UNIVERSITY OF LIVERPOOL

Assignment 1: Decision Tree Analysis and Case Study

DATA MINING AND MACHINE LEARNING (EBUS537)

Prepared by :

Name : Christina Ann Jacob

Student id : 201805307

Date: 10/12/2024

TABLE OF CONTENTS:

LIST OF FIGURES:

Data Exploration and Initial Insights

The training dataset,(myCarTrainDataset_2024.csv), has 500 records, each describing a car with five main details: price, doors, persons, boot, and accept.

Feature	**Description**	**Type**
price	Purchasing price of car	Categorical(high, low)
doors	Number of doors	Numeric(2,3,4,5)
persons	Seating capacity of the car	Numeric(2,4,5)
boot	Size of luggage boot	Categorical(small,med,big)
accept	Acceptability of car(Target)	Categorical:(yes,no)

No missing values were found, making the dataset clean and ready for analysis.

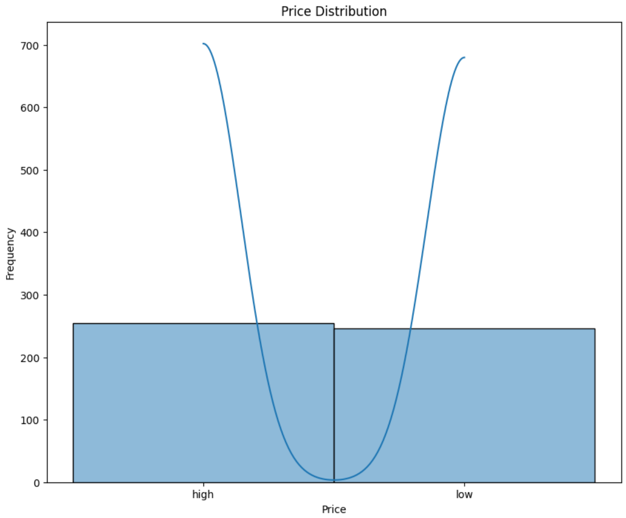
1. Numeric Features (Doors, Persons):

- The average number of doors is 3.73, so most cars have 3 or 4 doors.
- The average capacity is 3.66 people, ranging from 2 to 5 passengers .

2. Categorical Features (Price, Boot, Accept):

- Price is almost evenly split, with 254 cars marked as 'high' and 246 as 'low'.This balance is ideal for training a robust decision tree.
- For boot size, 'med' is more common, appearing 352 times compared to 'big'(148).
- In the accept category, 'no' is much more frequent (356) than 'yes'(144), meaning most cars are not considered acceptable,indicating class imbalance

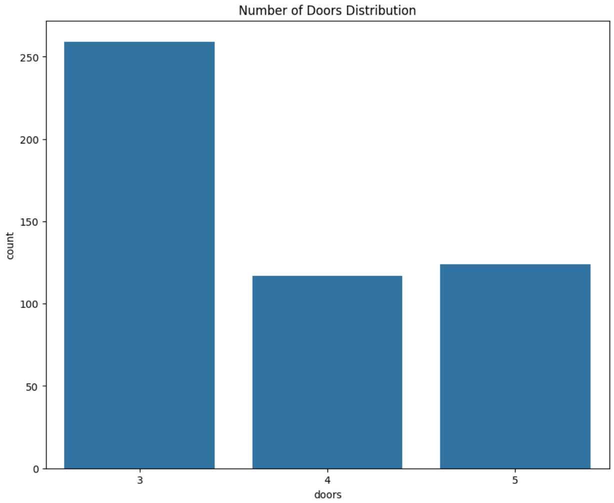
1. Price Distribution:



The histogram shows a nearly equal split between ‘high’ and ‘low’ prices, which is favorable for model training.

Fig 1:Price distribution

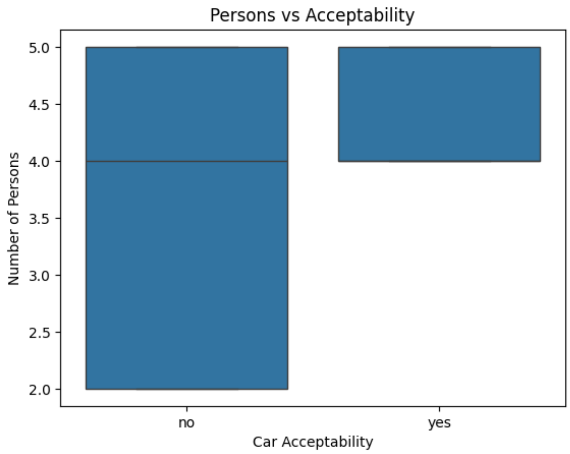
2. Doors Distribution:



A bar chart shows that cars with 3 doors are the most common, followed by those with 4 and 5 doors.

Fig 2: Number of Doors Distribution

3. Persons vs Acceptability :

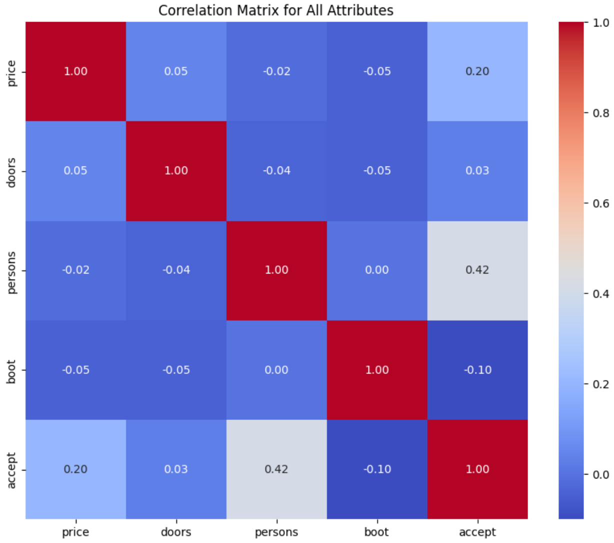


Cars that can accommodate 4 or more persons are more likely to be acceptable, highlighting “Persons” as a critical factor.

Higher capacity vehicles seem to be more practical for families or groups , leading to greater acceptability.

Fig 3: Persons vs Acceptability

4. Correlation Analysis :



	price	doors	persons	boot	accept
price	1.00	0.05	-0.02	-0.05	0.20
doors	0.05	1.00	-0.04	-0.05	0.03
persons	-0.02	-0.04	1.00	0.00	0.42
boot	-0.05	-0.05	0.00	1.00	-0.10
accept	0.20	0.03	0.42	-0.10	1.00

Price has a moderate positive correlation with accept (0.20), meaning higher prices might lead to more acceptance.Persons has a stronger positive correlation with accept (0.42), suggesting it’s an important factor for acceptance.

Fig 4: Correlation Matrix

Most features, like doors with boot or doors with accept, have very weak correlations (close to 0). This means these features provide unique information without much overlap.Overall, persons have the biggest impact on accept, while the others contribute independently.This helps us decide the feature to focus on.

Key insights:

- “Persons" has the strongest correlation with "accept," making it a strong candidate for an early split in the decision tree.
- The class imbalance(“no”>”yes”) may cause the decision tree to favor this class . Strategies such as class weighting or oversampling can be applied during tree building.
- Since there isn’t much of a link between doors and persons, it shows that each one gives us different info, which is great for building a fair decision tree.

Decision Tree Construction

STEP 1: Gini impurity of root node :

No: 356

Yes: 144

Total : 500

Gini(root) = 1 - (356/500)^2 - (144/500)^2 = 0.410

This value represents the impurity of the dataset before any splits.

Step 2 : Split analysis for features

Feature: Price

Gini(High)= 1 − (50/254)^2 − (204/254)^2)=1−(0.1969)^2−(0.8031)^2=0.3163

Gini(Low)= 1−(94/246)^2−(152/246)^2=1−(0.3821)^2−(0.6179)^2=0.4723

Weighted Gini for Price= (254/500)*0.3163+(246/500)*0.4723=0.393

Summary of Gini Impurities (All Features):

Feature	Gini impurity

Price	0.393
Doors	0.4092
Persons	0.326
Boot	0.4060

The attribute with the **lowest Gini impurity** is **persons** (**0.327**). Therefore, the root node splits on "persons."

Detailed Gini calculations for the remaining features ('Doors,' 'Boot') are provided in Appendix A.

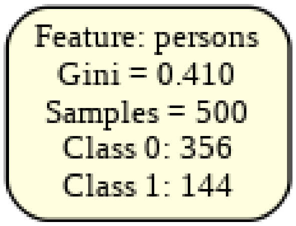


Fig 5 : Root node of decision tree

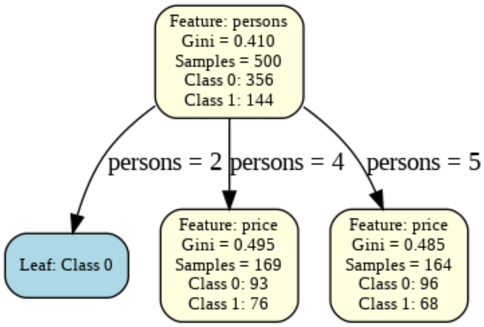


Fig 6 : Decision tree after first level of splitting

Step 3 : Recursive splitting

After splitting on the feature Persons, the dataset was divided into three branches: Persons=2, Persons=4, and Persons=5.

For Persons = 2

Since the Gini impurity is 0, this is a pure node, and no further splitting is required.

For Persons = 4

- **Total Samples: 169**
- **Class 0 (No): 93**
- **Class 1 (Yes): 76**

Gini Impurity for the Node

Gini(4) = 1 - (93/169)^2 - (76/169)^2 = 0.495

Feature Evaluation at this Node:

Feature	**Weighted gini**
Price	**0.454**
Doors	**0.4925**
Boot	**0.4922**

Best Split: Price (Gini = 0.454).

Similar calculations were performed, selecting Price (Gini = 0.4397) for the next split.

The node was split into two branches based on the values of Price (High and Low).

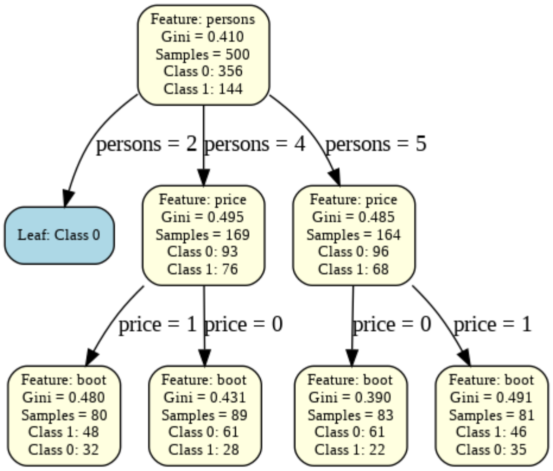


Fig 7 : Decision tree after second level of splitting

Step 4: Final Decision Tree

The process was repeated recursively until:

- Gini impurity reached 0 (pure nodes).
- No further splits were required.

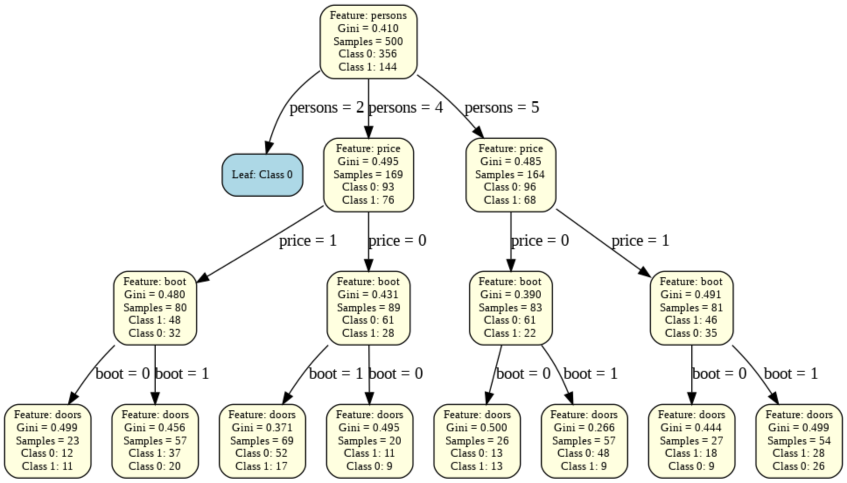


Fig 8 : Expanded decision tree showing the third level of recursive splits

Leaf nodes were assigned class labels based on the **majority voting system**. The final decision tree is shown below:

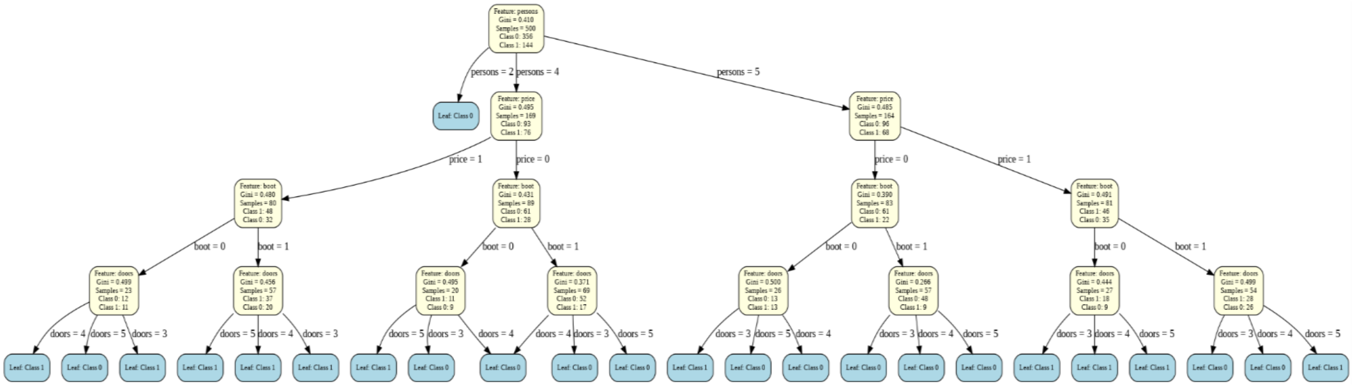


Fig 9 : Fully grown decision tree

The resulting decision tree is fully grown, with splits determined by the calculated Gini impurities at each step. The final tree ensures all possible splits have been considered, and predictions are made with the highest possible accuracy for the training dataset.

Appendix A contains the detailed Gini impurity calculations performed at each node and split level of the decision tree construction.

Post-Pruning of the Fully Grown Decision Tree

The fully grown decision tree was simplified to make it less complex and better at generalizing to new data. This was done by replacing subtrees with single nodes when all the outcomes in those parts were the same.

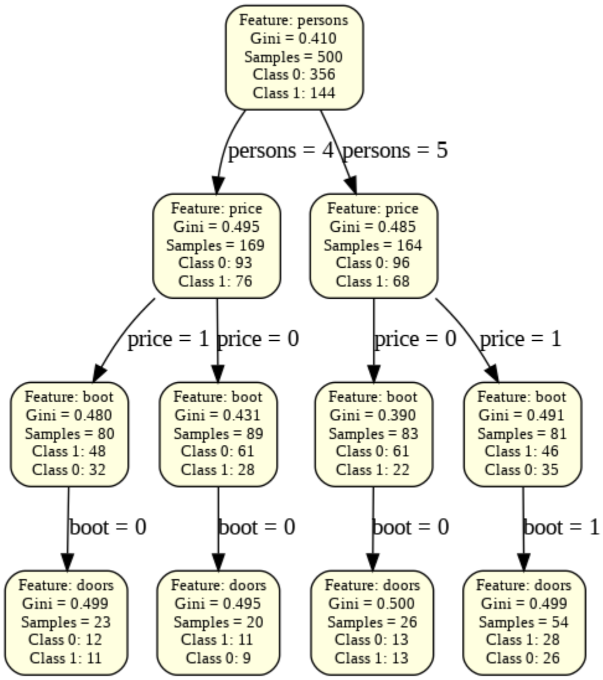
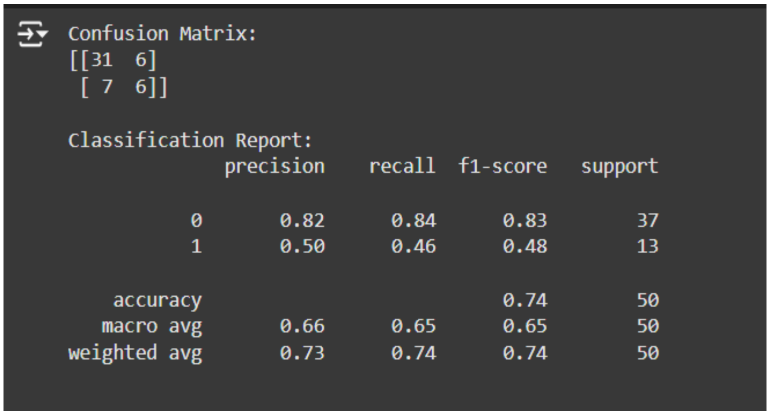


Fig 10 : Post-Pruned Decision Tree

After pruning,the tree depth remained 4 and the number of leaf nodes decreased from 25 to 17 reducing complexity.

Evaluation of the Post-Pruned Tree

The post-pruned decision tree was tested using the test dataset, yielding the following results:



```
Confusion Matrix:
[[31  6]
 [ 7  6]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.84      0.83        37
           1       0.50      0.46      0.48        13

    accuracy                           0.74        50
   macro avg       0.66      0.65      0.65        50
weighted avg       0.73      0.74      0.74        50
```

Fig 11 : *Confusion Matrix and Classification Report for the Post-Pruned Decision Tree*

True Negatives (Class 0): 31

False Negatives (Class 0 misclassified as 1): 6

True Positives (Class 1): 6

False Positives (Class 1 misclassified as 0): 7

The overall accuracy of the pruned tree is **74%**.

Class 0 (Unacceptable Cars):

- High precision (0.82) and recall (0.84) indicate strong performance in predicting this class.

Class 1 (Acceptable Cars):

- Lower precision (0.50) and recall (0.46) suggest potential challenges in accurately identifying acceptable cars likely due to class imbalance or overlapping features .

Key Insights :

- Pruning reduced the number of leaf nodes without affecting accuracy. This made the model easier to understand and faster to use.
- The pruned tree avoids being too specific to the training data, allowing it to make more reliable predictions on new, unseen data.
- The model performs less well for Class 1, showing the need for methods like balancing the data or **enhancing features** or considering alternate models like random forests.
- The pruned tree is a useful tool, especially for identifying unacceptable cars (Class 0). Its simpler design makes it easy for stakeholders to understand and trust the predictions.

Case Study: Development of Decision Tree Classification Algorithms in Predicting Mortality of COVID-19 Patients

The study by Zahra Mohammadi-Pirouz et al.(2024) focuses on using decision tree algorithms to predict the mortality of COVID-19 patients. The study focuses on supporting doctors in identifying highly risky patients and prioritization of clinical resources during this pandemic. The case will be analyzed in the sequence of the CRISP-DM framework.

Application of CRISP-DM Framework

1. Business Understanding

Objective: To develop a model that predicts COVID-19 mortality.This would help doctors focus on the patients who needed the most attention.

Motivation: The global COVID-19 pandemic overwhelmed healthcare systems, necessitating predictive tools to support critical decision-making in resource-constrained environments (Zahra Mohammadi-Pirouz et al., 2024). Decision trees were chosen because they are easy to understand and explain, essential for doctors who need clear, actionable insights.

2. Data Understanding

Dataset: The study used a dataset of COVID-19 patients, including:

Age and gender.

Health conditions (like diabetes or hypertension).

Clinical signs (like oxygen levels and inflammation markers).

Initial analysis revealed that age, oxygen saturation, and comorbidities were strongly associated with mortality, aligning with findings from other COVID-19 studies (WHO, 2020; Garg et al., 2020).

3. Data Preparation

Cleaning of data was done by imputation and treatment of missing values, which depended on the clinical relevance to improve data quality. Predictive features: age, comorbidities, oxygen saturation, were selected among others, to prevent overfitting . The data was transformed into a format and standard suitable for the decision tree algorithm.

4. Modeling

A decision tree classifier was chosen due to its simplicity, ability to handle non-linear relationships. The model provided clear, actionable rules, such as: *“If oxygen saturation < 90% and age > 65, mortality risk is high”* (Zahra Mohammadi-Pirouz et al., 2024).

The tree's depth and Gini impurity were tuned to balance complexity and accuracy. The decision tree offered insights into risk factors while maintaining predictive performance comparable to more complex models like random forests.

5. Evaluation

Accuracy, precision, recall, F1-score, and AUC-ROC were used to assess the model.

The decision tree achieved high accuracy and provided interpretable results, crucial for clinicians. The decision tree outperformed traditional statistical models like logistic regression in terms of interpretability and was competitive in accuracy with more advanced models.

6. Deployment

The model was added to hospital systems so doctors could get real-time predictions about which patients were at higher risk. This helped doctors take action early, improving patient care and making better use of resources.

The researchers pointed out that the model needed regular updates to keep up with new COVID-19 variants and changes in treatments.

Critical Insights

- Decision trees were perfect for healthcare because they provided simple, clear rules that doctors could easily understand and trust.
- The model didn’t perform as well for less common cases, like patients with a lower risk of mortality. This could be fixed by balancing the data or incorporating methods like random forests.
- While decision trees were great for this dataset, bigger datasets with more details might work better with a mix of decision trees and other advanced methods.
- This study showed how useful decision trees can be in healthcare, especially during emergencies like COVID-19, where quick and accurate decisions are crucial.

This case study shows how the CRISP-DM method helps create practical, data-driven solutions in healthcare. Using each step of the process allowed the researchers to develop a sound, understandable model that supported better decision-making during the COVID-19 crisis.

References

1.Zahra Mohammadi-Pirouz, Karimollah Hajian-Tilaki, Mahmoud Sadeghi Haddat-Zavareh, Abazar Amoozadeh and Bahrami, S. (2024). Development of decision tree classification algorithms in predicting mortality of COVID-19 patients. *International Journal of Emergency Medicine*, 17(1). doi:https://doi.org/10.1186/s12245-024-00681-7.

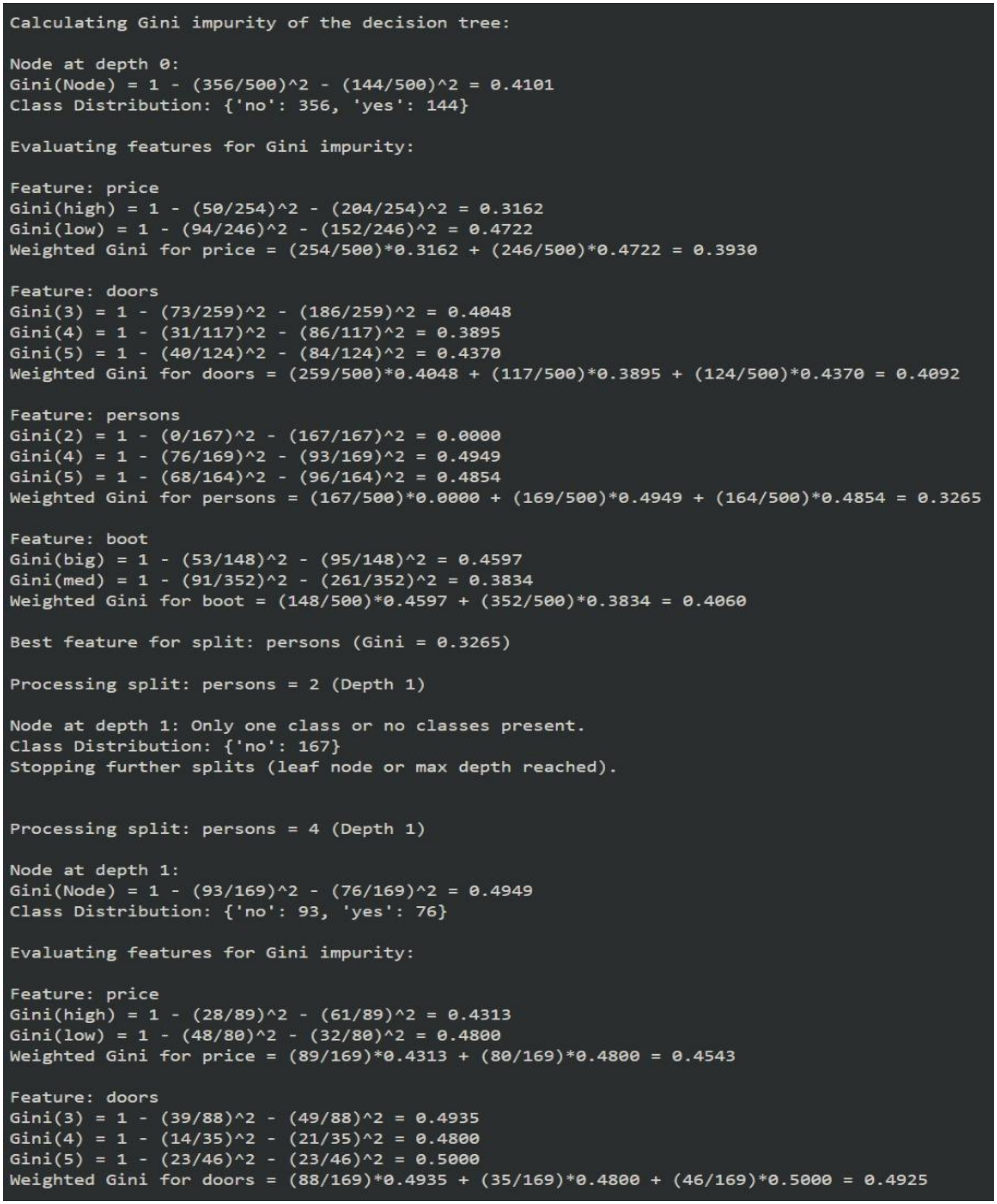
2.World Health Organization (WHO) (2020) *Clinical management of COVID-19: interim guidance, 27 May 2020*. Geneva: World Health Organization. Available at: https://iris.who.int/handle/10665/332196 (Accessed: 4 December 2024).

3.Garg, S. (2020). Hospitalization Rates and Characteristics of Patients Hospitalized with Laboratory-Confirmed Coronavirus Disease 2019 — COVID-NET, 14 States, March 1–30, 2020. *MMWR. Morbidity and Mortality Weekly Report*, [online] 69(15). doi:https://doi.org/10.15585/mmwr.mm6915e3.

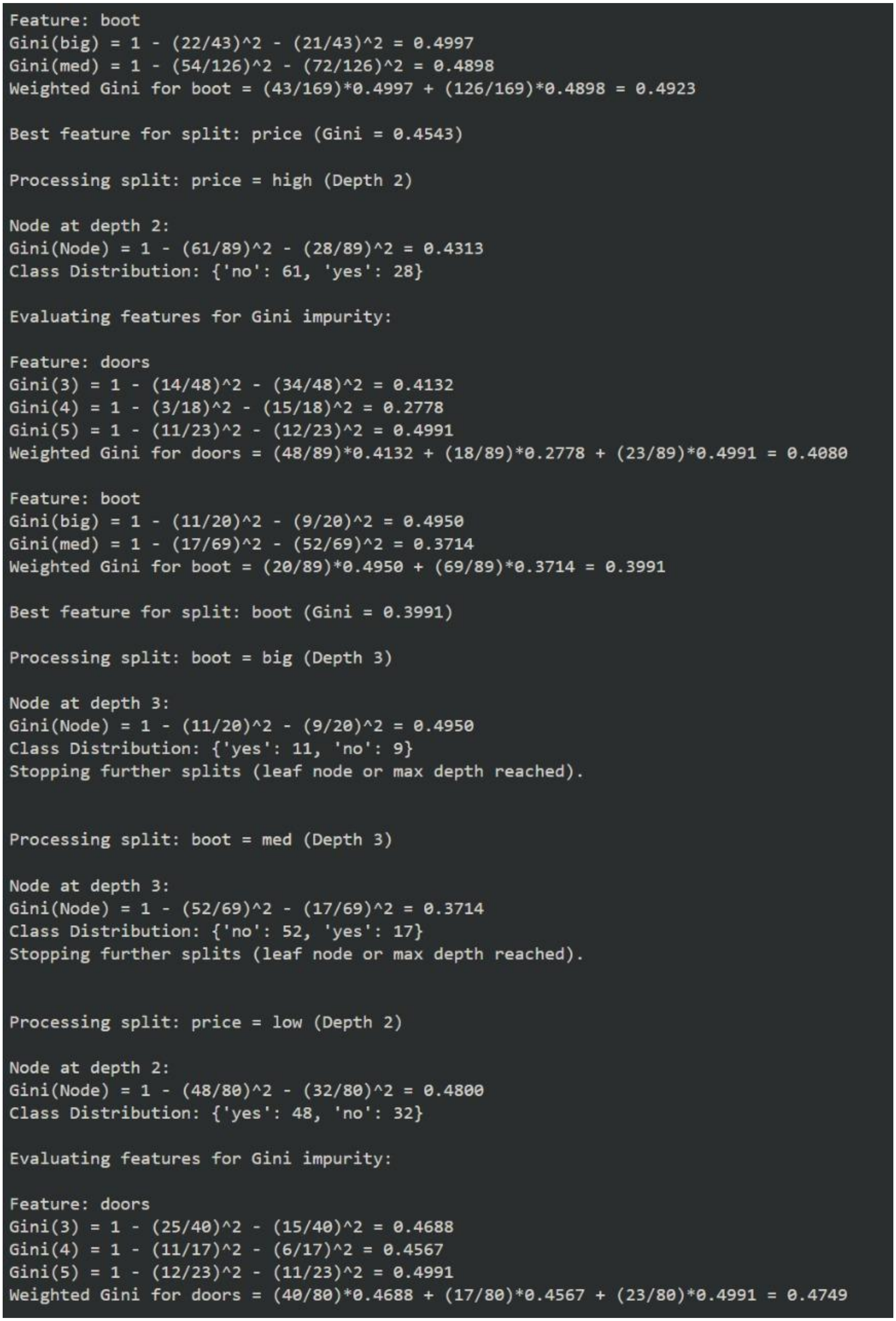
4.Hozan Akram Abdulqader and Abdulazeez, A.M. (2024). Review on Decision Tree Algorithm in Healthcare Applications. *Indonesian Journal of Computer Science*, 13(3). doi:https://doi.org/10.33022/ijcs.v13i3.4026.

5.Wirth, R. and Hipp, J. (2000). *CRISP-DM: Towards a standard process model for data mining | Request PDF*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/239585378_CRISP-DM_Towards_a_standard_process_model_for_data_mining [Accessed 9 Dec. 2024].

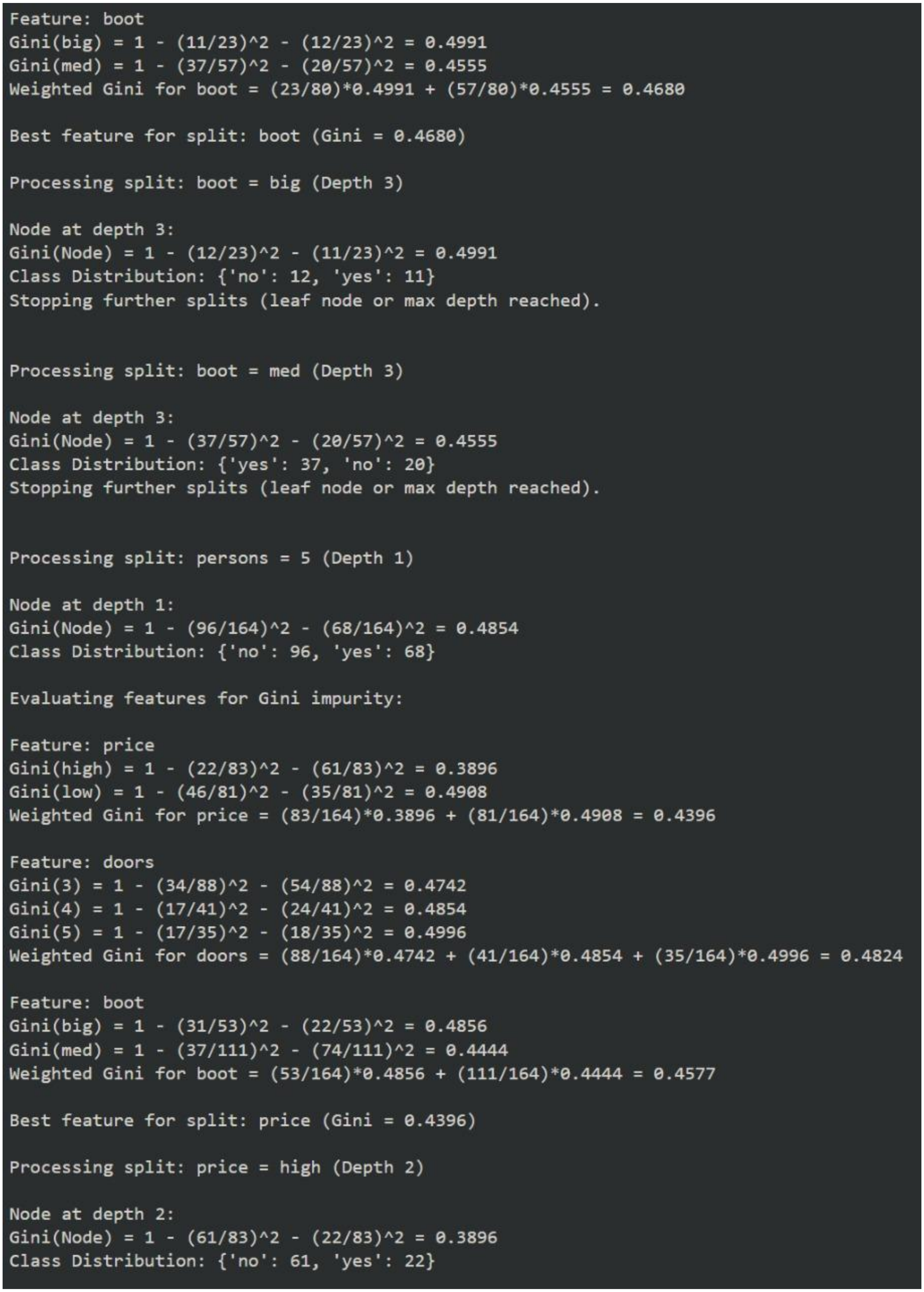
Appendix A: Detailed Gini Calculations



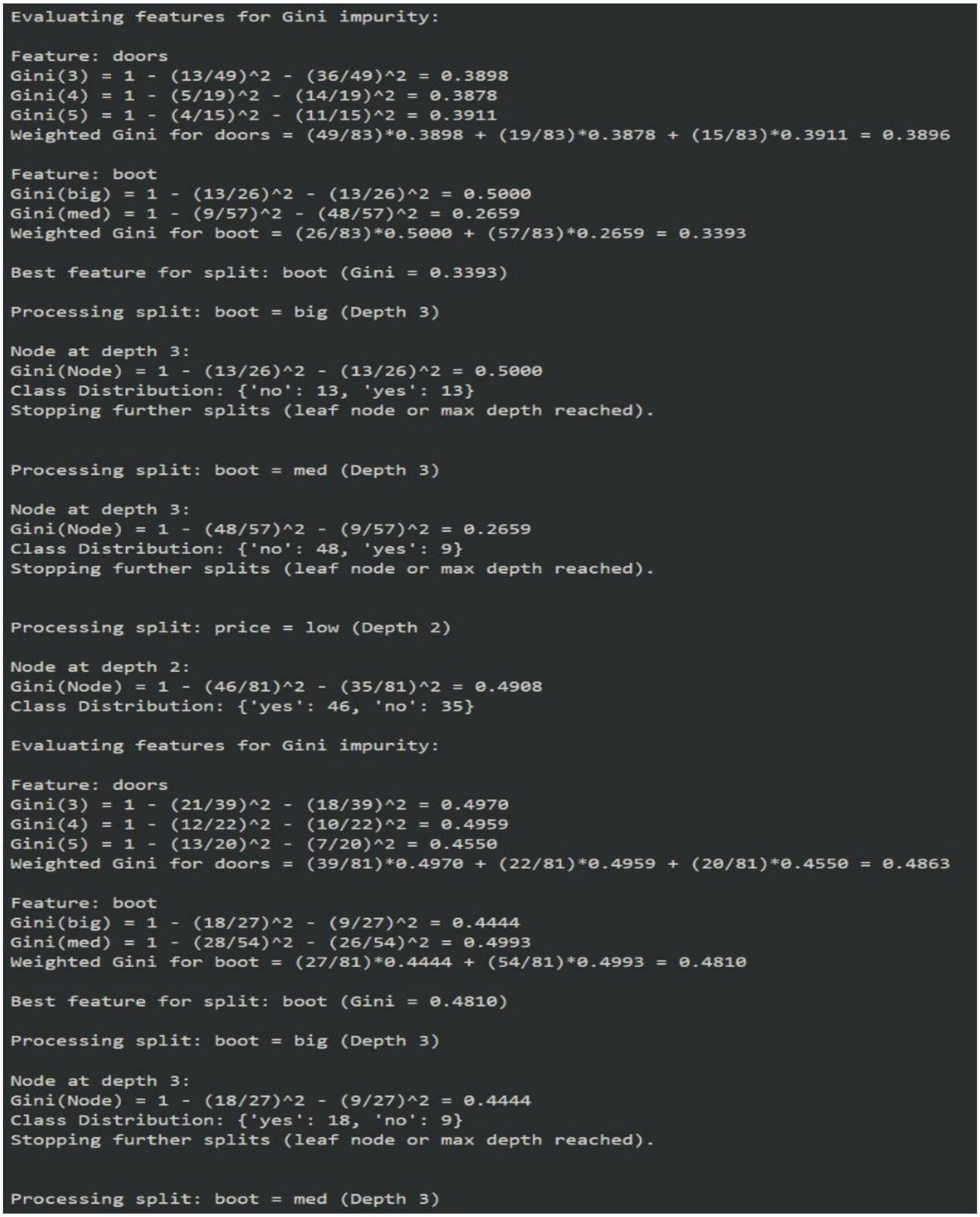
```
Calculating Gini impurity of the decision tree:

Node at depth 0:
Gini(Node) = 1 - (356/500)^2 - (144/500)^2 = 0.4101
Class Distribution: {'no': 356, 'yes': 144}

Evaluating features for Gini impurity:

Feature: price
Gini(high) = 1 - (50/254)^2 - (204/254)^2 = 0.3162
Gini(low) = 1 - (94/246)^2 - (152/246)^2 = 0.4722
Weighted Gini for price = (254/500)*0.3162 + (246/500)*0.4722 = 0.3930

Feature: doors
Gini(3) = 1 - (73/259)^2 - (186/259)^2 = 0.4048
Gini(4) = 1 - (31/117)^2 - (86/117)^2 = 0.3895
Gini(5) = 1 - (40/124)^2 - (84/124)^2 = 0.4370
Weighted Gini for doors = (259/500)*0.4048 + (117/500)*0.3895 + (124/500)*0.4370 = 0.4092

Feature: persons
Gini(2) = 1 - (0/167)^2 - (167/167)^2 = 0.0000
Gini(4) = 1 - (76/169)^2 - (93/169)^2 = 0.4949
Gini(5) = 1 - (68/164)^2 - (96/164)^2 = 0.4854
Weighted Gini for persons = (167/500)*0.0000 + (169/500)*0.4949 + (164/500)*0.4854 = 0.3265

Feature: boot
Gini(big) = 1 - (53/148)^2 - (95/148)^2 = 0.4597
Gini(med) = 1 - (91/352)^2 - (261/352)^2 = 0.3834
Weighted Gini for boot = (148/500)*0.4597 + (352/500)*0.3834 = 0.4060

Best feature for split: persons (Gini = 0.3265)

Processing split: persons = 2 (Depth 1)

Node at depth 1: Only one class or no classes present.
Class Distribution: {'no': 167}
Stopping further splits (leaf node or max depth reached).


Processing split: persons = 4 (Depth 1)

Node at depth 1:
Gini(Node) = 1 - (93/169)^2 - (76/169)^2 = 0.4949
Class Distribution: {'no': 93, 'yes': 76}

Evaluating features for Gini impurity:

Feature: price
Gini(high) = 1 - (28/89)^2 - (61/89)^2 = 0.4313
Gini(low) = 1 - (48/80)^2 - (32/80)^2 = 0.4800
Weighted Gini for price = (89/169)*0.4313 + (80/169)*0.4800 = 0.4543

Feature: doors
Gini(3) = 1 - (39/88)^2 - (49/88)^2 = 0.4935
Gini(4) = 1 - (14/35)^2 - (21/35)^2 = 0.4800
Gini(5) = 1 - (23/46)^2 - (23/46)^2 = 0.5000
Weighted Gini for doors = (88/169)*0.4935 + (35/169)*0.4800 + (46/169)*0.5000 = 0.4925
```



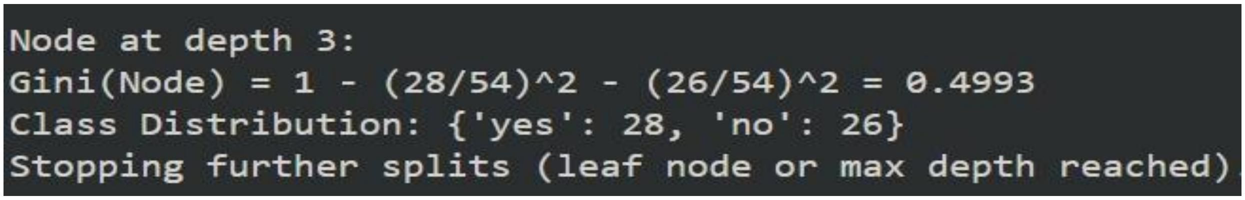
```
Feature: boot
Gini(big) = 1 - (22/43)^2 - (21/43)^2 = 0.4997
Gini(med) = 1 - (54/126)^2 - (72/126)^2 = 0.4898
Weighted Gini for boot = (43/169)*0.4997 + (126/169)*0.4898 = 0.4923

Best feature for split: price (Gini = 0.4543)

Processing split: price = high (Depth 2)

Node at depth 2:
Gini(Node) = 1 - (61/89)^2 - (28/89)^2 = 0.4313
Class Distribution: {'no': 61, 'yes': 28}

Evaluating features for Gini impurity:

Feature: doors
Gini(3) = 1 - (14/48)^2 - (34/48)^2 = 0.4132
Gini(4) = 1 - (3/18)^2 - (15/18)^2 = 0.2778
Gini(5) = 1 - (11/23)^2 - (12/23)^2 = 0.4991
Weighted Gini for doors = (48/89)*0.4132 + (18/89)*0.2778 + (23/89)*0.4991 = 0.4080

Feature: boot
Gini(big) = 1 - (11/20)^2 - (9/20)^2 = 0.4950
Gini(med) = 1 - (17/69)^2 - (52/69)^2 = 0.3714
Weighted Gini for boot = (20/89)*0.4950 + (69/89)*0.3714 = 0.3991

Best feature for split: boot (Gini = 0.3991)

Processing split: boot = big (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (11/20)^2 - (9/20)^2 = 0.4950
Class Distribution: {'yes': 11, 'no': 9}
Stopping further splits (leaf node or max depth reached).


Processing split: boot = med (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (52/69)^2 - (17/69)^2 = 0.3714
Class Distribution: {'no': 52, 'yes': 17}
Stopping further splits (leaf node or max depth reached).


Processing split: price = low (Depth 2)

Node at depth 2:
Gini(Node) = 1 - (48/80)^2 - (32/80)^2 = 0.4800
Class Distribution: {'yes': 48, 'no': 32}

Evaluating features for Gini impurity:

Feature: doors
Gini(3) = 1 - (25/40)^2 - (15/40)^2 = 0.4688
Gini(4) = 1 - (11/17)^2 - (6/17)^2 = 0.4567
Gini(5) = 1 - (12/23)^2 - (11/23)^2 = 0.4991
Weighted Gini for doors = (40/80)*0.4688 + (17/80)*0.4567 + (23/80)*0.4991 = 0.4749
```



```
Feature: boot
Gini(big) = 1 - (11/23)^2 - (12/23)^2 = 0.4991
Gini(med) = 1 - (37/57)^2 - (20/57)^2 = 0.4555
Weighted Gini for boot = (23/80)*0.4991 + (57/80)*0.4555 = 0.4680

Best feature for split: boot (Gini = 0.4680)

Processing split: boot = big (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (12/23)^2 - (11/23)^2 = 0.4991
Class Distribution: {'no': 12, 'yes': 11}
Stopping further splits (leaf node or max depth reached).


Processing split: boot = med (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (37/57)^2 - (20/57)^2 = 0.4555
Class Distribution: {'yes': 37, 'no': 20}
Stopping further splits (leaf node or max depth reached).


Processing split: persons = 5 (Depth 1)

Node at depth 1:
Gini(Node) = 1 - (96/164)^2 - (68/164)^2 = 0.4854
Class Distribution: {'no': 96, 'yes': 68}

Evaluating features for Gini impurity:

Feature: price
Gini(high) = 1 - (22/83)^2 - (61/83)^2 = 0.3896
Gini(low) = 1 - (46/81)^2 - (35/81)^2 = 0.4908
Weighted Gini for price = (83/164)*0.3896 + (81/164)*0.4908 = 0.4396

Feature: doors
Gini(3) = 1 - (34/88)^2 - (54/88)^2 = 0.4742
Gini(4) = 1 - (17/41)^2 - (24/41)^2 = 0.4854
Gini(5) = 1 - (17/35)^2 - (18/35)^2 = 0.4996
Weighted Gini for doors = (88/164)*0.4742 + (41/164)*0.4854 + (35/164)*0.4996 = 0.4824

Feature: boot
Gini(big) = 1 - (31/53)^2 - (22/53)^2 = 0.4856
Gini(med) = 1 - (37/111)^2 - (74/111)^2 = 0.4444
Weighted Gini for boot = (53/164)*0.4856 + (111/164)*0.4444 = 0.4577

Best feature for split: price (Gini = 0.4396)

Processing split: price = high (Depth 2)

Node at depth 2:
Gini(Node) = 1 - (61/83)^2 - (22/83)^2 = 0.3896
Class Distribution: {'no': 61, 'yes': 22}
```



```
Evaluating features for Gini impurity:

Feature: doors
Gini(3) = 1 - (13/49)^2 - (36/49)^2 = 0.3898
Gini(4) = 1 - (5/19)^2 - (14/19)^2 = 0.3878
Gini(5) = 1 - (4/15)^2 - (11/15)^2 = 0.3911
Weighted Gini for doors = (49/83)*0.3898 + (19/83)*0.3878 + (15/83)*0.3911 = 0.3896

Feature: boot
Gini(big) = 1 - (13/26)^2 - (13/26)^2 = 0.5000
Gini(med) = 1 - (9/57)^2 - (48/57)^2 = 0.2659
Weighted Gini for boot = (26/83)*0.5000 + (57/83)*0.2659 = 0.3393

Best feature for split: boot (Gini = 0.3393)

Processing split: boot = big (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (13/26)^2 - (13/26)^2 = 0.5000
Class Distribution: {'no': 13, 'yes': 13}
Stopping further splits (leaf node or max depth reached).


Processing split: boot = med (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (48/57)^2 - (9/57)^2 = 0.2659
Class Distribution: {'no': 48, 'yes': 9}
Stopping further splits (leaf node or max depth reached).


Processing split: price = low (Depth 2)

Node at depth 2:
Gini(Node) = 1 - (46/81)^2 - (35/81)^2 = 0.4908
Class Distribution: {'yes': 46, 'no': 35}

Evaluating features for Gini impurity:

Feature: doors
Gini(3) = 1 - (21/39)^2 - (18/39)^2 = 0.4970
Gini(4) = 1 - (12/22)^2 - (10/22)^2 = 0.4959
Gini(5) = 1 - (13/20)^2 - (7/20)^2 = 0.4550
Weighted Gini for doors = (39/81)*0.4970 + (22/81)*0.4959 + (20/81)*0.4550 = 0.4863

Feature: boot
Gini(big) = 1 - (18/27)^2 - (9/27)^2 = 0.4444
Gini(med) = 1 - (28/54)^2 - (26/54)^2 = 0.4993
Weighted Gini for boot = (27/81)*0.4444 + (54/81)*0.4993 = 0.4810

Best feature for split: boot (Gini = 0.4810)

Processing split: boot = big (Depth 3)

Node at depth 3:
Gini(Node) = 1 - (18/27)^2 - (9/27)^2 = 0.4444
Class Distribution: {'yes': 18, 'no': 9}
Stopping further splits (leaf node or max depth reached).


Processing split: boot = med (Depth 3)
```



```
Node at depth 3:
Gini(Node) = 1 - (28/54)^2 - (26/54)^2 = 0.4993
Class Distribution: {'yes': 28, 'no': 26}
Stopping further splits (leaf node or max depth reached).
```