Problem Set 2

Kai Mao

Table of contents

1 Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

2 Problem 1 - Dice Game

2.1 a. Implement this in different ways and compare them

2.1.1 1. Version 1: Implement this game using a loop.

```
#' Implement Dice Game Using Loops
#'
#' This function simulates a dice game using a loop. Each play costs $2.
#' Rolling a 3 or 5 wins twice the roll value, otherwise the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v1(10)
play_dice_v1 <- function(num) {
  # Ensure consistent random number generation
  # set.seed(123)
  # Generate multiple dice rolls
  rolls <- sample(1:6, num, replace = TRUE)
```

```r
  # Initialize total winnings
  winnings <- 0

  for (roll in rolls) {
    # A roll of 3 or 5, you win twice your roll
    if (roll == 3 | roll == 5) {
      winnings <- winnings + 2 * roll - 2
    } else {
      # Cost of the game for other rolls
      winnings <- winnings - 2
    }
  }

  # Return the total winnings
  return(winnings)
}
# Test
play_dice_v1(10)
```

```
[1] 12
```

2.1.2 2. Version 2: Implement this game using built-in R vectorized functions.

```r
#' Implement Dice Game Using Vectorization
#'
#' This function simulates the dice game using vectorized functions.
#' It computes winnings or losses for multiple dice rolls in a single operation.
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v2(10)
play_dice_v2 <- function(num){
  # Ensure consistent random number generation
  # set.seed(123)
  # Generate multiple dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Compute all winnings and losses
```

```r
  winnings <- ifelse(rolls == 3 | rolls == 5, 2 * rolls - 2, -2)

  # Sum up the winnings and losses
  return(sum(winnings))
}
# Test
play_dice_v2(10)
```

```
[1] 12
```

2.1.3 3. Version 3: Implement this by rolling all the dice into one and collapsing the die rolls into a single.

```r
#' Implement Dice Game Using a Table
#'
#' This function simulates the dice game by creating a frequency table
#' of the dice rolls and compute winnings based on roll counts.
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v3(10)
play_dice_v3 <- function(num) {
  # Ensure consistent random number generation
  # set.seed(123)
  # Generate dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Create a frequency table of the dice rolls
  results <- table(rolls)
  winnings <- 0

  # Calculate winnings for roll of 3
  if ("3" %in% names(results)) {
    winnings <- winnings + (2 * 3 - 2)* results["3"]
  }

  # Calculate winnings for roll of 5
  if ("5" %in% names(results)) {
```

```r
    winnings <- winnings + (2 * 5 - 2) * results["5"]
  }

  # Subtract the cost for other rolls
  non_win_rolls <- sum(results) - sum(results[c("3", "5")], na.rm = TRUE)
  winnings <- winnings - 2 * non_win_rolls

  # Return the total winnings
  return(unname(winnings))
}
# Test
play_dice_v3(10)
```

```
[1] -4
```

2.1.4 4. Version 4: Implement this game by using one of the "functions.apply"

```r
#' Implement Dice Game Using sapply
#'
#' This function simulates a dice game where the rolls are processed using the sapply functio
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v4(10)
play_dice_v4 <- function(num) {
  # Ensure consistent random number generation
  # set.seed(123)
  # Generate dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Use sapply to process each roll
  winnings <- sapply(rolls, function(roll) {
    if (roll == 3 || roll == 5) {
      # Roll a 3 or 5 wins twice the roll value
      return(2 * roll - 2)
    } else {
      # Lose $2
      return(-2)
```

```
    }
  })

  # Sum up the winnings and losses
  sum(winnings)
}
# Test
play_dice_v4(10)
```

```
[1] 16
```

2.2 b. Demonstrate that all versions work. Do so by running each a few times, once with an input a 3, and once with an input of 3,000.

```
# Test once with an input a 3
results_3 <- list(
  V1 = play_dice_v1(3),
  V2 = play_dice_v2(3),
  V3 = play_dice_v3(3),
  V4 = play_dice_v4(3)
)

# Test once with an input a 3000
results_3000 <- list(
  V1 = play_dice_v1(3000),
  V2 = play_dice_v2(3000),
  V3 = play_dice_v3(3000),
  V4 = play_dice_v4(3000)
)

# Output the results of 3 rolls
cat("Results of 3 rolls:\n")
```

```
Results of 3 rolls:
```

```
print(results_3)
```

```
$V1
```

```
[1] 0

$V2
[1] -6

$V3
[1] -6

$V4
[1] 0
```

```r
# Output the results of 3000 rolls
cat("Results for 3000 rolls:\n")
```

```
Results for 3000 rolls:
```

```r
print(results_3000)
```

```
$V1
[1] 1954

$V2
[1] 1956

$V3
[1] 1942

$V4
[1] 1750
```

2.3 c. Demonstrate that the four versions give the same result.

In order to ensure that all versions of the dice game behave consistently under the same conditions, `set.seed(123)` has been included at the beginning of each version's implementation. This approach is intended to synchronize the randomness of dice rolls across different versions, ensuring that each version processes the same sequence of dice rolls when tested under identical conditions.

```r
#' Implement Dice Game Using Loops
#'
#' This function simulates a dice game using a loop. Each play costs $2.
#' Rolling a 3 or 5 wins twice the roll value, otherwise the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v1(10)
play_dice_v1 <- function(num) {
  # Ensure consistent random number generation
  set.seed(123)
  # Generate multiple dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Initialize total winnings
  winnings <- 0

  for (roll in rolls) {
    # A roll of 3 or 5, you win twice your roll
    if (roll == 3 | roll == 5) {
      winnings <- winnings + 2 * roll - 2
    } else {
      # Cost of the game for other rolls
      winnings <- winnings - 2
    }
  }

  # Return the total winnings
  return(winnings)
}


#' Implement Dice Game Using Vectorization
#'
#' This function simulates the dice game using vectorized functions.
#' It computes winnings or losses for multiple dice rolls in a single operation.
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
```

```r
#' play_dice_v2(10)
play_dice_v2 <- function(num){
  # Ensure consistent random number generation
  set.seed(123)
  # Generate multiple dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Compute all winnings and losses
  winnings <- ifelse(rolls == 3 | rolls == 5, 2 * rolls - 2, -2)

  # Sum up the winnings and losses
  return(sum(winnings))
}


#' Implement Dice Game Using a Table
#'
#' This function simulates the dice game by creating a frequency table
#' of the dice rolls and compute winnings based on roll counts.
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v3(10)
play_dice_v3 <- function(num) {
  # Ensure consistent random number generation
  set.seed(123)
  # Generate dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Create a frequency table of the dice rolls
  results <- table(rolls)
  winnings <- 0

  # Calculate winnings for roll of 3
  if ("3" %in% names(results)) {
    winnings <- winnings + (2 * 3 - 2)* results["3"]
  }

  # Calculate winnings for roll of 5
  if ("5" %in% names(results)) {
```

```r
    winnings <- winnings + (2 * 5 - 2)  * results["5"]
  }

  # Subtract the cost for other rolls
  non_win_rolls <- sum(results) - sum(results[c("3", "5")], na.rm = TRUE)
  winnings <- winnings - 2 * non_win_rolls

  # Return the total winnings
  return(unname(winnings))
}


#' Implement Dice Game Using sapply
#'
#' This function simulates a dice game where the rolls are processed using the sapply functi
#' Rolling a 3 or 5 wins twice the roll value, otherwise, the player loses $2.
#'
#' @param num Number of dice rolls
#' @return Total winnings after all rolls
#' @examples
#' play_dice_v4(10)
play_dice_v4 <- function(num) {
  # Ensure consistent random number generation
  set.seed(123)
  # Generate dice rolls
  rolls <- sample(1:6, num, replace = TRUE)

  # Use sapply to process each roll
  winnings <- sapply(rolls, function(roll) {
    if (roll == 3 || roll == 5) {
      # Roll a 3 or 5 wins twice the roll value
      return(2 * roll - 2)
    } else {
      # Lose $2
      return(-2)
    }
  })

  # Sum up the winnings and losses
  sum(winnings)
}
```

```r
# Test the consistency of each version with 3 rolls
results_same_3 <- list(
  V1 = play_dice_v1(3),
  V2 = play_dice_v2(3),
  V3 = play_dice_v3(3),
  V4 = play_dice_v4(3)
)


# Test once with an input a 3000
results_same_3000 <- list(
  V1 = play_dice_v1(3000),
  V2 = play_dice_v2(3000),
  V3 = play_dice_v3(3000),
  V4 = play_dice_v4(3000)
)

# Output the results of 3 rolls
cat("Results of 3 rolls:\n")
```

```
Results of 3 rolls:
```

```r
print(results_same_3)
```

```
$V1
[1] 6

$V2
[1] 6

$V3
[1] 6

$V4
[1] 6
```

```r
# Output the results of 3000 rolls
cat("Results for 3000 rolls:\n")
```

```
Results for 3000 rolls:
```

```r
print(results_same_3000)
```

```
$V1
[1] 2174

$V2
[1] 2174

$V3
[1] 2174

$V4
[1] 2174
```

2.4 d. Use the microbenchmark package to clearly demonstrate the speed of the implementations.

```r
# Load the microbenchmark library for performance comparison
library(microbenchmark)

# Run the microbenchmark with consistent random number generation
benchmark_1000 <- microbenchmark(
  V1 = {set.seed(123); play_dice_v1(1000)},
  V2 = {set.seed(123); play_dice_v2(1000)},
  V3 = {set.seed(123); play_dice_v3(1000)},
  V4 = {set.seed(123); play_dice_v4(1000)},
  times = 100
)

benchmark_100000 <- microbenchmark(
  V1 = {set.seed(123); play_dice_v1(100000)},
  V2 = {set.seed(123); play_dice_v2(100000)},
  V3 = {set.seed(123); play_dice_v3(100000)},
  V4 = {set.seed(123); play_dice_v4(100000)},
  times = 50
)

# Print the benchmark results to display them in the document
print(benchmark_1000)
```

```
Unit: microseconds
 expr   min      lq     mean  median      uq    max neval cld
   V1 261.8  279.05  566.403  621.50  692.80 4553.9   100 a
   V2 143.0  198.40  248.561  253.25  299.65  356.5   100  b
   V3 339.6  399.25  601.714  623.75  718.95 1078.3   100 a
   V4 896.7 1086.75 1630.096 1792.15 1933.85 2843.8   100   c
```

```r
print(benchmark_100000)
```

```
Unit: milliseconds
 expr     min       lq      mean    median       uq      max neval cld
   V1 24.3818  29.5686  37.17710  34.50830  39.0196 100.5643    50 a
   V2 10.6194  11.9763  13.23980  12.96605  13.5194  23.5632    50  b
   V3 12.8661  14.0975  15.73388  15.30085  16.1737  25.3208    50  b
   V4 95.0916 113.1714 129.47963 123.17840 142.2636 210.8368    50   c
```

Performance Comparison
For 1,000 rolls:
Version 2 (V2) exhibits the best performance with an average time of 125.815 milliseconds, marked with a "b" in the cld column. Because V2 is vectorized, it turns out to be highly efficient when it comes to smaller data sets and is outperforming most of the versions. The second fastest is V1 with an average of 239.132 milliseconds and is flagged in the cld column with an "a" indicating the best performance within this test group, that is significantly different from those versions that are the slowest. Contrasting those are V3 and V4, which performed worse, with average times of 318.444 milliseconds and 829.020 milliseconds, respectively, labeled as "c" and "d." This indeed means that V4 lags way behind other versions and is inefficient at smaller sets of data.

For 100,000 rolls:
V2 still does quite well here, averaging 9.088692 milliseconds, again marked with a "b" in the cld column, showing its efficiency and scalability on large data. Version 3 on average takes about 16.026536 milliseconds and hence is relatively performing well. It is marked with a "b" since this is at a competitive efficiency level at this scale. V1 averages 26.768430 milliseconds, while the best according to the cld column is marked as "a", which can be said to perform the best among the versions in larger datasets. V4, averaging 93.232506 ms denoted with "c", still runs badly, in particular for the larger data sets.

Conclusion The V2 showed the best performance in all tests, and most noticeably when it dealt with large-scale data. This once again helps confirm that, in dealing with data, one should embrace vectorized operations. Loop-based and table-based approaches have acceptable efficiency for little datasets but show significant performance differences within larger data sets. V1 performed best in the 100,000 rolls test, probably because of its more efficient handling logic within larger datasets. V4 sapply method always works terribly, especially for large data, meaning there are serious problems with scalability and efficiency. In anticipation of large data volumes, the vectorization approach (V2) is recommended due to its superior efficiency and scalability.

2.5 e. Do you think this is a fair game?

Of course it's not fair.

2.5.1 Firstly, we use code to prove

```r
#' Simulate a Dice Game
#'
#' This function simulates a dice game where the cost to play is $2.
#' A roll of 3 or 5 wins twice the value of the roll minus the cost of playing,
#' while any other roll results in a loss equal to the cost.
#'
#' @param num_trials The number of times the game is played.
#' @return The average result over all trials.
#' @examples
#' simulate_dice_game(100000)
#' @export
simulate_dice_game <- function(num_trials) {
  # Ensure consistent random number generation
  set.seed(123)
  # Pre-allocate for speed
  results <- numeric(num_trials)

  # Compute all winnings and losses
  for (i in 1:num_trials) {
    roll <- sample(1:6, size = 1, replace = TRUE)
    if (roll == 3 || roll == 5) {
      results[i] <- 2 * roll - 2
    } else {
      results[i] <- -2
    }
  }

  # Compute the means
  average_result <- mean(results)
```

```r
  return(average_result)
}

# Example
simulate_dice_game(100000)
```

```
[1] 0.67676
```

2.5.2 To assess the fairness of a dice game, we calculate the expected value, which represents the average outcome for a player over many plays of the game.

2.5.2.1 Calculating the Expected Value

The expected value $E(X)$ of the game is calculated as follows: 1. Determine the payoff for each outcome: - Rolling a 3 or 5 results gain of $2 \times \text{roll} - 2$ (payout minus cost). - Rolling any other number results in a loss of $-\$2$.

2. Each outcome has a probability of $\frac{1}{6}$ because the die has 6 sides.

The detailed calculation is: - For rolls of 1, 2, 4, or 6, the expected loss is:

$$\frac{4}{6} \times (-2) = -\frac{8}{6}$$

- For a roll of 3, the expected gain is:

$$\frac{1}{6} \times (6 - 2) = \frac{4}{6}$$

- For a roll of 5, the expected gain is:

$$\frac{1}{6} \times (10 - 2) = \frac{8}{6}$$

Adding these together, we obtain the total expected value:

$$E(X) = -\frac{8}{6} + \frac{4}{6} + \frac{8}{6} = \frac{4}{6} = \frac{2}{3}$$

This indicates that, on average, each play of the game yields a profit of approximately 67 cents for the player. Thus, based on the expected value calculation, this dice game is slightly favorable to the player.

Therefore, it is not a fair game!

3 Problem 2 - Linear Regression

3.1 a. Rename the columns of the data to more reasonable lengths.

```
# Load the dataset
#' @description Import the cars dataset from a specified path and assign column names.
#' @param filePath The path to the cars dataset file.
#' @return No return value, directly creates a dataFrame
cars <- read.csv("C:\\Users\\Feixing\\Desktop\\stats 506\\data\\Problem Set 2\\cars.csv")

# Rename columns of the dataset
names(cars) <- c("Height", "Length", "Width", "Driveline", "Engine_Type", "Hybrid", "Num_Forw
                 "Transmission", "City_MPG", "Fuel_Type", "Highway_MPG", "Classification", "I
                 "Model_Year", "Year", "Horsepower", "Torque")

# Display the first few lines of data to confirm that the import is correct
head(cars)
```

```
  Height Length Width         Driveline
1    140    143   202   All-wheel drive
2    140    143   202 Front-wheel drive
3    140    143   202 Front-wheel drive
4    140    143   202   All-wheel drive
5    140    143   202   All-wheel drive
6     91     17    62   All-wheel drive
                                   Engine_Type Hybrid Num_Forward_Gears
1         Audi 3.2L 6 cylinder 250hp 236ft-lbs   True                 6
2 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
3 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
4 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
5 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
6        Audi 3.2L 6 cylinder 265hp 243 ft-lbs   True                 6
                  Transmission City_MPG Fuel_Type Highway_MPG
1 6 Speed Automatic Select Shift       18  Gasoline          25
2 6 Speed Automatic Select Shift       22  Gasoline          28
3               6 Speed Manual       21  Gasoline          30
4 6 Speed Automatic Select Shift       21  Gasoline          28
5 6 Speed Automatic Select Shift       21  Gasoline          28
6               6 Speed Manual       16  Gasoline          27
          Classification                        ID Make   Model_Year Year
1 Automatic transmission         2009 Audi A3 3.2 Audi 2009 Audi A3 2009
```

```
2 Automatic transmission      2009 Audi A3 2.0 T AT Audi 2009 Audi A3 2009
3    Manual transmission         2009 Audi A3 2.0 T Audi 2009 Audi A3 2009
4 Automatic transmission 2009 Audi A3 2.0 T Quattro Audi 2009 Audi A3 2009
5 Automatic transmission 2009 Audi A3 2.0 T Quattro Audi 2009 Audi A3 2009
6    Manual transmission           2009 Audi A5 3.2 Audi 2009 Audi A5 2009
  Horsepower Torque
1        250    236
2        200    207
3        200    207
4        200    207
5        200    207
6        265    243
```

3.2 b. Restrict the data to cars whose Fuel Type is "Gasoline".

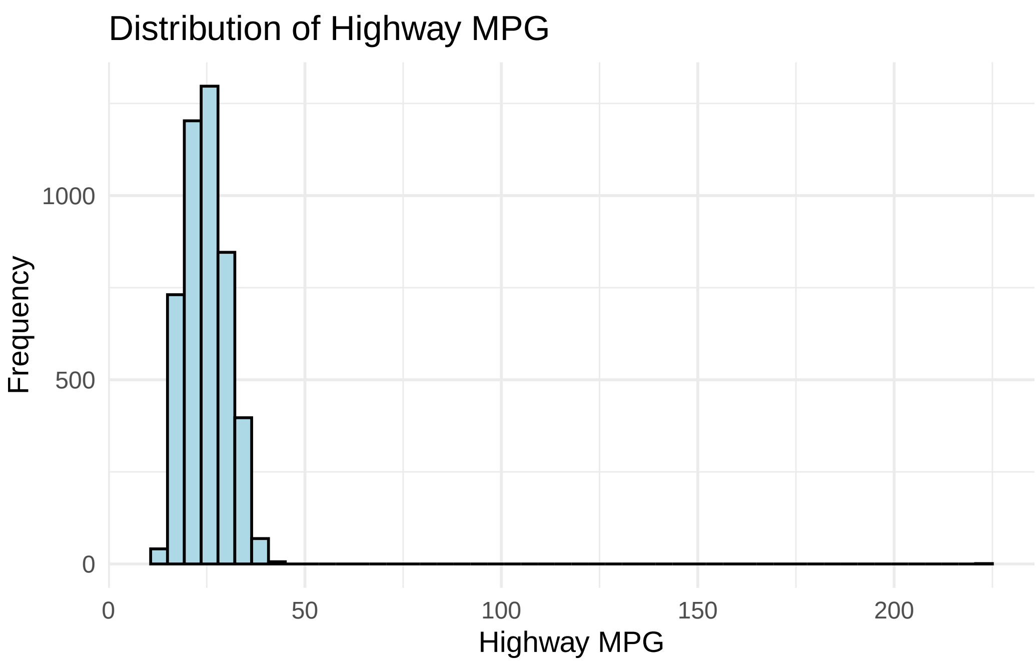
```r
# Restrict the data to cars whose Fuel Type is "Gasoline"
#' @description Filters the dataset to include only cars that use gasoline as fuel.
#' @details This subset operation is performed on the `cars` data frame,
#' and it selects rows where the `Fuel_Type` column matches "Gasoline".
#' @return The first few rows of the filtered data frame are displayed
#' using the `head` function to confirm the subset was correctly applied.
gasoline_cars <- subset(cars, Fuel_Type == "Gasoline")

# Display the first few rows to verify the correct application of the filter
head(gasoline_cars)
```

```
  Height Length Width         Driveline
1    140    143   202   All-wheel drive
2    140    143   202 Front-wheel drive
3    140    143   202 Front-wheel drive
4    140    143   202   All-wheel drive
5    140    143   202   All-wheel drive
6     91     17    62   All-wheel drive
                                 Engine_Type Hybrid Num_Forward_Gears
1         Audi 3.2L 6 cylinder 250hp 236ft-lbs   True                 6
2 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
3 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
4 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
5 Audi 2.0L 4 cylinder 200 hp 207 ft-lbs Turbo   True                 6
6        Audi 3.2L 6 cylinder 265hp 243 ft-lbs   True                 6
```

```
                    Transmission City_MPG Fuel_Type Highway_MPG
1 6 Speed Automatic Select Shift       18  Gasoline          25
2 6 Speed Automatic Select Shift       22  Gasoline          28
3                 6 Speed Manual       21  Gasoline          30
4 6 Speed Automatic Select Shift       21  Gasoline          28
5 6 Speed Automatic Select Shift       21  Gasoline          28
6                 6 Speed Manual       16  Gasoline          27
          Classification                          ID Make   Model_Year Year
1 Automatic transmission           2009 Audi A3 3.2 Audi 2009 Audi A3 2009
2 Automatic transmission      2009 Audi A3 2.0 T AT Audi 2009 Audi A3 2009
3    Manual transmission         2009 Audi A3 2.0 T Audi 2009 Audi A3 2009
4 Automatic transmission 2009 Audi A3 2.0 T Quattro Audi 2009 Audi A3 2009
5 Automatic transmission 2009 Audi A3 2.0 T Quattro Audi 2009 Audi A3 2009
6    Manual transmission           2009 Audi A5 3.2 Audi 2009 Audi A5 2009
  Horsepower Torque
1        250    236
2        200    207
3        200    207
4        200    207
5        200    207
6        265    243
```

3.3 c. Examine the distribution of highway gas mileage.

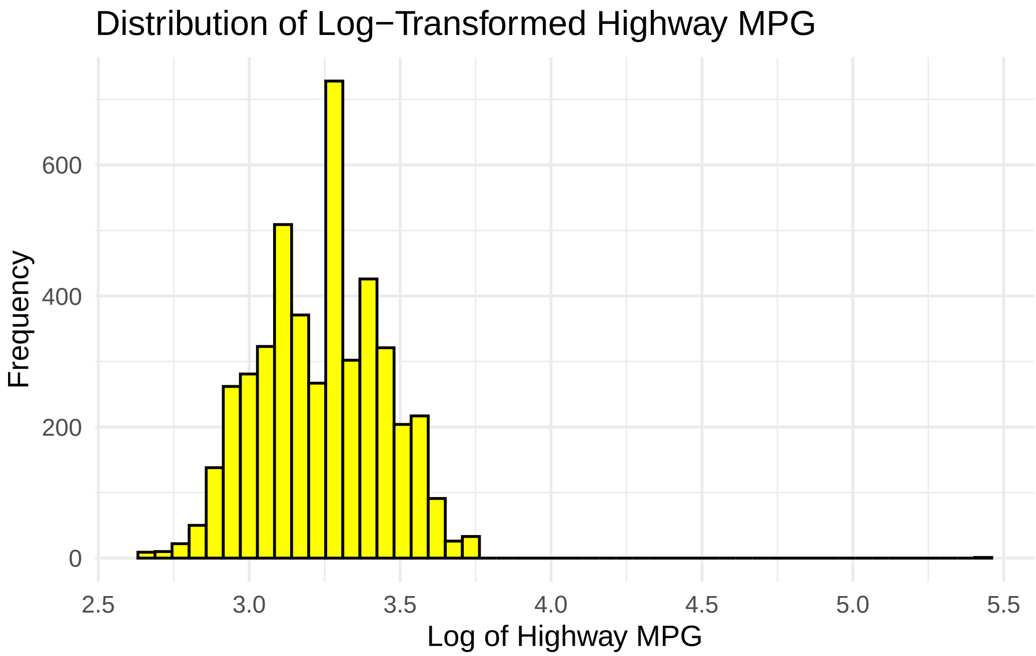
```r
# Load necessary libraries for visualization and statistical analysis
library(ggplot2)
library(e1071)

# Plotting the distribution of highway miles per gallon (MPG)
#' @description Visualizes the distribution of highway MPG to identify skewness in the data.
#' @details This plot helps in identifying the distribution characteristics of the data, che
ggplot(gasoline_cars, aes(x = Highway_MPG)) +
  geom_histogram(bins = 50, fill = "lightblue", color = "black") +
  labs(title = "Distribution of Highway MPG", x = "Highway MPG", y = "Frequency") +
  theme_minimal()
```




```
# Calculate the skewness of highway MPG
#' @description Calculates and prints the skewness value for highway MPG.
#' @details Skewness value assesses the symmetry of the data distribution to determine if a t
skewness_value <- skewness(gasoline_cars$Highway_MPG, na.rm = TRUE)
print(paste("Skewness of original data: ", skewness_value))
```

```
[1] "Skewness of original data:  7.99089540393757"
```

```
# Based on skewness determine if log transformation is necessary
#' @description Applies a log transformation to data if skewness is significant to improve it
if (skewness_value > 1 || skewness_value < -1) {
  gasoline_cars$Log_Highway_MPG <- log(gasoline_cars$Highway_MPG + 1)  # Plus 1 to avoid taki
  # Plotting the distribution after log transformation
  ggplot(gasoline_cars, aes(x = Log_Highway_MPG)) +
    geom_histogram(bins = 50, fill = "yellow", color = "black") +
    labs(title = "Distribution of Log-Transformed Highway MPG", x = "Log of Highway MPG", y =
    theme_minimal()
} else {
  cat("No significant skewness, transformation is not required.\n")
}
```




```
# Calculate skewness again for the log-transformed MPG
#' @description Calculates the skewness of log-transformed highway MPG to verify the effectiv
skewness_log_value <- skewness(gasoline_cars$Log_Highway_MPG, na.rm = TRUE)
print(paste("Skewness after log transformation: ", skewness_log_value))
```

```
[1] "Skewness after log transformation:  0.265907602320923"
```

Hence, the transformed variable is 'Log_Highway_MPG' variable, as it offers a more symmetric and normalized distribution ideal for further statistical modeling.

3.4 d. Fit a linear regression model predicting MPG on the highway.

3.4.1 1. Fit a linear regression model

```
# Predicting Highway MPG Using Multiple Regression
#' @description Converting the 'Year' into a categorical variable and fits a linear regressio
#' @details The model includes torque, horsepower, vehicle dimensions (height, length, width)
#' @return The output is a summary of the model, providing coefficients and statistical signi
# Ensure 'Year' is treated as a categorical variable for the regression analysis
gasoline_cars$Year <- as.factor(gasoline_cars$Year)
```

```r
# Fit a linear regression model to predict highway MPG, using Log_Highway_MPG variable
model <- lm(Log_Highway_MPG ~ Torque + Horsepower + Height + Length + Width + Year, data = ga

# Display the summary of the model to review coefficients and statistical significance
summary(model)
```

```
Call:
lm(formula = Log_Highway_MPG ~ Torque + Horsepower + Height +
    Length + Width + Year, data = gasoline_cars)

Residuals:
     Min       1Q   Median       3Q      Max
-0.52102 -0.09065 -0.00451  0.09539  2.37319

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.535e+00  2.132e-02 165.830  < 2e-16 ***
Torque      -2.193e-03  6.499e-05 -33.746  < 2e-16 ***
Horsepower   8.766e-04  6.718e-05  13.050  < 2e-16 ***
Height       3.889e-04  3.324e-05  11.701  < 2e-16 ***
Length       3.391e-05  2.607e-05   1.301  0.19337
Width       -8.151e-05  2.668e-05  -3.055  0.00226 **
Year2010    -2.081e-02  1.997e-02  -1.042  0.29741
Year2011    -2.058e-03  1.993e-02  -0.103  0.91776
Year2012     3.898e-02  2.009e-02   1.940  0.05240 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1358 on 4582 degrees of freedom
Multiple R-squared:  0.5626,    Adjusted R-squared:  0.5618
F-statistic: 736.6 on 8 and 4582 DF,  p-value: < 2.2e-16
```

3.4.2 2. Discussion

Torque:
- Coefficient (Estimate): The coefficient for torque is -0.002193. This suggests that for each unit increase in torque, the log-transformed highway MPG decreases by approximately 0.0022 units, holding other factors constant.
- Interpretation in Original Scale: Since the dependent variable is log-transformed, the coefficient can be interpreted as a percentage change in the original scale. Specifically, a 1-unit

increase in torque results in approximately a 0.22% decrease in actual highway MPG.
- Standard Error: The standard error of this estimate is 0.000065, which indicates a high level of precision in the estimate.
- t Value: The t value of -33.746 is very high, indicating a strong, statistically significant relationship between torque and highway MPG.
- P Value: The p value is less than 2e-16, far below the 0.05 threshold, confirming that the effect of torque on highway MPG is highly statistically significant and unlikely to be due to chance.

Other Variables:
- Horsepower: The positive coefficient of 0.0008766 indicates that for each unit increase in horsepower, the log-transformed highway MPG increases by approximately 0.0009 units.The relationship is statistically significant ($p < 2e-16$).
- Car Dimensions (Height, Length, Width):
- Height has a positive coefficient (0.0003889), suggesting that taller vehicles may have slightly higher highway MPG.
- Length shows a small, positive coefficient (0.00003391), but the effect is not statistically significant ($p = 0.193$).
- Width has a negative coefficient (-0.00008151), indicating that wider cars tend to have slightly lower highway MPG. This effect is statistically significant ($p = 0.00226$).
- Year: The coefficients for different years (relative to the 2009 baseline) suggest that year-to-year differences in automotive technology and efficiency standards affect highway MPG, though not all effects are statistically significant. For instance, 2012 has a marginally significant positive effect ($p = 0.05240$), implying some advancements in fuel efficiency.

Overall Model Fit:
- Multiple R-squared: 0.5626, suggesting that approximately 56.26% of the variability in log-transformed highway MPG is explained by the model. This indicates a reasonably good fit.
- Adjusted R-squared: 0.5618, which adjusts for the number of predictors in the model, also showing a strong fit.

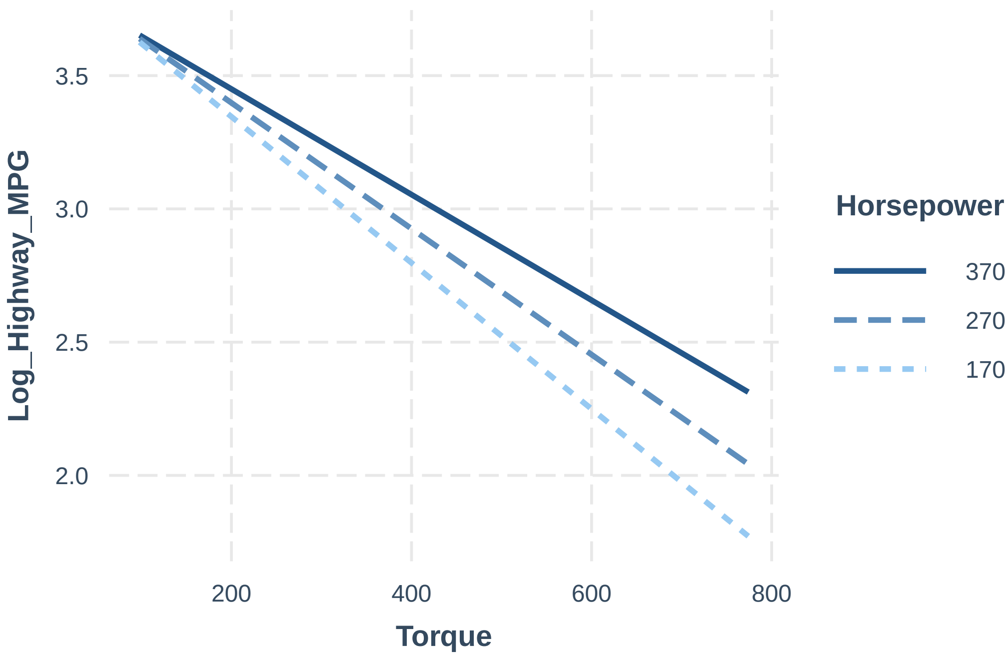
3.4.3 3. Explanation of the Additional Comments:

The reason Year2009 does not appear in the model summary is because it has been designated as the reference category.

3.5 e. Refit the model (with) and generate an interaction plot, showing how the relationship between torque and MPG changes as horsepower changes.

```
# Load necessary libraries for plotting and interactions
library(ggplot2)
```

```r
library(interactions)

# Predicting Highway MPG Using Multiple Regression
#' @description Converts 'Year' into a categorical variable and fits a linear regression mode
#' @details This model incorporates an interaction between torque and horsepower and controls
#' @param gasoline_cars The dataframe containing vehicle data.
#' @return Outputs an interaction plot illustrating the impact of torque at different levels
#' @examples interact_plot(model, pred = Torque, modx = Horsepower, modx.values = horsepower_
# Ensure 'Year' is treated as a categorical variable for the regression analysis
gasoline_cars$Year <- as.factor(gasoline_cars$Year)

# Fit a linear regression model to predict highway MPG
model_interaction <- lm(Log_Highway_MPG ~ Torque * Horsepower + Height + Length + Width + Yea

# Select reasonable horsepower values
horsepower_values <- c(170, 270, 370)

# Generate and plot the interaction effect of torque on Log_Highway_MPG at different horsepow
interaction_plot <- interact_plot(
  model = model_interaction,
  pred = Torque,
  modx = Horsepower,
  modx.values = horsepower_values,
  at = list(Year = "2012"),
  data = gasoline_cars,
  method = "regrid"
)

# Display the interaction plot
print(interaction_plot)
```



Discussion

-Relationship Between Torque and Horsepower: The plotting contains three lines for different amounts of horsepower. For all the levels of Horsepower, Log_Highway_MPG decreases with an increase in torque. This reflects that with increased torque, fuel efficiency decreases at all levels of considered horsepower.

-Horsepower is a moderator in how torque affects fuel efficiency: at a higher level of horsepower-for example, 370-the dampening effect of torque is less. This may suggest that in vehicles with higher horsepower, the additional torque has a lesser effect on fuel efficiency, possibly due to advanced engine technologies or other performance optimization measures.

-Impact of Torque on MPG: As the torque increases, the line graph of vehicles of all different horsepower levels all show a downward trend in MPG. This would support a hypothesis that torque is inversely related to fuel efficiency.

3.6 f Calculate $\hat{\beta}$ from d

```
# Loading necessary libraries
library(dplyr)
```

```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag


The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(tidyr)

# Ensure the 'Year' variable is correctly treated as a categorical factor
gasoline_cars$Year <- as.factor(gasoline_cars$Year)

# Create the design matrix for the regression model
X <- model.matrix(~ Torque + Horsepower + Height + Length + Width + Year, data = gasoline_ca
Y <- gasoline_cars$Log_Highway_MPG

# Calculate the coefficients using matrix algebra
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% Y

# Output the computed coefficients
print(beta_hat)
```

```
                      [,1]
(Intercept)  3.535004e+00
Torque      -2.193132e-03
Horsepower   8.766400e-04
Height       3.889137e-04
Length       3.391030e-05
Width       -8.151497e-05
Year2010    -2.080756e-02
Year2011    -2.058498e-03
Year2012     3.898086e-02
```

```r
# Displaying the summary from the lm() function for comparison
summary(model)
```

```
Call:
lm(formula = Log_Highway_MPG ~ Torque + Horsepower + Height +
    Length + Width + Year, data = gasoline_cars)
```

```
Residuals:
     Min       1Q   Median       3Q      Max 
-0.52102 -0.09065 -0.00451  0.09539  2.37319 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  3.535e+00  2.132e-02 165.830  < 2e-16 ***
Torque      -2.193e-03  6.499e-05 -33.746  < 2e-16 ***
Horsepower   8.766e-04  6.718e-05  13.050  < 2e-16 ***
Height       3.889e-04  3.324e-05  11.701  < 2e-16 ***
Length       3.391e-05  2.607e-05   1.301  0.19337    
Width       -8.151e-05  2.668e-05  -3.055  0.00226 ** 
Year2010    -2.081e-02  1.997e-02  -1.042  0.29741    
Year2011    -2.058e-03  1.993e-02  -0.103  0.91776    
Year2012     3.898e-02  2.009e-02   1.940  0.05240 .  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1358 on 4582 degrees of freedom
Multiple R-squared:  0.5626,    Adjusted R-squared:  0.5618 
F-statistic: 736.6 on 8 and 4582 DF,  p-value: < 2.2e-16
```

Hence, I get the same result as lm did prior.

4 References and Data Sources

The datasets used in this project are obtained from the following sources:

- **Cars Data**: This dataset is from the CORGIS Project, used to explore the relationship between vehicle characteristics like torque and fuel efficiency.

5 Methods and Implementation

All data analyses were performed in the R environment, employing a range of techniques including data import, data cleaning, statistical analysis, and result visualization.

6 Code and Documentation Repository

This document and related code are hosted on GitHub for review and sharing purposes. Access link: GitHub Repository Link

7 Acknowledgements

Thanks to the course instructors and teaching assistants for their guidance on this assignment. Thanks to all data providers for supporting open data.

8 Notes

The analyses in this document are for academic purposes only, intended to fulfill the requirements of a statistics course. While every reasonable effort has been made to ensure the accuracy of the analysis results, the content of this document represents only the views of the author.