American International University-Bangladesh (AIUB)

FACULTY OF SCIENCE and TECHNOLOGY

Final term Project
Fall 2023-24

INTRODUCTION TO DATA SCIENCE

Submitted To

Tohedul Islam

Submitted By

Group-3	
KAMIL AHMED	20-42058-1
MD. MORSHEDUL ISLAM	20-42645-1

Date: 25th Dec, 2023

Table of Contents

<u>Dataset Description:</u>

The "**Car Acceptability Classification Dataset**" available on Kaggle is a collection of categorical attributes used for classifying car acceptability. The dataset aims to assist in the classification of cars into different acceptability categories based on various features. Here are the key details:

Source:
The dataset is sourced from Kaggle, a well-known platform for data science and machine learning datasets. The direct link to access this dataset is [***Car Acceptability Classification Dataset***] (https://www.kaggle.com/datasets/subhajeetdas/car-acceptability-classification-dataset/data).

Attributes:
1. Buying_Price: Categorical data representing the buying price of the car with categories: vhigh, high, med, low.
2. Maintenance_Price: Categorical data indicating the maintenance price with categories: vhigh, high, med, low.

3. No_of_Doors: Categorical data specifying the number of doors available, with categories: 2, 3, 4, 5more.

4. Person_Capacity: Categorical data representing the seating capacity with categories: 2, 4, more.

5. Size_of_Luggage: Categorical data describing the luggage space with categories: small, med, big.

6. Safety: Categorical data indicating safety features with categories: low, med, high.

7. ***Car_Acceptability:*** Categorical data representing the target variable, the acceptability classification of the car, with categories: unacc, acc, good, vgood.

The dataset seems focused on categorical features, representing different aspects of a car that contribute to its acceptability classification. It appears suitable for classification tasks or predictive modeling aimed at determining the acceptability level of cars based on these attributes.

Providing a comprehensive understanding of these categorical attributes, this dataset can be valuable for exploratory analysis, classification models, or other data-driven approaches in the automotive domain.

Package Installation:

Code Segment:

```
install.packages("caret")

install.packages("klaR")

install.packages("gains")

install.packages("e1071")

install.packages("pROC")

install.packages("readr")

install.packages("tidyverse")
```

Description:

caret: - The `caret` package, short for Classification And Regression Training, is a versatile and comprehensive tool utilized for constructing, assessing, and fine-tuning predictive models. The software offers a cohesive interface for many machine learning methods and simplifies the processes of model training, testing, and performance assessment.

klaR: - `klaR` is a R package that is used for "Classification and Visualization". It offers a collection of classification algorithms and ways for visualizing classification models. The software incorporates a range of statistical approaches, including k-nearest neighbors (k-NN), Naive Bayes, and other classification methods.

gains: - The `gains` package provides functions for computing and visualizing cumulative gains curves and lift curves. These curves are commonly used in marketing and predictive modeling to evaluate model performance and compare the effectiveness of different models.

e1071: - `e1071` is a package that includes a wide range of functions for statistical learning, including support vector machines (SVM), Naive Bayes, clustering, and other machine learning methods. This package is multifunctional and can be used for applications such as classification, regression, and clustering.

pROC: - The `pROC` package is specifically built for analyzing and visualizing the performance of binary classification models. It offers routines to compute several metrics such as ROC (Receiver Operating Characteristic) curves, AUC (Area Under the ROC Curve), and confidence intervals for these measures.

readr: - `readr` is part of the tidyverse ecosystem and is used for efficient reading of rectangular data files (like CSV, TSV) into R. It provides faster and more consistent techniques for reading data compared to base R functions, making it suitable for data pretreatment and modification tasks.

tidyverse: - The `tidyverse` is a collection of R packages developed for data science, including data processing, visualization, and analysis. It consists of numerous programs (e.g., ggplot2, dplyr, tidyr) that work together following a uniform syntax, making data jobs more intuitive and efficient.

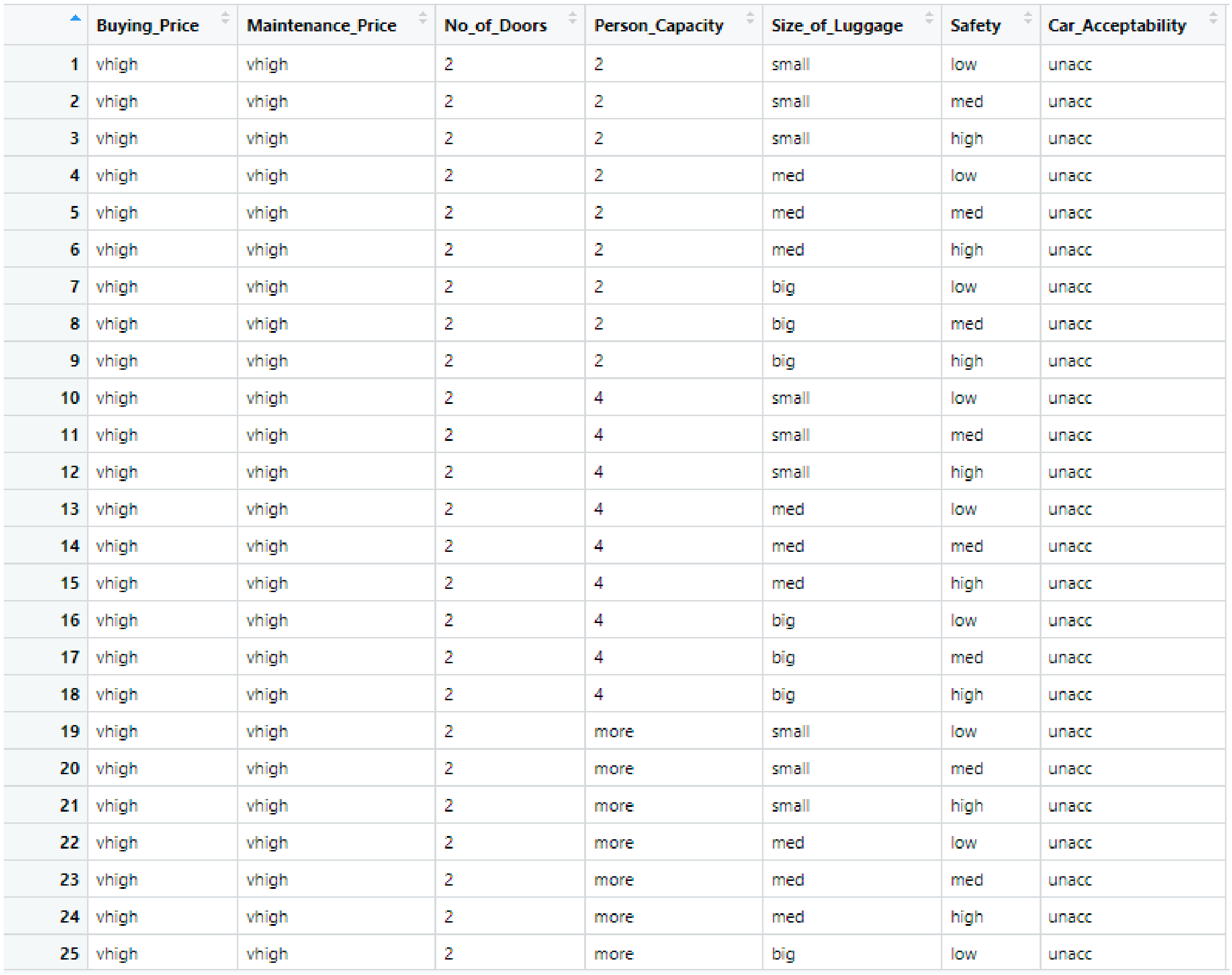
These packages offer a wide range of functions in data preprocessing, model development, evaluation, and visualization within the R programming language, catering to diverse elements of data science and machine learning processes.

Importing Dataset:

Code Segment:

```
data_src <- "D:/AIUB/11th Semester/Data Science/Final Project/car.csv"

CarAccept_Data <- read.csv(data_src, header = TRUE)

View(CarAccept_Data)
```

Output:



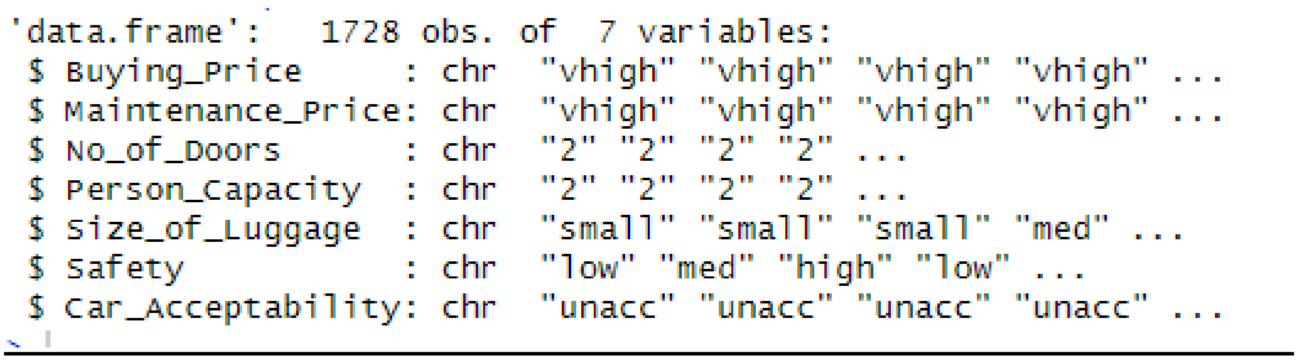
	Buying_Price	Maintenance_Price	No_of_Doors	Person_Capacity	Size_of_Luggage	Safety	Car_Acceptability
1	vhigh	vhigh	2	2	small	low	unacc
2	vhigh	vhigh	2	2	small	med	unacc
3	vhigh	vhigh	2	2	small	high	unacc
4	vhigh	vhigh	2	2	med	low	unacc
5	vhigh	vhigh	2	2	med	med	unacc
6	vhigh	vhigh	2	2	med	high	unacc
7	vhigh	vhigh	2	2	big	low	unacc
8	vhigh	vhigh	2	2	big	med	unacc
9	vhigh	vhigh	2	2	big	high	unacc
10	vhigh	vhigh	2	4	small	low	unacc
11	vhigh	vhigh	2	4	small	med	unacc
12	vhigh	vhigh	2	4	small	high	unacc
13	vhigh	vhigh	2	4	med	low	unacc
14	vhigh	vhigh	2	4	med	med	unacc
15	vhigh	vhigh	2	4	med	high	unacc
16	vhigh	vhigh	2	4	big	low	unacc
17	vhigh	vhigh	2	4	big	med	unacc
18	vhigh	vhigh	2	4	big	high	unacc
19	vhigh	vhigh	2	more	small	low	unacc
20	vhigh	vhigh	2	more	small	med	unacc
21	vhigh	vhigh	2	more	small	high	unacc
22	vhigh	vhigh	2	more	med	low	unacc
23	vhigh	vhigh	2	more	med	med	unacc
24	vhigh	vhigh	2	more	med	high	unacc
25	vhigh	vhigh	2	more	big	low	unacc

<u>Structure of the Dataset:</u>

<u>Code Segment:</u>

str(CarAccept_Data)

<u>Output:</u>



```
'data.frame':   1728 obs. of  7 variables:
 $ Buying_Price     : chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ Maintenance_Price: chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ No_of_Doors      : chr  "2" "2" "2" "2" ...
 $ Person_Capacity  : chr  "2" "2" "2" "2" ...
 $ Size_of_Luggage  : chr  "small" "small" "small" "med" ...
 $ Safety           : chr  "low" "med" "high" "low" ...
 $ Car_Acceptability: chr  "unacc" "unacc" "unacc" "unacc" ...
```

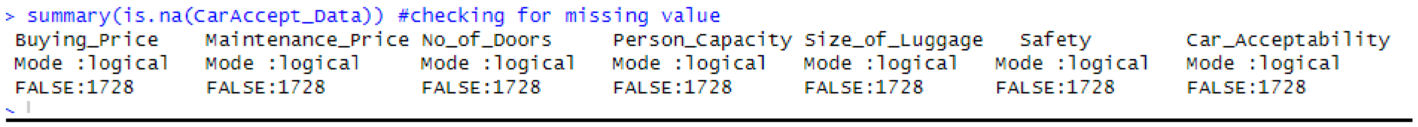
The car acceptability structure is a dataset designed for analyzing and predicting the acceptability level of cars based on various categorical attributes. All attributes within this dataset, including buying price, maintenance price, number of doors, seating capacity, luggage space, safety features, and car acceptability classification, are categorical in nature. Therefore, this dataset can be applied for classification algorithm, naïve bayes and chi-square test.

<u>Checking Missing Values:</u>

<u>Code Segment:</u>

summary(is.na(CarAccept_Data))

<u>Output:</u>



```
> summary(is.na(CarAccept_Data)) #checking for missing value
 Buying_Price    Maintenance_Price No_of_Doors     Person_Capacity Size_of_Luggage   Safety         Car_Acceptability
 Mode :logical   Mode :logical     Mode :logical   Mode :logical   Mode :logical   Mode :logical   Mode :logical    
 FALSE:1728      FALSE:1728        FALSE:1728      FALSE:1728      FALSE:1728      FALSE:1728      FALSE:1728       
```

There are no missing values in this dataset.

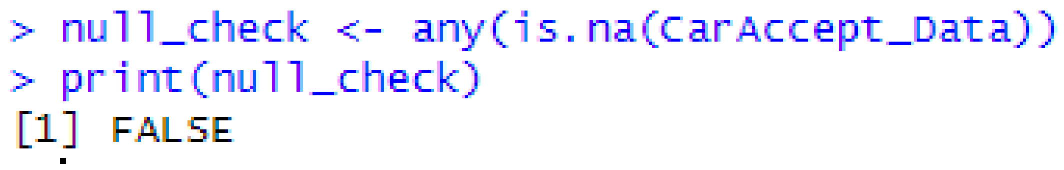
Checking NA Values:

Code Segment:

null_check <- any(is.na(CarAccept_Data))

print(null_check)

Output:



```
> null_check <- any(is.na(CarAccept_Data))
> print(null_check)
[1] FALSE
```

There are no null values in this dataset.

Applying Pearson Chi-Square Test:

Code Segment:

contingency_table1 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$Buying_Price)

contingency_table2 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$Maintenance_Price)

contingency_table3 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$No_of_Doors)

contingency_table4 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$Person_Capacity)

contingency_table5 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$Size_of_Luggage)

contingency_table6 <- table(CarAccept_Data$Car_Acceptability, CarAccept_Data$Safety)

chi_square_result1 <- chisq.test(contingency_table1)

chi_square_result2 <- chisq.test(contingency_table2)

chi_square_result3 <- chisq.test(contingency_table3)

chi_square_result4 <- chisq.test(contingency_table4)

chi_square_result5 <- chisq.test(contingency_table5)

chi_square_result6 <- chisq.test(contingency_table6)

print(chi_square_result1)

print(chi_square_result2)

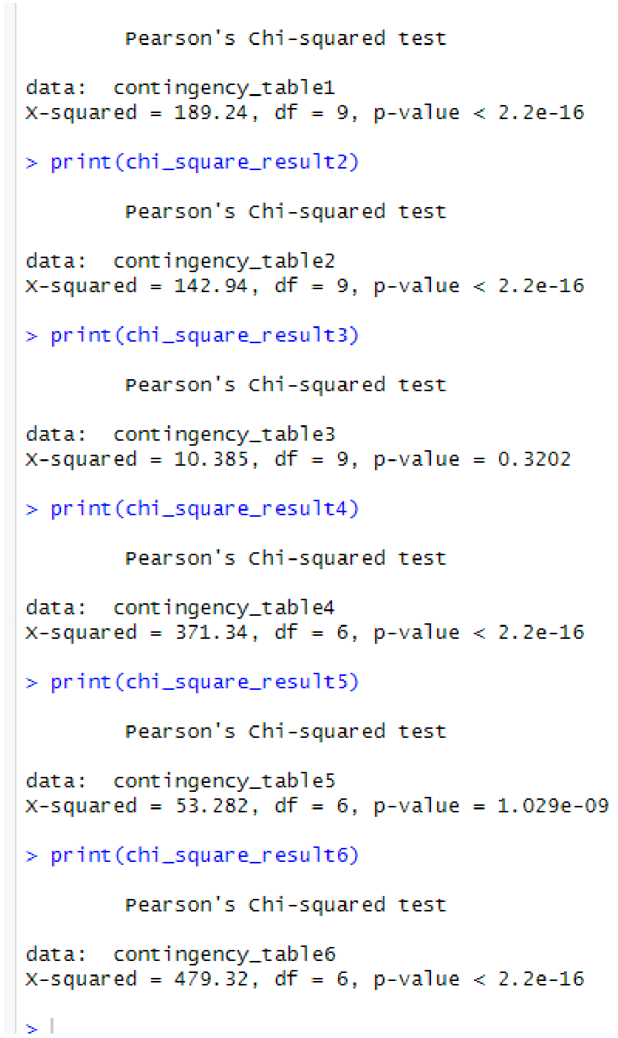
print(chi_square_result3)

print(chi_square_result4)

print(chi_square_result5)

print(chi_square_result6)

<u>Output:</u>



```
        Pearson's Chi-squared test

data:  contingency_table1
X-squared = 189.24, df = 9, p-value < 2.2e-16

> print(chi_square_result2)

        Pearson's Chi-squared test

data:  contingency_table2
X-squared = 142.94, df = 9, p-value < 2.2e-16

> print(chi_square_result3)

        Pearson's Chi-squared test

data:  contingency_table3
X-squared = 10.385, df = 9, p-value = 0.3202

> print(chi_square_result4)

        Pearson's Chi-squared test

data:  contingency_table4
X-squared = 371.34, df = 6, p-value < 2.2e-16

> print(chi_square_result5)

        Pearson's Chi-squared test

data:  contingency_table5
X-squared = 53.282, df = 6, p-value = 1.029e-09

> print(chi_square_result6)

        Pearson's Chi-squared test

data:  contingency_table6
X-squared = 479.32, df = 6, p-value < 2.2e-16

> 
```

The Chi-square test is a statistical test used to determine if there is a significant association between two categorical variables.

In the output of the Chi-squared tests, the p-value is a crucial indicator to determine the significance of the test. A small p-value (typically less than the chosen significance level, e.g., 0.05) suggests that you can reject the null hypothesis, indicating a significant association between the variables. In contrast, a large p-value suggests that there isn't enough evidence to reject the null hypothesis, indicating that the variables are independent.

Let's interpret each result based on the p-values:

chi_square_result1:

X-squared = 189.24, df = 9, p-value < 2.2e-16

Interpretation: There is a significant association between the variables represented by contingency_table1.

chi_square_result2:

X-squared = 142.94, df = 9, p-value < 2.2e-16

Interpretation: There is a significant association between the variables represented by contingency_table2.

chi_square_result3:

X-squared = 10.385, df = 9, p-value = 0.3202

Interpretation: There is no significant association between the variables represented by contingency_table3 (p-value > 0.05).

chi_square_result4:

X-squared = 371.34, df = 6, p-value < 2.2e-16

Interpretation: There is a significant association between the variables represented by contingency_table4.

chi_square_result5:

X-squared = 53.282, df = 6, p-value = 1.029e-09

Interpretation: There is a significant association between the variables represented by contingency_table5.

chi_square_result6:

X-squared = 479.32, df = 6, p-value < 2.2e-16

Interpretation: There is a significant association between the variables represented by contingency_table6.

In summary, based on the p-values:

contingency_table1, contingency_table2, contingency_table4, contingency_table5, and contingency_table6 show significant associations.

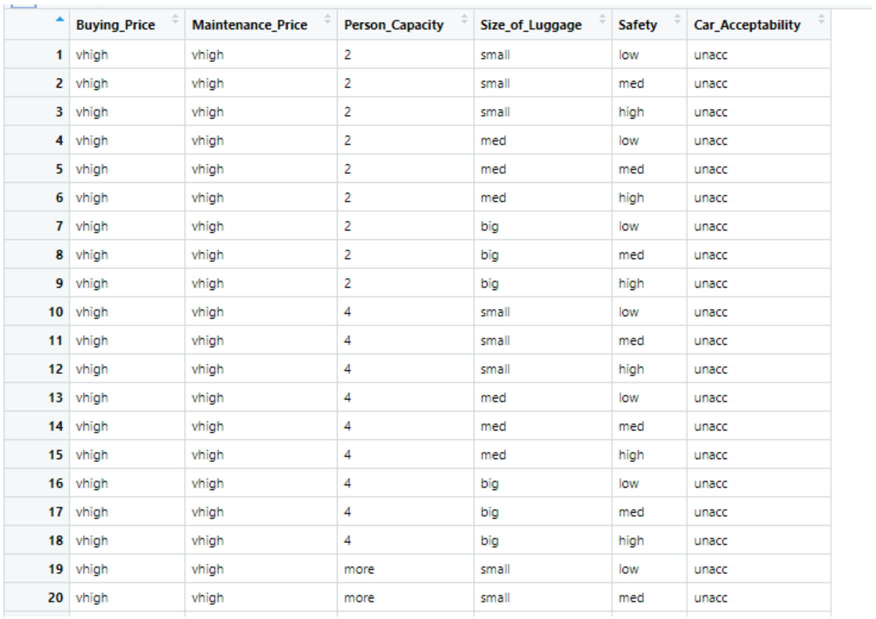
contingency_table3 does not show a significant association (p-value > 0.05).

<u>Filter Dataset:</u>

<u>Code Segment:</u>

```
significance_level <- 0.05

keep_attributes <- c("Buying_Price",

                    "Maintenance_Price",

                    "Person_Capacity",

                    "Size_of_Luggage",

                    "Safety",

                    "Car_Acceptability")

filtered_data <- CarAccept_Data[, keep_attributes]

View(filtered_data)
```

<u>Output:</u>



	Buying_Price	Maintenance_Price	Person_Capacity	Size_of_Luggage	Safety	Car_Acceptability
1	vhigh	vhigh	2	small	low	unacc
2	vhigh	vhigh	2	small	med	unacc
3	vhigh	vhigh	2	small	high	unacc
4	vhigh	vhigh	2	med	low	unacc
5	vhigh	vhigh	2	med	med	unacc
6	vhigh	vhigh	2	med	high	unacc
7	vhigh	vhigh	2	big	low	unacc
8	vhigh	vhigh	2	big	med	unacc
9	vhigh	vhigh	2	big	high	unacc
10	vhigh	vhigh	4	small	low	unacc
11	vhigh	vhigh	4	small	med	unacc
12	vhigh	vhigh	4	small	high	unacc
13	vhigh	vhigh	4	med	low	unacc
14	vhigh	vhigh	4	med	med	unacc
15	vhigh	vhigh	4	med	high	unacc
16	vhigh	vhigh	4	big	low	unacc
17	vhigh	vhigh	4	big	med	unacc
18	vhigh	vhigh	4	big	high	unacc
19	vhigh	vhigh	more	small	low	unacc
20	vhigh	vhigh	more	small	med	unacc

From the Pearson Chi-Square Test it has been found that,

"contingency_table3" which indicates "No_Of_Doors" attribute does not show a significant association (p-value > 0.05).

Thus the decision to remove or retain the attribute from the dataset depends on

the goals of the analysis and the significance of the relationship between variables.

As a result, the attribute "No_Of_Doors" has been removed from dataset.

Applying Naïve Bayes:

Code Segment:

install.packages("caret")

library(caret)

library(e1071)

naive_bayes <- naiveBayes(Car_Acceptability ~ ., data = train_data)

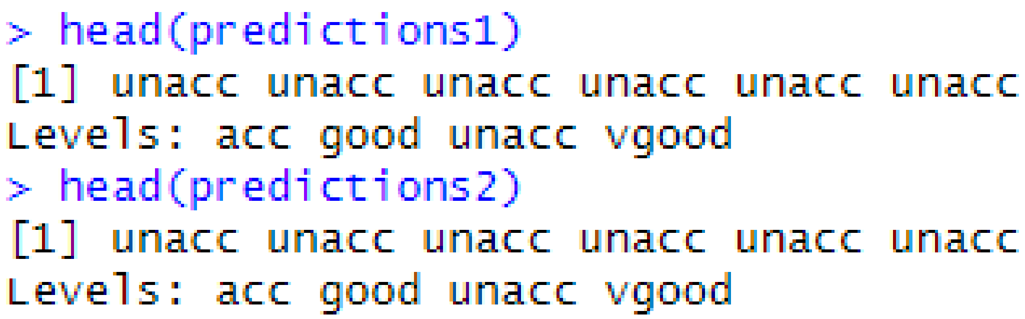
predictions1 <- predict(naive_bayes, newdata = test_data)

predictions2<- predict(naive_bayes, newdata = train_data)

head(predictions1)

head(predictions2)

Output:



```
> head(predictions1)
[1] unacc unacc unacc unacc unacc unacc
Levels: acc good unacc vgood
> head(predictions2)
[1] unacc unacc unacc unacc unacc unacc
Levels: acc good unacc vgood
```

Utilizes the naiveBayes function from the 'e1071' package to create a Naive Bayes classifier. The formula Car_Acceptability ~ . specifies the target variable 'Car_Acceptability' against all other available attributes in the 'train_data' dataset for model training. The predict function is used twice: predictions1 <- predict(naive_bayes, newdata = test_data) makes predictions on the 'test_data' using the trained Naive Bayes model and stores the predictions in 'predictions1'. predictions2 <- predict(naive_bayes, newdata = train_data) predicts outcomes on the 'train_data' dataset to evaluate the model's performance on the data it was trained on, and stores the predictions in 'predictions2'. Using head, it displays the first few predictions ('predictions1' and 'predictions2') generated by the Naive Bayes model on the testing and training datasets, respectively.

Dividing the data into training and test set:

Code Segment:

install.packages("caret")

library(caret)

set.seed(1)

split_CarIndex <- createDataPartition(filtered_data$Car_Acceptability, p = 0.8, list = FALSE)

train_data <- filtered_data[split_CarIndex,]

test_data <- filtered_data[-split_CarIndex,]

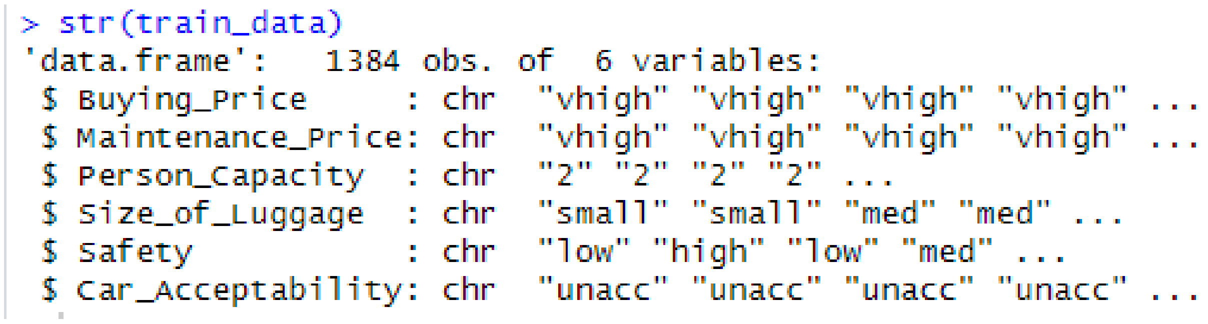
str(train_data)

str(test_data)

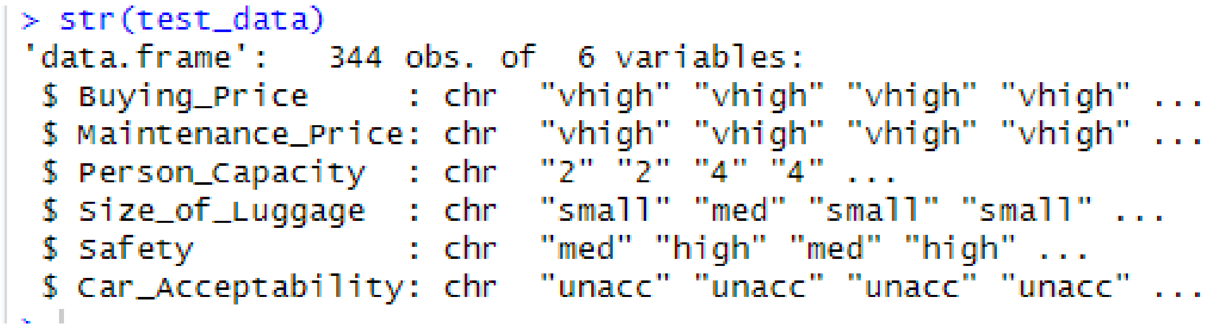
head(train_data)

head(test_data)

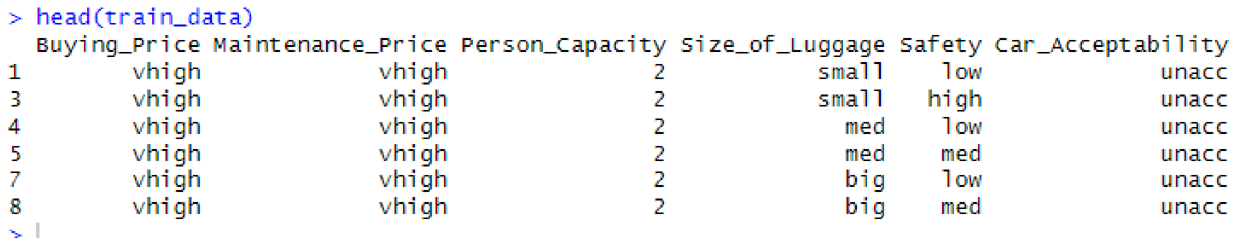
Output:



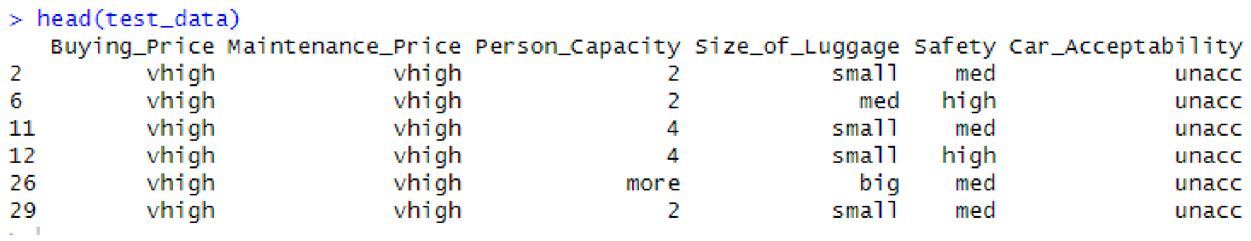
```
> str(train_data)
'data.frame':   1384 obs. of  6 variables:
 $ Buying_Price      : chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ Maintenance_Price: chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ Person_Capacity  : chr  "2" "2" "2" "2" ...
 $ Size_of_Luggage  : chr  "small" "small" "med" "med" ...
 $ Safety           : chr  "low" "high" "low" "med" ...
 $ Car_Acceptability: chr  "unacc" "unacc" "unacc" "unacc" ...
```



```
> str(test_data)
'data.frame':   344 obs. of  6 variables:
 $ Buying_Price      : chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ Maintenance_Price: chr  "vhigh" "vhigh" "vhigh" "vhigh" ...
 $ Person_Capacity  : chr  "2" "2" "4" "4" ...
 $ Size_of_Luggage  : chr  "small" "med" "small" "small" ...
 $ Safety           : chr  "med" "high" "med" "high" ...
 $ Car_Acceptability: chr  "unacc" "unacc" "unacc" "unacc" ...
```



```
> head(train_data)
  Buying_Price Maintenance_Price Person_Capacity Size_of_Luggage Safety Car_Acceptability
1        vhigh             vhigh               2           small    low             unacc
3        vhigh             vhigh               2           small   high             unacc
4        vhigh             vhigh               2             med    low             unacc
5        vhigh             vhigh               2             med    med             unacc
7        vhigh             vhigh               2             big    low             unacc
8        vhigh             vhigh               2             big    med             unacc
>
```



```
> head(test_data)
   Buying_Price Maintenance_Price Person_Capacity Size_of_Luggage Safety Car_Acceptability
2         vhigh             vhigh               2           small    med             unacc
6         vhigh             vhigh               2             med   high             unacc
11        vhigh             vhigh               4           small    med             unacc
12        vhigh             vhigh               4           small   high             unacc
26        vhigh             vhigh            more             big    med             unacc
29        vhigh             vhigh               2           small    med             unacc
```

It sets a specific seed (1 in this case) to ensure reproducibility of results across different runs. createDataPartition function from 'caret' is used to split the data based on the 'Car_Acceptability' variable. It creates an 80-20 split (80% for training and 20% for testing) while maintaining the distribution of classes using stratified sampling (by specifying list = FALSE). The resulting indices from createDataPartition are used to subset the 'filtered_data' into 'train_data' and 'test_data'. train_data contains 80% of the original data, used for training machine learning models. test_data contains the remaining 20%, used for evaluating model performance. The code displays the structure (str) of both 'train_data' and 'test_data', providing insights into the data types, attributes, and observations present in each set. Using head, it displays the first few rows of 'train_data' and 'test_data', offering a glimpse into the data content and structure.

<u>10-fold cross validation:</u>

<u>Code Segment:</u>

set.seed(1)

ctrl <- trainControl(method = "cv", number = 10)

Cross_Validation <- train(Car_Acceptability ~ .,

data = CarAccept_Data,

method = "naive_bayes",

trControl = ctrl)

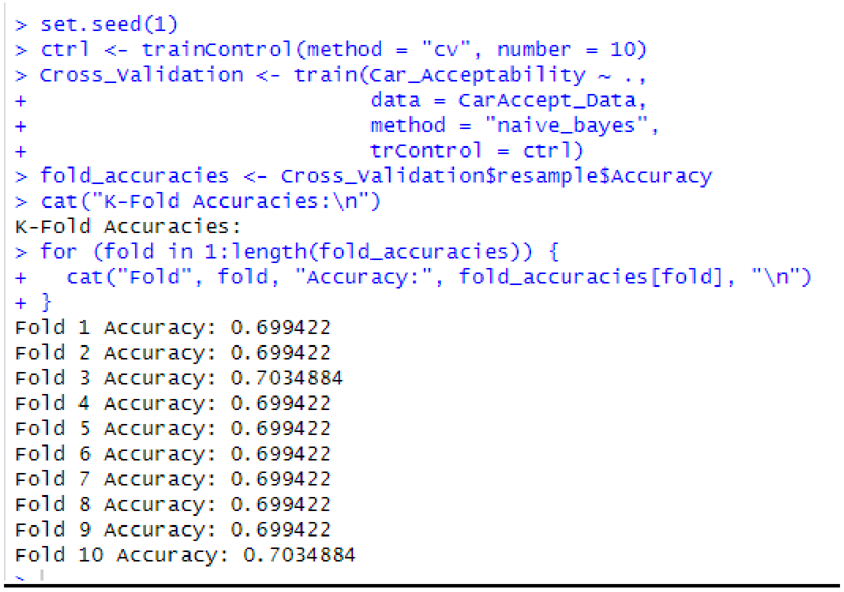
fold_accuracies <- Cross_Validation$resample$Accuracy

cat("K-Fold Accuracies:\n")

for (fold in 1:length(fold_accuracies)) {

cat("Fold", fold, "Accuracy:", fold_accuracies[fold], "\n") }

<u>Output:</u>



```
> set.seed(1)
> ctrl <- trainControl(method = "cv", number = 10)
> Cross_Validation <- train(Car_Acceptability ~ .,
+                           data = CarAccept_Data,
+                           method = "naive_bayes",
+                           trControl = ctrl)
> fold_accuracies <- Cross_Validation$resample$Accuracy
> cat("K-Fold Accuracies:\n")
K-Fold Accuracies:
> for (fold in 1:length(fold_accuracies)) {
+   cat("Fold", fold, "Accuracy:", fold_accuracies[fold], "\n")
+ }
Fold 1 Accuracy: 0.699422
Fold 2 Accuracy: 0.699422
Fold 3 Accuracy: 0.7034884
Fold 4 Accuracy: 0.699422
Fold 5 Accuracy: 0.699422
Fold 6 Accuracy: 0.699422
Fold 7 Accuracy: 0.699422
Fold 8 Accuracy: 0.699422
Fold 9 Accuracy: 0.699422
Fold 10 Accuracy: 0.7034884
```

Creates a 'trainControl' object named 'ctrl' using trainControl function, specifying 10-fold cross-validation (method = "cv", number = 10). This object defines the settings for the cross-validation procedure. Utilizes the train function from 'caret' to train a Naive Bayes classifier (method = "naive_bayes") using the 'CarAccept_Data' dataset. The formula Car_Acceptability ~ . indicates the prediction of the 'Car_Acceptability' variable based on all other available attributes. Retrieves the accuracy values for each fold from the cross-validated model stored in Cross_Validation$resample$Accuracy. Iterates through each fold, displaying the fold number and its respective accuracy on the console using a 'for' loop and cat function.

Generating the confusion matrix (for test data):

Code Segment:

Confusion_Matrix <- table(predictions1, test_data$Car_Acceptability)

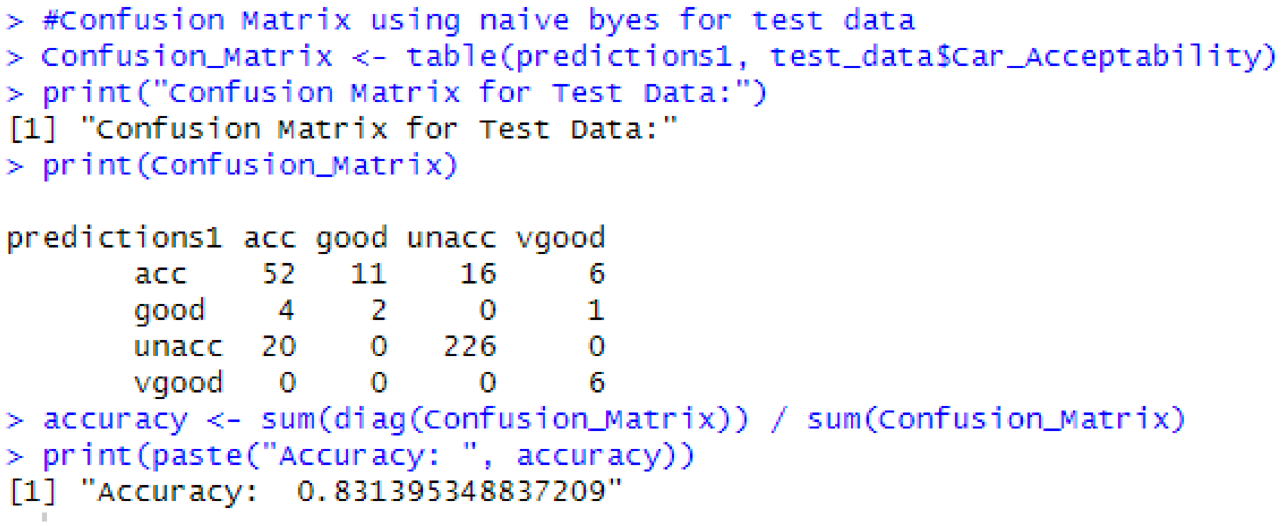
print("Confusion Matrix for Test Data:")

print(Confusion_Matrix)

accuracy <- sum(diag(Confusion_Matrix)) / sum(Confusion_Matrix)

print(paste("Accuracy: ", accuracy))

Output:



```
> #Confusion Matrix using naive byes for test data
> Confusion_Matrix <- table(predictions1, test_data$Car_Acceptability)
> print("Confusion Matrix for Test Data:")
[1] "Confusion Matrix for Test Data:"
> print(Confusion_Matrix)

predictions1 acc good unacc vgood
       acc     52   11    16     6
       good     4    2     0     1
       unacc   20    0   226     0
       vgood    0    0     0     6
> accuracy <- sum(diag(Confusion_Matrix)) / sum(Confusion_Matrix)
> print(paste("Accuracy: ", accuracy))
[1] "Accuracy:  0.831395348837209"
```

Generates a confusion matrix using the table function by comparing the predicted values ('predictions1') against the actual values from the 'Car_Acceptability' column in the test data ('test_data$Car_Acceptability'). This matrix assesses the model's performance by counting the number of correct and incorrect predictions for each class.

Generating the confusion matrix (for train data):

Code Segment:

Confusion_Matrix2 <- table(predictions2, train_data$Car_Acceptability)

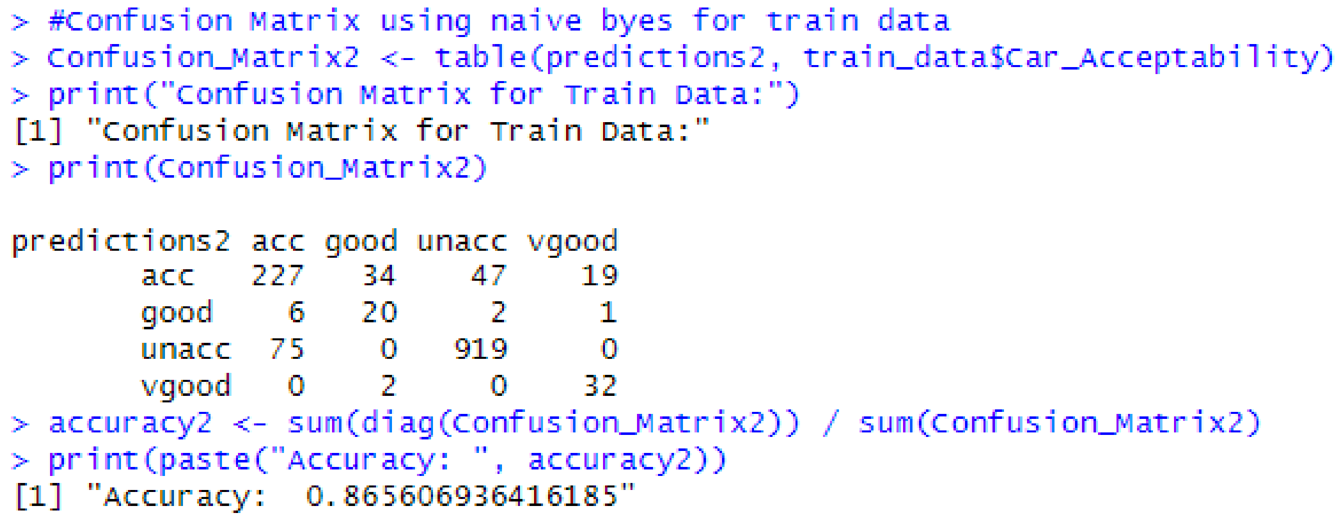
print("Confusion Matrix for Train Data:")

print(Confusion_Matrix2)

accuracy2 <- sum(diag(Confusion_Matrix2)) / sum(Confusion_Matrix2)

print(paste("Accuracy: ", accuracy2))

Output:



```
> #Confusion Matrix using naive byes for train data
> Confusion_Matrix2 <- table(predictions2, train_data$Car_Acceptability)
> print("Confusion Matrix for Train Data:")
[1] "Confusion Matrix for Train Data:"
> print(Confusion_Matrix2)

predictions2 acc good unacc vgood
       acc   227   34    47    19
       good    6   20     2     1
       unacc  75    0   919     0
       vgood   0    2     0    32
> accuracy2 <- sum(diag(Confusion_Matrix2)) / sum(Confusion_Matrix2)
> print(paste("Accuracy: ", accuracy2))
[1] "Accuracy:  0.865606936416185"
```

Generates a confusion matrix using the table function by comparing the predicted values ('predictions2') against the actual values from the 'Car_Acceptability' column in the test data ('test_data$Car_Acceptability'). This matrix assesses the model's performance by counting the number of correct and incorrect predictions for each class.

Reporting the Precision:

Code Segment:

Confusion_Matrix <- table(predictions1, test_data$Car_Acceptability)

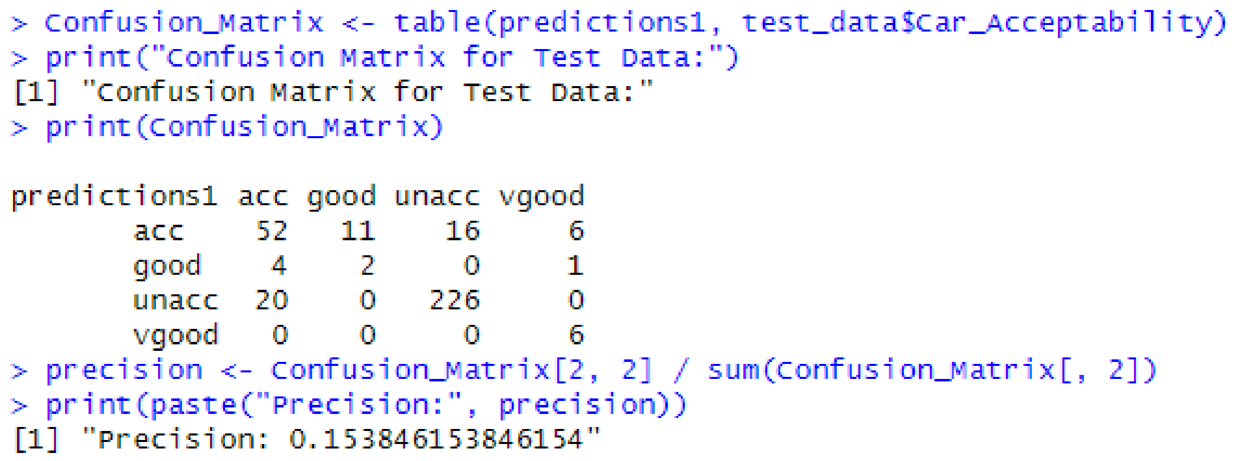
print("Confusion Matrix for Test Data:")

print(Confusion_Matrix)

precision <- Confusion_Matrix[2, 2] / sum(Confusion_Matrix[, 2])

print(paste("Precision:", precision))

Output:



```
> Confusion_Matrix <- table(predictions1, test_data$Car_Acceptability)
> print("Confusion Matrix for Test Data:")
[1] "Confusion Matrix for Test Data:"
> print(Confusion_Matrix)

predictions1 acc good unacc vgood
       acc    52   11    16     6
       good    4    2     0     1
       unacc  20    0   226     0
       vgood   0    0     0     6
> precision <- Confusion_Matrix[2, 2] / sum(Confusion_Matrix[, 2])
> print(paste("Precision:", precision))
[1] "Precision: 0.153846153846154"
```

Generates a confusion matrix using the table function by comparing the predicted values ('predictions1') against the actual values from the 'Car_Acceptability' column in the test data ('test_data$Car_Acceptability'). This matrix evaluates the model's performance by counting true positives, true negatives, false positives, and false negatives.

Calculates Precision, which measures the accuracy of positive predictions among the predicted positive instances. Computes Precision using the formula: True Positives / (True Positives + False Positives).

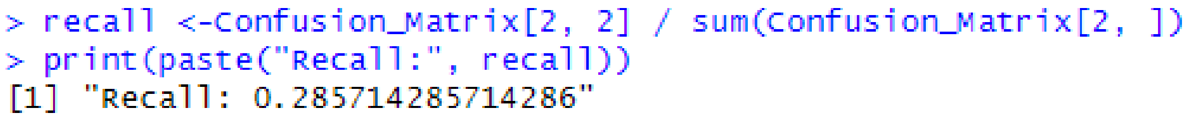
Reporting the Recall:

Code Segment:

recall <-Confusion_Matrix[2, 2] / sum(Confusion_Matrix[2,])

print(paste("Recall:", recall))

Output:



```
> recall <-Confusion_Matrix[2, 2] / sum(Confusion_Matrix[2, ])
> print(paste("Recall:", recall))
[1] "Recall: 0.285714285714286"
```

Computes Recall, which measures the coverage of actual positive instances captured by the model. Calculates Recall using the formula: True Positives / (True Positives + False Negatives).

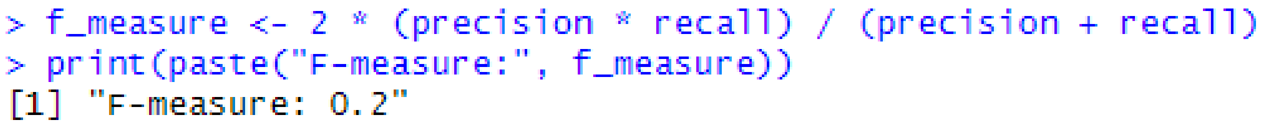
Reporting the F- measure value:

Code Segment:

f_measure <- 2 * (precision * recall) / (precision + recall)

print(paste("F-measure:", f_measure))

Output:



```
> f_measure <- 2 * (precision * recall) / (precision + recall)
> print(paste("F-measure:", f_measure))
[1] "F-measure: 0.2"
```

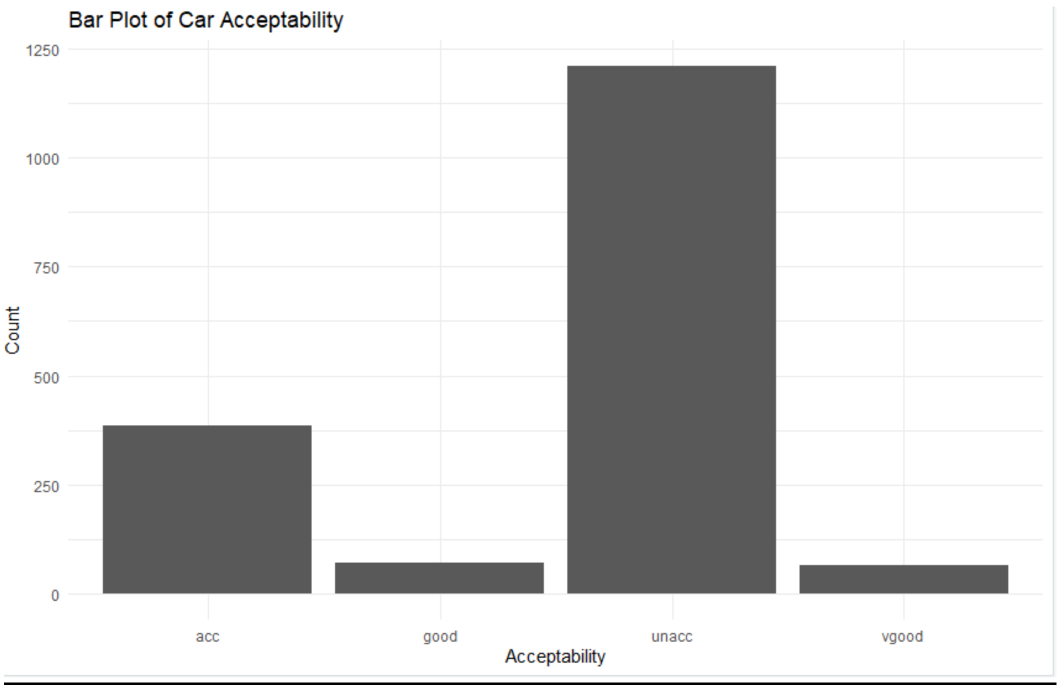
*Calculates F-measure, which combines Precision and Recall into a single metric, providing a balance between Precision and Recall. Computes F-measure using the formula: 2 * (Precision * Recall) / (Precision + Recall).*

Bar Plot of Car Acceptability:

Code Segment:

```
library(ggplot2)

ggplot(CarAccept_Data, aes(x = Car_Acceptability)) +
  geom_bar() +
  ggtitle("Bar Plot of Car Acceptability") +
  xlab("Acceptability") +
  ylab("Count") +
  theme_minimal()
```

Output:



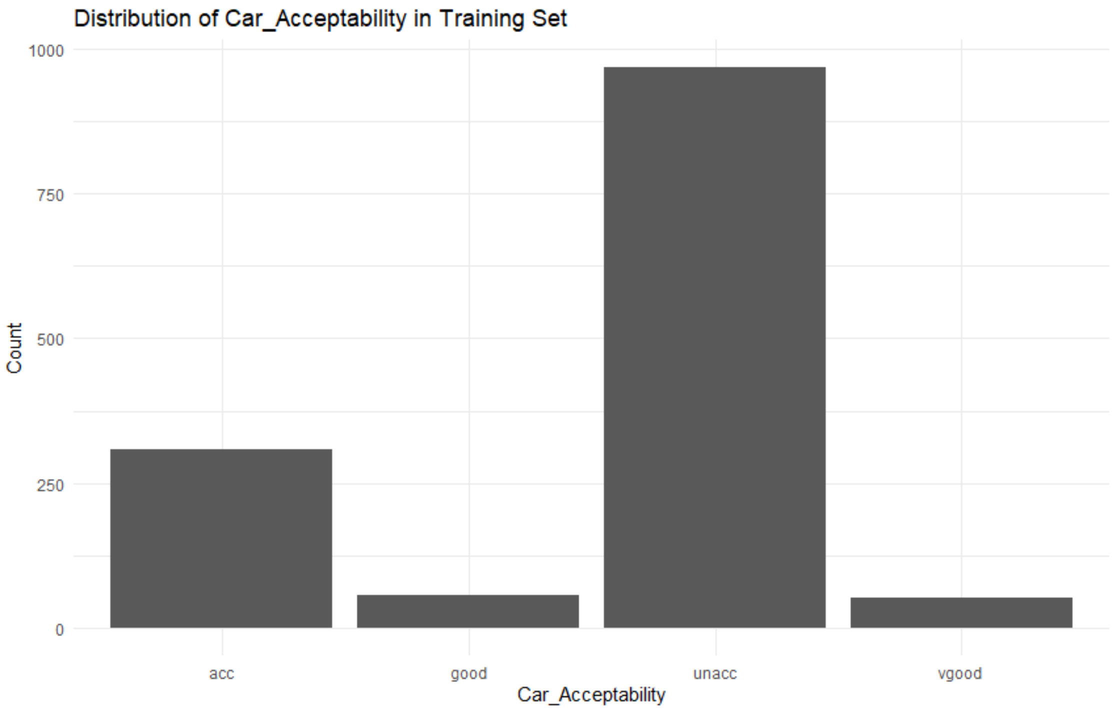
The x-axis of the plot represents the distinct categories or levels of car acceptability, potentially including categories such as 'unacc' (unacceptable), 'acc' (acceptable), 'good', and 'vgood' (very good). The y-axis displays the count or frequency of occurrences for each category of car acceptability, showing the number of instances belonging to each level. The height of each bar in the plot corresponds to the count of instances falling under each specific car acceptability category. Taller bars represent higher frequencies, indicating a greater number of occurrences for that particular acceptability level. In this dataset we can see that the unacc has highest count and vgood has the lowest count.

Bar Plot For Train Data :

Code Segment:

```
ggplot(train_data, aes(x = Car_Acceptability)) +
  geom_bar() +
  ggtitle("Distribution of Car_Acceptability in Training Set") +
  xlab("Car_Acceptability") +
  ylab("Count") +
  theme_minimal()
```

Output:



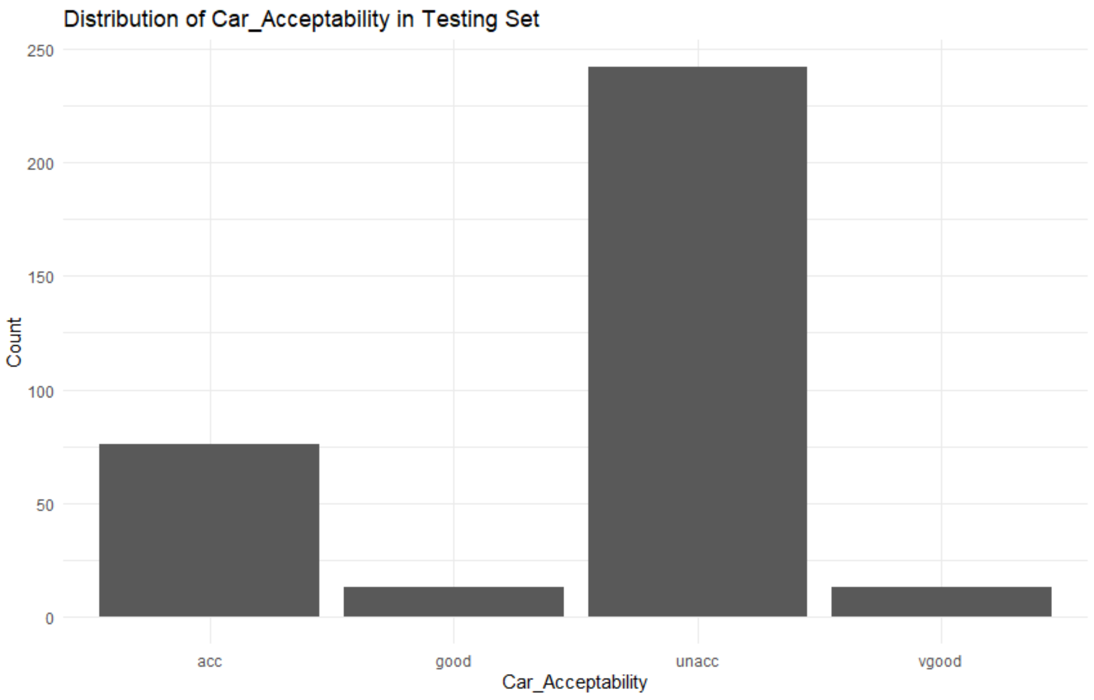
The x-axis of the plot represents the distinct categories or levels of car acceptability, potentially including categories such as 'unacc' (unacceptable), 'acc' (acceptable), 'good', and 'vgood' (very good). The y-axis displays the count or frequency of occurrences for each category of car acceptability, showing the number of instances belonging to each level. The height of each bar in the plot corresponds to the count of instances falling under each specific car acceptability category. Taller bars represent higher frequencies, indicating a greater number of occurrences for that particular acceptability level. In this dataset we can see that the unacc has highest count and vgood has the lowest count.

Bar Plot For Test Data :

Code Segment:

```
ggplot(test_data, aes(x = Car_Acceptability)) +
  geom_bar() +
  ggtitle("Distribution of Car_Acceptability in Testing Set") +
  xlab("Car_Acceptability") +
  ylab("Count") +
  theme_minimal()
```

Output:



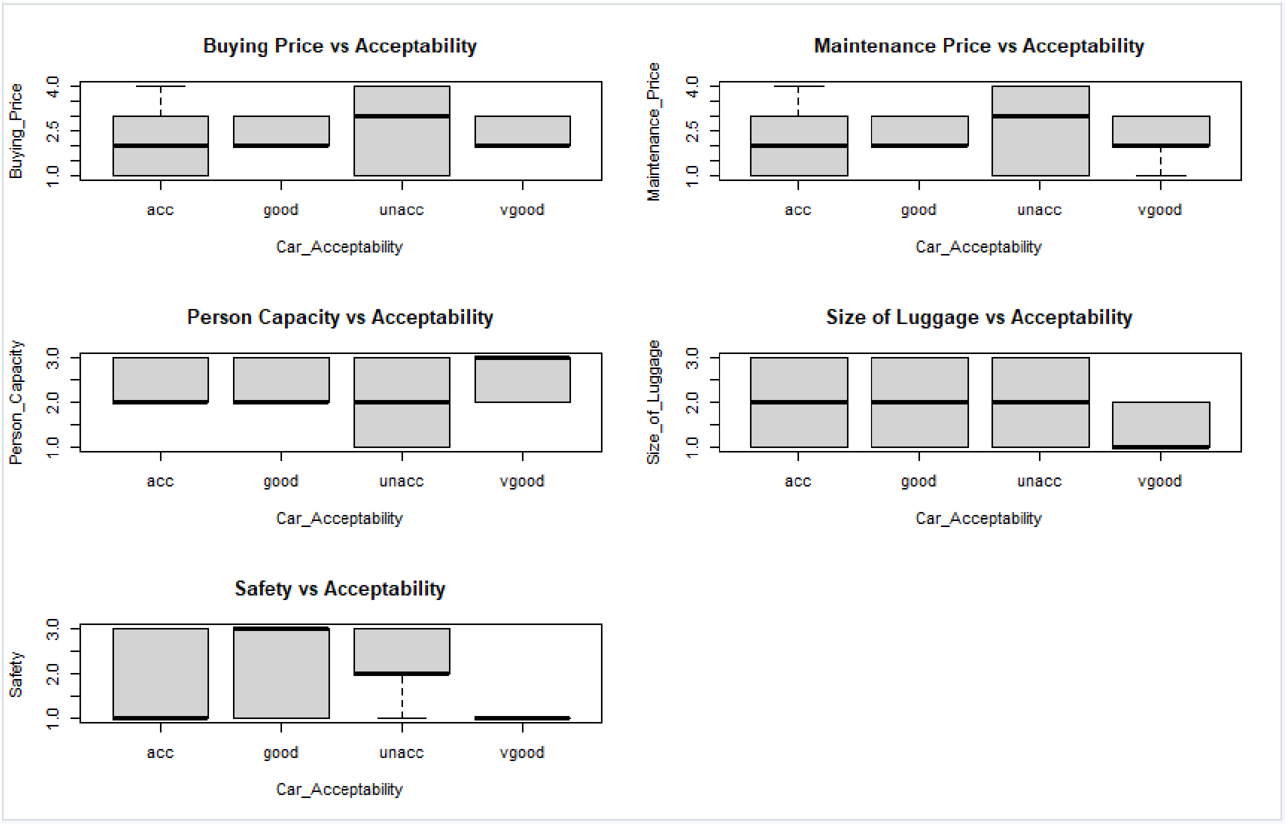
The x-axis of the plot represents the distinct categories or levels of car acceptability, potentially including categories such as 'unacc' (unacceptable), 'acc' (acceptable), 'good', and 'vgood' (very good). The y-axis displays the count or frequency of occurrences for each category of car acceptability, showing the number of instances belonging to each level. The height of each bar in the plot corresponds to the count of instances falling under each specific car acceptability category. Taller bars represent higher frequencies, indicating a greater number of occurrences for that particular acceptability level. In this dataset we can see that the unacc has highest count and vgood has the lowest count.

Box Plot For Buying Price, Maintenance, Person Capacity, Size of Luggage, Safety vs Acceptability:

Code Segment:

```
CarAccept_Data$Buying_Price <- as.factor(CarAccept_Data$Buying_Price)

CarAccept_Data$Maintenance_Price <- as.factor(CarAccept_Data$Maintenance_Price)

CarAccept_Data$Person_Capacity <- as.factor(CarAccept_Data$Person_Capacity)

CarAccept_Data$Size_of_Luggage <- as.factor(CarAccept_Data$Size_of_Luggage)

CarAccept_Data$Safety <- as.factor(CarAccept_Data$Safety)

CarAccept_Data$Car_Acceptability <- as.factor(CarAccept_Data$Car_Acceptability)


par(mfrow = c(3, 2))  # Adjust the grid layout as needed


boxplot(Buying_Price ~ Car_Acceptability, data = CarAccept_Data, main = "Buying Price vs Acceptability")

boxplot(Maintenance_Price ~ Car_Acceptability, data = CarAccept_Data, main = "Maintenance Price vs Acceptability")

boxplot(Person_Capacity ~ Car_Acceptability, data = CarAccept_Data, main = "Person Capacity vs Acceptability")

boxplot(Size_of_Luggage ~ Car_Acceptability, data = CarAccept_Data, main = "Size of Luggage vs Acceptability")

boxplot(Safety ~ Car_Acceptability, data = CarAccept_Data, main = "Safety vs Acceptability")
```

Output:



This code snippet begins by preparing the dataset ('CarAccept_Data') for visualization by converting categorical variables to factors, enabling the creation of box plots. It then generates individual box plots for each categorical attribute in relation to the 'Car_Acceptability' classes.

Buying Price vs. Acceptability: Displays how different buying price categories are distributed across various acceptability levels of cars.

Maintenance Price vs. Acceptability: Illustrates the distribution of maintenance price categories concerning car acceptability classes.

Person Capacity vs. Acceptability: Visualizes the distribution of seating capacity categories across different levels of car acceptability.

Size of Luggage vs. Acceptability: Represents the distribution of luggage size categories relative to different car acceptability levels.

Safety vs. Acceptability: Depicts the distribution of safety feature categories concerning different car acceptability classes.