Problem Set 2

Rajvi Jasani

GitHub Repository

This is the link to my GitHub repository https://github.com/rajvijasani/STATS506-Problem-Set-2.git

Problem 1 - Dice Game

a.

```r
#' Function to calculate total winnings version 1 (using a loop)
#'
#' @param n number of rolls
#' @param seed to control randomization (if a value is provided)
#'
#' @return total winnings
play_dice_v1 <- function(n, seed = NULL) {
  if (n < 0) {
    # checking for a negative input and showing appropriate error
    stop("number of rolls must be positive")
  }
  if (n == 0) {
    # if no die is rolled (game is not played), winnings=0
    return(0)
  }
  set.seed(seed)
  rolls <- sample(1:6, n, replace = TRUE)
  winnings <- 0

  for (i in rolls) {
    # -2 for the cost of a roll
    winnings <- winnings - 2
```

```r
    if (i == 3 | i == 5) {
      winnings <- winnings + (i * 2)
    }
    # any other roll wins nothing
  }
  return(winnings)
}

#' Function to calculate total winnings version 2 (using vectorization)
#'
#' @param n number of rolls
#' @param seed to control randomization (if a value is provided)
#'
#' @return total winnings
play_dice_v2 <- function(n, seed = NULL) {
  if (n < 0) {
    # checking for a negative input and showing appropriate error
    stop("number of rolls must be positive")
  }
  if (n == 0) {
    # if no die is rolled (game is not played), winnings=0
    return(0)
  }
  set.seed(seed)
  rolls <- sample(1:6, n, replace = TRUE)
  winnings <- 0
  # desired_rolls is a logical vector which stores
  # TRUE if a roll is 3 or 5 and FALSE otherwise
  desired_rolls <- (rolls == 3 | rolls == 5)
  # multiplying each roll with corresponding logical value and 2
  # [TRUE->1; FALSE->0]
  # and subtracting cost of roll
  # if roll is 3, (3*1*2)-2=4
  # if roll is 6, (6*0*2)-2=-2
  winnings_per_roll <- rolls * desired_rolls * 2 - 2
  return(sum(winnings_per_roll))
}

#' Function to calculate total winnings version 3 (using table function)
#'
#' @param n number of rolls
#' @param seed to control randomization (if a value is provided)
```

```
#'
#' @return total winnings
play_dice_v3 <- function(n, seed = NULL) {
  if (n < 0) {
    # checking for a negative input and showing appropriate error
    stop("number of rolls must be positive")
  }
  if (n == 0) {
    # if no die is rolled (game is not played), winnings=0
    return(0)
  }
  set.seed(seed)
  rolls <- sample(1:6, n, replace = TRUE)
  winnings <- 0
  # collecting the frequencies of rolls in a table form
  # factor() is used to include levels which have frequency=0 in the table
  rolls_count <- table(factor(rolls, levels = c(1, 2, 3, 4, 5, 6)))
  # as.numeric is used to only extract the frequency from the table
  winnings <- as.numeric(rolls_count[3]) * 6 + as.numeric(rolls_count[5]) * 10 - n * 2
  return(winnings)
}

#' Function to calculate total winnings version 4 (using vapply function)
#'
#' @param n number of rolls
#' @param seed to control randomization (if a value is provided)
#'
#' @return total winnings
play_dice_v4 <- function(n, seed = NULL) {
  if (n < 0) {
    # checking for a negative input and showing appropriate error
    stop("number of rolls must be positive")
  }
  if (n == 0) {
    # if no die is rolled (game is not played), winnings=0
    return(0)
  }
  set.seed(seed)
  rolls <- sample(1:6, n, replace = TRUE)
  winnings <- 0
  # applying function to return winnings (except cost of roll) of each roll
  # on all samples in the rolls vector,
```

```
  # summing the winnings and subtracting the costs all of rolls
  winnings <- sum(vapply(rolls, function(i) {
    if (i == 3 | i == 5) {
      return(i * 2)
    }
    return(0)
  }, 1)) - (n * 2)
  return(winnings)
}
```

Attribution of source for v3: Used ChatGPT to find a function to get the frequencies for all levels (even if they are zero) AND to find a function to only extract the frequency (without the name of level)

b.

Version 1

```
c(play_dice_v1(3), play_dice_v1(3000))
```

```
[1]    4 2136
```

Version 2

```
c(play_dice_v2(3), play_dice_v2(3000))
```

```
[1]    4 1796
```

Version 3

```
c(play_dice_v3(3), play_dice_v3(3000))
```

```
[1]    0 1602
```

Version 4

```
c(play_dice_v4(3), play_dice_v4(3000))
```

```
[1]    0 2164
```

c.

Results for n = 3, seed = 223

```
c(
  play_dice_v1(3, seed = 223),
  play_dice_v2(3, seed = 223),
  play_dice_v3(3, seed = 223),
  play_dice_v4(3, seed = 223)
)
```

```
[1] -6 -6 -6 -6
```

Results for n = 3000, seed = 223

```
c(
  play_dice_v1(3000, seed = 223),
  play_dice_v2(3000, seed = 223),
  play_dice_v3(3000, seed = 223),
  play_dice_v4(3000, seed = 223)
)
```

```
[1] 1652 1652 1652 1652
```

d.

Comparison of speeds with n = 1000

```
library(microbenchmark)
microbenchmark(
  version1 = play_dice_v1(1000, seed = 223),
  version2 = play_dice_v2(1000, seed = 223),
  version3 = play_dice_v3(1000, seed = 223),
  version4 = play_dice_v4(1000, seed = 223)
)
```

```
Unit: microseconds
     expr    min      lq     mean  median      uq    max neval
 version1  340.1  359.90  453.732  402.45  512.95 1166.1   100
 version2  129.0  155.45  195.170  180.95  231.65  370.0   100
 version3  329.3  362.05  492.425  422.40  572.55 1049.0   100
 version4 1252.3 1289.10 1660.092 1460.15 1967.20 3381.1   100
```

We see that there is only a small difference in speeds of version 1 (loop) and version 3 (table). Version 4 (vapply) is the slowest whereas version 2 (vectorization) is the fastest as expected.

Comparison of speeds with n = 100000

```
microbenchmark(
  version1 = play_dice_v1(100000, seed = 223),
  version2 = play_dice_v2(100000, seed = 223),
  version3 = play_dice_v3(100000, seed = 223),
  version4 = play_dice_v4(100000, seed = 223)
)
```

```
Unit: milliseconds
     expr      min        lq      mean    median        uq      max neval
 version1  32.5631  37.20690  45.57724  39.64330  45.18680 198.1683   100
 version2  11.1365  12.63385  14.06756  13.37725  14.74065  46.4822   100
 version3  16.0535  17.96375  20.60429  19.72305  21.61325  61.3299   100
 version4 123.2102 139.67105 154.08345 145.18730 160.84155 321.5470   100
```

As number of rolls increased, we can see a significant difference in speeds of version 1 (loop) and version 3 (table) which tells us that loops are not speed-wise efficient for big samples whereas tables, second fastest approach, can be useful for big samples. Version 4 (vapply) is still the slowest by a large margin whereas version 2 (vectorization) is the fastest as expected.

e.

We will be running 3 simulations of 10000 trials each but with different number of rolls just to be sure that number of rolls does not affect the winnings drastically.

```
# number of trials to run
trials <- 10000
# we will be running 3 different simulations with different number of rolls in each just to b
# vectors to save winning amounts (or losses) of each trail
all_winnings_1 <- vector(length = trials)
all_winnings_2 <- vector(length = trials)
all_winnings_3 <- vector(length = trials)
for (i in 1:trials) {
  # simulation with 10 rolls in each trial
  all_winnings_1[i] <- play_dice_v2(10)
  # simulation with 100 rolls in each trial
  all_winnings_2[i] <- play_dice_v2(100)
  # simulation with 1000 rolls in each trial
  all_winnings_3[i] <- play_dice_v2(1000)
```

```
}
c(mean(all_winnings_1),
  mean(all_winnings_2),
  mean(all_winnings_3))
```

```
[1]   6.9328  66.2358 670.1092
```

Each simulation gives a positive mean, i.e. over 10000 trials of 10, 100, or 1000 rolls, the player tends to win money on average. This makes the game biased towards the player, thus unfair. A fair game should have an average around 0, i.e. no gain no loss.

Problem 2 - Linear Regression

a.

```
setwd("E:/UM/STATS 506/Repos/STATS506-Problem-Set-2")
# assigning new column names for the data frame
col_names <- c(
  "height",
  "length",
  "width",
  "driveline",
  "engine_type",
  "hybrid",
  "gears",
  "transmission",
  "city_mpg",
  "fuel_type",
  "highway_mpg",
  "classification",
  "id",
  "make",
  "model_year",
  "year",
  "horsepower",
  "torque"
)
# importing the data set
cars <- read.csv("data/cars.csv", col.names = col_names)
```

b.

```
# number of observations before restricting the dataset
nrow(cars)
```

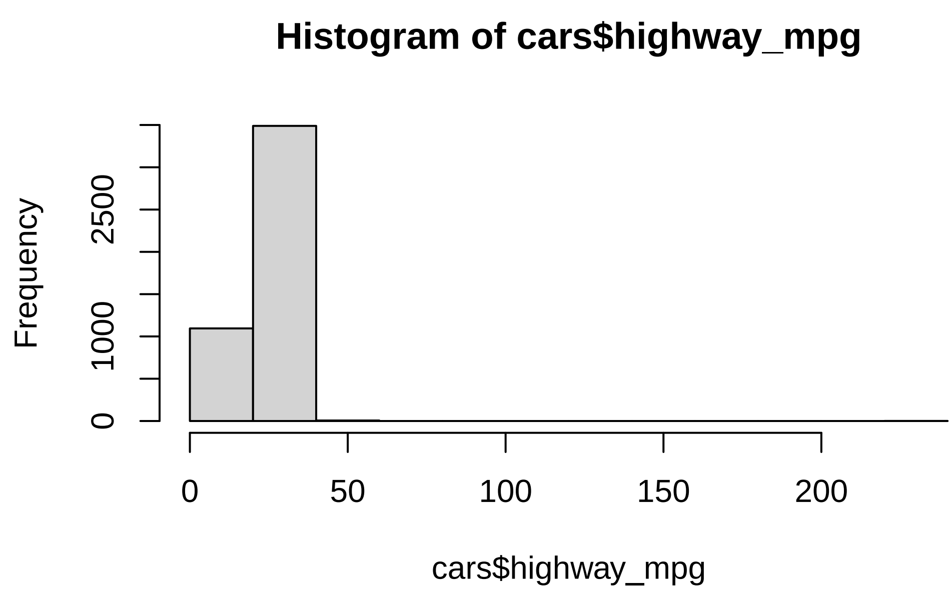
```
[1] 5076
```

```
cars <- cars[cars$fuel_type == "Gasoline", ]
# number of observations before restricting the dataset
nrow(cars)
```

```
[1] 4591
```

c.

```
# exploratory data analysis to check if highway_mpg variable needs transformation
hist(cars$highway_mpg)
```




```
c(min(cars$highway_mpg), max(cars$highway_mpg))
```

```
[1]  13 223
```

```
library(e1071)
skewness(cars$highway_mpg)
```

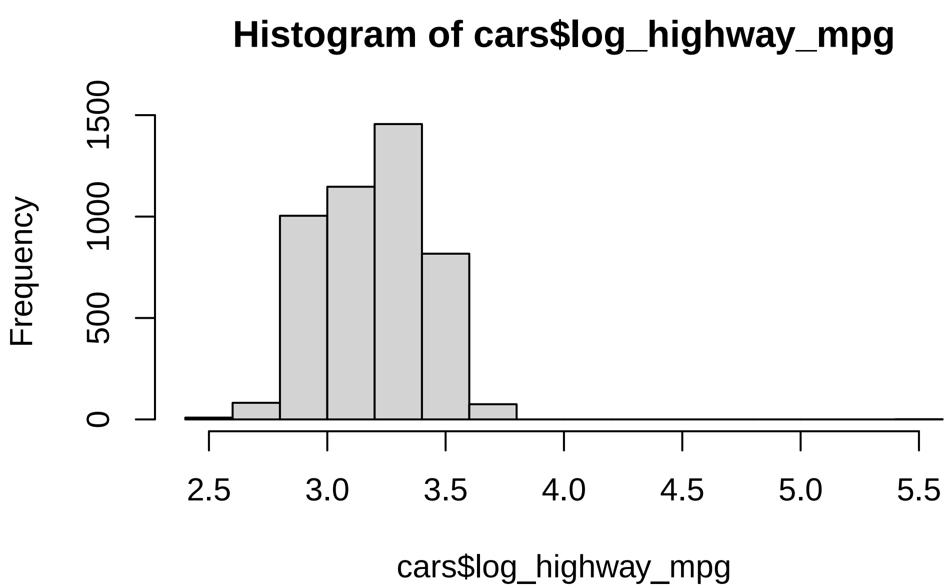
```
[1] 7.990895
```

```
tail(table(cars$highway_mpg))
```

```
 38  39  40  41  42 223
 14  12  27   3   3   1
```

From looking at the histogram, min-max values and skewness coefficient of the highway_mpg variable, I can notice that the distribution is skewed. After looking at the table of highway_mpg values I can understand that the maximum value is probably an outlier and unrealistic in a real-world case, but as the question does not mention if it is okay to remove this outlier, I am going to take it as a realistic value and remove the skewness by log transformation of the highway_mpg variable.

```
# log transformation of highway_mpg
cars$log_highway_mpg <- log(cars$highway_mpg)
hist(cars$log_highway_mpg)
```



```r
c(min(cars$log_highway_mpg), max(cars$log_highway_mpg))
```

```
[1] 2.564949 5.407172
```

Attribution of source: Used ChatGPT to find the library and function to calculate skewness.

d.

```r
# modeling a linear regression model
lm_model <- lm(log_highway_mpg ~ torque + horsepower + height + length +
                 width + as.factor(year),
               data = cars)
summary(lm_model)
```

```

Call:
lm(formula = log_highway_mpg ~ torque + horsepower + height +
    length + width + as.factor(year), data = cars)

Residuals:
     Min       1Q   Median       3Q      Max
-0.54759 -0.09385 -0.00414  0.09894  2.41852

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          3.507e+00  2.216e-02 158.236  < 2e-16 ***
torque              -2.294e-03  6.757e-05 -33.956  < 2e-16 ***
horsepower           9.238e-04  6.984e-05  13.227  < 2e-16 ***
height               4.050e-04  3.456e-05  11.719  < 2e-16 ***
length               3.475e-05  2.710e-05   1.282  0.19980
width               -8.722e-05  2.774e-05  -3.144  0.00168 **
as.factor(year)2010 -2.181e-02  2.076e-02  -1.051  0.29342
as.factor(year)2011 -2.430e-03  2.072e-02  -0.117  0.90665
as.factor(year)2012  4.012e-02  2.089e-02   1.921  0.05485 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1412 on 4582 degrees of freedom
Multiple R-squared:  0.5638,    Adjusted R-squared:  0.563
F-statistic: 740.3 on 8 and 4582 DF,  p-value: < 2.2e-16
```

The torque coefficient is -2.294e-03 with a near zero p-value. This means that torque has a strong, statistically significant, negative effect on highway mpg, i.e. as torque increases, highway mpg decreases.

e.

```
library(emmeans)
```

```
Welcome to emmeans.
Caution: You lose important information if you filter this package's results.
See '? untidy'
```

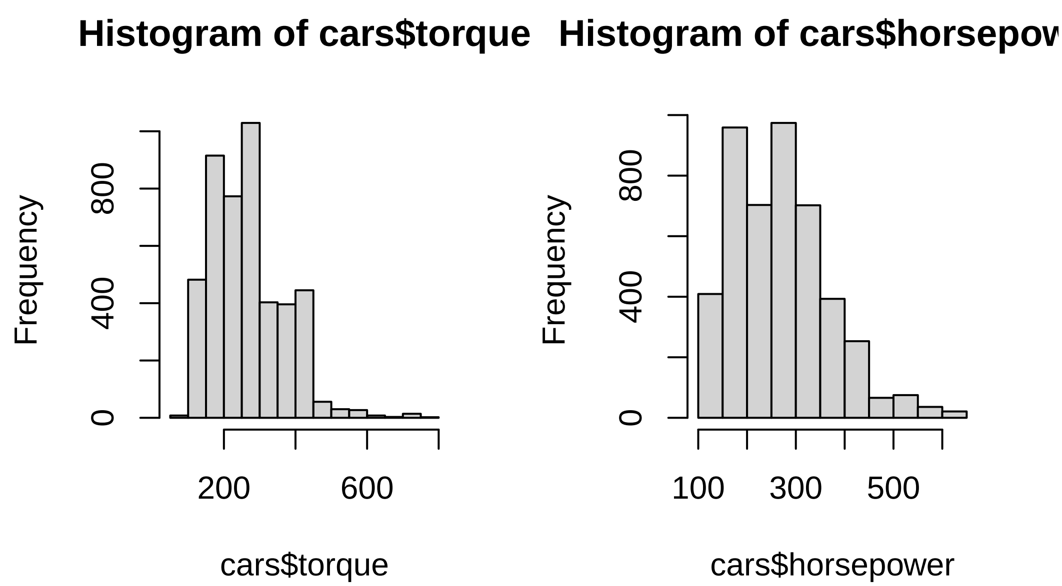
```
# modeling a linear regression model with interaction term
int_lm_model <- lm(log_highway_mpg ~ torque * horsepower + height + length +
                     width + as.factor(year),
                   data = cars)
summary(int_lm_model)
```

```

Call:
lm(formula = log_highway_mpg ~ torque * horsepower + height +
    length + width + as.factor(year), data = cars)

Residuals:
     Min       1Q   Median       3Q      Max
-0.55760 -0.08378 -0.00157  0.08194  2.45015

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)          3.854e+00  2.384e-02 161.669  < 2e-16 ***
torque              -3.533e-03  7.615e-05 -46.390  < 2e-16 ***
horsepower          -2.339e-04  7.632e-05  -3.064  0.00219 **
height               2.876e-04  3.215e-05   8.946  < 2e-16 ***
length               3.643e-05  2.500e-05   1.457  0.14525
width               -1.165e-04  2.561e-05  -4.548 5.55e-06 ***
as.factor(year)2010 -2.563e-02  1.915e-02  -1.338  0.18095
as.factor(year)2011 -5.886e-03  1.912e-02  -0.308  0.75822
as.factor(year)2012  3.640e-02  1.927e-02   1.889  0.05896 .
torque:horsepower    3.939e-06  1.391e-07  28.314  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1302 on 4581 degrees of freedom
Multiple R-squared:  0.6288,    Adjusted R-squared:  0.628
F-statistic: 862.1 on 9 and 4581 DF,  p-value: < 2.2e-16
```

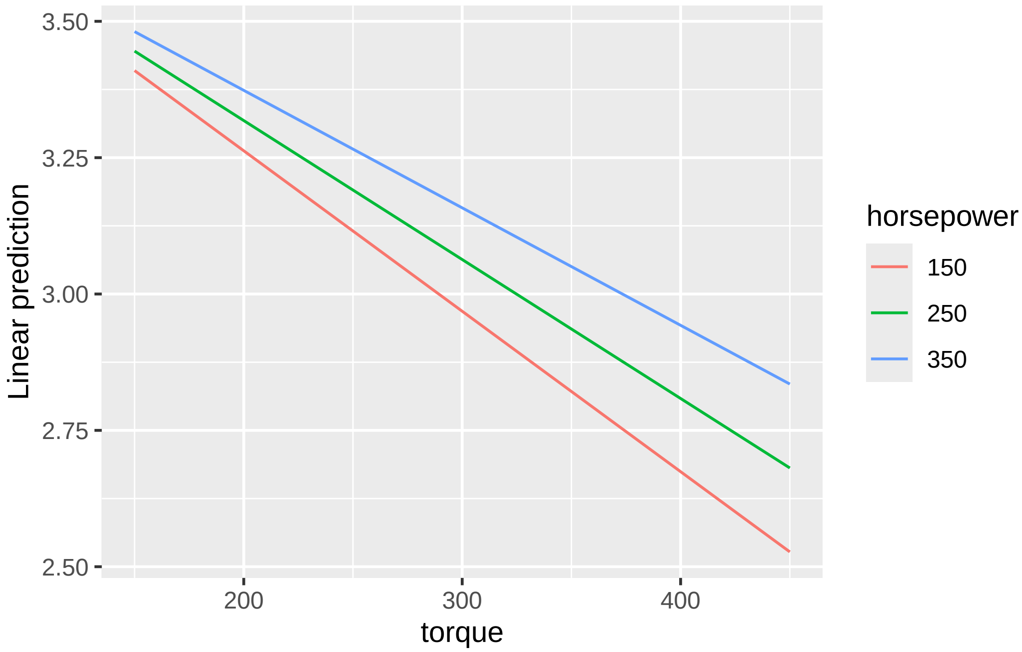
To choose reasonable values of torque and horsepower, we can look at their histograms

```
# to show 2 plots in one row
par(mfrow = c(1, 2))
hist(cars$torque)
hist(cars$horsepower)
```



Torque ranges from 150 to 450 approximately and I have taken horsepower values of 150, 250 and 350 as they cover the distribution of horsepower satisfactorily.

```
# plotting the interaction between torque an horsepower
emmip(int_lm_model,
      horsepower ~ torque,
      at = list(
        torque = seq(150, 450, 100),
        horsepower = c(150, 250, 350)
      ))
```



We can see that for all the chosen horsepower values, the trend in change of torque is consistent.

f.

$\hat{\beta} = (X^T X)^{-1} X^T y$

Attribution of source: Used ChatGPT for showing this formula on the doc.

```
# preparing design matrix X for manual calculation of beta
X <- model.matrix(log_highway_mpg ~ torque * horsepower + height + length +
                      width + as.factor(year),
                    data = cars)
y <- cars$log_highway_mpg
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
# combining lm coefficients and manual coefficients column-wise for ease of viewing
cbind(int_lm_model$coefficients, beta_hat)
```

```
                             [,1]          [,2]
(Intercept)         3.853858e+00  3.853858e+00
torque             -3.532564e-03 -3.532564e-03
horsepower         -2.338557e-04 -2.338557e-04
height              2.876238e-04  2.876238e-04
length              3.642612e-05  3.642612e-05
```

```
width                -1.165031e-04 -1.165031e-04
as.factor(year)2010 -2.562770e-02 -2.562770e-02
as.factor(year)2011 -5.886154e-03 -5.886154e-03
as.factor(year)2012  3.640253e-02  3.640253e-02
torque:horsepower    3.939306e-06  3.939306e-06
```

We can see that we get the same values of coefficients from the manual calculation of beta hat (column 2) as we got from the lm model (column 1).