CSE 343 : Machine Learning
Assignment 2 Decision Trees, Random Forests and Perceptron

Section A (Theory)

1.

(a) In a Random Forest, a balance between correlation and diversity among trees is key for optimal performance. Diversity ensures that trees make different errors which cancel out when aggregated, improving model accuracy. Correlation ensures that trees capture the true patterns in data, but too much leads to overfitting. The goal is to have trees that are somewhat correlated to capture data patterns but diverse enough to reduce overfitting and improve generalization.

(b) The "curse of dimensionality" in Naive Bayes refers to the problem that arises when the feature space is very high-dimensional. In such cases, the probability estimates may become unreliable due to the sparsity of data. To mitigate this, one can use dimensionality reduction techniques, feature selection, or smoothing techniques like Laplace smoothing.

(c) Naive Bayes may encounter problems with zero frequency if a categorical variable has a category in the test data set which was not observed in the training data set. This would lead to a zero probability estimate and thus an inability to make a prediction. To handle this, techniques such as smoothing (e.g., Laplace or Lidstone smoothing) can be used to assign a small non-zero probability to unseen features.

(d) Using Information Gain for splitting nodes in a decision tree can indeed be biased towards attributes with more levels (high cardinality), as they can result in more complex trees that overfit the data. An alternative criterion for attribute selection is Gini impurity, which is less biased towards attributes with more categories and often results in simpler trees.

2.

(a) Rahul's decision-making process can be visualized as a binary decision tree. To create a decision tree for Rahul's decision-making process, we need to consider the different conditions and their outcomes.
Here's how we can structure the decision tree:

First decision point: Is it raining?

- If No (let's denote the probability of no rain as P(No Rain)), go to outdoor activities.
- If Yes (probability of rain is then P(Rain)=1-P(No Rain)), go to indoor activities.

If outdoor:
Second decision point: Are more than 7 friends playing?

- If Yes (let's denote the probability of more than 7 friends playing as P(>7 Friends)), play football.
- If No (probability of 7 or fewer friends is P(≤7 Friends)=1-P(>7 Friends)), play badminton.

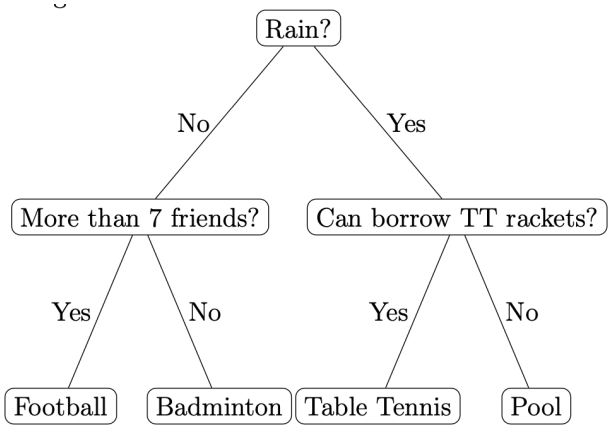
If indoor:
Third decision point: Can borrow TT rackets?

- If Yes (denote this probability as P(TT Rackets Available)), play table tennis.

- If No (probability of TT rackets not available is P(TT Rackets Not Available)=1−P(TT Rackets Available)), play pool.

Given these conditions, the possible outcomes and their conditional probabilities would be:

Play football outdoors: $P(\text{Football}) = P(\text{No Rain}) \cdot P(>7 \text{ Friends})$
Play badminton outdoors: $P(\text{Badminton}) = P(\text{No Rain}) \cdot P(\leq 7 \text{ Friends})$
Play table tennis indoors: $P(\text{Table Tennis}) = P(\text{Rain}) \cdot P(\text{TT Rackets Available})$
Play pool indoors: $P(\text{Pool}) = P(\text{Rain}) \cdot P(\text{TT Rackets Not Available})$



(b)

Let's define the events:

- R = It is raining.
- NR = It is not raining.
- PR = The app predicts 'Rainy'.
- PC = The app predicts 'Clear'.

Given:

- P(PR) = 0.3 (prob that the app predicts 'Rainy').
- P(PC) = 0.7 (prob that the app predicts 'Clear').
- P(PR|R) = 0.8 (prob that the app predicts 'Rainy' when it is raining).

- P(PC|NR) = 0.9 (prob that the app predicts 'Clear' when it is not raining).

Finding P(R|PR), the prob that it is actually raining given that the app predicts rain.

By total probability, P(PR) is given by:

$P(PR) = P(PR|R) \cdot P(R) + P(PR|NR) \cdot P(NR)$

But, $P(PR|NR) = 1 - P(PC|NR)$

$0.3 = 0.8 \cdot P(R) + 0.1 \cdot P(NR)$

And since $P(R) + P(NR) = 1$,

$0.3 = 0.8 \cdot P(R) + 0.1 \cdot (1 - P(R))$

$0.3 = 0.8 \cdot P(R) + 0.1 - 0.1 \cdot P(R)$ $0.3 = 0.7 \cdot P(R) + 0.1$

$0.2 = 0.7 \cdot P(R)$

$P(R) = 0.2/0.7$

$P(R) \approx 0.2857$

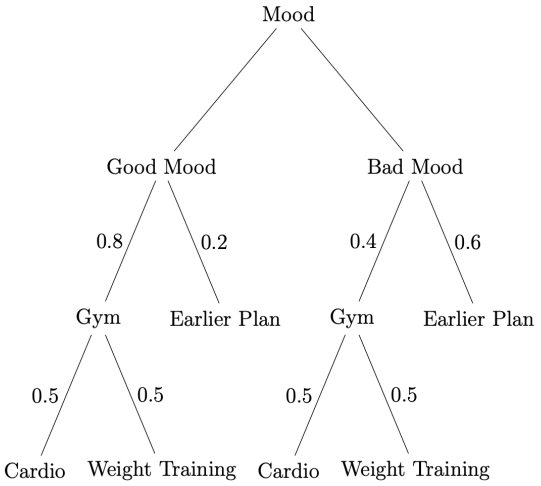
Now we can use Bayes' theorem to find P(R|PR):

$P(R|PR) = (P(PR|R) \cdot P(R)) / P(PR)$

$P(R|PR) = (0.8 \cdot 0.2857) \; 0.3 \approx 0.2286 / \; 0.3 \approx 0.762$

If the weather app forecasts rain, there is a 76.2% chance that it will indeed be a rainy day.

(c) Decision tree



Possible Outcomes and Their Conditional Probabilities:

- P(Go to Gym | Good Mood) = 0.8
- P(Stick to Earlier Plan | Good Mood) = 0.2
- P(Do Cardio | Go to Gym and Good Mood) = P(Do Weight Training | Go to Gym and Good Mood) = 0.5
- P(GotoGym|BadMood)=0.4
- P(Stick to Earlier Plan | Bad Mood) = 0.6
- P(Do Cardio | Go to Gym and Bad Mood) = P(Do Weight Training | Go to Gym and Bad Mood) = 0.5

(d) Given that Rahul had 7 hours of sleep, we have the following probabilities:

P (Good mood) = 0.6, P (Bad mood) = 0.4, P (F = 7|Good mood) = 0.7,

P (F = 7|Bad mood) = 0.45.

Firstly, we calculate the total probability of F = 7:

P(F = 7) = P(F = 7|Good mood) · P(Good mood) + P(F = 7|Bad mood) · P(Bad mood), P(F =7)=0.7·0.6+0.45·0.4,

P(F =7)=0.42+0.18 = 0.6.

Using Bayes' theorem,

P(Good mood|F = 7) = (P(F = 7|Good mood) · P(Good mood))/ P(F = 7)

P(Good mood|F = 7) = (0.7 · 0.6) /0.6

P (Good mood|F = 7) = 0.7.

The conditional probability of Rahul being in a bad mood given F = 7 is:

P(Bad mood|F = 7) = (P(F = 7|Bad mood) · P(Bad mood))/ P(F = 7)

P(Bad mood|F = 7) = (0.45 · 0.4)/ 0.6

P (Bad mood|F = 7) = 0.3.

Considering the probabilities of Rahul going to the gym based on his mood:

P (Gym|Good mood) = 0.8
P (Gym|Bad mood) = 0.4.

The total probability that Rahul will go to the gym is:
P (Gym) = P (Gym|Good mood) · P (Good mood|F = 7) + P (Gym|Bad mood) · P (Bad mood|F = 7)

P (Gym) = 0.8 · 0.7 + 0.4 · 0.3 = 0.68.

Now we calculate the probability that Rahul will stick to his earlier plan:
P (Earlier Plan) = 0.2 · 0.7 + 0.6 · 0.3 = 0.32.

At the gym, Rahul has two options: cardiovascular exercise or weight training. Since he's just as inclined to choose one as the other, the chance of him picking any one of these activities is half the overall likelihood of him going to the gym:
P (Cardio|Gym) = (½) · P (Gym), P (Cardio) = 0.34.

Similarly, the probability of doing weight training is the same as for cardio:
P (Weight Training|Gym) = P (Cardio|Gym),
P (Weight Training) = P (Cardio),
P (Weight Training) = 0.34.

Rahul is 32% likely to adhere to his original plan. There's a 34% probability he'll engage in cardio workouts and an equal 34% likelihood he'll opt for weight lifting when he's at the gym. The overall probability that he will end up going to the gym stands at 68%.

Section B (Library Implementation)

2.

(a) **Preprocessing**

- Loading the dataset
- Handling the missing values
  - Missing values were found in ‘ca’ and ‘thal’ columns. For ‘ca’ missing values were replaced with median value (‘ca’ being numerical) being and for ‘thal’ the missing values were replaced with mode (‘thal’ being categorical). This process is called **Imputation.**
- Converting the categorical variables that are represented as numerical types, to categorical types for analysis and the target variable 'num' was adjusted such that classes 1 to 4 are now represented as 1 (positive class) and class 0 is represented as 0 (negative class).

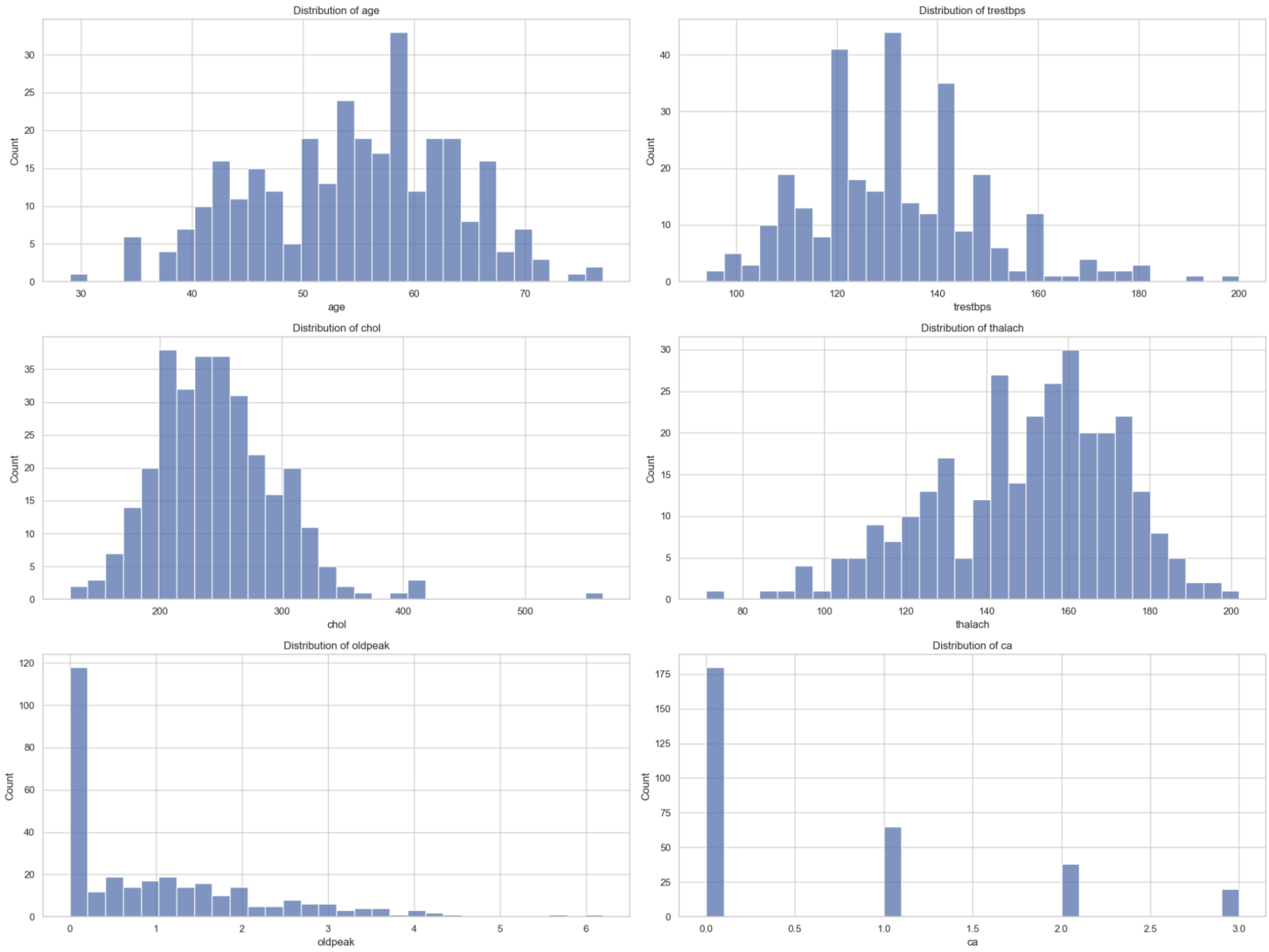
Exploratory Data Analysis

A general summary of the data

- The sex column has 2 unique categories, with the mode being 1.0 (likely representing males).
- The cp (chest pain type) column has 4 unique categories, with the most frequent being 4.0.
- The fbs (fasting blood sugar) column has 2 unique categories, with the mode being 0.0 (likely representing a fasting blood sugar below 120 mg/dl).
- The restecg (resting electrocardiographic results) column has 3 unique categories, with the most frequent being 0.0.
- The exang (exercise-induced angina) column has 2 unique categories, with the mode being 0.0 (likely representing 'no').
- The slope (the slope of the peak exercise ST segment) column has 3 unique categories, with the most frequent being 1.0.
- The thal column has 3 unique categories, with the most frequent being 3.0 (which might represent a specific thalassemia trait).

- The num (target variable) column has been converted into 2 classes with the most frequent being 0 (negative class).

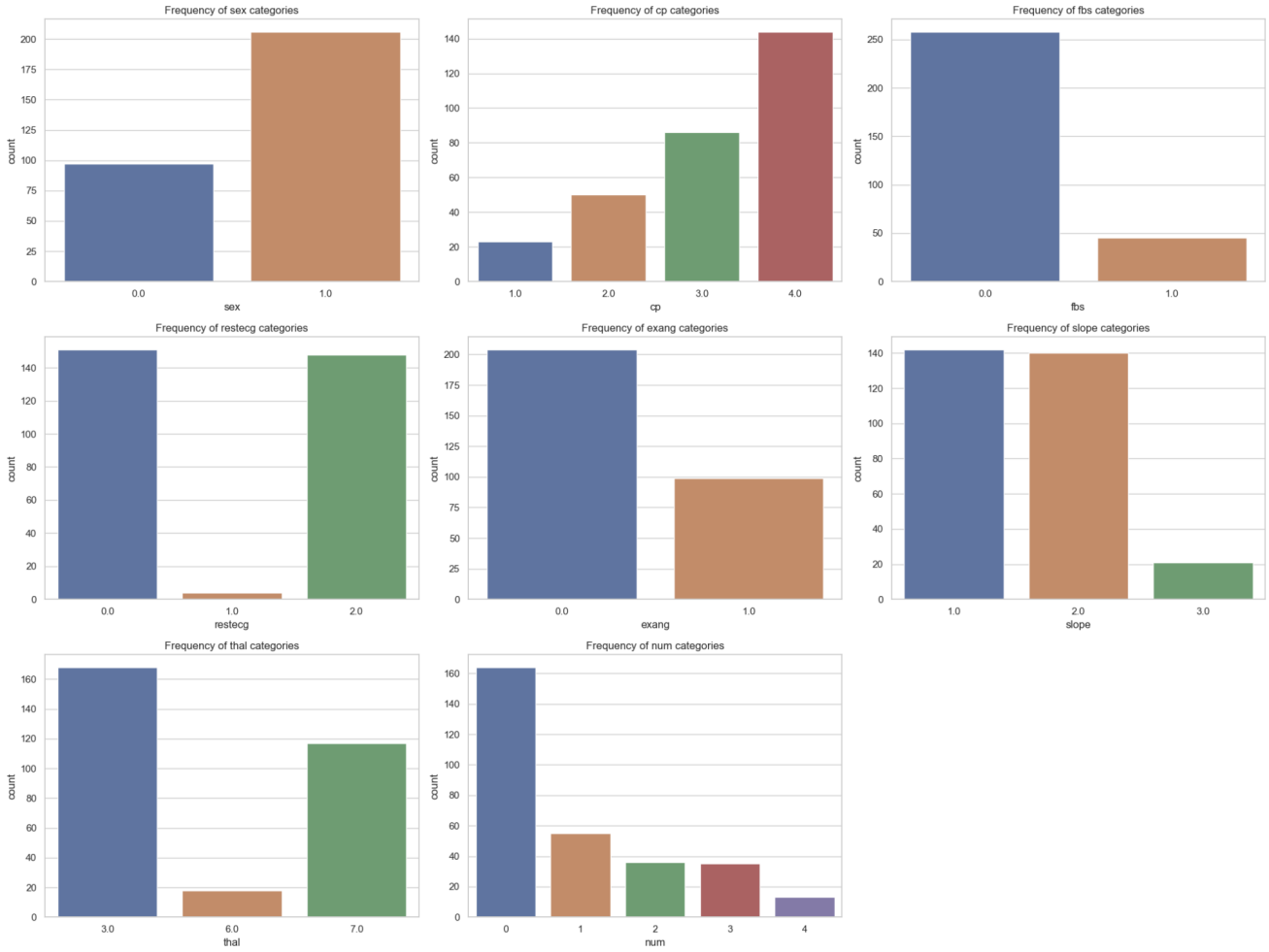
Histograms for Numerical Columns:



- Age: Appears to be normally distributed, with a slight right skew.
- Resting Blood Pressure (trestbps): Mostly normally distributed with a slight right skew.
- Serum Cholesterol (chol): Also has a normal distribution with a right skew, indicating a few higher values.
- Maximum Heart Rate Achieved (thalach): Appears to have a slight left skew.

- ST Depression (oldpeak): Right-skewed, with most values clustering near zero.
- Number of Major Vessels (ca): Many entries with a value of 0, and a few with values 1, 2, and 3, indicating a right-skewed distribution.

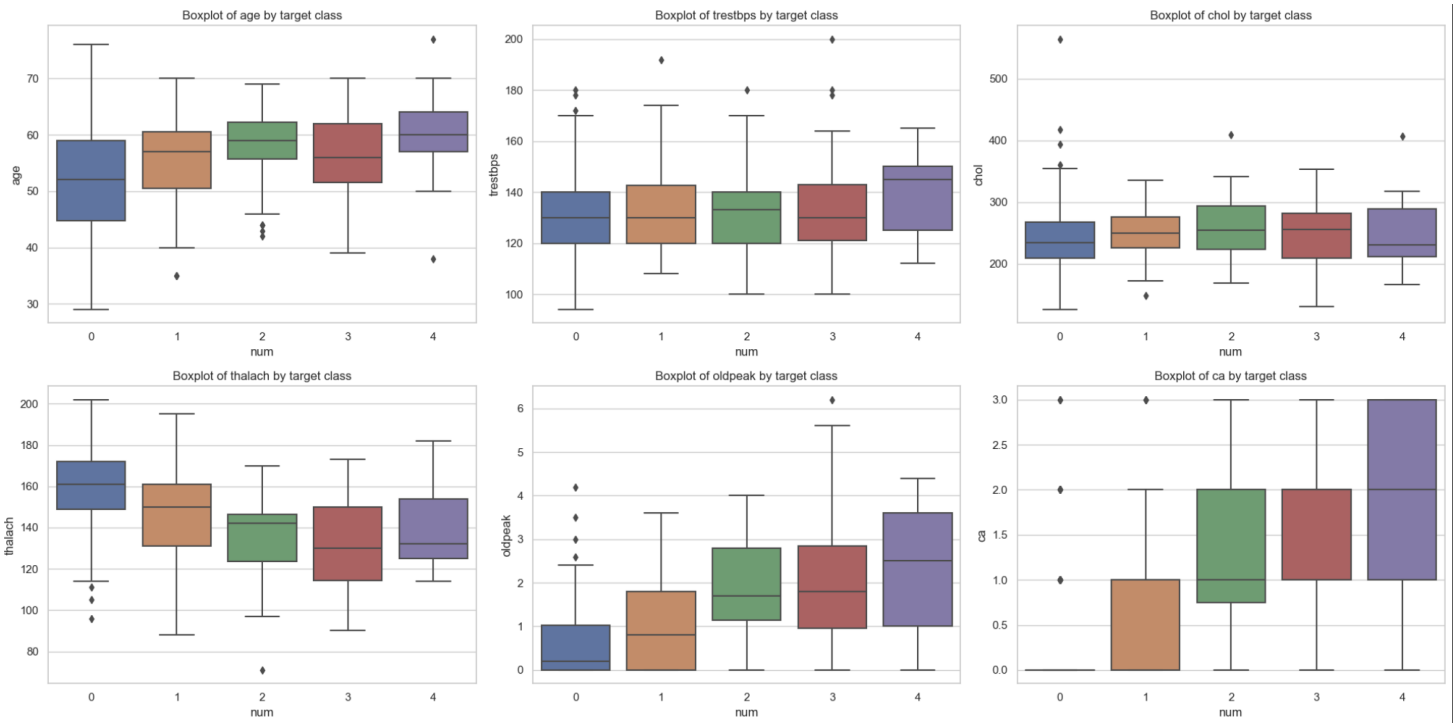
Bar Plots for Categorical Columns:



- Sex: More males than females in the dataset.
- Chest Pain Type (cp): Most individuals have a type '4', which could be asymptomatic or another specific type.
- Fasting Blood Sugar (fbs): Most individuals have a fasting blood sugar below 120 mg/dl.
- Resting ECG Results (restecg): Category '0' is the most common.
- Exercise Induced Angina (exang): More individuals do not experience angina induced by exercise.

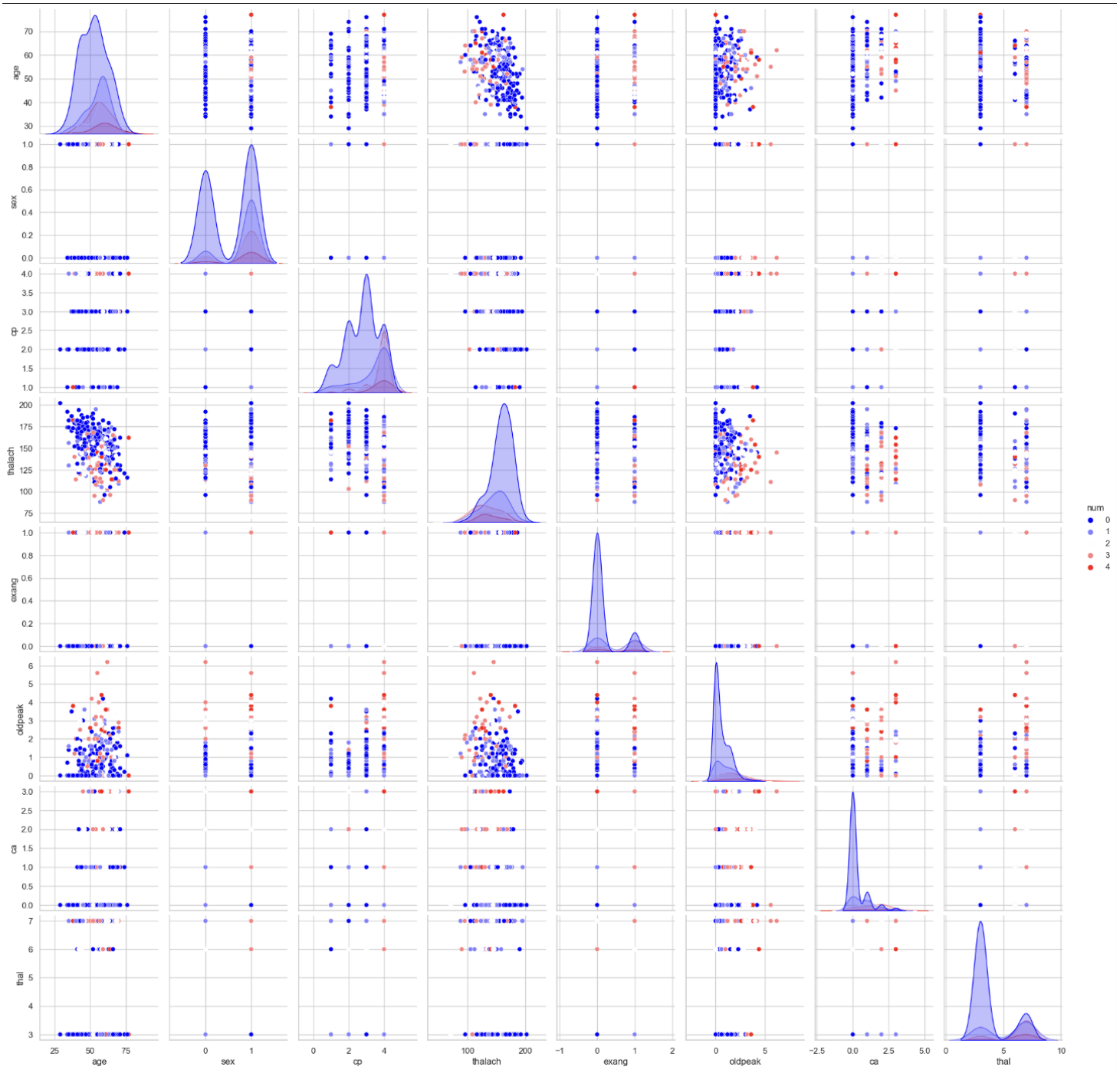
- Slope of the Peak Exercise ST Segment (slope): Category '1' is the most common.
- Thalassemia (thal): Category '3' is the most common.
- Target Variable (num): There are more individuals with no presence of heart disease (class 0) than with the presence (class 1).

Boxplots for Numerical Data by Target Class:



- These boxplots compare the distribution of numerical features across the two classes of the target variable (presence of heart disease).
- It seems that for some features like thalach, oldpeak, and ca, there are noticeable differences in the medians between the two classes, which might indicate their importance in predicting the presence of heart disease.

Pair Plots:



- Age: It's evident that older individuals are more prone to developing heart disease.
- Gender: Men (coded as 1) are more likely to experience heart disease compared to women (coded as 0).
- Chest Pain (cp): A greater degree of chest pain is linked to an increased likelihood of having heart disease.

- Maximum Heart Rate (thalach): Higher maximum heart rates are typically associated with a lower risk of heart disease.
- Exercise-induced Angina (exang): The presence of exercise-induced angina is indicative of heart disease.
- ST Depression (oldpeak): Greater levels of ST depression on an electrocardiogram (ECG) suggest a higher chance of having heart disease.
- Number of Major Vessels (ca): As the number of major blood vessels detected with issues increases, the probability of having heart disease also rises.
- Thalassemia (thal): Specific values of thalassemia are linked to a higher prevalence of heart disease.

(b) The dataset was split into train and test sets in the ratio 80:20.
Training set: 242 samples
Test set: 61 samples

(c) Both the 'entropy' and 'gini impurity' criteria for splitting in the decision tree resulted in the same accuracy score of approximately 75.41% on the test set. This means that, based on accuracy scores alone, neither criterion is superior for this particular dataset. Both have performed equally well in terms of predictive accuracy.
However, ‘gini’ can be a better choice as 'gini' tends to be faster because it doesn't require calculating logarithmic functions.

(d) In the pursuit of optimizing the performance of a decision tree classifier, a comprehensive hyperparameter tuning exercise was carried out using the Grid Search technique. The goal was to find the most effective combination of 'min samples split' and 'max features' while employing the 'gini' criterion for impurity.
The parameter grid was carefully constructed to explore a range of possibilities for the two hyperparameters under consideration:

Min Samples Split: The minimum number of samples required to split an internal node. The range was set from 2 to 9, allowing for an assessment of how small splits impact model performance.
Max Features: The number of features to consider when looking for the best split. Values ranged from 1 up to the total number of features present in the dataset, examining the influence of feature limitation on the classifier's accuracy.
A rigorous 5-fold cross-validation process was integrated into the Grid Search to ensure a robust evaluation of the model's performance across different subsets of the data.

Results
The Grid Search culminated in identifying the optimal hyperparameters for the decision tree classifier:

Min Samples Split: 3
Max Features: 5

When the classifier with these parameters was applied to the training set, it achieved an accuracy of approximately 79.74%. More importantly, the application of this "best estimator" to an independent test set yielded an accuracy of around 80.33%.

The modest increase in accuracy from the training to the test set suggests that the model, equipped with the identified hyperparameters, is generalizing well rather than overfitting to the training data. The selected 'min samples split' of 3 indicates that allowing splits on smaller groups of data contributes positively to the model's learning capability. On the other hand, restricting the 'max features' to 5 points to the model's ability to make the most out of a subset of the total available features, potentially avoiding the "curse of dimensionality" and enhancing computational efficiency.
The hyperparameter tuning process has demonstrated its value by not only improving the model's accuracy but also by striking a delicate balance between bias and variance. The decision tree classifier, with

its tuned parameters, stands validated for its predictive prowess on unseen data, marking a significant step towards a reliable and efficient classification model for this dataset.

(e) A detailed hyperparameter optimization using Grid Search was conducted for the Random Forest classifier. The process aimed to pinpoint the most effective settings to enhance model performance. The search resulted in the identification of an optimal configuration that led to notable accuracy and balanced classification metrics on the test dataset.

Hyperparameter Selection

The Grid Search method zeroed in on three critical hyperparameters that significantly influence model performance:

Number of Estimators (n_estimators): The model was configured to use 200 trees, creating a substantial ensemble that contributes to a robust prediction mechanism.

Maximum Depth (max_depth): The optimal depth was set to None, allowing each tree in the forest to grow until it reached the utmost level of class purity or until further splitting was untenable due to the minimum sample split constraint.

Minimum Samples Split (min_samples_split): The threshold was determined to be 8, ensuring that each internal node required a minimum of eight samples before it could be split into further branches.

Test Set Performance

Upon application to the test set, the fine-tuned Random Forest model demonstrated an accuracy of 88.52%. This high level of accuracy attests to the model's capability to generalize well to new data.

Classification Metrics

The model's performance was dissected further through a classification report, which illuminated its precision and recall for both classes in the dataset:

Class 0 (No Heart Disease): The model boasted a precision of 84% and a recall of 93%. These metrics indicate a high rate of correctly identified negatives and a strong ability to retrieve most actual negative cases.
Class 1 (Heart Disease Present): The precision achieved was 93%, with a recall of 84%. This reflects a high accuracy in positive predictions and a competent retrieval rate of actual positive cases.
Both classes achieved an F1-score of 89%, suggesting an equilibrium between precision and recall. The support numbers for Class 0 and Class 1 were 29 and 32, respectively.

Aggregate Performance
The macro and weighted averages across precision, recall, and F1-score hovered around 89%. This consistency in metrics showcases the model's equitable performance in classifying both conditions without bias.

Conclusion
The optimization process resulted in a Random Forest classifier that not only achieved high accuracy but also maintained commendable precision, recall, and F1-scores for both classes. The model's balanced performance across these metrics, coupled with its high accuracy, indicates its potential as a reliable diagnostic aid for heart disease. Further clinical validation is recommended to fully ascertain the model's applicability in a real-world healthcare setting.

Section C (Algorithm implementation using packages)

(a)

The MyDecisionTree classifier is a hand-crafted implementation of a decision tree for classification purposes, using the Python programming language and leveraging the NumPy library for numerical calculations and data manipulation. This classifier is structured to address classification problems through a series of methods that mimic the behavior of standard decision tree algorithms like those found in established libraries such as scikit-learn. Here's a breakdown of its components and functionalities:

Node Representation

In traditional decision tree algorithms, nodes are the basic units that store information and direct the flow of data through the tree. Each node represents a condition or decision, leading to either a classification (in the case of leaf nodes) or further subdivision (in non-leaf nodes). However, in this custom implementation, nodes are not explicitly created as objects but are instead represented by tuples or terminal values within a recursive structure.

Cost Function

The classifier includes a cost_function method that is capable of calculating either the Gini impurity or the entropy (determined by the method parameter). The Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. It is a criterion to minimize the probability of misclassification. The entropy, on the other hand, measures the level of uncertainty or disorder in the data and is also used for finding the best splits by maximizing information gain.

Splitting Mechanism
The make_split method is at the heart of the decision tree's operation, determining how to partition the data based on different features and their threshold values. This mechanism iterates over all features and their unique values to identify the split that results in the highest purity gain (or the largest reduction in cost, according to the cost function).

Tree Building Process
The fit method initiates the construction of the tree by calling the _build_tree private method, which employs a recursive approach to grow the tree. At each step, the method selects the best split and partitions the dataset into two subsets, which are then processed recursively until one of the stopping criteria is met: all data in a node belong to the same class, a maximum depth of the tree is reached, or no further gain is possible.

Prediction Process
Once built, the tree can make predictions using the predict method. It takes an input vector and traverses the tree, following the branches according to the splits until it reaches a leaf node. The prediction corresponds to the class label of the leaf node reached.

Model Evaluation
The score method allows for the evaluation of the classifier's performance. It calculates the accuracy by predicting class labels for a given dataset and comparing the predictions to the true labels.

Pruning
The prune method is an enhancement to prevent overfitting, which is common in fully grown trees. This method uses a validation set to

evaluate whether simplifying (pruning) parts of the tree could lead to the same or better performance. It works by tentatively replacing a subtree with a leaf node representing the most common class in that subtree and checking if the change leads to no decrease in accuracy on the validation set.

In practice, after the MyDecisionTree classifier is instantiated (with a specified maximum depth to avoid overfitting), it can be trained on a given dataset. The implementation is designed to encapsulate the intricacies of a decision tree within a user-friendly class that can be applied to real-world datasets for classification tasks, such as identifying diseases from medical data. The high accuracy achieved on such tasks would indicate that the classifier is effectively capturing the patterns and relationships in the data that are relevant to the classification objectives.

(b)
In this section, we detail the application and performance evaluation of a custom decision tree classifier, MyDecisionTree, implemented from scratch using Python with NumPy and Pandas libraries. The classifier was tested on a dataset related to thyroid conditions.

Dataset and Preprocessing
The dataset provided contains various features that could potentially influence the diagnosis of thyroid conditions. The initial examination of the dataset revealed a mixture of numerical and categorical features, with the target variable indicating the presence or absence of a thyroid condition.

Preprocessing Steps
The preprocessing of the dataset involved several key steps:

Binary Categorical Feature Conversion: Features with binary categorical values, represented as 'f' (false) and 't' (true), were converted to numerical format (0 and 1, respectively) for computational convenience.

Label Encoding: Non-binary categorical features were transformed into a numerical format using label encoding. This step is crucial for allowing mathematical operations and comparisons during the decision tree's training phase.

Handling Missing Values: The 'TBG' column contained zeros, which were interpreted as placeholders for missing values. These were replaced with the mean of the column to provide a reasonable estimate for the missing data.

Target Encoding: The target variable 'label' was encoded numerically, facilitating its use in the classification process.

Data Splitting: The dataset was divided into an 80-20 split to create training and validation sets. This separation allows the model to be trained on one portion of the data and independently evaluated on another.

Model Training and Evaluation

With the preprocessing complete, the MyDecisionTree classifier was instantiated with a maximum depth set to 5 to prevent overfitting and was trained on the training set.

Pre-pruning Performance

The initial evaluation of the classifier on the validation set yielded an accuracy of approximately 95.89%. This high accuracy

indicated that the classifier was capable of capturing the patterns in the data, enabling it to correctly classify the majority of cases.

Pruning
To further refine the model and potentially improve its generalizability, a pruning step was introduced. Pruning aims to remove parts of the tree that do not provide significant power in classifying instances, thereby simplifying the model and often enhancing its performance on unseen data.

Post-pruning Performance
After applying pruning, the classifier's accuracy on the validation set increased to approximately 97.50%. This improvement suggests that pruning was successful in eliminating overfitting, as the simpler model performed better on the validation data.

Conclusion
The MyDecisionTree classifier demonstrated robust performance in classifying thyroid conditions, achieving high accuracy both before and after pruning. The preprocessing steps were crucial in converting the dataset into a format amenable to the classifier's requirements, while pruning proved to be an effective technique for enhancing the model's accuracy and generalizability.

In summary, the custom decision tree classifier shows promise as a tool for medical diagnosis, with the potential for application in other classification domains pending further validation.