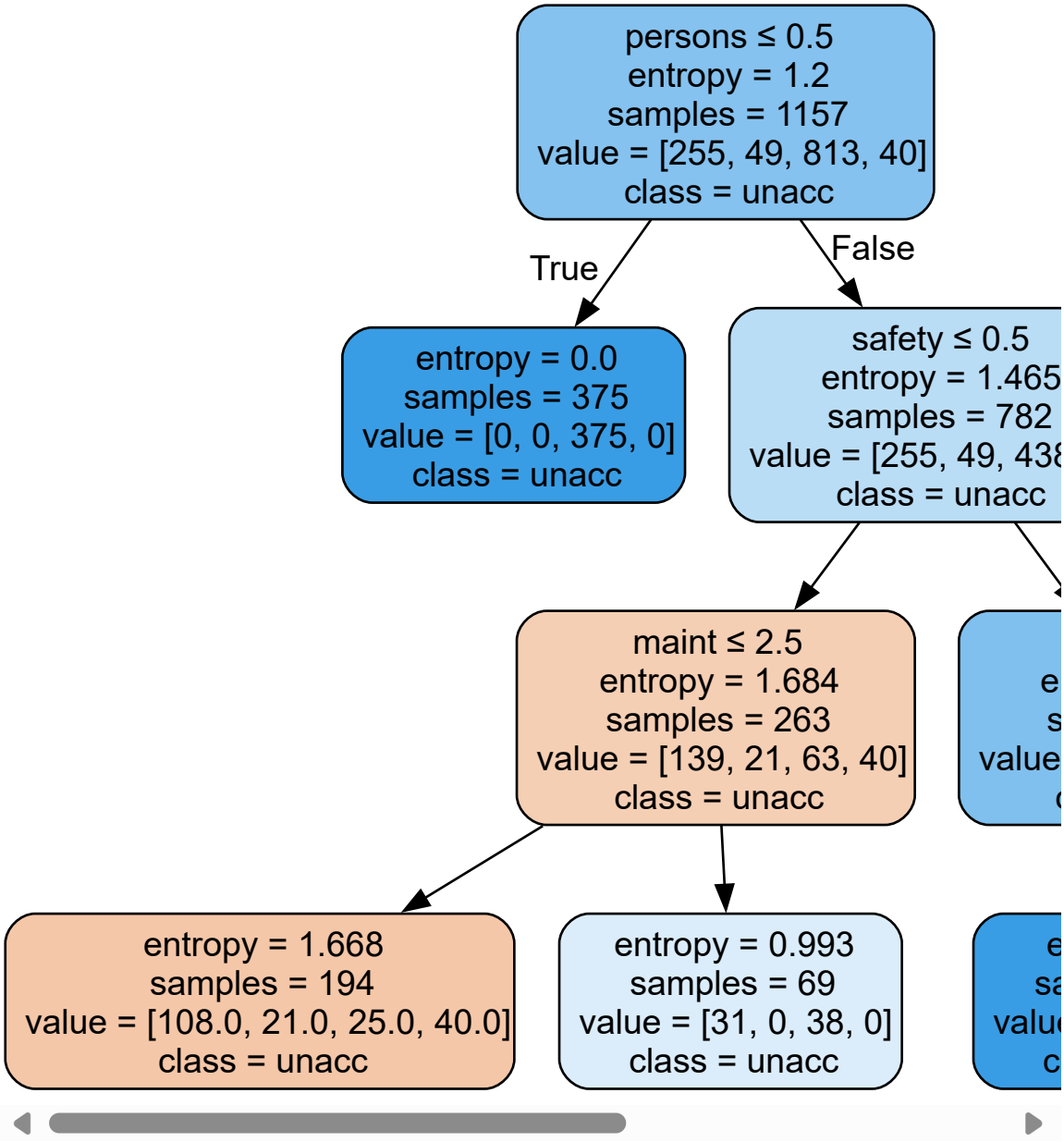
Visualize decision-trees with graphviz¶

In [62]:
```
import graphviz
dot_data = tree.export_graphviz(clf_en, out_file=None,
                                feature_names=X_train.columns,
                                class_names=y_train,
                                filled=True, rounded=True,
                                special_characters=True)

graph = graphviz.Source(dot_data)

graph
```

Out[62]:



Confusion matrix

In [63]:
```
# Print the Confusion Matrix and slice it into four pieces#
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

[[ 50   0  79   0]
 [  9   0  11   0]
 [ 10   0 387   0]
 [ 25   0   0   0]]
```

Classification Report

In [64]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_en))
```

```
              precision    recall  f1-score   support

         acc       0.53      0.39      0.45       129
        good       0.00      0.00      0.00        20
       unacc       0.81      0.97      0.89       397
       vgood       0.00      0.00      0.00        25

    accuracy                           0.77       571
   macro avg       0.34      0.34      0.33       571
weighted avg       0.68      0.77      0.72       571
```