HW2

Yoonseo Mok

2024-09-23

Problem 1

A

Version 1

```r
#' Version 1: Implement this game using a loop
#'
#' @param n The number of dice to roll.
#' @return Total winnings

v1 <- function(n) {
  tot_win <- 0
  for (i in 1:n) {
    rollnumb <- sample(1:6, 1)
    if (rollnumb %in%  3 | rollnumb %in% 5) {
      tot_win <- tot_win + 2 * rollnumb - 2
    } else {
      tot_win <- tot_win - 2
    }
  }
  return(tot_win)
}
```

Version 2

```r
#' Version 2: Implement this game using built-in R vectorized functions.
#'
#' @param n The number of dice to roll.
#' @return Total winnings

v2 <- function(n) {
  rollnumb <- sample(1:6, n,replace = T)
  win=ifelse(rollnumb %in% c(3,5),2*rollnumb-2,-2)
  total_win=sum(win)
  return(total_win)
}
```

Version 3

```r
#' Version 3: Implement this by rolling all the dice into one
#' and collapsing the die rolls into a single table()
#'
#' @param n The number of dice to roll.
#' @return Total winnings

v3 <- function(n) {
  rollnumb <- sample(1:6, n, replace = TRUE)
  rollnumb_table <- table(factor(rollnumb, levels = 1:6))

  tot_win <- sum((rollnumb_table[3] * (3*2 - 2)) + (rollnumb_table[5]* (5*2 - 2)), na.rm = TRUE)
  tot_lose <- sum(rollnumb_table[c(1,2,4,6)] * (-2), na.rm = TRUE)

  return(tot_win + tot_lose)
}
```

Version 4

```r
#' Version 4: Implement this game by using one of the "apply" functions.
#'
#' @param n The number of dice to roll.
#' @return Total winnings

v4 <- function(n) {
  rollnumb <- sample(1:6, n, replace = TRUE)
  winorlose <- sapply(rollnumb, function(x) {
    if (x %in%  3 | x %in% 5) {
      return(2*x-2)
    } else {
      return(-2)
    }
   }
  )
  return(sum(winorlose))
}
```

B

```r
# Demonstrate that all versions work
v1(3)
```

```
## [1] 4
```

```r
v1(3000)
```

```
## [1] 2030
```

```r
v2(3)
```

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

```r
v2(3000)
```

```
## [1] 2206
```

```r
v3(3)
```

```
## [1] 14
```

```r
v3(3000)
```

```
## [1] 2262
```

```r
v4(3)
```

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

```r
v4(3000)
```

```
## [1] 2060
```

All versions work

C

```r
# Demonstrate that the four versions give the same result.
set.seed(123)
v1(3)
```

```
## [1] 6
```

```r
set.seed(123)
v1(3000)
```

```
## [1] 2174
```

```r
set.seed(123)
v2(3)
```

```
## [1] 6
```

```r
set.seed(123)
v2(3000)
```

```
## [1] 2174
```

```r
set.seed(123)
v3(3)
```

```
## [1] 6
```

```r
set.seed(123)
v3(3000)
```

```
## [1] 2174
```

```r
set.seed(123)
v4(3)
```

```
## [1] 6
```

```r
set.seed(123)
v4(3000)
```

```
## [1] 2174
```

All four versions give the same result.

D

```r
library(microbenchmark)

# Demonstrate the speed of the implementations
# Low input of 1,000
x1<-1000
microbenchmark(v1(x1), v2(x1),v3(x1),v4(x1))
```

```
## Unit: microseconds
##    expr      min       lq       mean    median        uq      max neval cld
##  v1(x1) 2375.991 2448.110 2615.83034 2493.9890 2568.0965 5332.542   100 a
##  v2(x1)   49.938   51.660   54.46727   52.7875   56.4365   70.643   100  b
##  v3(x1)   62.238   64.493   70.73730   67.4655   75.1120  102.705   100  b
##  v4(x1)  767.233  792.079  849.24407  809.5245  838.6345 1579.771   100   c
```

```r
# High input of 100000
x2<-100000
microbenchmark(v1(x2), v2(x2),v3(x2),v4(x2))
```

```
## Unit: milliseconds
##    expr        min         lq       mean     median         uq        max neval
##  v1(x2) 242.566045 258.823918 286.226506 272.450535 296.418622 498.539623   100
##  v2(x2)   4.466294   4.611885   4.691368   4.666825   4.734680   5.465751   100
##  v3(x2)   3.934688   4.032452   4.706201   4.082719   4.134666  60.024164   100
##  v4(x2)  83.214543  87.810684 106.350487  96.932569 112.480958 227.768284   100
##  cld
##  a
##   b
##   b
##    c
```

Comparing the performance with low input of 1000, Version 2 function where we implemented the game using built-in R vectorized functions has the fastest performance. Comparing the performance with high input of 100000, Version 3 function where we Implemented the dice rolls into a single table() has the fastest performance. While version 3 is the fastest, version 2 function performance is similar. In both comparison, Version 1 function where we implemented the game using loop performs the slowest.

E

```r
# Evidence based upon a Monte Carlo simulation
simnumb <- 1000000
set.seed(123)

# Generate total winnings using version 2 function
total_net_result = v2(simnumb)

# Calculate average net result per game
average_net_result <- total_net_result / simnumb
average_net_result
```

```
## [1] 0.666316
```

```r
# Calculating using probability (just another way of providing evidence)
1/6*4+1/6*8+4/6*(-2)
```

```
## [1] 0.6666667
```

Based on the monte carlo simulation, we can see that we win 0.66. Therefore, we can conclude that this is a fair game since we are not loosing anything at least.

Problem 2

```r
# Importing the dataset
cars=read.csv("cars.csv")
```

A

```r
# Rename the columns
names(cars)=c("Height","Length","Width","Driveline","EngineType","Hybrid","NumbForwardGears",
              "Transmission","Citympg","FuelType","Highwaympg","Class","ID","Make","ModelYear","Year",
              "Horsepower","Torque")
```

B

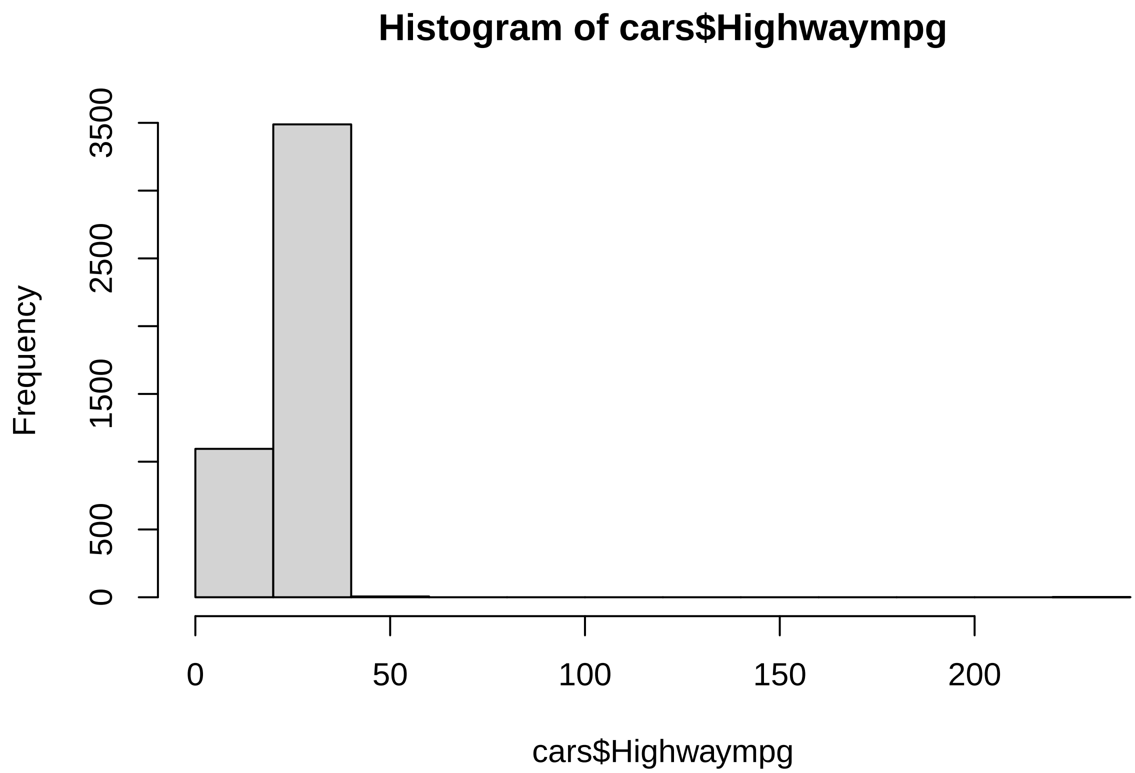
```r
# Restrict the data to cars whose Fuel Type is "Gasoline".
cars=cars[cars$FuelType %in% "Gasoline",]
```

C

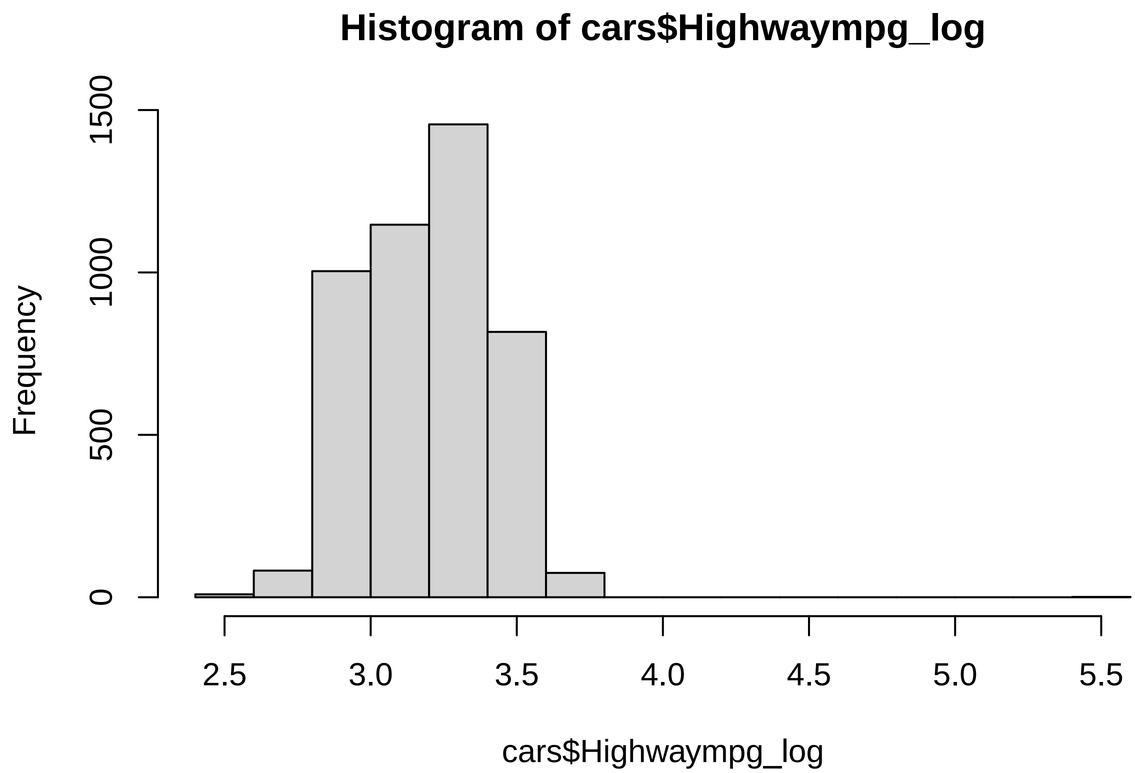
```r
# Checking highway mileage variable distribution
summary(cars$Highwaympg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   13.00   21.00   25.00   24.97   28.00  223.00
```

```r
hist(cars$Highwaympg)
```




```r
# Log transformation of highway mpg variable
cars$Highwaympg_log=log(cars$Highwaympg)
hist(cars$Highwaympg_log)
```



Highway gas mileage variable is slightly skewed. Although it is not extremely skewed, a log transformation could be used to normalize the highway mpg variable.

D

```
# Fit a linear regression model
model1=lm(Highwaympg_log ~ Torque+Horsepower+Height+Length+Width+factor(Year), data = cars)
summary(model1)
```

```
##
## Call:
## lm(formula = Highwaympg_log ~ Torque + Horsepower + Height +
##     Length + Width + 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 **
## factor(Year)2010 -2.181e-02  2.076e-02  -1.051  0.29342
## factor(Year)2011 -2.430e-03  2.072e-02  -0.117  0.90665
## 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
```

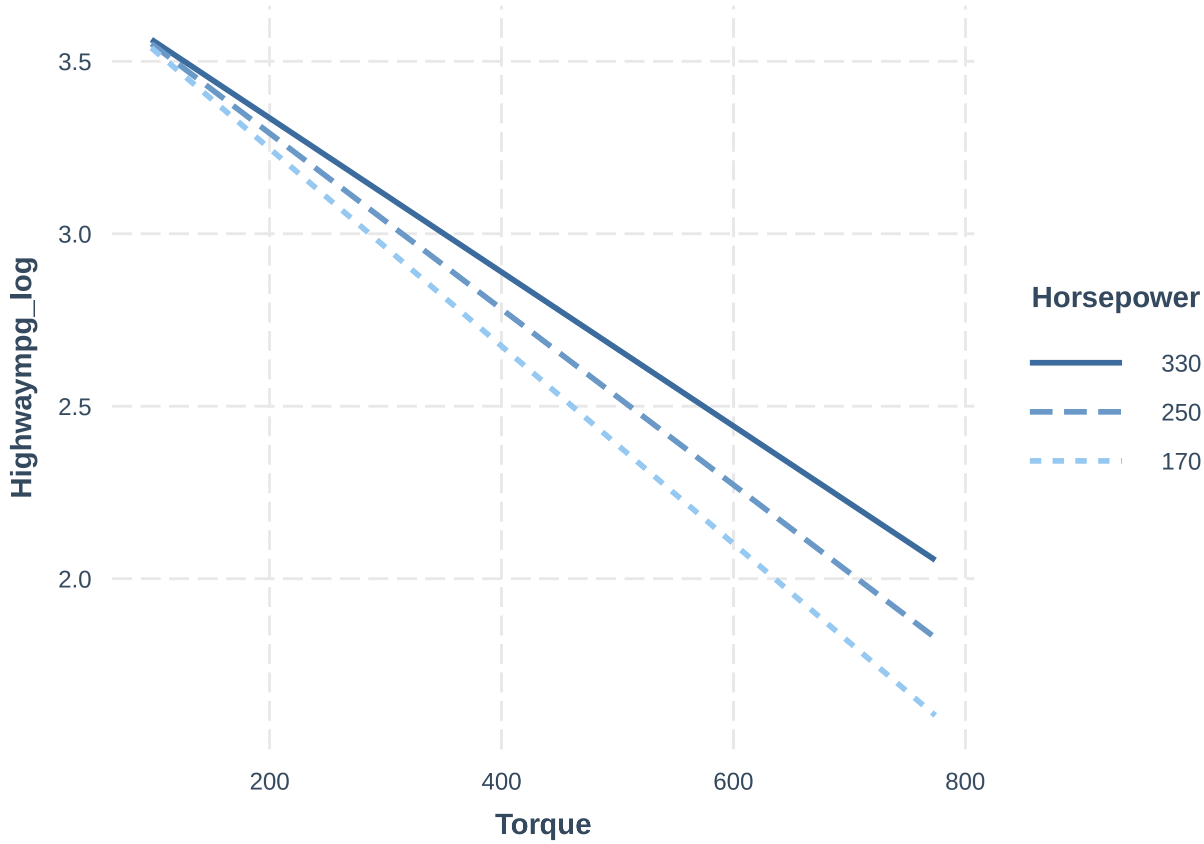
exp(-2.294e-03)=0.9977086 For every one-unit increase in the torque variable, highway gas mileage decreases by a factor of about 0.9977086, holding horsepower, all three dimensions of the car, and the year the car was released as constant.

E

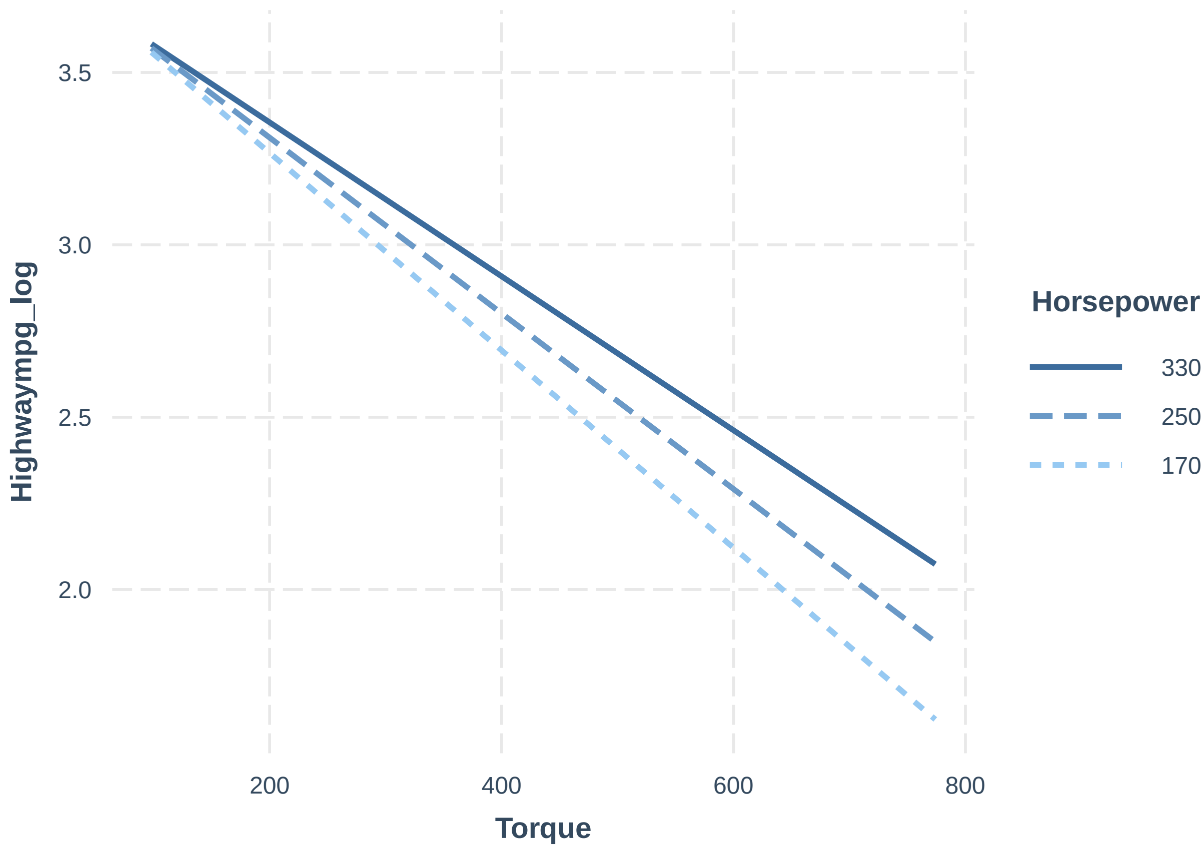
```r
# Fit a linear regression model with interaction
model2=lm(Highwaympg_log ~ Torque*Horsepower+Height+Length+Width+factor(Year), data = cars)
summary(model2)
```

```
##
## Call:
## lm(formula = Highwaympg_log ~ Torque * Horsepower + Height +
##     Length + Width + 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 ***
## factor(Year)2010  -2.563e-02  1.915e-02  -1.338  0.18095
## factor(Year)2011  -5.886e-03  1.912e-02  -0.308  0.75822
## 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
```

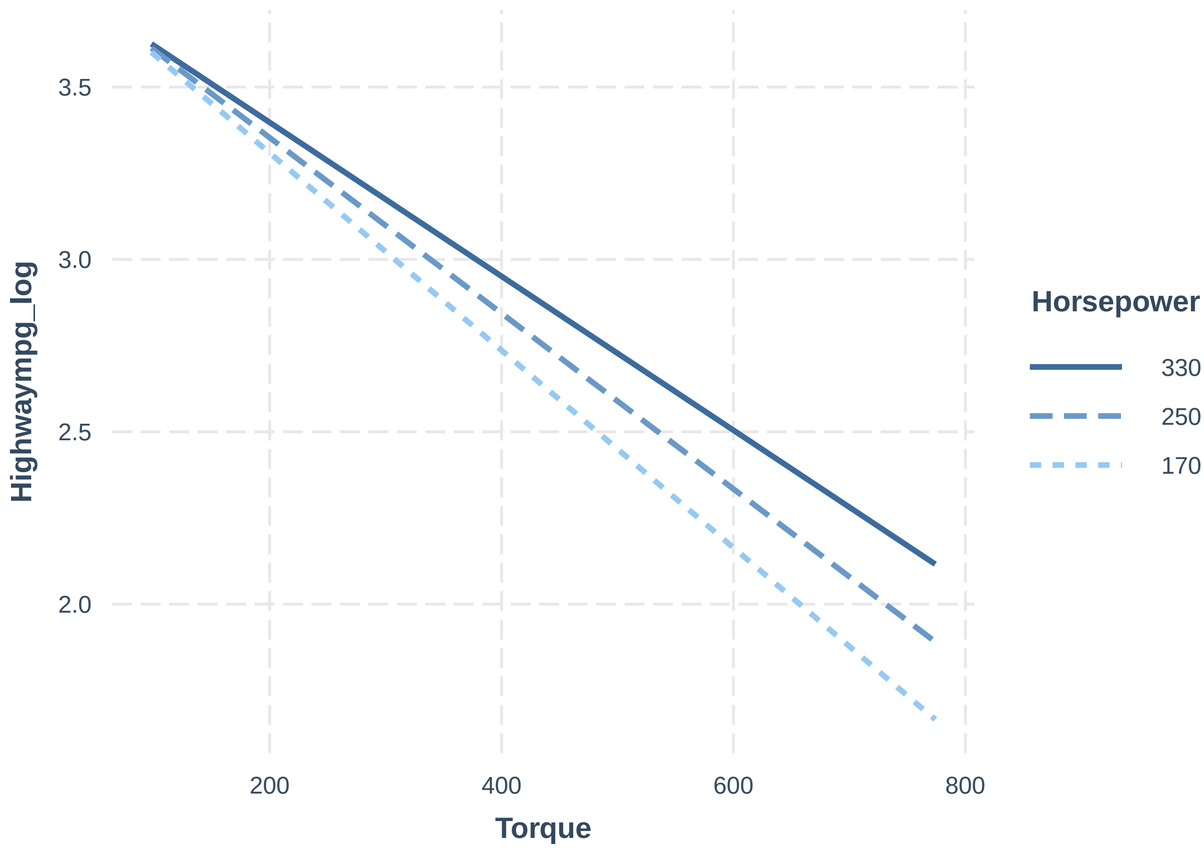
```r
library(interactions)

# Create interaction plot
interact_plot(model2, pred = Torque, modx = Horsepower,
              modx.values = c(170, 250, 330), at = list(Year = 2010), data=cars)
```




```r
interact_plot(model2, pred = Torque, modx = Horsepower,
              modx.values = c(170, 250, 330), at = list(Year = 2011), data=cars)
```




```r
interact_plot(model2, pred = Torque, modx = Horsepower,
              modx.values = c(170, 250, 330), at = list(Year = 2012), data=cars)
```



F

```r
# Creating design matrix
X <- model.matrix( ~ Torque + Horsepower + Height + Length + Width + factor(Year), data = cars)
Y <- cars$Highwaympg_log

# Calculate the beta coefficients using design matrix
solve(t(X) %*% X) %*% t(X) %*% Y
```

```
##                            [,1]
## (Intercept)        3.506922e+00
## Torque            -2.294331e-03
## Horsepower         9.238126e-04
## Height             4.049897e-04
## Length             3.475207e-05
## Width             -8.722295e-05
## factor(Year)2010 -2.181247e-02
## factor(Year)2011 -2.430359e-03
## factor(Year)2012  4.011528e-02
```

It gives out the same results as lm did prior