CSE343/ECE343: Machine Learning
Assignment-2 Decision Trees, Random Forests and Perceptron Max Marks: 25 (Programming: 15, Theory: 10) Due Date: 24/09/2023, 11:59 PM

Instructions

- Keep collaborations at high-level discussions. Copying/Plagiarism will be dealt with strictly.
- Late submission penalty: As per course policy.
- Your submission should be a single zip file 2020xxx_HW1.zip (Where *2020xxx* is your roll number). Include all the files (code and report with theory questions) arranged with proper names. A single .pdf report explaining your codes with results, relevant graphs, visualization and solution to theory questions should be there. The structure of submission should follow:

  2020xxx_HW2
  |− code_rollno.py/.ipynb
  |− report_rollno.pdf
  |− (All other files for submission)
- Anything not in the report will not be graded.
- Remember to turn in after uploading on Google Classroom. No excuses or issues would be taken regarding this after the deadline.
- Start the assignment early. Resolve all your doubts from TAs in their office hours at least two days before the deadline.
- Your code should be neat and well-commented.
- You have to do either Section B or C.
- Section A is mandatory.

—---

1. (4 points) Section A (Theoretical)

(a) (1 marks) Discuss the trade-off between correlation and diversity in Random Forests. Why is it important for the trees to be correlated up to a certain extent while maintaining diversity?

(b) (1 marks) When might the "curse of dimensionality" become an issue in Naive Bayes? What strategies can be employed to mitigate this problem in practice?

(c) (1 marks) What kind of problems will the Naive Bayes classifier face if it] encounters some value of attributes which was not present in the training dataset? Do you think this will affect the inference results? If yes, list some of the approaches to mitigate this problem, else give your reason why you think so that the results won’t be affected. Explain with an example.

(d) (1 marks) Do you think that while splitting a decision tree node using Information Gain might be biased if some attributes have more cardinality than others? If yes, mention some other criterion for attribute selection with proper explanation, else give reason why decision trees are unaffected by attributes having different cardinality of attributes. Explain with an example.

Solution::

(a) -

The trade-off between correlation and diversity in Random Forests is a critical aspect of the algorithm's design, and it impacts the overall performance and robustness of the model.

Correlation in Random Forests:

Correlation refers to the degree to which individual trees in a Random Forest produce similar predictions for the same input data. When trees in a Random Forest are highly correlated, they tend to make the same mistakes or errors on the same data points. This can lead to several problems:

- If the trees are highly correlated, their errors will likely be similar, and the ensemble won't be as effective at reducing variance.
- Overfitting the training data leads to a lack of generalization on unseen data. Random Forests are designed to mitigate overfitting, but a high correlation among trees can undermine this.

- If all the trees produce the exact predictions for a particular input, the model needs more diversity, and it may not be robust to different variations in the data.

  Diversity refers to the differences between the individual trees in the forest. When the trees are diverse, they capture different aspects of the data and make different errors. This allows the ensemble to reduce the overall variance and make more accurate predictions. However, if the trees are too diverse, the ensemble may suffer from high bias and fail to capture the underlying patterns in the data.

  Maintaining an optimal balance between correlation and diversity is essential for the Random Forest to achieve high performance. When the trees are too correlated, the ensemble may lose its predictive power and be unable to generalize well to new data. On the other hand, when the trees are too diverse, the ensemble may suffer from high bias and cannot capture the underlying patterns in the data.

  This balance ensures that the ensemble can capture the complexity of the data while reducing overfitting and variance. Techniques such as feature bagging and random subspace methods can help introduce diversity in the trees, while forms like bootstrap aggregating (bagging) can help reduce correlation among the trees. Regularization techniques, such as limiting the depth of the trees, can also be used to control the level of correlation and diversity in the ensemble.

(b) -

The "curse of dimensionality" occurs in high-dimensional data, leading to sparsity in the feature space for Naive Bayes. It causes computational complexity, data sparsity, and overfitting. Strategies to mitigate this include feature selection, reducing dimensions via PCA, LDA, or t-SNE, handling sparse data with techniques like Laplace smoothing, data preprocessing for normalization, scaling, and transformation, and using cross-validation and regularization to prevent overfitting. These approaches help reduce the curse of dimensionality and enhance the performance of Naive Bayes classifiers, especially in dealing with complex, high-dimensional datasets.

(c) -

The "zero-frequency problem" in Naive Bayes occurs when the classifier encounters an unseen attribute value during inference, leading to unreliable predictions. For instance, in text classification, if the word "innovation" appears in a test document but is absent from the training data, the standard Naive Bayes approach assigns it a zero probability, potentially causing misclassification.

To mitigate this, various strategies can be employed:

1. Laplace (Add-One) and Additive (Lidstone) Smoothing involves adding a small constant or a smoothing parameter to the count of each attribute value, preventing zero probabilities and improving robustness.

2. Good-turning smoothing estimates probabilities for unseen values based on the distribution

of seen values, providing better estimates for rare occurrences.

3. Backoff and Interpolation techniques combine probability estimates from higher-order and lower-order N-grams in language models, effectively handling unseen N-grams.

4. Creating a particular category for unknown values and assigning a small probability can help handle unseen matters during inference.

5. For continuous data, kernel density estimation can be used to estimate probabilities for unseen values.

The choice of strategy depends on the data's characteristics, with Laplace smoothing being a common and practical approach in many cases.

(d) -

When using Information Gain or similar metrics such as Gini Impurity or Gain Ratio to split decision tree nodes, a bias can arise due to differing attribute cardinalities. Attributes with higher cardinality often receive preferential treatment because they offer more opportunities for data splitting and enable finer distinctions between data points. This preference can lead to an imbalance where attributes with lower cardinality are overshadowed, potentially impacting the classification process.

For instance, when predicting customer churn in a telecom company, attributes like "CustomerID" and "Phone Number," with high cardinality due to unique values, may dominate the splitting process over an attribute like "Monthly Usage," which has lower cardinality. This can result in overfitting and the failure to capture essential patterns related to churn, like the correlation between "Monthly Usage" and churn.

To counteract this issue, alternative attribute selection criteria less sensitive to cardinality imbalances can be employed:

1. Gain Ratio: This criterion normalizes Information Gain by intrinsic attribute information, mitigating bias towards high-cardinality attributes and ensuring a balanced influence across attributes.

2. Gini Impurity: This metric measures misclassification probability and is less affected by cardinality differences, serving as a robust alternative for decision tree construction.

3. Random Forests: By aggregating predictions from multiple trees and utilizing random feature selection, Random Forests can mitigate cardinality-based biases through ensemble learning techniques.

4. Constraints on Max Leaf Nodes or Max Depth: Limiting the depth or number of leaf nodes in the tree can prevent overfitting and reduce the influence of high-cardinality attributes during tree construction.

In conclusion, decision trees are susceptible to biases caused by differing attribute cardinalities, particularly when employing default splitting criteria like Information Gain. By adopting appropriate attribute selection criteria and implementing tree regularization techniques, this bias can be alleviated, ensuring that decision trees focus on relevant attributes and capture meaningful patterns within the data.

2. (5 points) Rahul's decision-making process for playing a sport is based on the weather and certain conditions. Given that he prefers outdoor activities, if there is no rain, he will play outdoor. For outdoor activities, if more than 7 of his friends are playing, he will opt for football; otherwise, he will choose badminton. On the other hand, if it's indoor due to rain, he will play table tennis (TT) only if he can borrow TT rackets from a friend; otherwise, he will play pool.

(a) (1 marks) Draw decision tree for the given scenario and write the all possible out comes and their conditional probabilities expression.

(b) (1 mark) Rahul decides to rely on a weather prediction app that claims to accurately forecast 'Rainy' and 'Clear' days. On any given day, the app predicts 'Rainy' with a probability of 0.3 (30) and it predicts 'Clear' with a probability of 0.7 (70percent)The app's accuracy for predicting 'Rainy' days is 80 percent, and its accuracy for predicting 'Clear' days is 90percent. What is the probability that it's going to rain on a day, given that the app predicts 'Rainy'?"

(c) (1.5 marks) Rahul's friend Ram has asked Rahul to come along with him for gym (where he can go irrespective of weather conditions). But Rahul's likelihood of going to the gym depends on his mood. If he is in good mood then there is a 80 percent chance that he will go to the gym and 20 percent chance that that he will stick to his earlier plan whereas, if he is in bad mood there is a 40percent chance that he will go to the gym and 60 percent chance that he will stick to his earlier plan. At gym he will either do cardiological exercise or weight training, both of which are equally likely. Draw the new decision tree and write down all possible outcomes and their conditional probabilities (in expression form).

(d) (1.5 marks) The probability of Rahul having good mood is 0.6 and bad mood is 0.4, and his mood is determined by factor F (amount of sleep he had last night). Assume that Rahul had 7 hours of sleep on the night before he is making this decision (Given P(F = 7 | Good mood) = 0.7 and P(F = 7 | Bad mood) = 0.45). Find the probability of all the possible outcomes and state the most likely outcome.

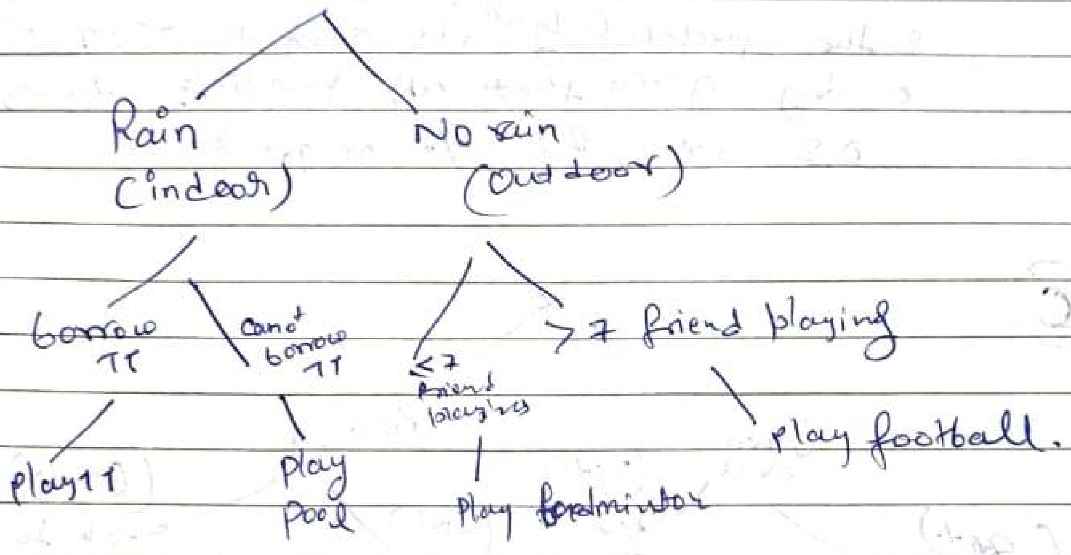
Solution::

Mohammad ShariQ
2020220

Assignment-2

Q2

(a)



expression

Play TT: P(Play TT | Rain, can borrow TT)

Play pool: P(Play pool | Rain, cannot borrow TT)

Play football: P((Play football) | No Rain, >7 friends playing)

Play badminton: P (play badminton | No Rain <=7 friends playing).

(b) P(Rainy) = 0.3 or 30%

P(clear) = 0.7 or 70%

Accuracy of rainy days → P(Predict Rainy day | Rainy) = 0.8 or 80%

Accuracy of clear days → P(predict Clear day | clear) = 0.9, or 90%

we find: P(Rainy day | predict Rainy day) =

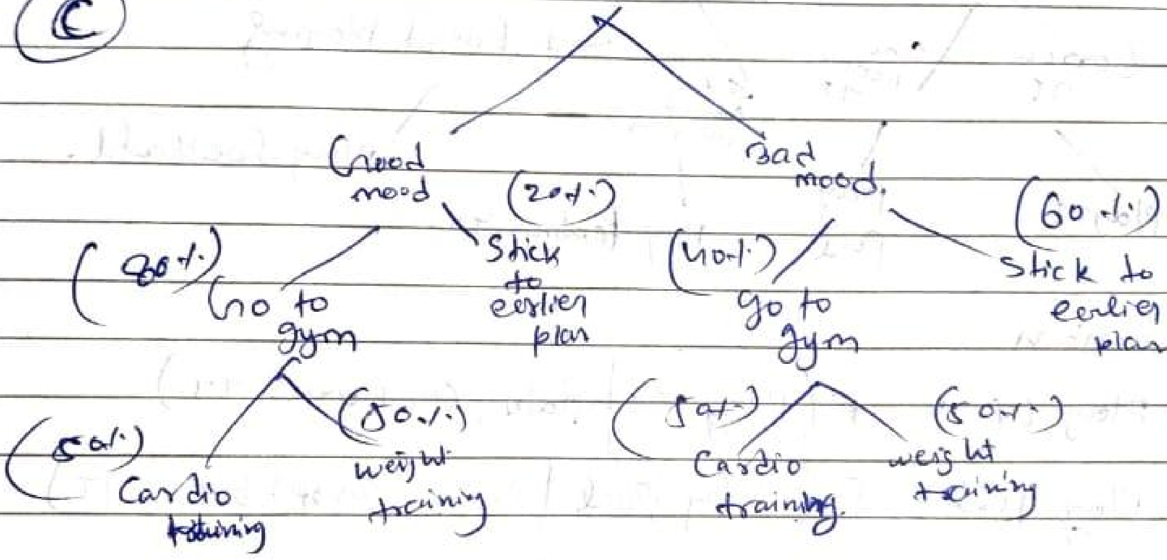
$$\frac{P(\text{Predict rainy day} \mid \text{rainy}) \times P(\text{rainy})}{P(\text{predict rainy day})}$$

$$= \frac{0.8 \times 0.3}{0.3} = 0.8$$

or 80% that its going to rain

So the probabilility is (80%) its going to rain on a day given that app predicts Rainy is 0.8 or 80% or greater.

(C)



→ P(cardio training | Goodmood, Gotogym) = 0.5

→ P(Weight training | Goodmood, go to gym) = 0.5.

→ P(stick to earlier plan | Good mood, Not go to gym) = 0.2

→ P(go to gym | Good mood) = 0.8

→ P(cardio training | Bad mood & gotogym) = 0.5

→ P(weight training | Badmood & gotogym) = 0.5

→ P(stick to earlier plan | Bad mood; notgotogym) = 0.6

→ P(gotogym | Badmood) = 0.4

Delta

(d) $P(F=7 \mid \text{Goodmood}) = 0.7$

$P(F=7 \mid \text{Bad mood}) = 0.45$

$P(F=7) = P(F=7 \mid G) \times P(G) + P(F=7 \mid B) \times P(B)$

$= 0.7 \times 0.6 + 0.45 \times 0.4 = 0.6$

$$P(G \mid F=7) = \frac{P(F=7 \mid G) \times P(G)}{P(F=7)}$$

$$= \frac{0.7 \times 0.6}{0.6} = 0.7$$

$$P(B \mid F=7) = \frac{P(F=7 \mid B) \times P(B)}{P(F=7)}$$

$$= \frac{0.45 \times 0.4}{0.6} = 0.3$$

hence most likely that Rahul is in good with 70% probability when he had 7 hours to sleep.

3. (15 points) Section B (Library Implementation)

Decision Tree and Random Forests

Perform classification task on the heart disease dataset using only the relevant attributes mentioned on the repository website.

Dataset: Heart Disease - UCI Machine Learning Repository

(a) (2 marks) Preprocess the dataset if required and perform Exploratory Data Analysis

(b) (1 marks) Split the dataset into train and test sets in the ratio 80:20

(c) (3 marks) Train decision trees using ‘entropy’ and ‘gini impurity’ as the splitting criterion and report the best criterion for attribute selection based on the accuracy scores.

(d) (4 marks) Now taking the best criterion for attribute selection in part c, perform hyperparameter search for the parameters min_samples_split and max_features using Grid Search. Select the best combination of the hyperparameters using the test data scores.

(e) (5 marks) Finally train a random forest classifier for the same dataset. Perform Grid Search for the parameters n_estimators, max_depth and min_samples_split. Report the best combination of hyperparameters and present the classification re port on the test data.

Code:

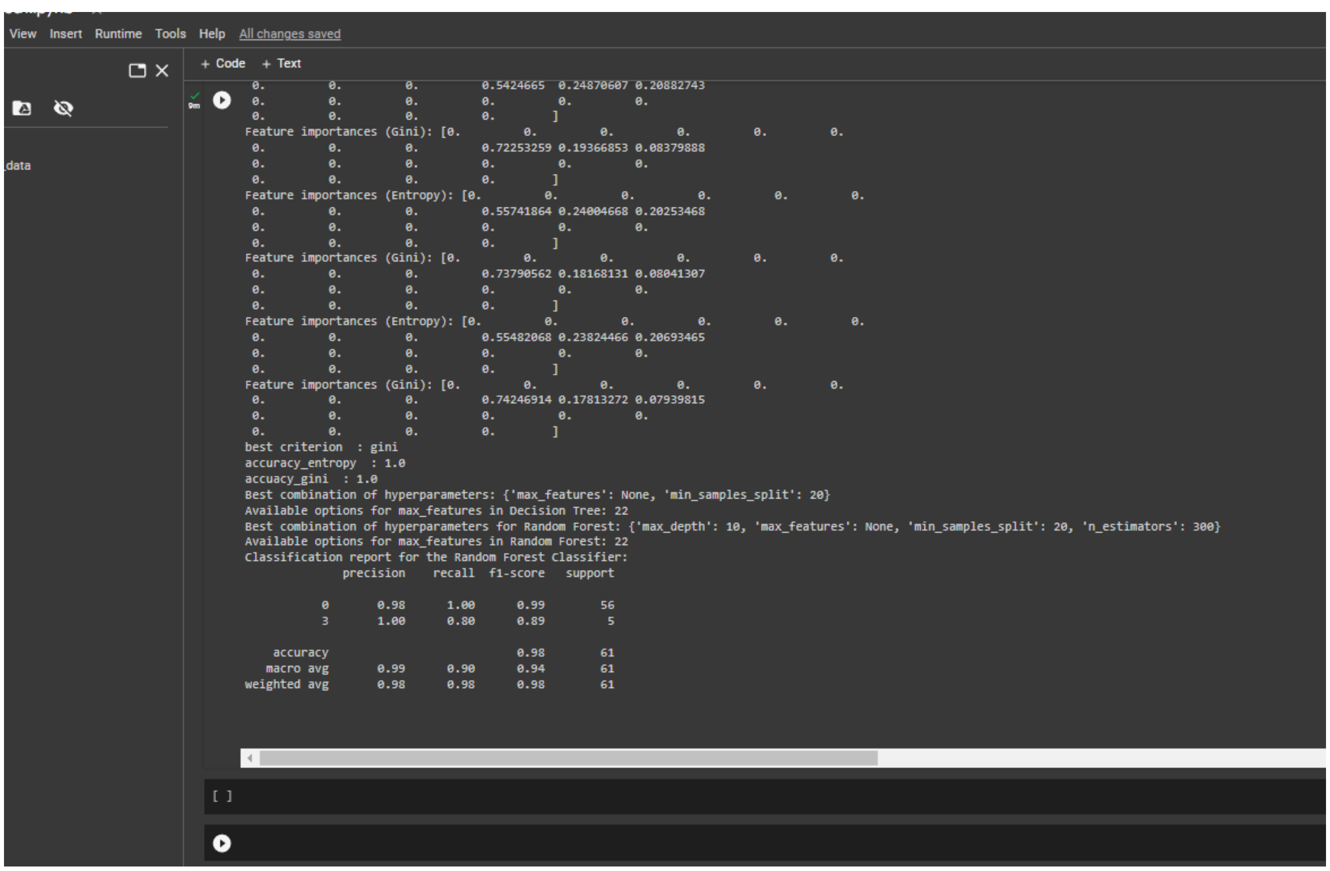
```
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import StratifiedKFold


def load_data_file(file_path):
    names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
   'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

    data = pd.read_csv(file_path, names=names)

    return data


def load_data():

                                                                     url                 =
   'https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/proce
   ssed.cleveland.data'

    names = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
   'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num']

    data = pd.read_csv(url, names=names)

    return data


    # Modify the handle_missing_values function

def handle_missing_values(data):

    data = data.replace('?', pd.NA)

    data = data.apply(pd.to_numeric, errors='coerce')

    data.fillna(data.mean(), inplace=True)  # Filling missing values with mean

    return data



def preprocess_data(data):

    data = pd.get_dummies(data, columns=['cp', 'restecg', 'slope', 'thal'])

    scaler = StandardScaler()

    scaled_data = scaler.fit_transform(data.drop('num', axis=1))

    data = pd.DataFrame(scaled_data, columns=data.columns[:-1])

    data['num'] = data['num'].astype(int)

    return data


def visualize_data(data):

    sns.pairplot(data, hue='num')
```

```python
    plt.title('Pairplot for the Dataset')
    plt.show()

    plt.figure(figsize=(12, 8))
    sns.heatmap(data.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Heatmap')
    plt.show()

def split_data(data):
    X = data.drop('num', axis=1)
    y = data['num']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)  # Implementing stratified sampling
    return X_train, X_test, y_train, y_test

def train_decision_trees(X_train, y_train, X_test, y_test):
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  # Implementing cross-validation
    dt_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
    dt_gini = DecisionTreeClassifier(criterion='gini', random_state=42)

    for train_index, test_index in skf.split(X_train, y_train):
        X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[test_index]
        y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[test_index]

        dt_entropy.fit(X_train_fold, y_train_fold)
        dt_gini.fit(X_train_fold, y_train_fold)

        accuracy_entropy = dt_entropy.score(X_val_fold, y_val_fold)
        accuracy_gini = dt_gini.score(X_val_fold, y_val_fold)
        print("Feature importances (Entropy):", dt_entropy.feature_importances_)
        print("Feature importances (Gini):", dt_gini.feature_importances_)

    if accuracy_entropy > accuracy_gini:
        best_criterion = 'entropy'
    else:
        best_criterion = 'gini'

    return best_criterion, accuracy_entropy, accuracy_gini




def grid_search_decision_trees(X_train, y_train, best_criterion):
    parameters_grid = {
        'min_samples_split': [20, 50, 100],
        'max_features': ['sqrt', 'log2', None]
    }

     grid_search = GridSearchCV(DecisionTreeClassifier(criterion=best_criterion,
   random_state=42), parameters_grid, cv=5)
    grid_search.fit(X_train, y_train)
    best_params = grid_search.best_params_

     max_features_options_dt = X_train.shape[1] if best_params['max_features'] is
   None else best_params['max_features']

    return best_params, max_features_options_dt

def grid_search_random_forest(X_train, y_train):

    randomf_param_grid = {
        'n_estimators': [50, 100, 300],

        'max_depth': [10, 20, 50],

        'min_samples_split': [50, 100, 20],

        'max_features': ['sqrt', 'log2', None]

    }


    rf_grid_search = GridSearchCV(RandomForestClassifier(random_state=42),


    randomf_param_grid, cv=5)


    rf_grid_search.fit(X_train, y_train)

    best_rf_params = rf_grid_search.best_params_

     max_features_options_rf = X_train.shape[1] if best_rf_params['max_features']
   is None else best_rf_params['max_features']


    return best_rf_params, max_features_options_rf


def train_random_forest(X_train, y_train, X_test, y_test, best_rf_params):

    rf_classifier = RandomForestClassifier(**best_rf_params, random_state=42)

    rf_classifier.fit(X_train, y_train)

    y_pred = rf_classifier.predict(X_test)

    print("Classification report for the Random Forest Classifier:")

    print(classification_report(y_test, y_pred))





if __name__ == '__main__':


  # file_path = 'your_local_file_path_here.csv'  # Replace with the correct file
   path

  #   data = load_data(file_path)
```

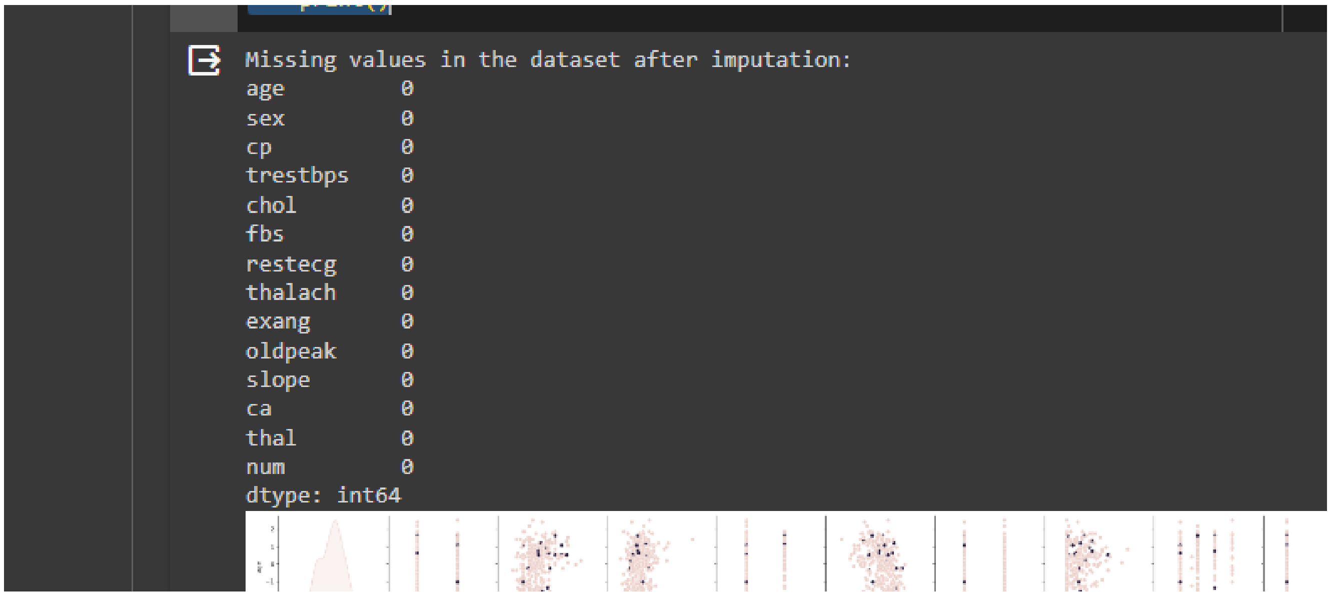
```
## loading data via online dataset
  data = load_data()
  print("Missing values in the dataset after imputation:")
  print(data.isnull().sum())

  data = handle_missing_values(data)
  data = preprocess_data(data)
  visualize_data(data)

  X_train, X_test, y_train, y_test = split_data(data)

  print("X-train data")
  print(X_train)
  print()
  print()
  print("X-test data")
  print(X_test)
  print()
  print()
  print("Y-test data")
  print(y_test)
  print()
  print()
  print("Y-train data")
  print(y_train)



  print()
  print()
```

```
 best_criterion,accuracy_entropy,accuracy_gini = train_decision_trees(X_train,
y_train, X_test, y_test)

 print(f"best criterion  : {best_criterion}")
 print(f"accuracy_entropy  : {accuracy_entropy}")
 print(f"accuacy_gini  : {accuracy_gini}")

  best_params, max_features_options_dt = grid_search_decision_trees(X_train,
y_train, best_criterion)
 print(f"Best combination of hyperparameters: {best_params}")
        print(f"Available options for max_features in Decision Tree:
{max_features_options_dt}")

  best_rf_params, max_features_options_rf = grid_search_random_forest(X_train,
y_train)
       print(f"Best combination of hyperparameters for Random Forest:
{best_rf_params}")
        print(f"Available options for max_features in Random Forest:
{max_features_options_rf}")

 train_random_forest(X_train, y_train, X_test, y_test, best_rf_params)
 print()
 print()
```

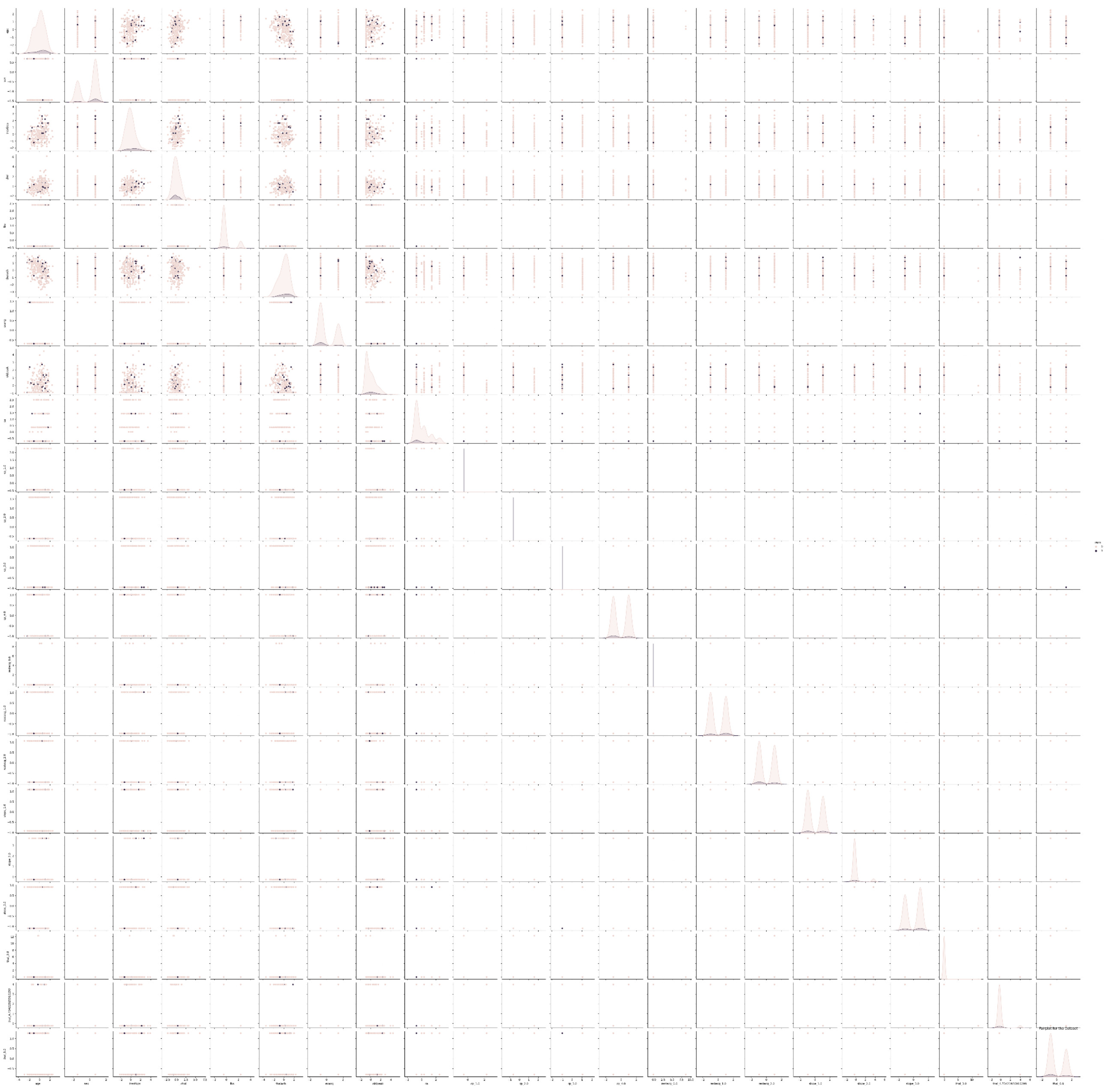
Results:

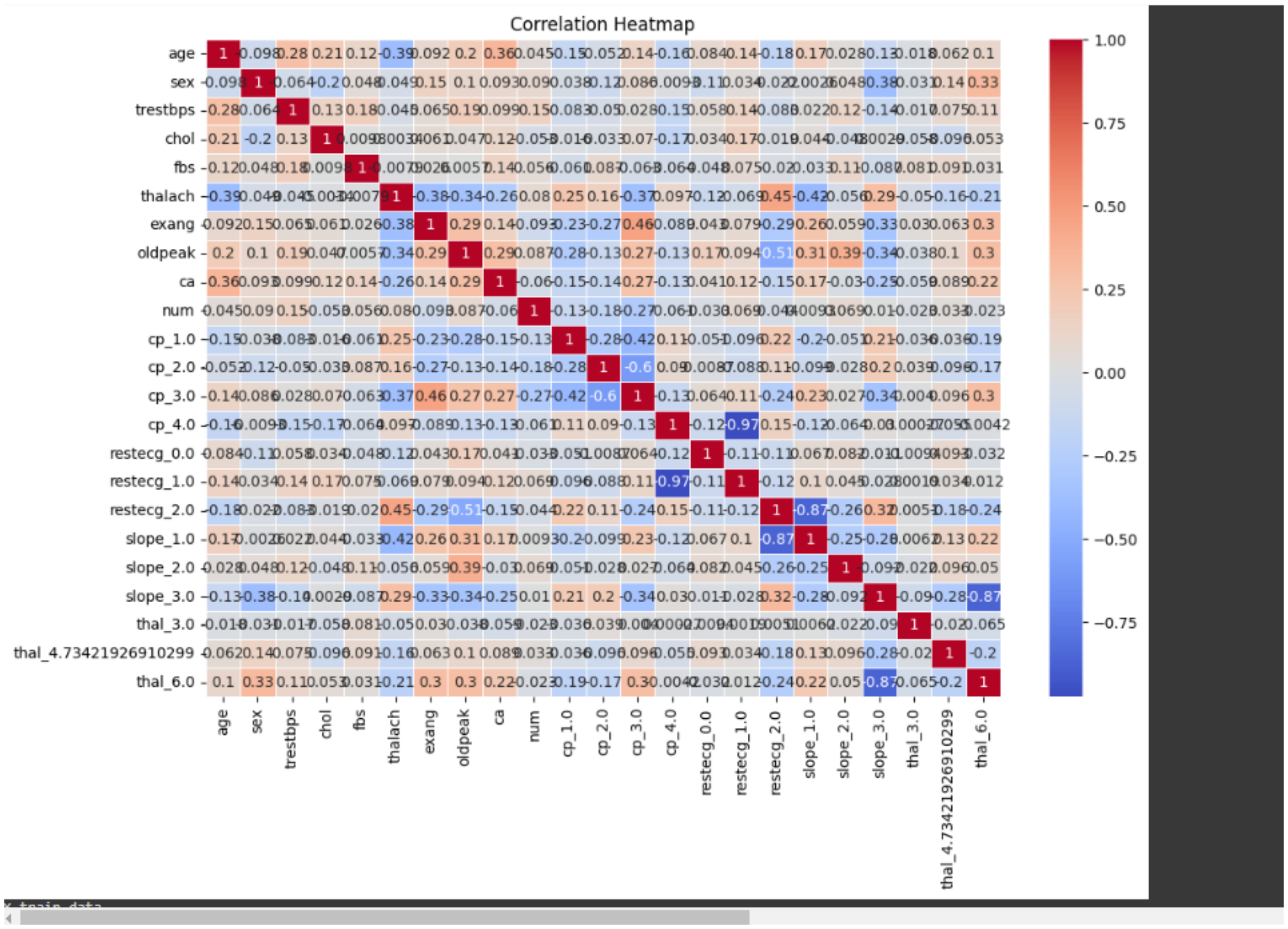


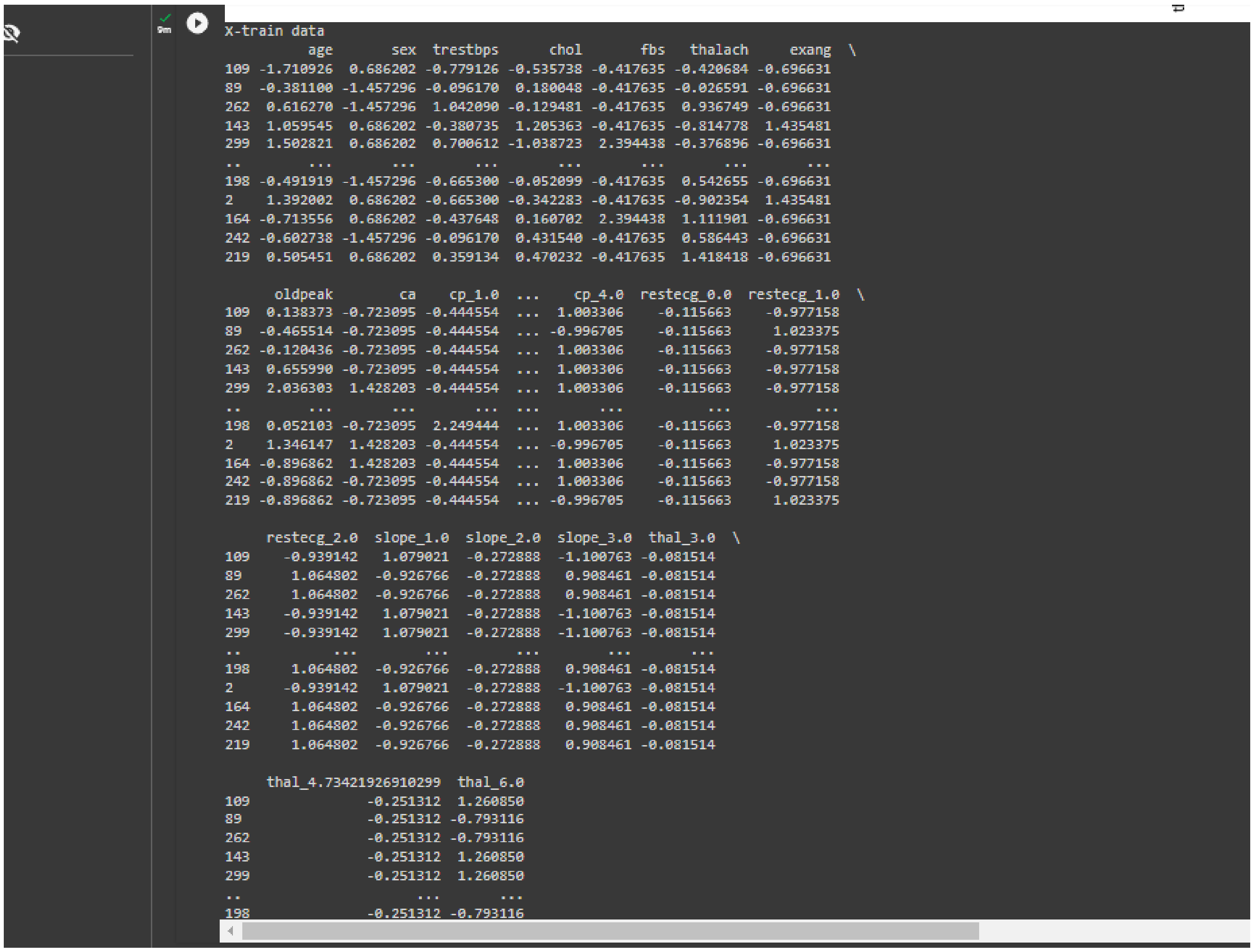
```
 0.         0.         0.         0.5424665  0.24870607 0.20882743
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Gini): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.72253259 0.19366853 0.08379888
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Entropy): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.55741864 0.24004668 0.20253468
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Gini): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.73790562 0.18168131 0.08041307
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Entropy): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.55482068 0.23824466 0.20693465
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Gini): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.74246914 0.17813272 0.07939815
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
best criterion  : gini
accuracy_entropy  : 1.0
accuacy_gini  : 1.0
Best combination of hyperparameters: {'max_features': None, 'min_samples_split': 20}
Available options for max_features in Decision Tree: 22
Best combination of hyperparameters for Random Forest: {'max_depth': 10, 'max_features': None, 'min_samples_split': 20, 'n_estimators': 300}
Available options for max_features in Random Forest: 22
Classification report for the Random Forest Classifier:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        56
           3       1.00      0.80      0.89         5

    accuracy                           0.98        61
   macro avg       0.99      0.90      0.94        61
weighted avg       0.98      0.98      0.98        61
```



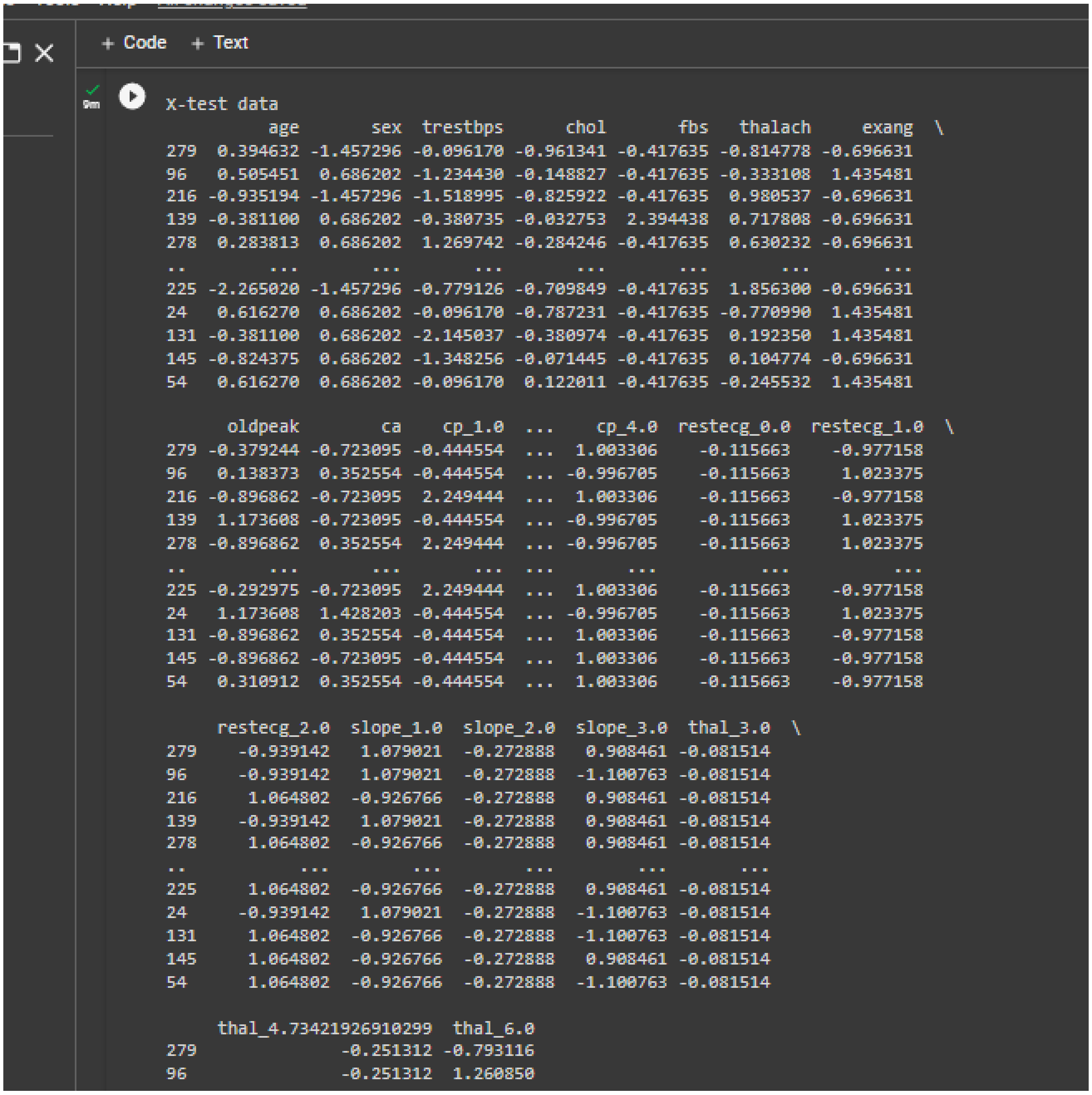
```
Missing values in the dataset after imputation:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
num         0
dtype: int64
```


Pairplot for the Dataset

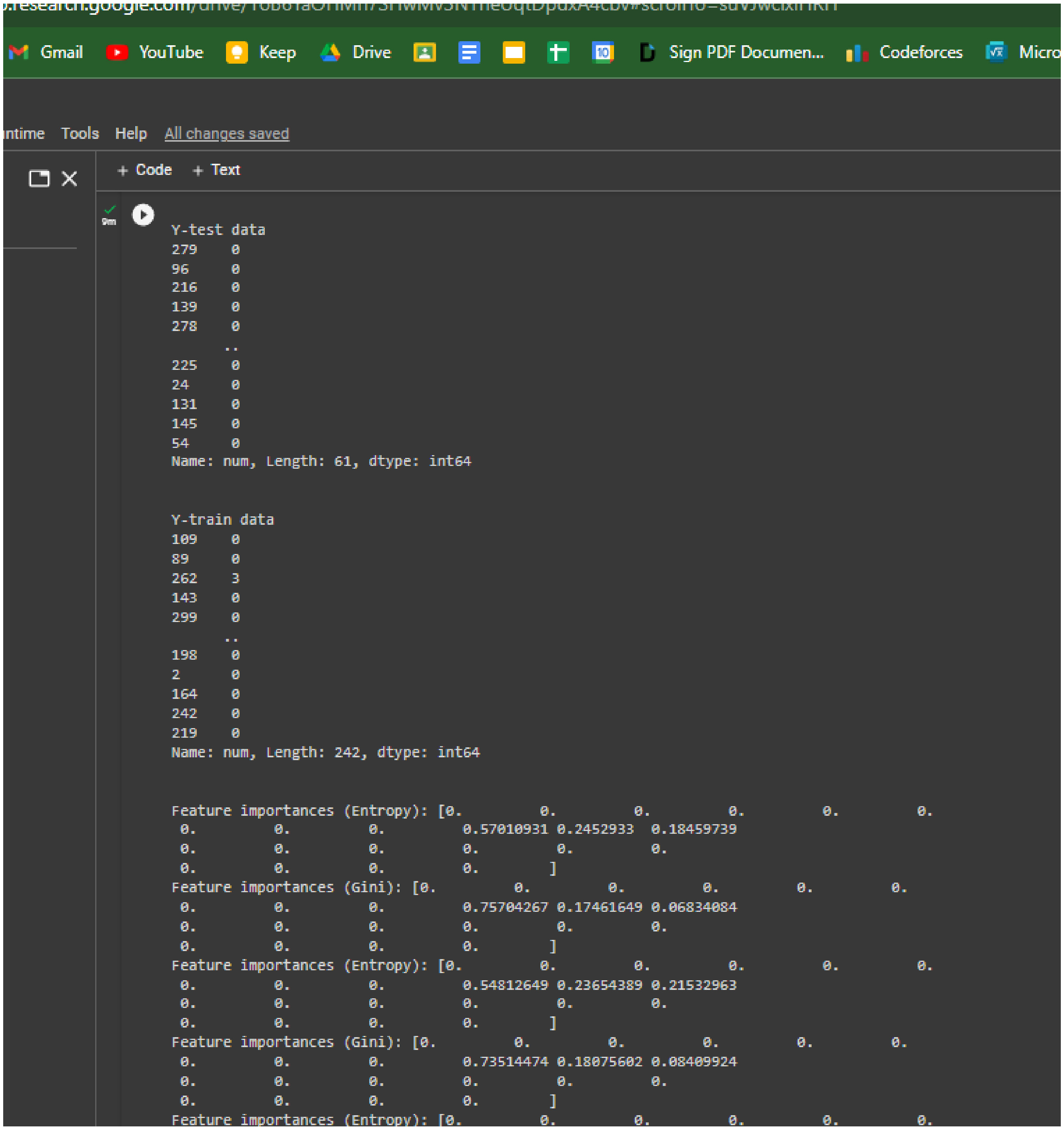

Correlation Heatmap
age
sex
trestbps
chol
fbs
thalach
exang
oldpeak
ca
num
cp_1.0
cp_2.0
cp_3.0
cp_4.0
restecg_0.0
restecg_1.0
restecg_2.0
slope_1.0
slope_2.0
slope_3.0
thal_3.0
thal_4.73421926910299
thal_6.0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



```
X-train data
          age       sex  trestbps      chol       fbs   thalach     exang  \
109 -1.710926  0.686202 -0.779126 -0.535738 -0.417635 -0.420684 -0.696631
89  -0.381100 -1.457296 -0.096170  0.180048 -0.417635 -0.026591 -0.696631
262  0.616270 -1.457296  1.042090 -0.129481 -0.417635  0.936749 -0.696631
143  1.059545  0.686202 -0.380735  1.205363 -0.417635 -0.814778  1.435481
299  1.502821  0.686202  0.700612 -1.038723  2.394438 -0.376896 -0.696631
..        ...       ...       ...       ...       ...       ...       ...
198 -0.491919 -1.457296 -0.665300 -0.052099 -0.417635  0.542655 -0.696631
2    1.392002  0.686202 -0.665300 -0.342283 -0.417635 -0.902354  1.435481
164 -0.713556  0.686202 -0.437648  0.160702  2.394438  1.111901 -0.696631
242 -0.602738 -1.457296 -0.096170  0.431540 -0.417635  0.586443 -0.696631
219  0.505451  0.686202  0.359134  0.470232 -0.417635  1.418418 -0.696631

      oldpeak        ca    cp_1.0  ...    cp_4.0  restecg_0.0  restecg_1.0  \
109  0.138373 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
89  -0.465514 -0.723095 -0.444554  ... -0.996705    -0.115663     1.023375
262 -0.120436 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
143  0.655990 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
299  2.036303  1.428203 -0.444554  ...  1.003306    -0.115663    -0.977158
..        ...       ...       ...  ...       ...          ...          ...
198  0.052103 -0.723095  2.249444  ...  1.003306    -0.115663    -0.977158
2    1.346147  1.428203 -0.444554  ... -0.996705    -0.115663     1.023375
164 -0.896862  1.428203 -0.444554  ...  1.003306    -0.115663    -0.977158
242 -0.896862 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
219 -0.896862 -0.723095 -0.444554  ... -0.996705    -0.115663     1.023375

     restecg_2.0  slope_1.0  slope_2.0  slope_3.0  thal_3.0  \
109    -0.939142   1.079021  -0.272888  -1.100763 -0.081514
89      1.064802  -0.926766  -0.272888   0.908461 -0.081514
262     1.064802  -0.926766  -0.272888   0.908461 -0.081514
143    -0.939142   1.079021  -0.272888  -1.100763 -0.081514
299    -0.939142   1.079021  -0.272888  -1.100763 -0.081514
..           ...        ...        ...        ...       ...
198     1.064802  -0.926766  -0.272888   0.908461 -0.081514
2      -0.939142   1.079021  -0.272888  -1.100763 -0.081514
164     1.064802  -0.926766  -0.272888   0.908461 -0.081514
242     1.064802  -0.926766  -0.272888   0.908461 -0.081514
219     1.064802  -0.926766  -0.272888   0.908461 -0.081514

     thal_4.73421926910299  thal_6.0
109              -0.251312  1.260850
89               -0.251312 -0.793116
262              -0.251312 -0.793116
143              -0.251312  1.260850
299              -0.251312  1.260850
..                     ...       ...
198              -0.251312 -0.793116
```



```
X-test data
          age       sex  trestbps      chol       fbs   thalach     exang  \
279  0.394632 -1.457296 -0.096170 -0.961341 -0.417635 -0.814778 -0.696631
96   0.505451  0.686202 -1.234430 -0.148827 -0.417635 -0.333108  1.435481
216 -0.935194 -1.457296 -1.518995 -0.825922 -0.417635  0.980537 -0.696631
139 -0.381100  0.686202 -0.380735 -0.032753  2.394438  0.717808 -0.696631
278  0.283813  0.686202  1.269742 -0.284246 -0.417635  0.630232 -0.696631
..        ...       ...       ...       ...       ...       ...       ...
225 -2.265020 -1.457296 -0.779126 -0.709849 -0.417635  1.856300 -0.696631
24   0.616270  0.686202 -0.096170 -0.787231 -0.417635 -0.770990  1.435481
131 -0.381100  0.686202 -2.145037 -0.380974 -0.417635  0.192350  1.435481
145 -0.824375  0.686202 -1.348256 -0.071445 -0.417635  0.104774 -0.696631
54   0.616270  0.686202 -0.096170  0.122011 -0.417635 -0.245532  1.435481

      oldpeak        ca    cp_1.0  ...    cp_4.0  restecg_0.0  restecg_1.0  \
279 -0.379244 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
96   0.138373  0.352554 -0.444554  ... -0.996705    -0.115663     1.023375
216 -0.896862 -0.723095  2.249444  ...  1.003306    -0.115663    -0.977158
139  1.173608 -0.723095 -0.444554  ... -0.996705    -0.115663     1.023375
278 -0.896862  0.352554  2.249444  ... -0.996705    -0.115663     1.023375
..        ...       ...       ...  ...       ...          ...          ...
225 -0.292975 -0.723095  2.249444  ...  1.003306    -0.115663    -0.977158
24   1.173608  1.428203 -0.444554  ... -0.996705    -0.115663     1.023375
131 -0.896862  0.352554 -0.444554  ...  1.003306    -0.115663    -0.977158
145 -0.896862 -0.723095 -0.444554  ...  1.003306    -0.115663    -0.977158
54   0.310912  0.352554 -0.444554  ...  1.003306    -0.115663    -0.977158

     restecg_2.0  slope_1.0  slope_2.0  slope_3.0  thal_3.0  \
279    -0.939142   1.079021  -0.272888   0.908461 -0.081514
96     -0.939142   1.079021  -0.272888  -1.100763 -0.081514
216     1.064802  -0.926766  -0.272888   0.908461 -0.081514
139    -0.939142   1.079021  -0.272888   0.908461 -0.081514
278     1.064802  -0.926766  -0.272888   0.908461 -0.081514
..           ...        ...        ...        ...       ...
225     1.064802  -0.926766  -0.272888   0.908461 -0.081514
24     -0.939142   1.079021  -0.272888  -1.100763 -0.081514
131     1.064802  -0.926766  -0.272888  -1.100763 -0.081514
145     1.064802  -0.926766  -0.272888   0.908461 -0.081514
54      1.064802  -0.926766  -0.272888  -1.100763 -0.081514

     thal_4.73421926910299  thal_6.0
279              -0.251312 -0.793116
96               -0.251312  1.260850
```



```
Y-test data
279    0
96     0
216    0
139    0
278    0
      ..
225    0
24     0
131    0
145    0
54     0
Name: num, Length: 61, dtype: int64


Y-train data
109    0
89     0
262    3
143    0
299    0
      ..
198    0
2      0
164    0
242    0
219    0
Name: num, Length: 242, dtype: int64


Feature importances (Entropy): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.57010931 0.2452933  0.18459739
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Gini): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.75704267 0.17461649 0.06834084
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Entropy): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.54812649 0.23654389 0.21532963
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Gini): [0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.73514474 0.18075602 0.08409924
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
Feature importances (Entropy): [0.         0.         0.         0.         0.         0.
```

4. (15 points) Section C (Algorithm implementation using packages)

1. Implement a Decision Tree from Scratch for Classification. Create a Decision Tree classifier from scratch using the NumPy and Pandas libraries. Design a class called MyDecisionTree (specifically for solving classification problems.)

(a) (10 marks)Include the following functions in MyDecisionTree class:

(a) cost_function(): Develop a cost function that could be either the Gini index or Information gain. This function should compute the impurity of a node or the gain achieved by a potential split.
Gini index: 1- sum(proportion)2
proportion = number of values/count of rows

(b) make_split(): Define the basic mechanism of splitting a node in the Decision Tree such that it selects the best feature and value to split on.

(c) max_depth(): Define the maximum depth of the tree.

(d) Pruning (optional): define a function to remove branches that doesn't con tribute much for improving accuracy.

(e) predict()

(f) score(): Create a function to evaluate the DT (performance metric) Provide proper documentation (well-commented codes)

(b) (5 marks) Additionally, you may demonstrate the effectiveness of your MyDecision Tree class by training and evaluating it on the given dataset.

Code:

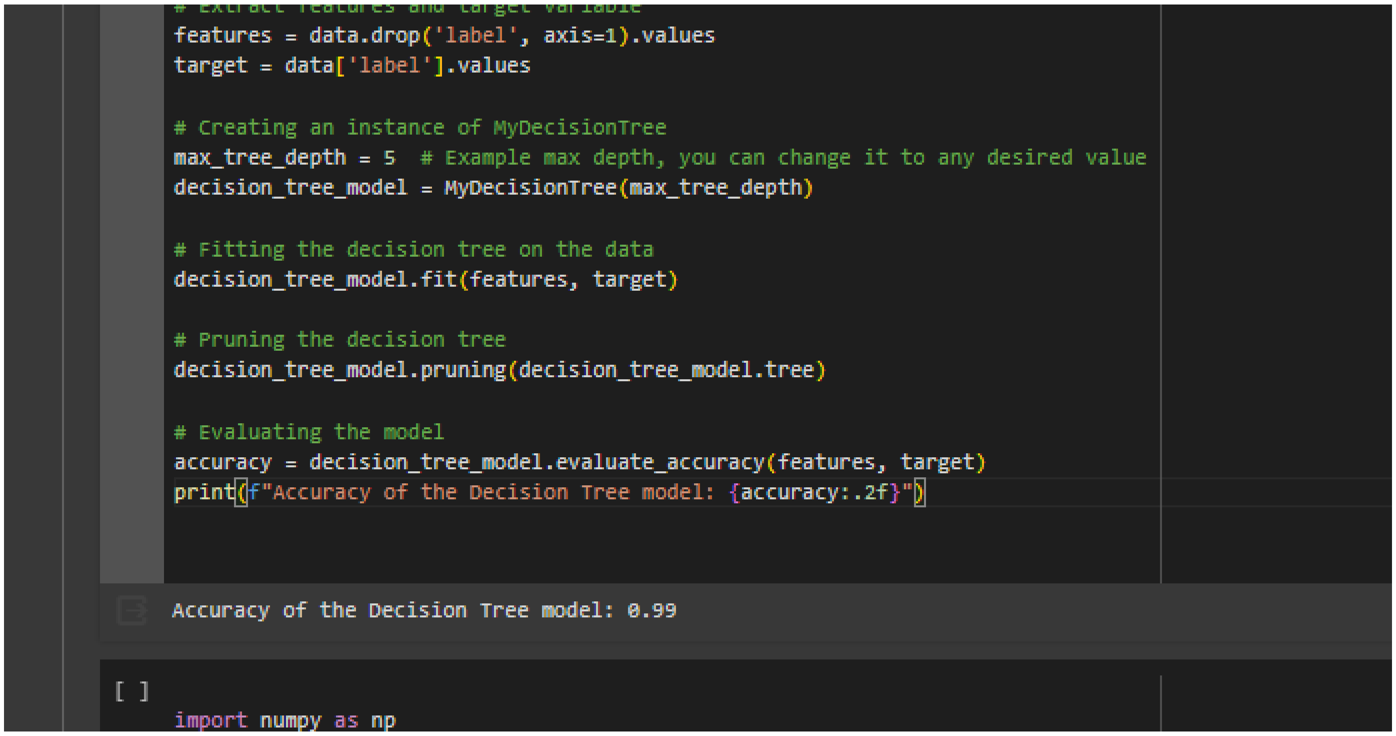
```
import numpy as np

import pandas as pd


class MyDecisionTree:

    def __init__(self, max_depth):

        self.max_depth = max_depth


    def gini_index(self, groups, classes):

        total_instances = float(sum([len(group) for group in groups]))
```

```python
        gini = 0.0
        for group in groups:
            group_size = float(len(group))
            if group_size == 0:
                continue
            score = 0.0
            for class_val in classes:
                p = [row[-1] for row in group].count(class_val) / group_size
                score += p * p
            gini += (1.0 - score) * (group_size / total_instances)
        return gini

    def make_split(self, dataset):
        class_values = list(set(row[-1] for row in dataset))
        best_index, best_value, best_score, best_groups = 999, 999, 999, None
        for index in range(len(dataset[0]) - 1):
            for row in dataset:
                groups = self.test_split(index, row[index], dataset)
                gini = self.gini_index(groups, class_values)
                if gini < best_score:
                    best_index, best_value, best_score, best_groups = index, row[index], gini, groups
        return {'index': best_index, 'value': best_value, 'groups': best_groups}



    def split(self, node, max_depth, min_size, depth):
        left, right = node['groups']
        del (node['groups'])
        if not left or not right:
            node['left'] = node['right'] = self.to_terminal(left + right)
            return
        if depth >= max_depth:
            node['left'], node['right'] = self.to_terminal(left),
    self.to_terminal(right)
            return
        if len(left) <= min_size:
            node['left'] = self.to_terminal(left)
        else:
            node['left'] = self.make_split(left)
            self.split(node['left'], max_depth, min_size, depth + 1)
        if len(right) <= min_size:
            node['right'] = self.to_terminal(right)
        else:
            node['right'] = self.make_split(right)
            self.split(node['right'], max_depth, min_size, depth + 1)

    def build_tree(self, train, max_depth, min_size):
        root = self.make_split(train)
        self.split(root, max_depth, min_size, 1)
        return root

    def test_split(self, index, value, dataset):
        left, right = list(), list()
        for row in dataset:
            if row[index] < value:
                left.append(row)
            else:
                right.append(row)
        return left, right

    def to_terminal(self, group):
        outcomes = [row[-1] for row in group]
        return max(set(outcomes), key=outcomes.count)
```

```python
def fit(self, features, target):
    dataset = np.column_stack((features, target))
    self.tree = self.build_tree(dataset, self.max_depth, 1)


def pruning(self, tree, depth=0):
    if 'groups' in tree:
        if not tree['left'] or not tree['right']:
            tree['left'] = tree['right'] = self.to_terminal(tree['groups'])
        if depth >= self.max_depth:
            tree['left'] = self.to_terminal(tree['groups'][0])
            tree['right'] = self.to_terminal(tree['groups'][1])
        if isinstance(tree['left'], dict):
            self.pruning(tree['left'], depth + 1)
        if isinstance(tree['right'], dict):
            self.pruning(tree['right'], depth + 1)


def predict(self, node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return self.predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return self.predict(node['right'], row)
        else:
            return node['right']
    def evaluate_accuracy(self, features, target):
        predictions = []
        for row in features:
            prediction = self.predict(self.tree, row)
            predictions.append(prediction)
        predictions = np.array(predictions)
        accuracy = np.mean(predictions == target)
        return accuracy


# Load the dataset

data = pd.read_csv('/content/Thyroid data - Sheet1.csv')

# Preprocess the data

# Mapping the classes to binary values (0 and 1)
data['label'] = data['label'].map({'negative': 0, 'hyperthyroid': 1, 'T3 toxic':
   1, 'goitre': 1})

# Extract features and target variable
features = data.drop('label', axis=1).values
target = data['label'].values

# Creating an instance of MyDecisionTree
max_tree_depth = 5
decision_tree_model = MyDecisionTree(max_tree_depth)
```

```
# Fitting the decision tree on the data
decision_tree_model.fit(features, target)


# Pruning the decision tree
decision_tree_model.pruning(decision_tree_model.tree)


# Evaluating the model
accuracy = decision_tree_model.evaluate_accuracy(features, target)
print(f"Accuracy of the Decision Tree model: {accuracy:.2f}")
```

Result:



```
# Extract features and target variable
features = data.drop('label', axis=1).values
target = data['label'].values

# Creating an instance of MyDecisionTree
max_tree_depth = 5  # Example max depth, you can change it to any desired value
decision_tree_model = MyDecisionTree(max_tree_depth)

# Fitting the decision tree on the data
decision_tree_model.fit(features, target)

# Pruning the decision tree
decision_tree_model.pruning(decision_tree_model.tree)

# Evaluating the model
accuracy = decision_tree_model.evaluate_accuracy(features, target)
print(f"Accuracy of the Decision Tree model: {accuracy:.2f}")
```

```
Accuracy of the Decision Tree model: 0.99
```

```
[ ]
import numpy as np
```