Practicum2.Rmd

Nithya sarabudla

2024-03-17

Problem 1

2.Explore the dataset as you see fit and that allows you to get a sense of the data and get comfortable with it.

```r
# Setting seed for reproducibility
set.seed(111)

# Reading the data from a CSV file
mushroom_data <- read.csv("/Users/nithyasarabudla/DA5030/mushrooms.csv", na.strings="?")

# Checking the basic structure of the data
str(mushroom_data)
```

```
## 'data.frame':    8124 obs. of  23 variables:
##  $ class                   : chr  "p" "e" "e" "p" ...
##  $ cap.shape               : chr  "x" "x" "b" "x" ...
##  $ cap.surface             : chr  "s" "s" "s" "y" ...
##  $ cap.color               : chr  "n" "y" "w" "w" ...
##  $ bruises                 : chr  "t" "t" "t" "t" ...
##  $ odor                    : chr  "p" "a" "l" "p" ...
##  $ gill.attachment         : chr  "f" "f" "f" "f" ...
##  $ gill.spacing            : chr  "c" "c" "c" "c" ...
##  $ gill.size               : chr  "n" "b" "b" "n" ...
##  $ gill.color              : chr  "k" "k" "n" "n" ...
##  $ stalk.shape             : chr  "e" "e" "e" "e" ...
##  $ stalk.root              : chr  "e" "c" "c" "e" ...
##  $ stalk.surface.above.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.surface.below.ring: chr  "s" "s" "s" "s" ...
##  $ stalk.color.above.ring  : chr  "w" "w" "w" "w" ...
##  $ stalk.color.below.ring  : chr  "w" "w" "w" "w" ...
##  $ veil.type               : chr  "p" "p" "p" "p" ...
##  $ veil.color              : chr  "w" "w" "w" "w" ...
##  $ ring.number             : chr  "o" "o" "o" "o" ...
##  $ ring.type               : chr  "p" "p" "p" "p" ...
##  $ spore.print.color       : chr  "k" "n" "n" "k" ...
##  $ population              : chr  "s" "n" "n" "s" ...
##  $ habitat                 : chr  "u" "g" "m" "u" ...
```

```r
# Checking the dimensions of the data
cat("Dimensions of the dataset", dim(mushroom_data), "\n")
```

```
## Dimensions of the dataset 8124 23
```

```r
# Checking all the unique values from veil.type
unique(mushroom_data$veil.type)
```

```
## [1] "p"
```

As there is just one distinct value, it is better to drop this column

```r
sum(is.na(mushroom_data$stalk.root))
```

```
## [1] 2480
```

As there are more than 20% missing values in this column, it is better to drop this column too.

```r
# Since the veil.type has just 1 distinct value and there are several missing values in stalk.root, dro
cols_to_remove <- c("veil.type", "stalk.root")
mushroom_data <- mushroom_data[, !names(mushroom_data) %in% cols_to_remove]
# Converting the processed data to data frame by converting the columns to factors
mushroom_data <- as.data.frame(lapply(mushroom_data, as.factor))
```

3.Split the combined data set 70/30% so you retain 30% for validation using random sampling without replacement. Use a fixed seed so you produce the same results each time you run the code. Going forward you will use the 70% data set for training and the 30% data set for validation and determine accuracy.

```r
# Using library caret for splitting
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```r
# Setting seed for reproductibility
set.seed(111)
# Splitting the data frame in 70% and 30% ratios without replacement
index <- createDataPartition(mushroom_data$class, p=0.7, list=FALSE)

df_train <- mushroom_data[index,]
df_valid <- mushroom_data[-index,]
```

```r
# Checking value counts for class in training set
table(df_train$class)
```

```
##
##    e    p
## 2946 2742
```

```
# Checking value counts for class in validation set
table(df_valid$class)
```

```
##
##    e    p
## 1262 1174
```

```
# Checking the dimensions of training and testing sets
cat("Dimensions of the training set:", dim(df_train), "\n")
```

```
## Dimensions of the training set: 5688 21
```

```
cat("Dimensions of the testing set:", dim(df_valid), "\n")
```

```
## Dimensions of the testing set: 2436 21
```

4.Using the Naive Bayes Classification algorithm from the KlaR package, build a binary classifier that predicts the poisonous mushrooms. You need to transform continuous variables into categorical variables by binning, using equal size bins from min to max (ignore this if there is no numerical variables in the data).

```
library(klaR)
```

```
## Loading required package: MASS
```

```
# Building and fitting the Naive Bayes model usin training data
nb_model <- NaiveBayes(class ~ ., data = df_train)
# Validating using the validation data
nb_predictions <- predict(nb_model, newdata = df_valid)
```

5.Build a confusion matrix for the classifier from (4) and comment on it, e.g., explain what it means.

```
# A function to compute various metrics using confusion matrix and print them
compute_metrics <- function(conf_matrix) {
  TP <- conf_matrix[2, 2]
  TN <- conf_matrix[1, 1]
  FP <- conf_matrix[1, 2]
  FN <- conf_matrix[2, 1]

  accuracy <- (TP + TN) / sum(conf_matrix)
  precision <- TP / (TP + FP)
  recall <- TP / (TP + FN)
  f1_score <- 2 * precision * recall / (precision + recall)

  cat("Accuracy:", accuracy, "\n")
  cat("Precision:", precision, "\n")
  cat("Recall:", recall, "\n")
  cat("F1 Score:", f1_score, "\n")
}
```

```r
# Computing the confusion matrix and printing the metrics
confusion_matrix <- table(nb_predictions$class, df_valid$class)
print("NAIVE BAYES MODEL")
```

```
## [1] "NAIVE BAYES MODEL"
```

```r
compute_metrics(confusion_matrix)
```

```
## Accuracy: 0.9417077
## Precision: 0.8858603
## Recall: 0.9923664
## F1 Score: 0.9360936
```

```r
print(confusion_matrix)
```

```
##
##        e    p
##   e 1254  134
##   p    8 1040
```

- True Positives (TP): 1040 (correctly predicted poisonous mushrooms)
- True Negatives (TN): 1254 (correctly predicted non-poisonous mushrooms)
- False Positives (FP): 134 (incorrectly predicted as poisonous)
- False Negatives (FN): 8 (incorrectly predicted as non-poisonous)

The high number of true positives and true negatives indicates the model's ability to accurately classify mushrooms. The low number of false positives and false negatives suggests that the model's predictions are mostly correct, with only a few misclassifications.

6.Compare the results of Naive Bayes with the performance of RIPPER algorithm, which algorithm works better on this dataset? how do you interpret the results?

```r
# install.packages("C50")
library(C50)

# Building RIPPER model on the training data
ripper_model <- C5.0(df_train[, -which(names(df_train) == "class")], df_train$class)
# Validating using the validation data
ripper_predictions <- predict(ripper_model, newdata = df_valid)

# Computing the confusion matrix and printing the metrics
ripper_confusion_matrix <- table(ripper_predictions, df_valid$class)
print("RIPPER MODEL")
```

```
## [1] "RIPPER MODEL"
```

```r
compute_metrics(ripper_confusion_matrix)
```

```
## Accuracy: 1
## Precision: 1
## Recall: 1
## F1 Score: 1
```

```r
print(ripper_confusion_matrix)
```

```
##
## ripper_predictions    e    p
##                  e 1262    0
##                  p    0 1174
```

RIPPER Model:

- True Positives (TP): 1174
- True Negatives (TN): 1262
- False Positives (FP): 0
- False Negatives (FN): 0

Comparison:

Both models have high numbers of true positives and true negatives, indicating their ability to correctly classify mushrooms. The Naive Bayes model has a few false positives and false negatives, while the RIPPER model has none. This suggests that the RIPPER model is more conservative in its predictions.

Contrast:

The Naive Bayes model has a higher number of false positives and false negatives compared to the RIPPER model, which has none. This indicates that the Naive Bayes model may be slightly more prone to misclassification.

The RIPPER model's perfect classification in the validation set suggests that it may have overfit the training data, as it perfectly classifies even unseen data. This could be a concern if the model is not generalizing well to new data.

Overall, both models perform well, but the Naive Bayes model shows a few misclassifications, while the RIPPER model demonstrates perfect classification on the validation set, possibly indicating overfitting.

Problem 2

2.Explore the dataset as you see fit and that allows you to get a sense of the data and get comfortable with it.

```r
# Reading the data from a CSV file
bc_data <- read.csv("/Users/nithyasarabudla/DA5030/Wisonsin_breast_cancer_data.csv")
# Checking the basic structure of data
str(bc_data)
```

```
## 'data.frame':    569 obs. of  33 variables:
##  $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844
##  $ diagnosis              : chr  "M" "M" "M" "M" ...
##  $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
##  $ area_mean              : num  1001 1326 1203 386 1297 ...
##  $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave.points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se              : num  1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst             : num  2019 1956 1709 568 1575 ...
##  $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
##  $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
##  $ X                      : logi  NA NA NA NA NA NA ...
```

```
# Checking for the missing values in data.
as.matrix(colSums(is.na(bc_data)))
```

```
##                         [,1]
## id                         0
## diagnosis                  0
## radius_mean                0
## texture_mean               0
## perimeter_mean             0
## area_mean                  0
## smoothness_mean            0
## compactness_mean           0
## concavity_mean             0
## concave.points_mean        0
## symmetry_mean              0
## fractal_dimension_mean     0
## radius_se                  0
## texture_se                 0
## perimeter_se               0
## area_se                    0
## smoothness_se              0
## compactness_se             0
## concavity_se               0
## concave.points_se          0
## symmetry_se                0
## fractal_dimension_se       0
## radius_worst               0
## texture_worst              0
## perimeter_worst            0
## area_worst                  0
## smoothness_worst            0
## compactness_worst           0
## concavity_worst             0
## concave.points_worst        0
## symmetry_worst              0
## fractal_dimension_worst     0
## X                         569
```

```
# Total missing values in data
sum(is.na(bc_data))
```

```
## [1] 569
```

```
cat("Dimensions of the dataset", dim(bc_data), "\n")
```

```
## Dimensions of the dataset 569 33
```

All the missing values are from this unknown `X` column which is completely empty. So, better to drop it. And the `id` column also serves no purpose in predicting the cancer state of an individual. So, dropping it.

```
# Dropping both the "X" and "id" columns, then converting the label `diagnosis` to factor.
bc_data <- bc_data[, !(names(bc_data) == "X")]
bc_data <- bc_data[, !(names(bc_data) == "id")]
bc_data$diagnosis <- factor(bc_data$diagnosis)
# Printing the head
head(bc_data)
```

```
##   diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   compactness_mean concavity_mean concave.points_mean symmetry_mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   fractal_dimension_mean radius_se texture_se perimeter_se area_se
## 1                0.07871    1.0950     0.9053        8.589  153.40
## 2                0.05667    0.5435     0.7339        3.398   74.08
## 3                0.05999    0.7456     0.7869        4.585   94.03
## 4                0.09744    0.4956     1.1560        3.445   27.23
## 5                0.05883    0.7572     0.7813        5.438   94.44
## 6                0.07613    0.3345     0.8902        2.217   27.19
##   smoothness_se compactness_se concavity_se concave.points_se symmetry_se
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   fractal_dimension_se radius_worst texture_worst perimeter_worst area_worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   smoothness_worst compactness_worst concavity_worst concave.points_worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   symmetry_worst fractal_dimension_worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

3.Split the combined data set 75/25% so you retain 25% for validation using random sampling without replacement. Use a fixed seed so you produce the same results each time you run the code. Going forward you will use the 75% data set for training and the 25% data set for validation and determine accuracy.

```
# Setting seed for reproductibility
set.seed(111)
# Splitting the data frame in 75% and 25% ratios without replacement
index <- createDataPartition(bc_data$diagnosis, p=0.75, list=FALSE)

df_train <- bc_data[index,]
df_valid <- bc_data[-index,]
```

```
# Checking value counts Dimensions of the dataset
table(df_train$diagnosis)
```

```
##
##   B   M
## 268 159
```

```
# Checking value counts of label in validation data
table(df_valid$diagnosis)
```

```
##
##  B  M
## 89 53
```

```r
# Checking dimensions
cat("Dimensions of the training set:", dim(df_train), "\n")
```

```
## Dimensions of the training set: 427 31
```

```r
cat("Dimensions of the testing set:", dim(df_valid), "\n")
```

```
## Dimensions of the testing set: 142 31
```

4.Create a full logistic regression model of the same features as in the original data (i.e., do not eliminate any features regardless of p-value) to predict the Diagnosis variable. Be sure to either use some encoding for categorical features or convert them to factor variables and ensure that the glm function does the dummy coding (ignore this if there is no categorical variable in the data).

```r
# Building and fitting the logistic regression model on training data
logit_model <- glm(diagnosis ~ ., data = df_train, family = binomial)
# Validating it using validation data
logit_predictions <- predict(logit_model, newdata = df_valid, type = "response")
```

5.Build a confusion matrix for the classifier from (4) and comment on it, e.g., explain what it means.

```r
# Classifying the probabilities into classes using 0.5 as threshold.
predicted_classes <- ifelse(as.numeric(logit_predictions) > 0.5, "M", "B")

# Computing the confusion matrix and printing the metrics
logit_confusion_matrix <- table(predicted_classes, df_valid$diagnosis)
print("LOGISTIC REGREESION MODEL")
```

```
## [1] "LOGISTIC REGREESION MODEL"
```

```r
compute_metrics(logit_confusion_matrix)
```

```
## Accuracy: 0.9507042
## Precision: 0.9622642
## Recall: 0.9107143
## F1 Score: 0.9357798
```

```r
print(logit_confusion_matrix)
```

```
##
## predicted_classes  B  M
##                 B 84  2
##                 M  5 51
```

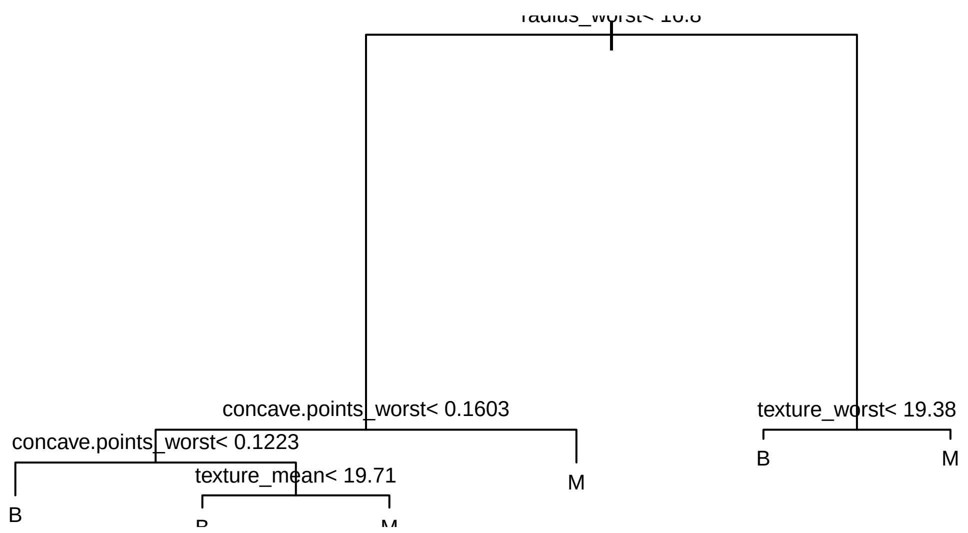
- The logistic regression model achieves high accuracy, precision, recall, and F1 score, indicating its effectiveness in classifying breast cancer cases as benign (B) or malignant (M).
- With a low number of false positives (2) and false negatives (5), the model demonstrates strong performance in correctly identifying both benign and malignant cases.

- The high precision indicates that when the model predicts a case as malignant, it is correct approximately 96.23% of the time, reducing the chances of unnecessary treatment for benign cases.
- The recall score of 91.07% indicates that the model effectively identifies the majority of actual malignant cases.

6.Create a Decision Tree model from rpart package, build a classifier that predicts the Diagnosis variable.

```r
# Importing the required library for decision tree
library(rpart)

# Building and fitting tree model on training data
tree_model <- rpart(diagnosis ~ ., data = df_train)
# Plotting the tree
plot(tree_model)
text(tree_model, cex = 0.7)
```



7.Build a confusion matrix for the classifier from (6) and comment on it, e.g., explain what it means.

```r
# Validating tree model on validation data
tree_predictions <- predict(tree_model, newdata = df_valid, type = "class")

# Computing the confusion matrix and printing the metrics
tree_confusion_matrix <- table(tree_predictions, df_valid$diagnosis)
print("DECISION TREE MODEL")
```

```
## [1] "DECISION TREE MODEL"
```

```
compute_metrics(tree_confusion_matrix)
```

```
## Accuracy: 0.9225352
## Precision: 0.9056604
## Recall: 0.8888889
## F1 Score: 0.8971963
```

```
print(tree_confusion_matrix)
```

```
##
## tree_predictions  B  M
##                B 83  5
##                M  6 48
```

- The decision tree model achieves good accuracy, precision, recall, and F1 score, indicating its effectiveness in classifying breast cancer cases as benign (B) or malignant (M).
- The model shows a slightly higher number of false positives (5) compared to the logistic regression model, suggesting that it may be less conservative in classifying cases as malignant.
- However, the decision tree model also has a slightly higher number of false negatives (6) compared to the logistic regression model, indicating that it may miss a few malignant cases.

8.Use a boosting ensemble (C5.0 decision tree with 10 boosting iterations) to predict the Diagnosis variable.

```
# Importing the required library for boosting model
library(C50)

set.seed(111)
# Train the boosting ensemble model with 10 iterations
boost_model <- C5.0(diagnosis ~ ., data = df_train, trials = 10)
```

9.Build a confusion matrix for the classifier from (8) and comment on it, e.g., explain what it means.

```
# Make predictions on the validation set
boost_predictions <- predict(boost_model, newdata = df_valid)

# Computing the confusion matrix and printing the metrics
boost_confusion_matrix <- table(boost_predictions, df_valid$diagnosis)
print("BOOSTING MODEL")
```

```
## [1] "BOOSTING MODEL"
```

```
compute_metrics(boost_confusion_matrix)
```

```
## Accuracy: 0.9788732
## Precision: 0.9811321
## Recall: 0.962963
## F1 Score: 0.9719626
```

```
print(boost_confusion_matrix)
```

```
##
## boost_predictions  B  M
##                 B 87  1
##                 M  2 52
```

- The model shows very few false positives (1) and false negatives (2), suggesting that it is excellent at correctly classifying both benign and malignant cases.
- The high precision indicates that when the model predicts a case as malignant, it is correct approximately 98.11% of the time, reducing the chances of unnecessary treatment for benign cases.
- The recall score of 96.30% indicates that the model effectively identifies the majority of actual malignant cases.

10. Build a function called predictOutcomeClass() that predicts the same Outcome variable and that combines the three predictive models from (4), (6), and (8) into a simple ensemble and uses majority vote to determine the final prediction using the individual predictions.

```
set.seed(111)
# A function to predict outcome class based on all three models.
predictOutcomeClass <- function(logistic_model, tree_model, boost_model, data) {
  # Predict using logistic regression model
  logistic_pred <- predict(logistic_model, newdata = data, type = "response")
  logistic_pred <- ifelse(logistic_pred > 0.5, "M", "B")

  # Predict using decision tree model
  tree_pred <- predict(tree_model, newdata = data, type = "class")

  # Predict using boosting ensemble model
  boost_pred <- predict(boost_model, newdata = data)

  # Combine predictions into a data frame
  predictions_df <- data.frame(Logistic = logistic_pred, Tree = tree_pred, Boost = boost_pred)

  # Use row-wise majority vote to determine final prediction
  final_prediction <- apply(predictions_df, 1, function(x) {
    tab <- table(x)
    as.character(names(tab)[which.max(tab)])
  })

  return(final_prediction)
}
```

11.Using the ensemble model from (10), predict the Diagnosis of the following individual (you can impute the missing values/columns using median): Radius_mean: 14.5 | Texture_mean: 17.0 | Perimeter_mean: 87.5 | Area_mean: 561.3 | Smoothness_mean: 0.098 | Compactness_mean: 0.105 | Concavity_mean: 0.085 | Concave_points_mean: 0.050 | Symmetry_mean: 0.180 | Fractal_dimension_mean: 0.065 | Radius_se: 0.351 | Texture_se: 1.015 | Perimeter_se: 2.457 | Area_se: 26.15 | Smoothness_se: 0.005 | Compactness_se: 0.022 | Concavity_se: 0.036 | Concave_points_se: 0.013 | Symmetry_se: 0.030 | Fractal_dimension_se: 0.005 | Radius_worst: 16.5 | Texture_worst: 25.3 | Perimeter_worst: 114.8 | Area_worst: 733.5 | Smoothness_worst: 0.155 | Compactness_worst: 0.220 | Concavity_worst: missing | Concave_points_worst: missing | Symmetry_worst: 0.360 | Fractal_dimension_worst: 0.110

```r
individual <- data.frame(
  radius_mean = 14.5,
  texture_mean = 17.0,
  perimeter_mean = 87.5,
  area_mean = 561.3,
  smoothness_mean = 0.098,
  compactness_mean = 0.105,
  concavity_mean = 0.085,
  concave.points_mean = 0.050,
  symmetry_mean = 0.180,
  fractal_dimension_mean = 0.065,
  radius_se = 0.351,
  texture_se = 1.015,
  perimeter_se = 2.457,
  area_se = 26.15,
  smoothness_se = 0.005,
  compactness_se = 0.022,
  concavity_se = 0.036,
  concave.points_se = 0.013,
  symmetry_se = 0.030,
  fractal_dimension_se = 0.005,
  radius_worst = 16.5,
  texture_worst = 25.3,
  perimeter_worst = 114.8,
  area_worst = 733.5,
  smoothness_worst = 0.155,
  compactness_worst = 0.220,
  concavity_worst = NA,  # missing value
  concave.points_worst = NA,  # missing value
  symmetry_worst = 0.360,
  fractal_dimension_worst = 0.110
)

# Impute missing values with median
individual$concavity_worst <- median(df_train$concavity_worst, na.rm = TRUE)
individual$concave.points_worst <- median(df_train$concave.points_worst, na.rm = TRUE)

# Predict the diagnosis using the ensemble model
prediction <- predictOutcomeClass(logit_model, tree_model, boost_model, individual)

# Print the prediction
cat("Prediction:", prediction, "\n")
```

```
## Prediction: B
```

The outcome is Benign class.

12. (3 bonus points) build a Random Forest ensemble with 100 trees and compare it's results over the test split with the boosting ensemble from (8).

```r
# Import required library for random forest model
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Train the Random Forest model on training data
rf_model <- randomForest(diagnosis ~ ., data = df_train, ntree = 100)

# Make predictions on the validation set
rf_predictions <- predict(rf_model, newdata = df_valid)

# Calculate accuracy for Random Forest
rf_accuracy <- mean(rf_predictions == df_valid$diagnosis)
cat("Random Forest Accuracy:", rf_accuracy, "\n")
```

```
## Random Forest Accuracy: 0.971831
```

```r
# Make predictions using the boosting ensemble model
boost_predictions <- predict(boost_model, newdata = df_valid)

# Calculate accuracy for Boosting Ensemble
boost_accuracy <- mean(boost_predictions == df_valid$diagnosis)
cat("Boosting Ensemble Accuracy:", boost_accuracy, "\n")
```

```
## Boosting Ensemble Accuracy: 0.9788732
```

The Random Forest model achieves an accuracy of 97.18%, while the Boosting Ensemble model achieves a slightly higher accuracy of 97.89%. This indicates that the Boosting Ensemble model performs slightly better in classifying breast cancer cases compared to the Random Forest model.

Problem 3

```r
# Importing the data from downloaded data, where NAs are represented by "?" character.
cars.df <- read.csv("/Users/nithyasarabudla/DA5030/automobile/imports-85.data", na.strings = "?")

# As the columns names are not provided in the data, those are manually set.
colnames(cars.df) <- c(
  "symboling", "normalized_losses", "make", "fuel_type", "aspiration",
  "num_of_doors", "body_style", "drive_wheels", "engine_location",
  "wheel_base", "length", "width", "height", "curb_weight",
  "engine_type", "num_of_cylinders", "engine_size", "fuel_system",
  "bore", "stroke", "compression_ratio", "horsepower", "peak_rpm",
  "city_mpg", "highway_mpg", "price"
)
# Checking the basic structure of data
str(cars.df)
```

```
## 'data.frame':    204 obs. of  26 variables:
##  $ symboling        : int  3 1 2 2 2 1 1 1 0 2 ...
##  $ normalized_losses: int  NA NA 164 164 NA 158 NA 158 NA 192 ...
##  $ make             : chr  "alfa-romero" "alfa-romero" "audi" "audi" ...
##  $ fuel_type        : chr  "gas" "gas" "gas" "gas" ...
##  $ aspiration       : chr  "std" "std" "std" "std" ...
##  $ num_of_doors     : chr  "two" "two" "four" "four" ...
##  $ body_style       : chr  "convertible" "hatchback" "sedan" "sedan" ...
##  $ drive_wheels     : chr  "rwd" "rwd" "fwd" "4wd" ...
##  $ engine_location  : chr  "front" "front" "front" "front" ...
##  $ wheel_base       : num  88.6 94.5 99.8 99.4 99.8 ...
##  $ length           : num  169 171 177 177 177 ...
##  $ width            : num  64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4 67.9 64.8 ...
##  $ height           : num  48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 52 54.3 ...
##  $ curb_weight      : int  2548 2823 2337 2824 2507 2844 2954 3086 3053 2395 ...
##  $ engine_type      : chr  "dohc" "ohcv" "ohc" "ohc" ...
##  $ num_of_cylinders : chr  "four" "six" "four" "five" ...
##  $ engine_size      : int  130 152 109 136 136 136 136 131 131 108 ...
##  $ fuel_system      : chr  "mpfi" "mpfi" "mpfi" "mpfi" ...
##  $ bore             : num  3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13 3.13 3.5 ...
##  $ stroke           : num  2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 3.4 2.8 ...
##  $ compression_ratio: num  9 9 10 8 8.5 8.5 8.5 8.3 7 8.8 ...
##  $ horsepower       : int  111 154 102 115 110 110 110 140 160 101 ...
##  $ peak_rpm         : int  5000 5000 5500 5500 5500 5500 5500 5500 5500 5800 ...
##  $ city_mpg         : int  21 19 24 18 19 19 19 17 16 23 ...
##  $ highway_mpg      : int  27 26 30 22 25 25 25 20 22 29 ...
##  $ price            : int  16500 16500 13950 17450 15250 17710 18920 23875 NA 16430 ...
```

```r
# Required categorical columns
categorical_columns <- c("num_of_doors", "num_of_cylinders", "engine_location")
# All numeric type columns
numeric_columns <- sapply(cars.df, is.numeric)
# All integer type columns
integer_columns <- sapply(cars.df, is.integer)
# merging integer type and numeric type into one.
num_int_columns <- names(cars.df)[numeric_columns | integer_columns]
# Finally, merging all the required columns
selected_columns <- c(categorical_columns, num_int_columns)

# Filtering the required columns.
cars.df <- cars.df[, selected_columns]
```

```r
as.matrix(sapply(cars.df, FUN = function (x) sum(is.na(x))))
```

```
##                   [,1]
## num_of_doors         2
## num_of_cylinders     0
## engine_location      0
## symboling            0
## normalized_losses   40
## wheel_base           0
## length               0
## width                0
## height                0
## curb_weight           0
## engine_size           0
## bore                  4
## stroke                4
## compression_ratio     0
## horsepower            2
## peak_rpm              2
## city_mpg              0
## highway_mpg           0
## price                 4
```

```
# For numeric/integer columns, impute the missing values with the column's median value.
for (col in num_int_columns) {
  cars.df[[col]] <- ifelse(is.na(cars.df[[col]]), median(cars.df[[col]], na.rm = TRUE), cars.df[[col]])
}
```

```
# For the categorical data, the missing value is present in "num_of_doors" column, it is better to impu
mode_num_of_doors <- names(which.max(table(cars.df$num_of_doors)))
cars.df$num_of_doors[is.na(cars.df$num_of_doors)] <- mode_num_of_doors
```

```
cat("Total missing values after imputation:", sum(is.na(cars.df)), "\n")
```

```
## Total missing values after imputation: 0
```

```
# Encode the written numbers with actual numeric numbers.
door_mapping <- c("two" = 2, "four" = 4)
cylinder_mapping <- c("two" = 2, "three" = 3, "four" = 4, "five" = 5, "six" = 6, "eight" = 8, "twelve" =

cars.df$num_of_doors <- as.integer(door_mapping[cars.df$num_of_doors])
cars.df$num_of_cylinders <- as.integer(cylinder_mapping[cars.df$num_of_cylinders])
# Factorize the engine location column.
cars.df$engine_location <- as.integer(factor(cars.df$engine_location, levels = c("front", "rear")))
```

```
# Checking the structure and types of data
str(cars.df)
```

```
## 'data.frame':    204 obs. of  19 variables:
##  $ num_of_doors     : int  2 2 4 4 2 4 4 4 2 2 ...
##  $ num_of_cylinders : int  4 6 4 5 5 5 5 5 5 4 ...
##  $ engine_location  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ symboling        : int  3 1 2 2 2 1 1 1 0 2 ...
##  $ normalized_losses: num  115 115 164 164 115 158 115 158 115 192 ...
##  $ wheel_base       : num  88.6 94.5 99.8 99.4 99.8 ...
##  $ length           : num  169 171 177 177 177 ...
##  $ width            : num  64.1 65.5 66.2 66.4 66.3 71.4 71.4 71.4 67.9 64.8 ...
##  $ height           : num  48.8 52.4 54.3 54.3 53.1 55.7 55.7 55.9 52 54.3 ...
##  $ curb_weight      : int  2548 2823 2337 2824 2507 2844 2954 3086 3053 2395 ...
##  $ engine_size      : int  130 152 109 136 136 136 136 131 131 108 ...
##  $ bore             : num  3.47 2.68 3.19 3.19 3.19 3.19 3.19 3.13 3.13 3.5 ...
##  $ stroke           : num  2.68 3.47 3.4 3.4 3.4 3.4 3.4 3.4 3.4 2.8 ...
##  $ compression_ratio: num  9 9 10 8 8.5 8.5 8.5 8.3 7 8.8 ...
##  $ horsepower       : num  111 154 102 115 110 110 110 140 160 101 ...
##  $ peak_rpm         : num  5000 5000 5500 5500 5500 5500 5500 5500 5500 5800 ...
##  $ city_mpg         : int  21 19 24 18 19 19 19 17 16 23 ...
##  $ highway_mpg      : int  27 26 30 22 25 25 25 20 22 29 ...
##  $ price            : num  16500 16500 13950 17450 15250 ...
```

2. Are there outliers in any one of the features in the data set? How do you identify outliers? Remove them but create a second data set with outliers removed called cars.no.df. Keep the original data set cars.df.

```
# Duplicate cars.df into cars.no.df
cars.no.df <- cars.df

# Function to detect outliers using IQR method, ignoring specified columns
detect_outliers_ignore <- function(x) {
  q1 <- quantile(x, 0.25)
  q3 <- quantile(x, 0.75)
  # Compute the inter quartile range
  iqr <- q3 - q1
  lower_bound <- q1 - 1.5 * iqr
  upper_bound <- q3 + 1.5 * iqr
  outliers <- x[x < lower_bound | x > upper_bound]
  return(list(count = length(outliers), outliers = outliers))
}

# Iterate over each column and remove outliers from cars.no.df
for (col in names(cars.no.df)) {
  # If the column is categorical, ignore it.
  if (!(col %in% categorical_columns)) {
    outliers_info <- detect_outliers_ignore(cars.no.df[[col]])
    cat("Outliers in", col, ":", outliers_info$count, "\n")
    # Remove the outlier rows from cars.no.df
    cars.no.df <- cars.no.df[!cars.no.df[[col]] %in% outliers_info$outliers, ]
  }
}
```

```
## Outliers in symboling : 0
## Outliers in normalized_losses : 8
## Outliers in wheel_base : 3
## Outliers in length : 1
## Outliers in width : 11
## Outliers in height : 0
## Outliers in curb_weight : 2
## Outliers in engine_size : 5
## Outliers in bore : 0
## Outliers in stroke : 23
## Outliers in compression_ratio : 27
## Outliers in horsepower : 0
## Outliers in peak_rpm : 2
## Outliers in city_mpg : 1
## Outliers in highway_mpg : 0
## Outliers in price : 3
```

The Interquartile Range (IQR) is a robust measure of statistical dispersion that is often used to identify and remove outliers from a dataset. To calculate the IQR, the dataset is first divided into quartiles, with the first quartile (Q1) representing the value below which 25% of the data falls, and the third quartile (Q3) representing the value below which 75% of the data falls. The IQR is then calculated as the difference between Q3 and Q1. Outliers are typically defined as values that fall below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR, indicating that they are unusually high or low compared to the rest of the data. By identifying and removing outliers using the IQR, the dataset can be cleaned to improve the accuracy and reliability of statistical analyses and machine learning models.

```r
cat("Number of Unique values in each column: \n")
```

```
## Number of Unique values in each column:
```

```r
for (col in names(cars.no.df)) {
  cat(col, ":", length(unique(cars.no.df[, col])), "\n")
}
```

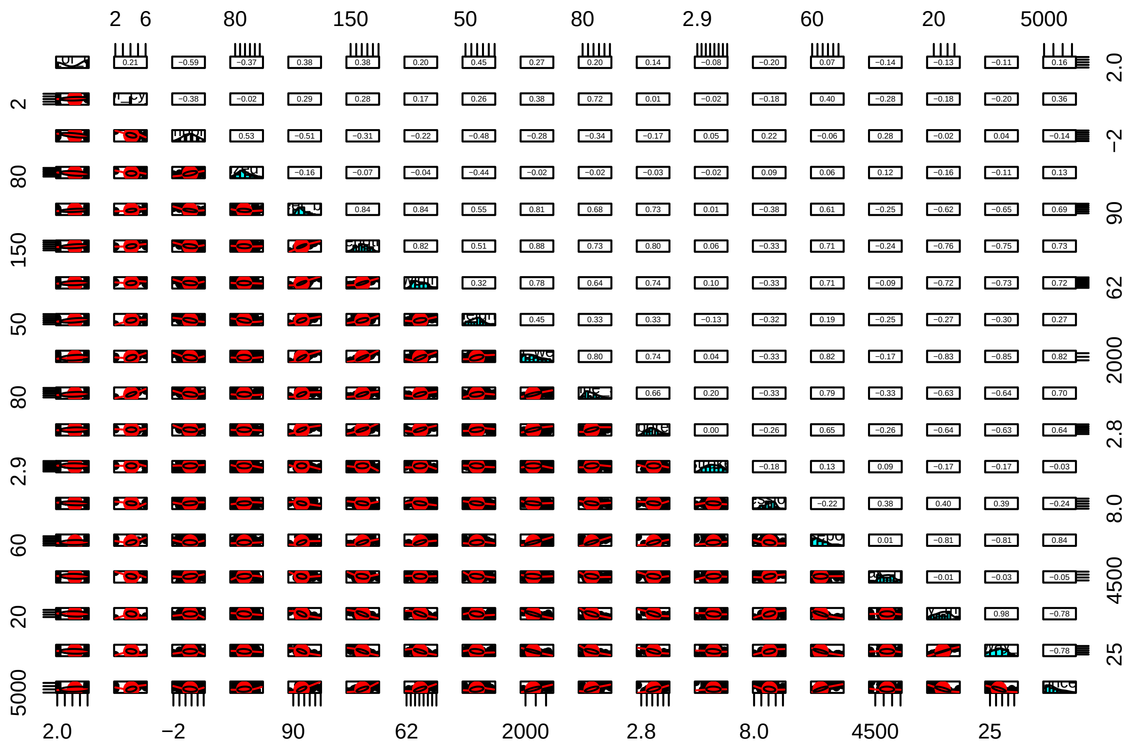
```
## num_of_doors : 2
## num_of_cylinders : 4
## engine_location : 1
## symboling : 6
## normalized_losses : 37
## wheel_base : 29
## length : 48
## width : 27
## height : 36
## curb_weight : 102
## engine_size : 25
## bore : 22
## stroke : 20
## compression_ratio : 17
## horsepower : 30
## peak_rpm : 11
## city_mpg : 18
## highway_mpg : 20
## price : 108
```

There is only 1 unique value in `engine_location` after removing the outliers. We can drop this column.

```r
# Dropping the engine location from cars.no.df
cars.no.df$engine_location <- NULL
```

Using pairs.panel , what are the distributions of each of the features in the data set with outliers removed (cars.no.df)? Are they reasonably normal so you can apply a statistical learner such as regression? Can you normalize features through a log, inverse, or square-root transform? State which features should be transformed and then transform as needed and build a new data set, cars.tx.

```r
# Importing the required packages for pairs.panels
# install.packages("psych")
library(psych)
# Plot the cars.no.df using pairs.panels
pairs.panels(cars.no.df)
```



```r
# Build a linear regression model
model <- lm(price ~ ., data = cars.no.df)
summary(model)
```

```
##
## Call:
## lm(formula = price ~ ., data = cars.no.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3924.7  -869.8  -165.3   818.9  5208.6
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -3.791e+04  2.309e+04  -1.642   0.1038
## num_of_doors       8.104e+01  2.572e+02   0.315   0.7533
## num_of_cylinders   2.715e+03  1.608e+03   1.689   0.0944 .
## symboling         -1.744e+02  2.846e+02  -0.613   0.5412
## normalized_losses  1.471e+01  8.997e+00   1.635   0.1051
## wheel_base         2.126e+02  1.149e+02   1.851   0.0671 .
## length            -6.666e+01  5.312e+01  -1.255   0.2125
## width              1.162e+02  2.917e+02   0.398   0.6913
## height            -4.442e+01  1.228e+02  -0.362   0.7182
## curb_weight        2.341e+00  1.580e+00   1.481   0.1417
## engine_size       -1.062e+02  6.553e+01  -1.621   0.1083
## bore               5.724e+03  3.946e+03   1.450   0.1501
## stroke             2.269e+02  2.218e+03   0.102   0.9187
## compression_ratio  6.310e+02  6.035e+02   1.046   0.2983
## horsepower         7.626e+01  1.704e+01   4.474 2.04e-05 ***
## peak_rpm          -1.908e-01  6.060e-01  -0.315   0.7536
## city_mpg          -3.412e+02  1.988e+02  -1.716   0.0893 .
## highway_mpg        1.564e+02  1.837e+02   0.852   0.3964
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1879 on 100 degrees of freedom
## Multiple R-squared:  0.8115, Adjusted R-squared:  0.7794
## F-statistic: 25.32 on 17 and 100 DF,  p-value: < 2.2e-16
```

- The overall model seems to be significant with a p-value less than 0.05, indicating that at least one of the predictors has a significant effect on the price.
- The Adjusted R-squared value of 0.7794 suggests that the model explains about 77.94% of the variance in the response variable, which is quite good.
- Looking at the coefficients, we see that 'horsepower' has a strong positive effect on price, as indicated by its high coefficient and low p-value. This suggests that as 'horsepower' increases, the price of the car tends to increase.
- 'num_of_doors', 'symboling', 'length', 'width', 'height', 'curb_weight', 'engine_size', 'bore', 'stroke', 'compression_ratio', 'peak_rpm', 'city_mpg', and 'highway_mpg' do not seem to have a significant effect on price, as their p-values are higher than 0.05.
- It's also worth noting that 'num_of_cylinders' has a marginally significant effect on price, with a p-value of 0.0944.

```r
# Importing required library for computing skewness
library(e1071)

# Identify skewed columns
skewed_columns <- names(Filter(function(x) abs(skewness(x, na.rm = TRUE)) > 0.5, cars.no.df))

# Apply log transformation to skewed columns
cars.tx <- cars.no.df
cars.tx[skewed_columns] <- lapply(cars.tx[skewed_columns], function(x) log(x + 1))
```

4.What are the correlations to the response variable (price) for cars.no.df? Are there collinearities? Build a full correlation matrix.

```r
# Calculate correlations
correlations <- cor(cars.no.df)

# Correlation with the response variable (price)
price_correlations <- correlations["price", ]
print(price_correlations)
```

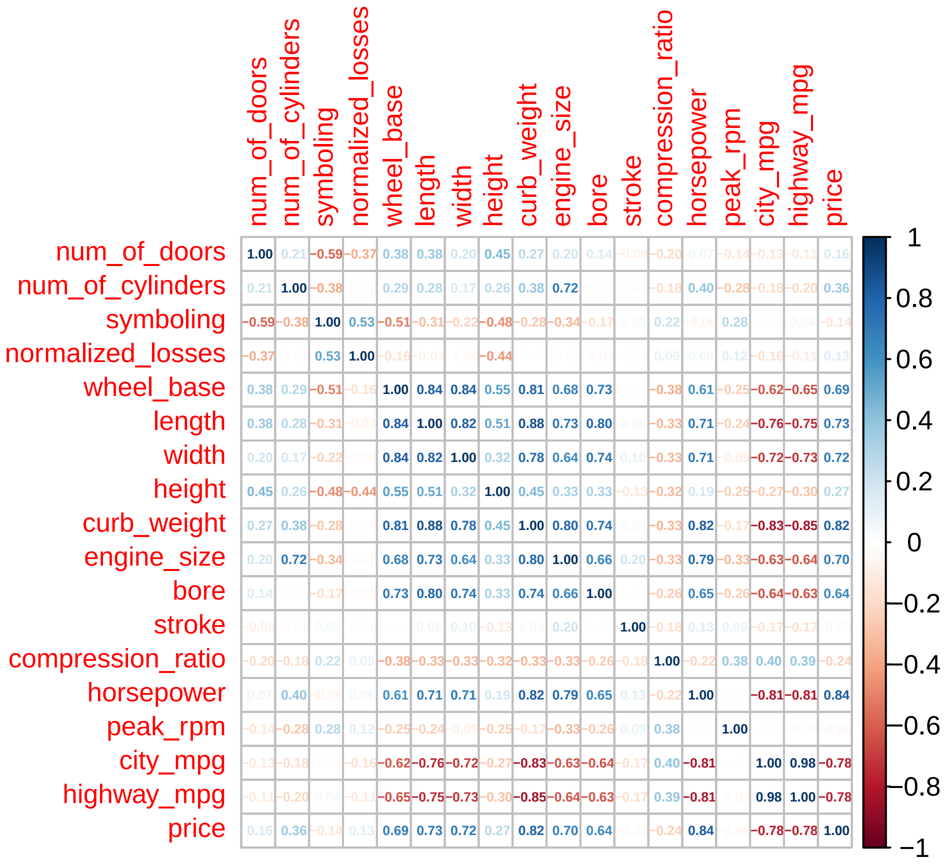
```
##      num_of_doors  num_of_cylinders         symboling normalized_losses
##        0.16349079        0.36016733       -0.14422856        0.12700891
##        wheel_base            length             width            height
##        0.69060654        0.72699798        0.71546459        0.27154459
##       curb_weight       engine_size              bore            stroke
##        0.81717998        0.69901983        0.63927008       -0.03474395
## compression_ratio        horsepower          peak_rpm          city_mpg
##       -0.23770534       0.83917128       -0.04539356       -0.78460588
##       highway_mpg            price
##       -0.77772616       1.00000000
```

These correlations reveal the relationships between various car features and their prices in the dataset. Features like engine size, curb weight, length, width, and height show strong positive correlations with price, indicating that larger and heavier cars tend to be more expensive. Similarly, horsepower exhibits a strong positive correlation, suggesting that cars with higher horsepower command higher prices. On the other hand, city and highway miles per gallon (MPG) exhibit strong negative correlations with price, indicating that cars with better fuel efficiency are often more affordable.

```r
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
corrplot(correlations, method="number", number.cex=0.4, tl.cex=0.8)
```



5. Split each of the three data sets, cars.no.df, cars.df, and cars.tx 70%/30% so you retain 30% for testing using random sampling without replacement. Call the datasets, cars.training and cars.testing, cars.no.training and cars.no.testing, and cars.tx.training and cars.tx.testing.

```r
# Set seed for reproductibility
set.seed(111)
```

```r
# Split cars.df
index <- createDataPartition(cars.df$price, p = 0.7, list = FALSE)
cars.training <- cars.df[index, ]
cars.testing <- cars.df[-index, ]

# Split cars.no.df
index <- createDataPartition(cars.no.df$price, p = 0.7, list = FALSE)
cars.no.training <- cars.no.df[index, ]
cars.no.testing <- cars.no.df[-index, ]

# Split cars.tx
index <- createDataPartition(cars.tx$price, p = 0.7, list = FALSE)
cars.tx.training <- cars.tx[index, ]
cars.tx.testing <- cars.tx[-index, ]
```

6. Build three ideal multiple regression models for cars.training, cars.no.training, and cars.tx.training using backward elimination based on p-value for predicting price.

```r
# Building and Fitting multiple regression models on training datasets

# For cars.training
model_training <- lm(price ~ ., data = cars.training)
final_model_training <- step(model_training, direction = "backward")

# For cars.no.training
model_no_training <- lm(price ~ ., data = cars.no.training)
final_model_no_training <- step(model_no_training, direction = "backward")

# For cars.tx.training
model_tx_training <- lm(price ~ ., data = cars.tx.training)
final_model_tx_training <- step(model_tx_training, direction = "backward")
```

7. Build a Regression Tree model using rpart package for predicting price: one with cars.training, one with cars.no.training, and one with cars.tx.training.

```r
library(rpart)

# Build a regression tree model for cars.training
tree_model_training <- rpart(price ~ ., data = cars.training, method = "anova")

# Build a regression tree model for cars.no.training
tree_model_no_training <- rpart(price ~ ., data = cars.no.training, method = "anova")

# Build a regression tree model for cars.tx.training
tree_model_tx_training <- rpart(price ~ ., data = cars.tx.training, method = "anova")
```

8. Provide an analysis of all the 6 models (using their respective testing data sets), including Adjusted R-Squared and RMSE. Which of these models is the best? Why?

```r
# Define a function to calculate RMSE
rmse <- function(predicted, actual) {
  sqrt(mean((predicted - actual)^2))
}

# Define a function to extract adjusted R2 score
adj_r_squared <- function(model, data) {
  1 - (1 - summary(model)$adj.r.squared) * ((nrow(data) - 1) / (nrow(data) - length(model$coefficients)
}

# Calculate RMSE for each model using their testing datasets
results <- data.frame(
  Model = c("Multiple Regression (cars.training)", "Multiple Regression (cars.no.training)", "Multiple 
          "Regression Tree (cars.training)", "Regression Tree (cars.no.training)", "Regression Tree (
  Adjusted_R_Squared = c(
    adj_r_squared(final_model_training, cars.testing),
    adj_r_squared(final_model_no_training, cars.no.testing),
    adj_r_squared(final_model_tx_training, cars.tx.testing),
    adj_r_squared(tree_model_training, cars.testing),
    adj_r_squared(tree_model_no_training, cars.no.testing),
    adj_r_squared(tree_model_tx_training, cars.tx.testing)
  ),
  RMSE = c(
    rmse(predict(final_model_training, newdata = cars.testing), cars.testing$price),
    rmse(predict(final_model_no_training, newdata = cars.no.testing), cars.no.testing$price),
    rmse(predict(final_model_tx_training, newdata = cars.tx.testing), cars.tx.testing$price),
    rmse(predict(tree_model_training, newdata = cars.testing), cars.testing$price),
    rmse(predict(tree_model_no_training, newdata = cars.no.testing), cars.no.testing$price),
    rmse(predict(tree_model_tx_training, newdata = cars.tx.testing), cars.tx.testing$price)
  )
)

# Print the results
print(results)
```

Based on the obtained results, the best model for predicting price appears to be the Multiple Regression model trained on the cars.tx.training dataset. This model has the highest Adjusted R-Squared value of 0.7902639, indicating a good fit to the data. Additionally, it has a very low RMSE (Root Mean Squared Error) of 0.1690655, suggesting that it has accurate predictions compared to the actual prices in the testing dataset.

9. Using each of the regression models, how one unit change in highway-mpg translates into the price prediction? how about city-mpg? (do not apply backward feature elimination for only for this part).

```
# For Multiple Regression models
coef_training <- coef(model_training)
coef_no_training <- coef(model_no_training)
coef_tx_training <- coef(model_tx_training)

# Extract coefficients for highway-mpg and city-mpg
coef_highway_mpg_training <- coef_training["highway_mpg"]
coef_highway_mpg_no_training <- coef_no_training["highway_mpg"]
coef_highway_mpg_tx_training <- coef_tx_training["highway_mpg"]

coef_city_mpg_training <- coef_training["city_mpg"]
```

```r
coef_city_mpg_no_training <- coef_no_training["city_mpg"]
coef_city_mpg_tx_training <- coef_tx_training["city_mpg"]

# Interpretation
cat("Multiple Regression (cars.training):\n")
```

```
## Multiple Regression (cars.training):
```

```r
cat("One unit change in highway-mpg leads to a change of", coef_highway_mpg_training, "in price.\n")
```

```
## One unit change in highway-mpg leads to a change of -164.1985 in price.
```

```r
cat("One unit change in city-mpg leads to a change of", coef_city_mpg_training, "in price.\n\n")
```

```
## One unit change in city-mpg leads to a change of 38.49221 in price.
```

```r
cat("Multiple Regression (cars.no.training):\n")
```

```
## Multiple Regression (cars.no.training):
```

```r
cat("One unit change in highway-mpg leads to a change of", coef_highway_mpg_no_training, "in price.\n")
```

```
## One unit change in highway-mpg leads to a change of 125.4263 in price.
```

```r
cat("One unit change in city-mpg leads to a change of", coef_city_mpg_no_training, "in price.\n\n")
```

```
## One unit change in city-mpg leads to a change of -374.1585 in price.
```

```r
cat("Multiple Regression (cars.tx.training):\n")
```

```
## Multiple Regression (cars.tx.training):
```

```r
cat("One unit change in highway-mpg leads to a change of", coef_highway_mpg_tx_training, "in price.\n")
```

```
## One unit change in highway-mpg leads to a change of 0.02174656 in price.
```

```r
cat("One unit change in city-mpg leads to a change of", coef_city_mpg_tx_training, "in price.\n\n")
```

```
## One unit change in city-mpg leads to a change of -0.01991628 in price.
```

10. For each of the predictions, calculate the 95% prediction interval for the price. (Exclude Regression Trees)

```r
calculate_prediction_interval <- function(model, data) {
  # Predicted values
  predictions <- predict(model, newdata = data, interval = "prediction" ,level = 0.95)
  # Prediction interval
  prediction_interval <- data.frame(
    Lower = predictions[, "lwr"],
    Upper = predictions[, "upr"]
  )
  return(prediction_interval)
}


prediction_interval_training <- calculate_prediction_interval(final_model_training, cars.training)
prediction_interval_no_training <- calculate_prediction_interval(final_model_no_training, cars.no.traini
prediction_interval_tx_training <- calculate_prediction_interval(final_model_tx_training, cars.tx.traini

# Print the prediction intervals
cat("95% Prediction Interval for Multiple Regression (cars.training):\n")
```

```
## 95% Prediction Interval for Multiple Regression (cars.training):
```

```r
print(head(prediction_interval_training))
```

```
##       Lower    Upper
## 2  9705.236 24205.24
## 3  4774.575 18622.51
## 4  8403.365 22398.42
## 5  8172.938 22071.56
## 7 11458.830 26141.91
## 8 11387.733 26195.26
```

```r
cat("\n95% Prediction Interval for Multiple Regression (cars.no.training):\n")
```

```
##
## 95% Prediction Interval for Multiple Regression (cars.no.training):
```

```r
print(head(prediction_interval_no_training))
```

```
##        Lower     Upper
## 2  10009.617 18052.258
## 3   8920.104 16253.701
## 5   9663.749 16767.842
## 13 11280.957 18567.434
## 19  2529.963  9754.776
## 21  2005.033  9130.898
```

```r
cat("\n95% Prediction Interval for Multiple Regression (cars.tx.training):\n")
```

```
##
## 95% Prediction Interval for Multiple Regression (cars.tx.training):
```

```r
print(head(prediction_interval_tx_training))
```

```
##       Lower    Upper
## 2  9.301859 9.960951
## 4  9.166557 9.794101
## 12 9.121817 9.756383
## 20 8.508623 9.136578
## 21 8.460953 9.089284
## 22 8.460953 9.089284
```

Each data frame explains the 95% prediction intervals for price for each of the three multiple regression models.