HW2

Q1:

Prompt 1: "I have a dataset of email messages collected over the past five years that have been manually labeled as 'spam' or 'not spam'. How should I use Decision Tree Induction to automatically classify new email messages based on this data?"

Certainly! Here's a step-by-step guide to using Decision Tree Induction for email spam classification:

Data Preprocessing:

Tokenization: Break down the email content into individual words or tokens.

Remove Stopwords: Filter out common words like 'and', 'the', 'is', etc., which might not be significant for classification.

Feature Extraction: Convert the tokens into numerical features. Common methods include:

Bag of Words (BoW): Represents text based on the frequency of words.

Term Frequency-Inverse Document Frequency (TF-IDF): Weighs terms based on their importance in the document relative to the entire dataset.

Data Splitting:

Partition the dataset into training and testing sets. A typical split might be 70-30 or 80-20, with the larger portion for training.

Building the Decision Tree:

Train the decision tree on the training set.

Choose an algorithm like ID3 (which uses information gain) or CART (which can use Gini impurity).

Define criteria for splitting nodes and set a stopping criterion to prevent overfitting. This could be based on maximum tree depth, minimum samples per leaf, or a minimum increase in information gain.

Model Evaluation:

Assess the decision tree's performance using the testing set.

Use metrics such as accuracy, precision, recall, and F1-score to gauge the model's effectiveness.

For a more robust evaluation, consider employing cross-validation. This involves splitting the data multiple times and evaluating the model on different subsets.

Tree Pruning (Optional):

To combat overfitting, consider pruning the tree. This entails removing sections of the tree that offer little predictive power.

Optimization:

If the model's performance isn't up to par, think about:

Adjusting hyperparameters (e.g., max depth, min samples per leaf).

Exploring ensemble methods like Random Forest, which constructs multiple decision trees and aggregates their outcomes.

Deployment:

Once content with the model's performance, deploy it to automatically classify new incoming emails.

Set up a system to monitor the model's real-world performance. Given that the nature of spam can change over time, it's vital to ensure the model remains effective and to retrain it if necessary.

In summary, Decision Trees offer an interpretable and straightforward method for classification. However, they can be susceptible to overfitting, especially with intricate datasets. Regular monitoring and potential retraining are essential to ensure continued effectiveness.

Prompt 2: "Can you explain the step-by-step process of implementing a Decision Tree Induction algorithm for classifying email messages as spam or not spam?"

Of course! Implementing a Decision Tree Induction algorithm for classifying email messages involves several systematic steps. Here's a detailed breakdown:

1. **Data Collection**:
    - Gather a dataset of email messages that have been labeled as 'spam' or 'not spam'.
2. **Data Preprocessing**:
    - **Tokenization**: Convert the email content into individual words or tokens.
    - **Remove Stopwords**: Filter out common words such as 'and', 'the', 'is', which might not be significant for classification.
    - **Feature Extraction**: Convert the tokens into numerical features. Common methods include:
        - **Bag of Words (BoW)**: Represents text based on the frequency of words.
        - **Term Frequency-Inverse Document Frequency (TF-IDF)**: Weighs terms based on their importance in the document relative to the entire dataset.
    - **Label Encoding**: Convert the labels ('spam' and 'not spam') into numerical values, e.g., 0 for 'not spam' and 1 for 'spam'.
    - **Handle Imbalanced Data**: If there's a significant difference between the number of 'spam' and 'not spam' labels, consider techniques such as oversampling, undersampling, or SMOTE.
3. **Data Splitting**:
    - Divide the dataset into training and testing sets. A common split might be 70-30 or 80-20, with the larger portion for training.
4. **Building the Decision Tree**:
    - Use a Decision Tree algorithm like ID3, C4.5, or CART.
    - Define the criteria for splitting nodes. Common criteria include:
        - **Information Gain**: Measures the reduction in entropy achieved by partitioning the dataset.
        - **Gini Impurity**: Measures the frequency at which a randomly chosen element would be incorrectly classified.
    - Set stopping criteria to prevent overfitting, such as maximum tree depth, minimum samples per leaf, or a minimum increase in the chosen criterion (e.g., information gain).
5. **Training the Model**:
    - Use the training dataset to train the Decision Tree.
    - The algorithm will recursively split the data based on the best attribute at each level, building the tree from the top down.
6. **Model Evaluation**:
    - Use the testing set to evaluate the model's performance.
    - Calculate metrics such as accuracy, precision, recall, and F1-score to understand the model's effectiveness.

- Consider using cross-validation for a more comprehensive evaluation.

7. **Tree Pruning (Optional)**:
    - To reduce overfitting, consider pruning branches of the tree that add little predictive power.
    - Pruning can be done by reducing the tree's depth or by cutting branches that result in a minimal increase in the chosen criterion.
8. **Optimization**:
    - Adjust hyperparameters, if necessary, to improve performance.
    - Consider ensemble methods like Random Forest if a single tree doesn't provide satisfactory results.
9. **Deployment**:
    - Once satisfied with the model's performance, integrate it into the email system to automatically classify incoming emails as 'spam' or 'not spam'.
    - Regularly retrain the model with new data to ensure its continued effectiveness.
10. **Monitoring and Maintenance**:

- Continuously monitor the model's performance in real-world scenarios.
- Collect feedback and retrain the model periodically to account for evolving spam tactics.

In essence, Decision Tree Induction provides a clear and interpretable method for classifying data. However, care must be taken to preprocess the data correctly, choose the right criteria for splitting, and prevent overfitting to ensure the best results.

Prompt 3: I'm tasked with using Decision Tree Induction to automatically classify new email messages as either "spam" or "not spam." Can you provide a step-by-step approach to tackling this problem?

Final prompt

ST

using Decision Tree Induction to automatically classify new email messages as either "spam" or "not spam." Can you provide a step-by-step approach to tackling this problem?

Certainly! Using Decision Tree Induction to classify email messages as "spam" or "not spam" is a common and effective approach in machine learning. Here's a step-by-step approach to tackle this problem:

1. **Data Collection**:
   - Gather a labeled dataset of email messages. Each email should be labeled as either "spam" or "not spam" (ham). The dataset should include features like the email's subject, sender, body text, and any other relevant attributes.
2. **Data Preprocessing**:
   - Clean and preprocess the email data. This may involve:
     - Removing HTML tags, special characters, and unnecessary whitespace.
     - Tokenizing the text (splitting it into words or phrases).
     - Removing stopwords (common words like "the," "and," "is").
     - Stemming or lemmatization to reduce words to their root form.
     - Feature extraction: Convert the text data into numerical format using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings.
3. **Data Splitting**:
   - Split the dataset into two parts: a training set and a testing/validation set. The training set is used to train the decision tree, and the testing set is used to evaluate its performance.
4. **Decision Tree Induction**:
   - Train a decision tree classifier using the training data. Decision tree induction involves recursively splitting the data based on the most informative features to create a tree-like structure. Popular algorithms for decision tree induction include ID3, C4.5, and CART.
   - Choose an appropriate criterion for splitting nodes, such as Gini impurity or information gain.
5. **Model Evaluation**:
   - Evaluate the performance of the decision tree model using the testing/validation set. Common evaluation metrics for binary classification include accuracy, precision, F1-score, and ROC-AUC.

Regenerate

F1-score, and ROC-AUC.

6. **Hyperparameter Tuning** (Optional):
   - Experiment with different hyperparameters of the decision tree algorithm, such as the maximum depth of the tree or the minimum number of samples required to split a node, to optimize the model's performance.
7. **Feature Importance** (Optional):
   - Analyze the feature importance scores provided by the decision tree model to understand which features are most relevant in classifying spam emails. This can help refine the feature selection process.
8. **Model Deployment**:
   - Once you are satisfied with the model's performance, you can deploy it in a production environment to automatically classify incoming email messages as spam or not spam.
9. **Continuous Monitoring**:
   - Monitor the model's performance in the production environment and periodically retrain it with new data to ensure its accuracy over time.
10. **Handling Imbalanced Data** (Optional):
    - If your dataset is highly imbalanced, meaning you have many more non-spam (ham) emails than spam emails, consider techniques like resampling, using different evaluation metrics, or exploring advanced algorithms like Random Forest or Gradient Boosting to improve model performance.

a) Correct Parts in ChatGPT's Answer:

Data Preprocessing: The steps related to data preprocessing, such as cleaning, tokenization, and feature extraction, are correctly mentioned.

Data Splitting: The separation of data into training and testing sets is correctly described.

Model Evaluation: ChatGPT correctly identifies common evaluation metrics for binary classification.

Hyperparameter Tuning is correctly mentioned.

Model Deployment: The importance of deploying the model in a production environment is correctly emphasized.

Continuous Monitoring: The need for monitoring and regular retraining of the model is correctly highlighted.

Most of the steps chatgpt mentioned are correct.

Wrong:

Chatgpt mentioned feature importance as optional. While feature importance analysis is a crucial step in machine learning model development, the extent to which you delve into it can be tailored to the needs of your project. However, it should not be entirely omitted if you aim to build a robust and interpretable model.

Hyperparameter tuning is generally not optional but highly recommended when developing machine learning models, including decision trees for spam classification.

Which parts need to be elaborated:

Data-preprocessing such as missing values, outliers, and duplicates needs further elaborations. Data Splitting e:g how to choose the right split ratio. Decision Tree Induction e:g how to choose the right algorithm, how to tune hyperparameters. Train a Decision Tree model on the training data using an appropriate algorithm (e.g., ID3, C4.5, CART) but it should mention which algorithm I can use in a particular scenario. Model Evaluation How to choose the right evaluation metrics. The process of training a decision tree and the choice of criteria for splitting nodes should be elaborated further. Handling Imbalanced Data: While ChatGPT mentions this as an optional step, more details about techniques to address imbalanced data, such as oversampling, undersampling, or using different evaluation metrics, could be provided.

Missing Parts

- Aggregation: We could aggregate the number of spam words in the email body and the number of links in the email body to create a new feature that represents the overall spaminess of the email.
- Dimensionality reduction: We could use PCA to reduce the number of features in the dataset and improve the performance of the decision tree model.
- Feature subset selection: We could use a technique such as recursive feature elimination to identify the most important features in the dataset and remove the redundant and irrelevant features.
- Feature creation: We could create a new feature that represents the presence of certain spam words in the email body.
- Discretization: We could discretize the feature that represents the number of spam words in the email body into two categories: "low" and "high."
- Normalization: We could normalize the features to a common range, such as 0 to 1.
- Measures of similarity and dissimilarity: We could calculate the cosine similarity between two email messages. This information could be used to create a new feature that represents the similarity between two email messages.
- Correlation: We could calculate the correlation between the feature that represents the number of spam words in the email body and the feature that represents the number of links in the email body. This information could be used to identify redundant features.

How should training records be split? Method for specifying test condition, depending on attribute types.

1. How should the splitting procedure stop?
2. How to measure best split, Measures of Node Impurity e:g Gini index, entropy, classification error.

   Visualizing the Tree: It can be beneficial to visualize the decision tree to understand its structure and interpretability. Tools and techniques for visualizing the tree should be considered.

   Cross-Validation: It's essential to use cross-validation techniques to assess the model's performance and ensure it generalizes well to unseen data. Cross-validation is an important step that should be part of the training and evaluation process.

- A detailed explanation about how decision trees make decisions (i.e., entropy, information gain, or Gini impurity) is missing.
- Overfitting, underfitting, pruning.
- It would be valuable to discuss the interpretability of decision trees and how to explain their predictions, especially in cases where the model's decisions may need to be justified.

Q2:

Gini Index for Gender

$$Gini\ (Male) = 1 - \left(\left(\frac{2}{3}\right)^2 + \left(\frac{1}{3}\right)^2\right) = 0.45$$

$$Gini\ (Female) = 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right) = 0.5$$

$$Gini\ (Gender) = \frac{3}{5} \times 0.45 + \frac{2}{5} \times 0.5$$

$$= 0.47$$

Computing Gini Index for age

Survived	Yes		No		No		Yes		No			
	Age											
	8		26		32		45		55			
	-1		17		29		38.5		50		60	
	$\leq$	$>$	$\leq$	$>$	$\leq$	$>$	$\leq$	$>$	$\leq$	$>$	$\leq$	$>$
Yes	0	2	1	1	1	1	1	1	2	0	2	0
No	0	3	0	3	1	2	2	1	2	1	3	0
Gini	0.48		0.3		0.46		0.46		0.4		0.48	

$$\text{Gini (split-8)} = \frac{0}{5}\left(1-\left(\left(\frac{0}{0}\right)^2+\left(\frac{0}{0}\right)^2\right)\right)+\frac{5}{5}\left(1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2\right)$$

$$= 1-\frac{4}{25}-\frac{9}{25}$$

$$= 0.48$$

$$\text{Gini (split-17)} = \frac{1}{5}\left(1-\left(\frac{1}{1}\right)^2-\left(\frac{0}{1}\right)^2\right)+\frac{4}{5}\left(1-\left(\frac{1}{4}\right)^2-\left(\frac{3}{4}\right)^2\right)$$

$$= \frac{1}{5}(1-1-0)+\frac{4}{5}\left(1-\frac{1}{16}-\frac{9}{16}\right)$$

$$= 0.3$$

$$\text{Gini (split-29)} = \frac{2}{5}\left(1-\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^2\right)+\frac{3}{5}\left(1-\left(\frac{1}{3}\right)^2-\left(\frac{2}{3}\right)^2\right)$$

$$= \frac{2}{5}\left(1-\frac{1}{2}\right)+\frac{3}{5}\left(1-\frac{1}{9}-\frac{4}{9}\right)$$

$$= \frac{2}{5}\times\frac{1}{2}+\frac{3}{5}\times\frac{4}{9}$$

$$= \frac{1}{5}+\frac{4}{15}$$

$$= 0.46.$$

$$\text{Gini (split-38.5)} = \frac{3}{5}\left(1-\left(\frac{1}{3}\right)^2-\left(\frac{2}{3}\right)^2\right)+\frac{2}{5}\left(1-\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^2\right)$$

$$= \frac{3}{5}\left(1-\frac{1}{9}-\frac{4}{9}\right)+\frac{2}{5}\left(1-\frac{1}{4}-\frac{1}{4}\right)$$

$$= \frac{3}{5}\left(1-\frac{5}{9}\right)+\frac{2}{5}\left(1-\frac{1}{2}\right)$$

$$= \frac{3}{5}\left(\frac{4}{9}\right)+\frac{2}{5}\left(\frac{1}{2}\right)$$

$$= \frac{4}{15}+\frac{1}{5}$$

$$= 0.46$$

$$= 0.46$$

$$\text{Gini (split-50)} = \frac{4}{5}\left(1-\left(\frac{2}{4}\right)^2-\left(\frac{2}{4}\right)^2\right)+\frac{1}{5}\left(1-\left(\frac{0}{1}\right)^2-\left(\frac{1}{1}\right)^2\right)$$

$$= \frac{4}{5}\left(1-\frac{4}{16}-\frac{4}{16}\right)+\frac{1}{5}\left(1-0-1\right)$$

$$= \frac{4}{5}\left(1-\frac{1}{2}\right)+0$$

$$= 0.4$$

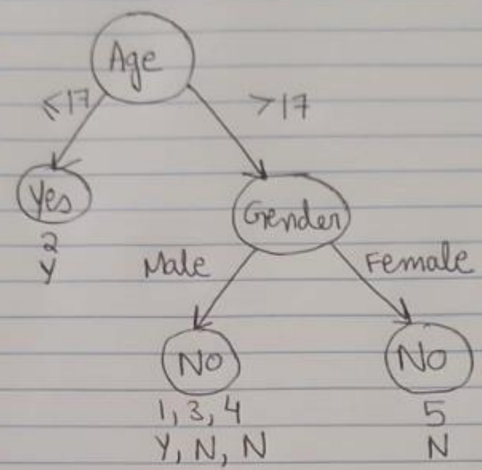
$$\text{Gini (split-60)} = \frac{5}{5}\left(1-\left(\frac{2}{5}\right)^2-\left(\frac{3}{5}\right)^2\right)+\frac{0}{5}\left(1-0-0\right.$$

$$= 1-\frac{4}{25}-\frac{9}{25}$$

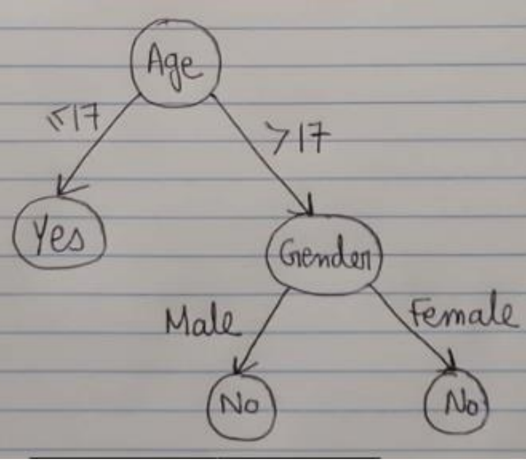
$$= 1-\frac{13}{25}$$

$$= 0.48$$

choosing 17 as split position because it has least gini.



class attribute is summed and followed majority rule.



3.1. C5.0 and CART are two well-known decision tree algorithms. Read the published literature about these two algorithms and answer the following question: for each algorithm, provide an overview describing how the algorithm works, discuss the impurity measure it uses for the attribute split test condition, and discuss one advantage and one disadvantage of the algorithm. Provide the references to the published literature to justify your answers.

1- C5.0 Algorithm:

Decision trees construction of C5.0 from a set of training dataset is as same as C4.5, using the idea of entropy and information gain.C5.0 model calculates the information gain for each attribute and select the maximum gain value as root node or the best splitting attribute.

C5.0 model works by splitting the sample based on the attribute that provides the maximum information gain. The C5.0 model can split samples on basis of the biggest information gain attribute. .The sample subset that is get from the former split will be split afterward. The process will continue until the sample subset cannot be split and is usually according to another attribute. Finally, examine the lowest level split, those sample subsets that don't have remarkable contribution to the model will be rejected.

The C5.0 algorithm, like its predecessor C4.5, uses an impurity measure to determine the best attribute for splitting the data at each node of the decision tree. This impurity measure helps in evaluating how well a given attribute separates the training examples based on their target classification.

For C5.0, the primary impurity measure used is entropy, which is a measure of impurity. The entropy is defined as:

Entropy(t) = - $\sum_j p(j \mid t) \log_2 p(j \mid t)$

Using entropy, the algorithm calculates the information gain for each attribute, which is the difference in entropy before and after a hypothetical split based on that attribute.

Information gain tells us how important a given attribute of the feature vectors is.

Information Gain = entropy(parent) – [average entropy(children)]

Advantage of C5.0

- C5.0 has the several advantages:
- It can handle all types of attributes.
- The C5.0 rules set have noticeably lower error rates on unseen cases.
- It is much faster to complete the rules set construction task.

Disadvantages of C5.0

- Over fitting happens when algorithm model picks up data with uncommon characteristics.
- Small changes in training data can result in large changes to decision logic.

<u>CART</u>

Cart can handle both classification (using Gini impurity) and regression (using variance) tasks. It always performs binary splits, meaning each node in the tree splits into exactly two child nodes. For classification, uses Gini impurity to evaluate the quality of a split.

And for regression uses variance or mean squared error to evaluate splits. Employs a post-pruning approach, where the fully grown tree is pruned from the bottom up to avoid overfitting. This is typically done using a validation set to determine which branches to prune.

Output for classification is leaf nodes represent the majority class of the samples that fall into that leaf and for regression leaf nodes predict a continuous value, typically the mean of the target values of the samples in that leaf.

CART uses GINI Index to determine in which attribute the branch should be generated. The strategy is to choose the attribute whose GINI Index is minimum after splitting.

$GINI_{split} = \sum^{k}_{i=1} n_i/n \; GINI(i)$

The attribute that has the minimum Gini index is selected as the splitting attribute.

There are some advantages and disadvantages of the CART algorithm.

Advantages:

- It can handle both categorical and numerical variables.
- Classification process is done with less calculation.
- CART algorithm will itself identify the most significant variables and eliminate non-significant ones.
- It can also easily handle outliers.

Diadvantages:

- One of the disadvantages of CART algorithm is it can split only by one variable.
- It may have unstable decision tree.
- Insignificant modification of learning sample such as eliminating several observations and cause changes in decision tree: increasers decrease of tree complexity, changes in splitting variables and values.

Table 3.1 Comparisons between Two Decision Tree Algorithms: C5.0 and CART

	C5.0 Algorithm	CART Algorithm
Type of data	Categorical, continuous, dates, times and timestamps data	Continuous and nominal data
Speed	Highest	Average
Pruning	Pre pruning (Pessimistic pruning)	Post pruning (Cost complexity pruning)
Boosting	Supported	Supported
Missing values	Can handle	Can handle
Splitting criteria	Multi split	Binary split
Formula	Use entropy and information gain	Use Gini diversity index

Customer Card Classification Based on C5.0 & CART Algorithms. (n.d.).

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=87b9df85c16d81c09d399f15f06aacfce8021df8

COMPARISON OF DATA MINING CLASSIFICATION ALGORITHMS: C5.0 AND CART FOR CAR EVALUATION AND CREDIT CARD INFORMATION DATASETS. (n.d.).

PROBLEM 3.2

df <- read.csv("C:/Users/mehre/test.r/HW2/Wholesale customers data.csv")

View(df)

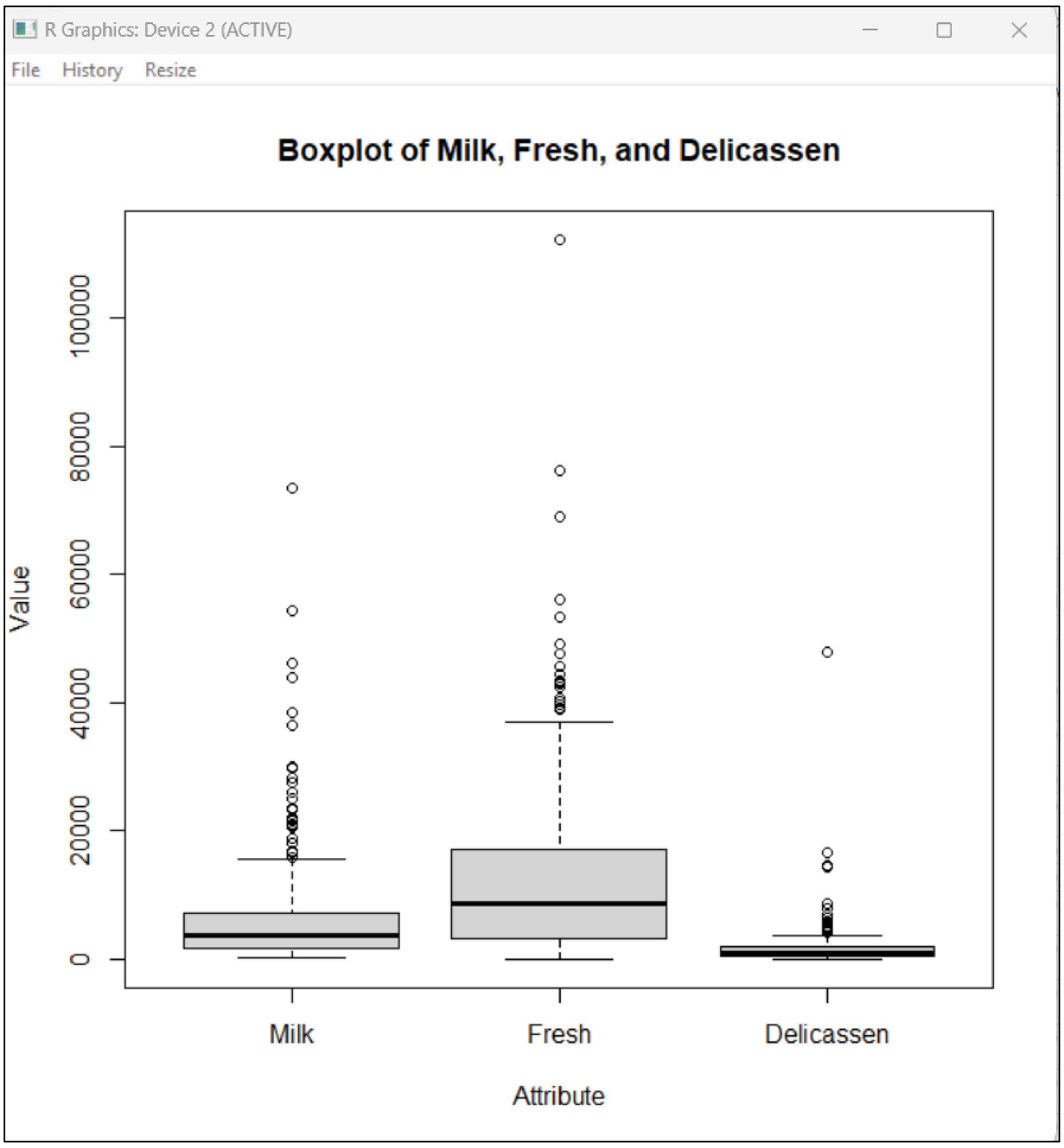
Filter

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
1	2	3	12669	9656	7561	214	2674	1338
2	2	3	7057	9810	9568	1762	3293	1776
3	2	3	6353	8808	7684	2405	3516	7844
4	1	3	13265	1196	4221	6404	507	1788
5	2	3	22615	5410	7198	3915	1777	5185
6	2	3	9413	8259	5126	666	1795	1451
7	2	3	12126	3199	6975	480	3140	545
8	2	3	7579	4956	9426	1669	3321	2566
9	1	3	5963	3648	6192	425	1716	750
10	2	3	6006	11093	18881	1159	7425	2098
11	2	3	3366	5403	12974	4400	5977	1744
12	2	3	13146	1124	4523	1420	549	497
13	2	3	31714	12319	11757	287	3881	2931
14	2	3	21217	6208	14982	3095	6707	602
15	2	3	24653	9465	12091	294	5058	2168
16	1	3	10253	1114	3821	397	964	412
17	2	3	1020	8816	12121	134	4508	1080
18	1	3	5876	6157	2933	839	370	4478

Showing 1 to 19 of 440 entries, 8 total columns

a. Using the Boxplot visualization method, in a single figure, draw a boxplot of each of the following attributes: Milk, Fresh, and Delicatessen.

```
options(scipen=999)

f1=boxplot(df$Milk, df$Fresh, df$Delicassen,

        names = c('Milk', 'Fresh', 'Delicassen'),

        main = 'Boxplot of Milk, Fresh, and Delicassen',

        ylab = 'Value',

        xlab = 'Attribute'

)
```



b) From the boxplots of the three attributes of Task (a), identify which attributes have outliers, which attribute values are outliers, and justify your answers. If there are outliers, write your R code to remove the entire tuples containing the outliers from the dataset, and print the dataset after those tuples have been removed.

```
get_outliers <- function(data) {
  Q1 <- quantile(data, 0.25)
  Q3 <- quantile(data, 0.75)
  IQR <- Q3 - Q1
  outliers <- data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))]
```

```
  return(outliers)
}

# Extract outliers for each of the specified columns
milk_outliers <- get_outliers(df$Milk)
fresh_outliers <- get_outliers(df$Fresh)
delicassen_outliers <- get_outliers(df$Delicassen)

milk_outliers_count <- length(milk_outliers)
fresh_outliers_count <- length(fresh_outliers)
delicassen_outliers_count <- length(delicassen_outliers)

# Display the count and the outlier values for each column
cat("Number of outliers in Milk:", milk_outliers_count, "\n")
cat("Outlier values for Milk:", milk_outliers, "\n\n")

cat("Number of outliers in Fresh:", fresh_outliers_count, "\n")
cat("Outlier values for Fresh:", fresh_outliers, "\n\n")

cat("Number of outliers in Delicassen:", delicassen_outliers_count, "\n")
cat("Outlier values for Delicassen:", delicassen_outliers, "\n")
```

```
> # Display the count and the outlier values for each column
> cat("Number of outliers in Milk:", milk_outliers_count, "\n")
Number of outliers in Milk: 28
> cat("Outlier values for Milk:", milk_outliers, "\n\n")
Outlier values for Milk: 36423 20484 15729 22044 54259 21412 29892 38369 20959 46197 73498 27472 16729 15726 25862 29627 43950 28326 16599 23133 17972 23527 20655 25071 1678
4 18664 21858 16687

>
> cat("Number of outliers in Fresh:", fresh_outliers_count, "\n")
Number of outliers in Fresh: 20
> cat("Outlier values for Fresh:", fresh_outliers, "\n\n")
Outlier values for Fresh: 43088 56159 44466 40721 43265 56082 76237 42312 45640 112151 47493 56083 53205 49063 68951 40254 42786 39679 38793 39228

>
> cat("Number of outliers in Delicassen:", delicassen_outliers_count, "\n")
Number of outliers in Delicassen: 27
> cat("Outlier values for Delicassen:", delicassen_outliers, "\n")
Outlier values for Delicassen: 7844 5185 4478 4334 16523 5778 5206 4626 5864 4985 6465 14472 14351 5130 4430 6250 8550 47943 6854 6372 5121 5609 4100 4829 5120 5137 4365
>
```

```
# Rows with outliers for each column
milk_outlier_rows <- df$Milk %in% milk_outliers
fresh_outlier_rows <- df$Fresh %in% fresh_outliers
delicassen_outlier_rows <- df$Delicassen %in% delicassen_outliers
```

Combine the outlier rows for all columns

all_outlier_rows <- milk_outlier_rows | fresh_outlier_rows | delicassen_outlier_rows

Count the number of rows with outliers

sum(all_outlier_rows)

df_no_outliers <- df[!(df$Milk %in% milk_outliers |

df$Fresh %in% fresh_outliers |

df$Delicassen %in% delicassen_outliers),]

View(df_no_outliers)

Filter

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
1	2	3	12669	9656	7561	214	2674	1338
2	2	3	7057	9810	9568	1762	3293	1776
4	1	3	13265	1196	4221	6404	507	1788
6	2	3	9413	8259	5126	666	1795	1451
7	2	3	12126	3199	6975	480	3140	545
8	2	3	7579	4956	9426	1669	3321	2566
9	1	3	5963	3648	6192	425	1716	750
10	2	3	6006	11093	18881	1159	7425	2098
11	2	3	3366	5403	12974	4400	5977	1744
12	2	3	13146	1124	4523	1420	549	497
13	2	3	31714	12319	11757	287	3881	2931
14	2	3	21217	6208	14982	3095	6707	602
15	2	3	24653	9465	12091	294	5058	2168
16	1	3	10253	1114	3821	397	964	412
17	2	3	1020	8816	12121	134	4508	1080
19	2	3	18601	6327	10099	2205	2767	3181
20	1	3	7780	2495	9464	669	2518	501
21	2	3	17546	4519	4602	1066	2259	2124

Showing 1 to 19 of 380 entries, 8 total columns

Count rows for each specified Region value

region_counts <- table(df_no_outliers$Region[df_no_outliers$Region %in% c(2, 3, 1)])

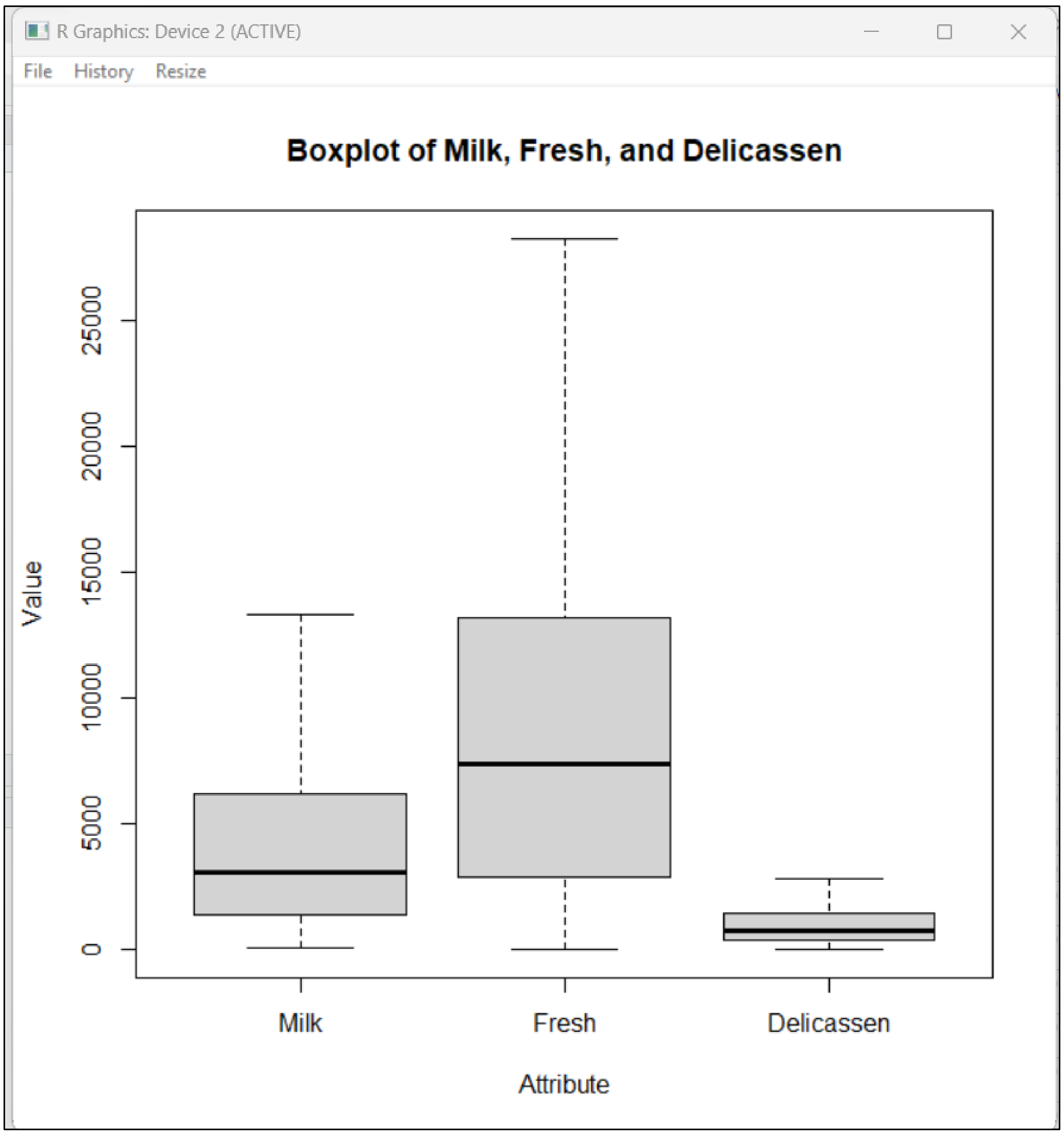
Print the counts

```
print(region_counts)
# Display the dimensions of the original and cleaned dataframes
cat("Original dataframe dimensions:", dim(df), "\n")
cat("Dataframe dimensions after outlier removal:", dim(df_no_outliers), "\n")
View(df_no_outliers)
df_no_outliers <- df
repeat {
 # Identify outliers for each column
 milk_outliers <- boxplot.stats(df_no_outliers$Milk)$out
 fresh_outliers <- boxplot.stats(df_no_outliers$Fresh)$out
 delicassen_outliers <- boxplot.stats(df_no_outliers$Delicassen)$out

 # Create a logical index for rows with outliers
 outlier_index <- df_no_outliers$Milk %in% milk_outliers |
  df_no_outliers$Fresh %in% fresh_outliers |
  df_no_outliers$Delicassen %in% delicassen_outliers

 # If no outliers are found, break the loop
 if (sum(outlier_index) == 0) {
  break
 }

 # Otherwise, remove the outliers and continue
 df_no_outliers <- df_no_outliers[!outlier_index, ]
}
```

c) Using the preprocessed dataset obtained from Task (b), repeat Task (a) and provide your interpretation of the new boxplots.

```
f2=boxplot(df_no_outliers$Milk, df_no_outliers$Fresh, df_no_outliers$Delicassen,
        names = c('Milk', 'Fresh', 'Delicassen'),
        main = 'Boxplot of Milk, Fresh, and Delicassen',
```

```
        ylab = 'Value',
        xlab = 'Attribute'
)
```



After removing outliers, you can see there is complete difference in boxplot with outliers,

This data distribution was mostly towards upper viscor so I applied formula Q1 <- quantile(data, 0.25)

Q3 <- quantile(data, 0.75)

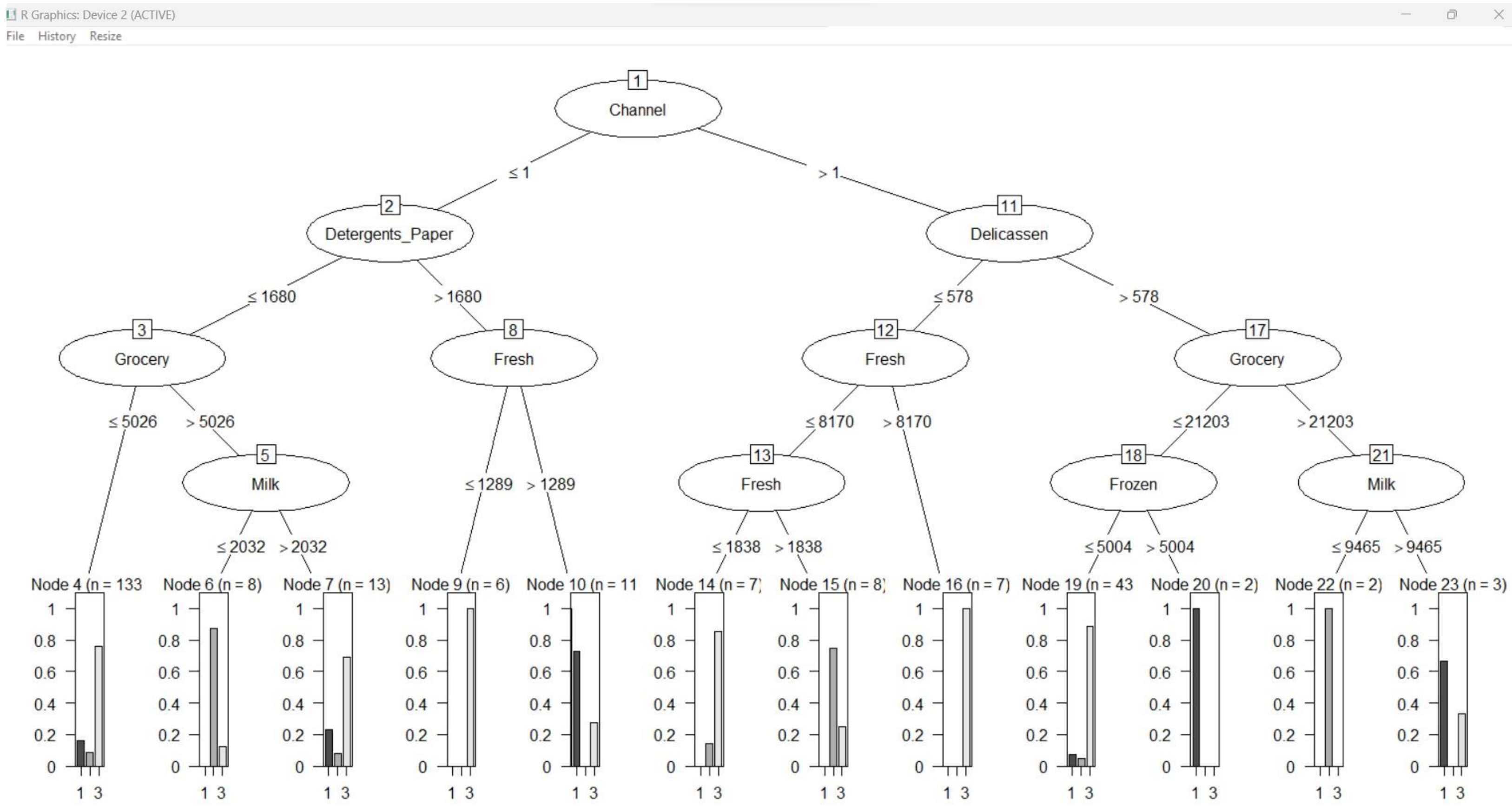
IQR <- Q3 - Q1

outliers <- data[(data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))] to calculate values and removing values above the viscor.

- The "lower viscor" would be the minimum value or the 25th percentile ($1Q1$), which is 1533. This represents the value below which 25% of the data lies.
- The "upper viscor" determined using the interquartile range (IQR) method is 15,675.5. This value suggests that, based on the distribution of the data provided, any viscosity value above 15,675.5 might be considered unusually high or potentially an outlier.

d) Using the preprocessed dataset obtained from Task (b) and using the C5.0 algorithm (available from the package C5.0), build a decision tree that classifies the tuples based on the class attribute "Region" in the dataset. Print the resulting decision tree in the graphical format. Then evaluate the error rate using k-fold cross-validation with k = 3. For each fold, print the confusion matrix to standard output, then calculate, print, and store the error rate.

```
library(caret)

# Load the C5.0 package

library(C50)

df2 <- df_no_outliers[order(runif(nrow(df_no_outliers))),]

tail(df2,25)

df2$Region <- factor(df2$Region)

# Split  data into training and testing sets

set.seed(123)

split_idx <- sample(1:nrow(df2), 0.7 * nrow(df2))  # 70% training, 30% testing

df2_train <- df2[split_idx, ]

df2_test <- df2[-split_idx, ]

# Train the C5.0 model with the correct formula

C50_model <- C5.0(Region ~ ., data = df2_train)


# Print the summary of the model

summary(C50_model)

plot(C50_model)
```




```
#C5.0 Predict

C50_predict<- predict(C50_model,df2_test)

C50_predict

#Compare

table(df2_test[,2],C50_predict)

#C5.0 Train Improve performance

C50_model<- C5.0(df2_train[,-2],df2_train[,2],trials=10)

summary(C50_model)

#C5.0 Predict Class

C50_predict_class<- predict(C50_model,df2_test, method = "class")

C50_predict_class

#Compare

table(df2_test[,2],C50_predict_class)

#C5.0 Predict Prob

C50_predict_prob<- predict(C50_model,df2_test, method = "prob")


C50_predict_prob
```

#Compare

table(df2_test[,2],C50_predict_prob)

```
>
> C50_predict_prob<- predict(C50_model,df2_test, method = "prob")
> C50_predict_prob
  [1] 2 1 2 1 2 1 2 2 2 1 1 3 1 2 1 1 1 2 2 2 2 3 1 1 1 1 2 2 3 2 1 2 2 1 1 1 1 1 3 3 3 2 2 1 1 2 2 1 3 1 2 2 1 2 2 1 3 1 2 2 2 3 1 2 3 1 3 3 3 3 1 2 1 2 3 1 1 1 3 2 1 3
 [83] 2 2 3 3 3 3 2 1 2 1 2 2 2 2 1 2 1 3 3 1 3 1 2 3 1 3 3 1 2 2 3 2 3 3 3 1 1 1 3 2 1 1 1 2 3 2 2 2 3 3 1 2 2 3 2 1 2 1 3 3 3 2 3 1 1 2 3 2 3 1 2 1 1 3 3 1 1 1 2 2 1 2
[165] 2 1 2 3 2 1 2 2 1 2 1 2 1 1 2 2 2 1 3 2 3 1 2 1 1 2 1 3 3 3 2 2 1 2 1 3 3 1 1 3 2 2 2 1 1 1 2 2 1 1 1 3 3 1 1 1 3 2
Levels: 1 2 3
```

Define the number of folds (k)

k <- 3

Create a training control object for k-fold cross-validation

train_control <- trainControl(method = "cv", number = k)

store error rates for C5.0

error_rates_C50 <- numeric(k)

Perform k-fold cross-validation

for (fold in 1:k) {

 # Create training and testing subsets for the current fold

 set_index <- createFolds(df2$Region, k = k, list = TRUE)

 df2_train <- df2[-set_index[[fold]],]

 df2_test <- df2[set_index[[fold]],]

 # Train the C5.0 model

 C50_model <- C5.0(Region ~ ., data = df2_train)

 # Make predictions on the test data

 C50_predict <- predict(C50_model, newdata = df2_test)

 # Create a confusion matrix

 confusion_matrix <- table(df2_test$Region, C50_predict)

 # Print the confusion matrix for the current fold

 cat("Confusion Matrix (Fold", fold, "):\n")

 print(confusion_matrix)

 # Calculate the error rate for the current fold

 error_rate_C50 <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)

 # Print and store the error rate

 cat("Error Rate (Fold", fold, "):", error_rate_C50, "\n")

 error_rates_C50[fold] <- error_rate_C50

}

Calculate and print the average error rate across all folds

avg_error_rate_C50 <- mean(error_rates_C50)

cat("Average Error Rate:", avg_error_rate_C50, "\n")

cat("Length of error_rates_C50:", length(error_rates_C50), "\n")

```
Confusion Matrix (Fold 1 ):
   C50_predict
     1  2  3
  1 71  0 11
  2  2 78  2
  3 19  9 54
Error Rate (Fold 1 ): 0.1747967
Confusion Matrix (Fold 2 ):
   C50_predict
     1  2  3
  1 74  2  6
  2  0 77  5
  3 16 19 47
Error Rate (Fold 2 ): 0.195122
Confusion Matrix (Fold 3 ):
   C50_predict
     1  2  3
  1 71  2  9
  2  0 82  0
  3 13 19 50
Error Rate (Fold 3 ): 0.1747967
>
> # Calculate and print the average error rate across all folds
> avg_error_rate_C50 <- mean(error_rates_C50)
> cat("Average Error Rate:", avg_error_rate_C50, "\n")
Average Error Rate: 0.1815718
>
> cat("Length of error_rates_C50:", length(error_rates_C50), "\n")
Length of error_rates_C50: 3
>
```

e) Repeat Task (d) using the CART algorithm (available from the package 'rpart').

library(rpart)

library(caret)

library(rpart.plot)

index <- createDataPartition(df_no_outliers$Region, p=0.8, list=FALSE)

x_train <- df_no_outliers[index,]

x_test <- df_no_outliers[-index,]

y_train <- df_no_outliers$Region[index]

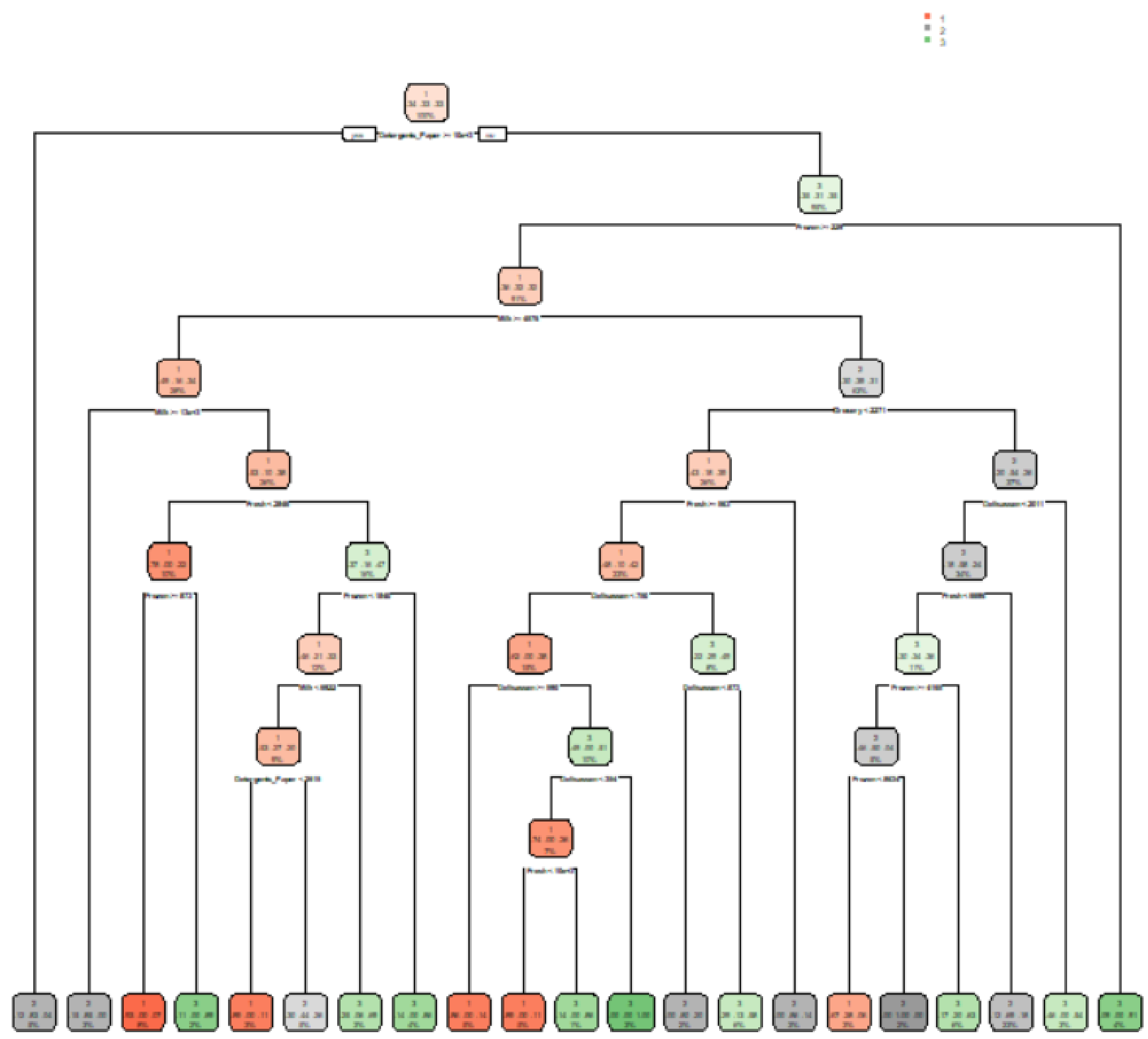
y_test <- df_no_outliers$Region[-index]

Shuffle dataframe

df2 <- df_no_outliers[order(runif(nrow(df_no_outliers))),]

tail(df2, 25)

```
# Split your data into training and testing sets

set.seed(123)  # For reproducibility

split_idx <- sample(1:nrow(df2), 0.7 * nrow(df2))  # 70% training, 30% testing

df2_train <- df2[split_idx, ]

df2_test <- df2[-split_idx, ]

# Train the CART model

cart_model <- rpart(Region ~ ., data = df2_train, method = "class")

# CART Predict

cart_predict <- predict(cart_model, df2_test, type = "class")

cart_predict

# Compare predictions

table(df2_test[, 2], cart_predict)

cart_model <- rpart(Region ~ ., data = df2_train, method = "class")

# Plot the CART decision tree

rpart.plot(cart_model)
```



```
# Set seed for reproducibility
set.seed(123)

# Define the number of folds (k)
k <- 3

# Create a training control object for k-fold cross-validation
train_control <- trainControl(method = "cv", number = k)

# Initialize a vector to store error rates
```

```
error_rates <- numeric(k)

for (fold in 1:k) {
  # Create training and testing subsets for the current fold
  set_index <- createFolds(df2$Region, k = k, list = TRUE)
  df2_train <- df2[-set_index[[fold]], ]
  df2_test <- df2[set_index[[fold]], ]

  # Train the CART model
  CART_model <- rpart(Region ~ ., data = df2_train, method = "class")

  # Make predictions on the test data
  CART_predict <- predict(CART_model, newdata = df2_test, type = "vector")

  # Create a confusion matrix
  confusion_matrix <- table(df2_test$Region, CART_predict)

  # Print the confusion matrix for the current fold
  cat("Confusion Matrix (Fold", fold, "):\n")
  print(confusion_matrix)

  # Calculate the error rate for the current fold
  error_rate_CART <- 1 - sum(diag(confusion_matrix)) / sum(confusion_matrix)

  # Print and store the error rate
  cat("Error Rate (Fold", fold, "):", error_rate_CART, "\n")
  error_rates[fold] <- error_rate_CART
}

# Calculate and print the average error rate across all folds
avg_error_rate_CART <- mean(error_rates)
cat("Average Error Rate:", avg_error_rate_CART, "\n")
```

cat("Length of error_rates_CART:", length(error_rates), "\n")

```
Confusion Matrix (Fold 1 ):
   CART_predict
     1  2  3
  1 62  9 11
  2  3 64 15
  3 19 26 37
Error Rate (Fold 1 ): 0.3373984
Confusion Matrix (Fold 2 ):
   CART_predict
     1  2  3
  1 54 21  9
  2  3 73  4
  3 13 30 39
Error Rate (Fold 2 ): 0.3252033
Confusion Matrix (Fold 3 ):
   CART_predict
     1  2  3
  1 63  5 11
  2  8 70  7
  3 25 15 42
Error Rate (Fold 3 ): 0.2886179
>
> # Calculate and print the average error rate across all folds
> avg_error_rate_CART <- mean(error_rates)
> cat("Average Error Rate:", avg_error_rate_CART, "\n")
Average Error Rate: 0.3170732
>
>
> cat("Length of error_rates_CART:", length(error_rates), "\n")
Length of error_rates_CART: 3
```

Calculate the difference in error rates

diff_error_rates <- error_rate_CART - error_rate_C50

F) Once you have carried out the above tasks (a)-(e), use hypothesis testing as discussed in Chapter 3 in the textbook to determine whether or not the error rate difference between the two classification algorithms is statistically significant given the confidence level of 98%. Your R program must print the confidence level, calculate and print the confidence interval of the error difference, and print a message to indicate whether or not the error rate difference is significant based on the calculated confidence interval and which model (the tree produced by C5.0 or the tree produced by CART) is your selected model. Note that this question asks for a two-sided confidence interval, not a one-sided one, so be careful when reading the probability table or using the appropriate R command.

Calculate the standard error of the difference

se_diff <- sqrt((error_rate_C50 * (1 - error_rate_C50) / nrow(df2_test)) +

(error_rate_CART * (1 - error_rate_CART) / nrow(df2_test)))

```
# Calculate the test statistic (t-value)
t_stat <- diff_error_rates / se_diff

# Calculate the degrees of freedom
df <- 2 * nrow(df2_test) - 2

# Calculate the two-sided p-value
p_value <- 2 * (1 - pt(abs(t_stat), df))

# Define the desired confidence level
conf_level <- 0.98

# Calculate the critical value
alpha <- 1 - conf_level
critical_value <- qt(1 - alpha / 2, df)

# Calculate and print the confidence interval of the error difference
margin_of_error <- critical_value * se_diff
lower_bound <- diff_error_rates - margin_of_error
upper_bound <- diff_error_rates + margin_of_error

# Print the confidence level
cat("Confidence Level:", conf_level * 100, "%\n")

# Print the confidence interval
cat("Confidence Interval of the Error Difference: [", lower_bound, ",", upper_bound, "]\n")



# Check the significance of the error rate difference and print the appropriate message
if (lower_bound > 0 && upper_bound < 0) {
  cat("The error rate difference is statistically significant at", conf_level * 100, "% confidence level.\n")
```

```
} else {

  cat("The confidence interval contains 0; therefore, the difference may not be statistically significant at", conf_level * 100, "% confidence level.\n")

}


# Indicate the selected model based on the error rate difference

if (diff_error_rates > 0) {

  cat("The C5.0 model has a lower error rate and is the selected model.\n")

} else if (diff_error_rates < 0) {

  cat("The CART model has a lower error rate and is the selected model.\n")

} else {

  cat("Both models have the same error rate.\n")

}
```

```
> # Print the confidence level
> cat("Confidence Level:", conf_level * 100, "%\n")
Confidence Level: 98 %
>
>
> # Print the confidence interval
> cat("Confidence Interval of the Error Difference: [", lower_bound, ",", upper_bound, "]\n")
Confidence Interval of the Error Difference: [ 0.07235355 , 0.2528497 ]
>
> # Check the significance of the error rate difference and print the appropriate message
> # Check the significance of the error rate difference and print the appropriate message
> if (lower_bound > 0 && upper_bound < 0) {
+   cat("The error rate difference is statistically significant at", conf_level * 100, "% confidence level.\n")
+ } else {
+   cat("The error rate difference is not statistically significant at", conf_level * 100, "% confidence level.\n")
+ }
The error rate difference is not statistically significant at 98 % confidence level.
>
>
> # Indicate the selected model based on the error rate difference
> if (diff_error_rates > 0) {
+   cat("The C5.0 model has a lower error rate and is the selected model.\n")
+ } else if (diff_error_rates < 0) {
+   cat("The CART model has a lower error rate and is the selected model.\n")
+ } else {
+   cat("Both models have the same error rate.\n")
+ }
The C5.0 model has a lower error rate and is the selected model.
>
```

```
###############LAST TASK#######################

# Determine the selected model

selected_model <- ifelse(error_rate_C50 < error_rate_CART, C50_model, CART_model)
```

```
# Function to predict class label for a new tuple
predict_class_label <- function(new_tuple) {
  predicted_label <- predict(selected_model, newdata = new_tuple, type = "class")
  return(predicted_label)
}
# Define a function to predict class label for a given tuple
predict_class_label <- function(tuple) {
  return(predict(CART_model, newdata = tuple, type = "class"))
}

# Function to predict class label for a given tuple
predict_class_label <- function(tuple, model) {
  predicted_label <- predict(model, newdata = tuple, type = "class")
  return(predicted_label)
}

# Function to predict class label for a given tuple
predict_class_label <- function(tuple, model) {
  predicted_label <- predict(model, newdata = tuple, type = "class")
  return(predicted_label)
}

# Testing the function with three different input tuples
tuple1 <- data.frame(Channel=2, Fresh=5283, Milk=13316, Grocery=20399, Frozen=1809, Detergents_Paper=8752, Delicassen=172)
predicted_label1 <- predict_class_label(tuple1, C50_model)
cat("For tuple1:\n")
print(tuple1)
cat("Predicted class label:", predicted_label1, "\n\n")

tuple2 <- data.frame(Channel=1, Fresh=11537, Milk=138, Grocery=5838, Frozen=18119, Detergents_Paper=13381, Delicassen=9806)
predicted_label2 <- predict_class_label(tuple2, C50_model)
```

cat("For tuple2:\n")

print(tuple2)

cat("Predicted class label:", predicted_label2, "\n\n")

tuple3 <- data.frame(Channel=1, Region=1, Fresh=15000, Milk=2000, Grocery=1000, Frozen=3000, Detergents_Paper=100, Delicassen=1000)

predicted_label3 <- predict_class_label(tuple3, C50_model)

cat("For tuple3:\n")

print(tuple3)

cat("Predicted class label:", predicted_label3, "\n\n")

```
> # Testing the function with three different input tuples
> tuple1 <- data.frame(Channel=2, Fresh=5283, Milk=13316, Grocery=20399, Frozen=1809, Detergents_Paper=8752, Delicassen=172)
> predicted_label1 <- predict_class_label(tuple1, C50_model)
> cat("For tuple1:\n")
For tuple1:
> print(tuple1)
  Channel Fresh  Milk Grocery Frozen Detergents_Paper Delicassen
1       2  5283 13316   20399   1809             8752        172
> cat("Predicted class label:", predicted_label1, "\n\n")
Predicted class label: 2

>
> tuple2 <- data.frame(Channel=1, Fresh=11537, Milk=138, Grocery=5838, Frozen=18119, Detergents_Paper=13381, Delicassen=9806)
> predicted_label2 <- predict_class_label(tuple2, C50_model)
> cat("For tuple2:\n")
For tuple2:
> print(tuple2)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       1 11537  138    5838  18119            13381       9806
> cat("Predicted class label:", predicted_label2, "\n\n")
Predicted class label: 1

>
> tuple3 <- data.frame(Channel=1, Region=1, Fresh=15000, Milk=2000, Grocery=1000, Frozen=3000, Detergents_Paper=100, Delicassen=1000)
> predicted_label3 <- predict_class_label(tuple3, C50_model)
> cat("For tuple3:\n")
For tuple3:
> print(tuple3)
  Channel Region Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       1      1 15000 2000    1000   3000              100       1000
> cat("Predicted class label:", predicted_label3, "\n\n")
Predicted class label: 3

>
```

##2nd time

Testing the function with three different input tuples

tuple1 <- data.frame(Channel=2, Fresh=2127, Milk=2053, Grocery=65580, Frozen=9049, Detergents_Paper=8171, Delicassen=18954)

predicted_label1 <- predict_class_label(tuple1, C50_model)

cat("For tuple1:\n")

print(tuple1)

cat("Predicted class label:", predicted_label1, "\n\n")

tuple2 <- data.frame(Channel=2, Fresh=15377, Milk=37748, Grocery=58438, Frozen=18159, Detergents_Paper=30381, Delicassen=8061)

predicted_label2 <- predict_class_label(tuple2, C50_model)

```
cat("For tuple2:\n")

print(tuple2)

cat("Predicted class label:", predicted_label2, "\n\n")


tuple3 <- data.frame(Channel=1, Fresh=1500, Milk=200, Grocery=100, Frozen=300, Detergents_Paper=1007, Delicassen=1080)

predicted_label3 <- predict_class_label(tuple3, C50_model)

cat("For tuple3:\n")

print(tuple3)

cat("Predicted class label:", predicted_label3, "\n\n")
```

```
> # Testing the function with three different input tuples
> tuple1 <- data.frame(Channel=2, Fresh=2127, Milk=2053, Grocery=65580, Frozen=9049, Detergents_Paper=8171, Delicassen=18954)
> predicted_label1 <- predict_class_label(tuple1, C50_model)
> cat("For tuple1:\n")
For tuple1:
> print(tuple1)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       2  2127 2053   65580   9049             8171      18954
> cat("Predicted class label:", predicted_label1, "\n\n")
Predicted class label: 1

>
> tuple2 <- data.frame(Channel=2, Fresh=15377, Milk=37748, Grocery=58438, Frozen=18159, Detergents_Paper=30381, Delicassen=8061)
> predicted_label2 <- predict_class_label(tuple2, C50_model)
> cat("For tuple2:\n")
For tuple2:
> print(tuple2)
  Channel Fresh  Milk Grocery Frozen Detergents_Paper Delicassen
1       2 15377 37748   58438  18159            30381       8061
> cat("Predicted class label:", predicted_label2, "\n\n")
Predicted class label: 1

>
> tuple3 <- data.frame(Channel=1, Fresh=1500, Milk=200, Grocery=100, Frozen=300, Detergents_Paper=1007, Delicassen=1080)
> predicted_label3 <- predict_class_label(tuple3, C50_model)
> cat("For tuple3:\n")
For tuple3:
> print(tuple3)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       1  1500  200     100    300             1007       1080
> cat("Predicted class label:", predicted_label3, "\n\n")
Predicted class label: 3

> |
```

```
# Testing 3RD TIME

tuple1 <- data.frame(Channel=1, Fresh=2787, Milk=1698, Grocery=2510, Frozen=65, Detergents_Paper=477, Delicassen=52)

predicted_label1 <- predict_class_label(tuple1, C50_model)

cat("For tuple1:\n")

print(tuple1)

cat("Predicted class label:", predicted_label1, "\n\n")


tuple2 <- data.frame(Channel=1, Fresh=2799, Milk=1700, Grocery=2520, Frozen=68, Detergents_Paper=480, Delicassen=60)

predicted_label2 <- predict_class_label(tuple2, C50_model)

cat("For tuple2:\n")

print(tuple2)
```

cat("Predicted class label:", predicted_label2, "\n\n")

tuple3 <- data.frame(Channel=2, Fresh=2137, Milk=3737, Grocery=19172, Frozen=1274, Detergents_Paper=17120, Delicassen=1059)

predicted_label3 <- predict_class_label(tuple3, C50_model)

cat("For tuple3:\n")

print(tuple3)

cat("Predicted class label:", predicted_label3, "\n\n")

```
> # Testing 3RD TIME
> tuple1 <- data.frame(Channel=1, Fresh=2787, Milk=1698, Grocery=2510, Frozen=65, Detergents_Paper=477, Delicassen=52)
> predicted_label1 <- predict_class_label(tuple1, C50_model)
> cat("For tuple1:\n")
For tuple1:
> print(tuple1)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       1  2787 1698    2510     65              477         52
> cat("Predicted class label:", predicted_label1, "\n\n")
Predicted class label: 3

>
> tuple2 <- data.frame(Channel=1, Fresh=2799, Milk=1700, Grocery=2520, Frozen=68, Detergents_Paper=480, Delicassen=60)
> predicted_label2 <- predict_class_label(tuple2, C50_model)
> cat("For tuple2:\n")
For tuple2:
> print(tuple2)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       1  2799 1700    2520     68              480         60
> cat("Predicted class label:", predicted_label2, "\n\n")
Predicted class label: 3

>
>
> tuple3 <- data.frame(Channel=2,  Fresh=2137, Milk=3737, Grocery=19172, Frozen=1274, Detergents_Paper=17120, Delicassen=1059)
> predicted_label3 <- predict_class_label(tuple3, C50_model)
> cat("For tuple3:\n")
For tuple3:
> print(tuple3)
  Channel Fresh Milk Grocery Frozen Detergents_Paper Delicassen
1       2  2137 3737   19172   1274            17120       1059
> cat("Predicted class label:", predicted_label3, "\n\n")
Predicted class label: 2

> |
```