# Statistics 506- Problem Set #2

Garrett Pinkston

## Problem 1 - Dice Game

**a)** Build the following versions

- Version 1: Implement this game using a loop.

```
#' Simulate a Dice Game using Loops
#'
#' This function simulates a simple dice game where a player rolls a six-sided
#' die "nrolls" times using a "for" loop.
#'
#' The player gains money if the result of the roll is a 3 or 5, but loses $2
#' for each roll regardless.
#'
#' @param nrolls The number of dice rolls to simulate (an integer).
#'
#' @return The final amount of money (wallet) after all rolls are completed.
#' A positive value indicates profit, and a negative value indicates loss.
#' @export
#'
#' @examples
#' # Simulate 10 rolls of the dice
#' loopDiceGame(10)
#'
#' Simulate 100 rolls of the dice
#' loopDiceGame(100)
loopDiceGame <- function(nrolls){
  # set initial payout to 0
  wallet <- 0

  # loop through and simulate each roll individually
  for (i in 1:nrolls) {
```

```r
    #sample a number from 1 to 6 with replacement
    roll <- sample(1:6,1)

    # if a roll is 3 or 5, double it and add the amount to the wallet
    if ((roll == 3) || (roll == 5)) {
      wallet <- wallet + (2 * roll)
    }

    #regardless of outcome, subtract 2 bucks for playing the game
    wallet <- wallet - 2
  }
  return(wallet)
}

set.seed(123)
loopDiceGame(3000)
```

```
[1] 2174
```

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

```r
#' Simulate a Dice Game using Vectorized Functions
#'
#' This function simulates a simple dice game where a player rolls a six-sided
#' die "nrolls" times using vectorized functions in R.
#'
#' The player gains money if the result of the roll is a 3 or 5, but loses $2
#' for each roll regardless.
#'
#' @param nrolls The number of dice rolls to simulate (an integer).
#'
#' @return The final amount of money (wallet) after all rolls are completed.
#' A positive value indicates profit, and a negative value indicates loss.
#' @export
#'
#' @examples
#' # Simulate 10 rolls of the dice
#' vectorizedDiceGame(10)
#'
#' # Simulate 100 rolls of the dice
#' vectorizedDiceGame(100)
```

```r
vectorizedDiceGame <- function(nrolls){
  # set initial payout to 0
  wallet <- 0

  # expand all rolls into a single vector (values 1-6) with size nrolls
  allRolls <- sample(1:6,nrolls,replace=TRUE)

  # take all instances where roll value is 3 or 5 then double it
  # and store it into wallet
  wallet <- (allRolls == 3 | allRolls == 5) * 2 *allRolls

  # sum the entire vector and then subtract cost to play ($2 per roll)
  wallet <- sum(wallet) - (2*nrolls)

  # return final result
  return(wallet)
}

set.seed(123)
vectorizedDiceGame(3000)
```

```
[1] 2174
```

- Version 3: Implement this by rolling all the dice into one and collapsing the die rolls into a single `table()`. (Hint: Be careful indexing the table - what happens if you make a table of a single dice roll? You may need to look to other resources for how to solve this.)

```r
#' Simulate a Dice Game using a Table
#'
#' This function simulates a simple dice game where a player rolls a six-sided
#' die "nrolls" times by collapsing all dice rolls into a single table.
#'
#' The player gains money if the result of the roll is a 3 or 5,
#' but loses $2 for each roll regardless.
#'
#' @param nrolls The number of dice rolls to simulate (an integer).
#'
#' @return The final amount of money (wallet) after all rolls are completed.
#' A positive value indicates profit, and a negative value indicates loss.
#' @export
#'
```

```r
#' @examples
#' # Simulate 10 rolls of the dice
#' tableDiceGame(10)
#'
#' # Simulate 100 rolls of the dice
#' tableDiceGame(100)
tableDiceGame <- function(nrolls){
  # set initial count to 0
  wallet <- 0

  # create a vector of all rolls (values 1-6) with size nrolls
  allRolls <- sample(1:6,nrolls,replace=TRUE)

  # collapse all roll counts into table. We need to ensure all counts appear in
  # the table (even if num wasn't rolled), so we will use factor
  allRolls <- factor(allRolls,levels=1:6)
  rollsTable <- table(allRolls)

  # calculate profit for rolling 3's and 5's respectively
  # profit = numOccurences * numberRolled * payoff (double)
  profitForThrees <- as.numeric(rollsTable["3"]) * 2 * 3
  profitForFives <- as.numeric(rollsTable["5"]) * 2 * 5

  # Add profit into the wallet
  wallet <- profitForThrees + profitForFives

  # subtract cost to play (2 * number of rolls)
  wallet <- wallet - (2 * nrolls)

  return(wallet)
}
set.seed(123)
tableDiceGame(3000)
```

```
[1] 2174
```

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

```r
#' Simulate a Dice Game using an "apply" function
#'
#' This function simulates a simple dice game where a player rolls a six-sided
#' die "nrolls" times using vapply to ensure all values are integers.
```

```r
#'
#' The player gains money if the result of the roll is a 3 or 5,
#' but loses $2 for each roll regardless.
#'
#' @param nrolls The number of dice rolls to simulate (an integer).
#'
#' @return The final amount of money (wallet) after all rolls are completed.
#' A positive value indicates profit, and a negative value indicates loss.
#' @export
#'
#' @examples
#' # Simulate 10 rolls of the dice
#' applyDiceGame(10)
#'
#' # Simulate 100 rolls of the dice
#' applyDiceGame(100)
applyDiceGame <- function(nrolls){

  # set initial payout to 0
  wallet <- 0

  # create a vector of all rolls (values 1-6) with size nrolls
  allRolls <- sample(1:6,nrolls,replace=TRUE)

  # Use Vapply to apply function returning 2 times the value if the roll is a
  # 3 or 5, and 0 otherwise, storing results. Make sure all numbers are integers.
  allProfits <- vapply(allRolls,function(individualRoll){
      if ((individualRoll == 3) || (individualRoll == 5)) {
        return(as.integer(2 * individualRoll))
      } else{
        return(0L)
      }
  },integer(1))

  # sum all profits (vector) and store value
  wallet <- sum(allProfits)

  # subtract cost to play (2 * number of rolls)
  wallet <- wallet - (2 * nrolls)

  return(wallet)
}
```

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

```
[1] 2174
```

**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.

```r
# run loop dice game for 3 and 3000
loopDiceGame(3)
```

```
[1] -6
```

```r
loopDiceGame(3000)
```

```
[1] 1418
```

```r
# run vectorized dice game for 3 and 3000
vectorizedDiceGame(3)
```

```
[1] -6
```

```r
vectorizedDiceGame(3000)
```

```
[1] 2306
```

```r
# run table dice game for 3 and 3000
tableDiceGame(3)
```

```
[1] 0
```

```r
tableDiceGame(3000)
```

```
[1] 2084
```

```r
# run "apply" dice game for 3 and 3000
applyDiceGame(3)
```

```
[1] 10
```

```r
applyDiceGame(3000)
```

```
[1] 1972
```

As we can observe, all functions work and provide output. 8 functions were tested and 8 reasonable pieces of output were returned.

**c)** Demonstrate that the four versions give the same result. Test with inputs 3 and 3,000. (You will need to add a way to control the randomization.)

```r
# set seed to random value, run all games for 3 rolls
set.seed(100)
loopDiceGame(3)
```

```
[1] 0
```

```r
set.seed(100)
vectorizedDiceGame(3)
```

```
[1] 0
```

```r
set.seed(100)
tableDiceGame(3)
```

```
[1] 0
```

```r
set.seed(100)
applyDiceGame(3)
```

```
[1] 0
```

```r
# now set seed to random value, run all games for 3000 rolls
set.seed(100)
loopDiceGame(3000)
```

```
[1] 2062
```

```r
set.seed(100)
vectorizedDiceGame(3000)
```

```
[1] 2062
```

```r
set.seed(100)
tableDiceGame(3000)
```

```
[1] 2062
```

```r
set.seed(100)
applyDiceGame(3000)
```

```
[1] 2062
```

As we can see, all of the outputs match, which ensures the functions work correctly.

**d)** Use the microbenchmark package to clearly demonstrate the speed of the implementations. Compare performance with a low input (1,000) and a large input (100,000). Discuss the results.

```r
# load in microbenchmark
library(microbenchmark)

# use the microbenchmark function to run this experiment for low input 1000
smallBenchmark <- microbenchmark(
  loop = loopDiceGame(1000), # 1000 rolls of loop dice game
  vectorized = vectorizedDiceGame(1000), # 1000 rolls of vectorized dice game
  table = tableDiceGame(1000), # 1000 rolls of table dice game
  vapply = applyDiceGame(1000), # 1000 rolls of apply dice game
  times = 100 #repeat this process 100 times
)
```

```
Warning in microbenchmark(loop = loopDiceGame(1000), vectorized =
vectorizedDiceGame(1000), : less accurate nanosecond times to avoid potential
integer overflows
```

```
print(summary(smallBenchmark))
```

```
        expr      min        lq       mean   median        uq      max neval
1       loop 1764.681 1906.6230 2019.00974 1944.712 2014.4735 2897.921   100
2 vectorized   26.199   28.3515   30.44906   29.561   30.7295   54.325   100
3      table   64.370   70.3150   78.42070   73.923   83.3530  154.365   100
4     vapply  235.053  255.0405  297.67271  264.983  276.9960 3298.286   100
```

```
# use the microbenchmark function to run this experiment for large input 100000

largeBenchmark <- microbenchmark(
  loop = loopDiceGame(100000), # 100000 rolls of loop dice game
  vectorized = vectorizedDiceGame(100000), # 100000 rolls of vectorized dice game
  table = tableDiceGame(100000), # 100000 rolls of table dice game
  vapply = applyDiceGame(100000), # 100000 rolls of apply dice game
  times = 10  #repeat this process only 10 times (due to large sample sizes)
)

print(summary(largeBenchmark))
```

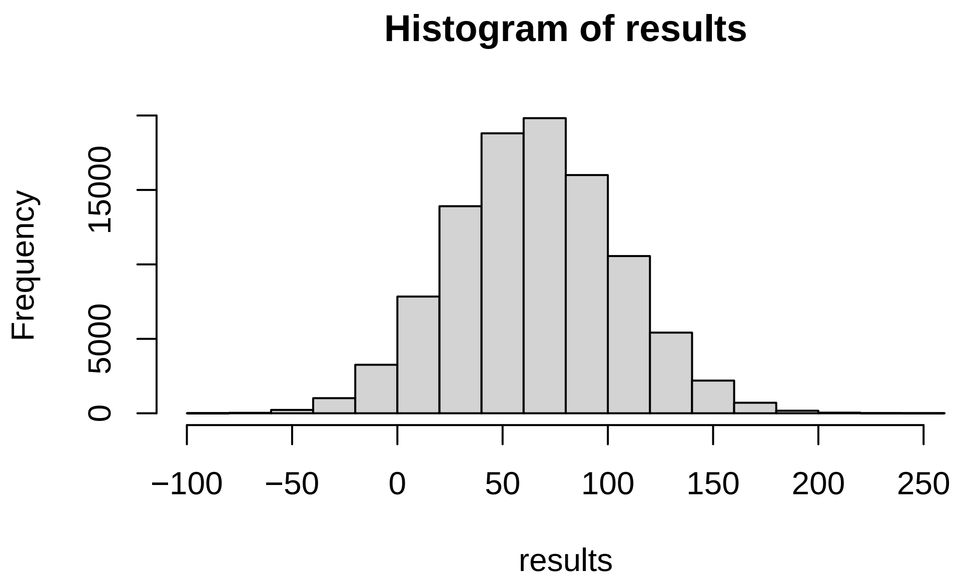
```
        expr        min         lq       mean     median         uq        max
1       loop 192.463758 193.169901 194.768880 194.280201 196.376224 199.096410
2 vectorized   2.552332   2.574923   2.594960   2.595607   2.618219   2.640236
3      table   4.695771   4.704012   4.776279   4.745688   4.789333   5.057637
4     vapply  26.047505  26.203961  28.434935  27.174124  27.320719  42.925278
  neval
1    10
2    10
3    10
4    10
```

As we can see, the outputs for the vectorized dice game performed significantly faster than all other functions examined. The median scores of vectorized were 2.65 and 29.9 milliseconds, and were nearly double the speed of the next fastest times. Next, table boasted impressive times of 4 and 66 milliseconds. After that, vapply succeeded in times of 28 and 271 milliseconds- significantly slower than the functions examined before. In dead last, the loop function performed in 223 and 2204 milliseconds, significantly slower than any other function.

**e)** Do you think this is a fair game? Defend your decision with evidence based upon a Monte Carlo simulation.

```r
#' Monte Carlo Simulation of a Dice Game
#'
#' This function runs a Monte Carlo simulation of a dice game, simulating
#' multiple rounds of dice rolls.
#'
#' It performs "nsims" simulations, each with "nrolls" dice rolls, using the
#' "vectorizedDiceGame" function for each simulation.
#'
#' @param nrolls The number of dice rolls in each simulation (an integer).
#' @param nsims The number of simulations to perform (an integer).
#'
#' @return A numeric vector where each element represents the result of one
#' simulation (total winnings or losses).
#' @export
#'
#' @examples
#' # Perform 10 simulations with 20 rolls in each
#' monteCarloDice(20, 10)
#'
#' # Perform 100 simulations with 50 rolls in each
#' monteCarloDice(50, 100)
#'
monteCarloDice <- function(nrolls, nsims){
  # create an empty vector to store results in
  results <- vector()

  # loop through, simulating each game
  for(i in 1:nsims){
    # use vectorized dice game for simulation since it is the fastest
    # storing results in the results vector
    results[i] <- vectorizedDiceGame(nrolls)
  }
  # after simulation return the vector
  return(results)
}

#simulate 100 rolls with 100000 experiments
results <- monteCarloDice(100,100000)

#plot results
```

```r
hist(results)
```



```r
# generate a 5 number summary of resulting distribution above
summary(results)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 -98.00   40.00   66.00   66.47   92.00  242.00 
```

Based on the outcome of the Monte Carlo Experiment, I would say that this is not a fair game (meaning not a zero-sum game) because the median of the distribution lies at 66. This means on average, you can expect to win about $66.72 if you roll the dice 100 times. If you were playing this game, you would gladly accept because of the high probability of making money. However, if you were the casino or institution hosting this game, you would likely lose money due to the high average payoffs.

## Problem 2 - Linear Regression

a) The names of the variables in this data are way too long. Rename the columns of the data to more reasonable lengths.

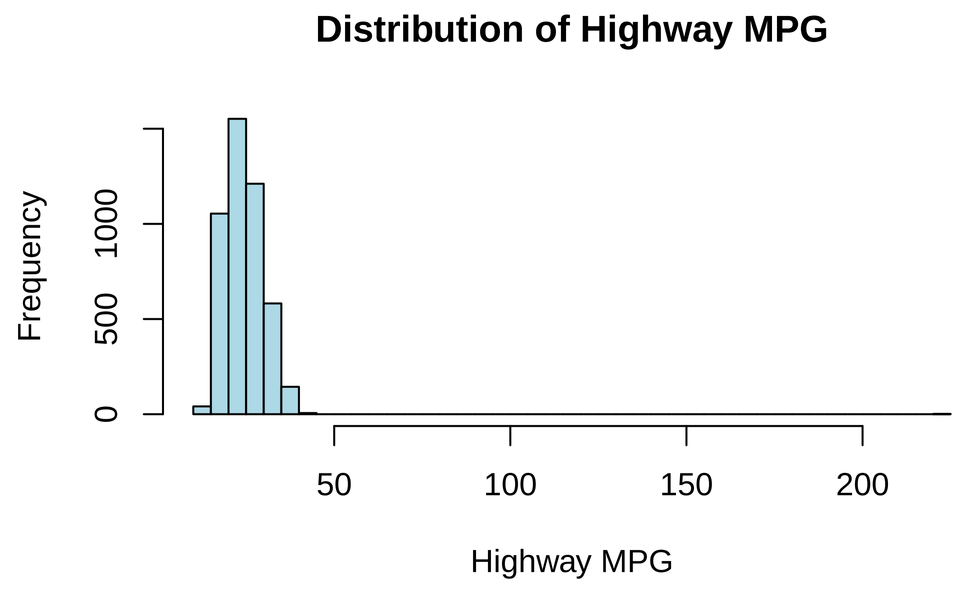
```r
# Load data
df <- read.csv('/Users/garrettpinkston/Desktop/Michigan/STAT506/Data/cars.csv')

# Write descriptions of all corresponding columns
descriptors <- c("Height","Length","Width","Driveline","EngineType",
                 "Hybrid","Gears","Transmission","CityMPG","FuelType",
                 "HighwayMPG", "Classification", "ID", "Make", "Model",
                 "Year", "Horsepower", "Torque")

#replace column names with descriptions
colnames(df) <- descriptors
```

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

```r
# Keep all entries of just Gasoline
df <- df[df$FuelType == 'Gasoline',]
```

c) Examine the distribution of highway gas mileage. Consider whether a transformation could be used. If so, generate the transformed variable and use this variable going forward. If not, provide a short justification.

```r
# use histogram function and only look at HighwayMPG variable
hist(df$HighwayMPG, main = "Distribution of Highway MPG",
     xlab = "Highway MPG", col = "lightblue", breaks = 50)
```

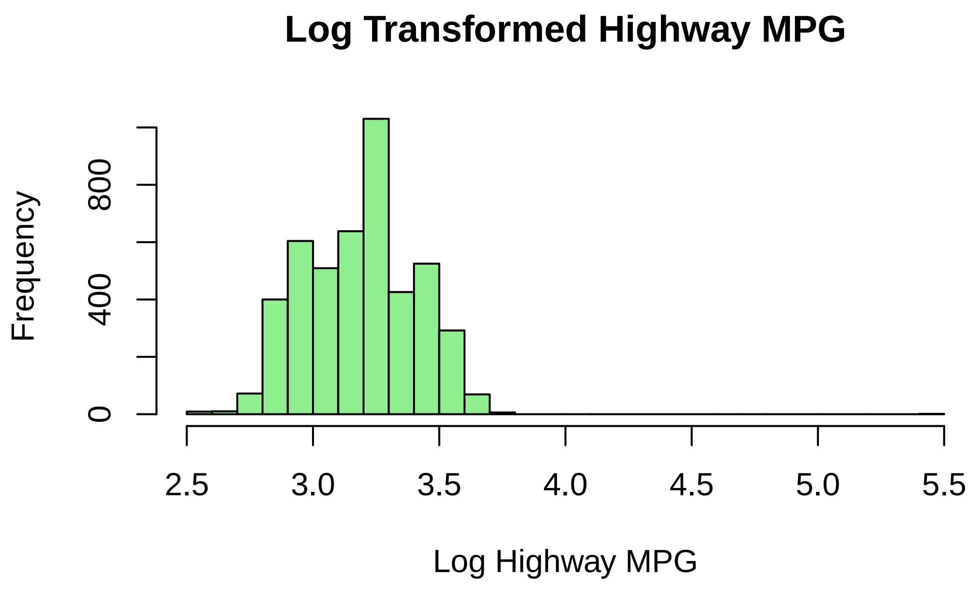


```r
summary(df$HighwayMPG)
```

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

Based on the histogram above, the median of the distribution is approximately 25 (residing the tallest bar). Because there are more visible bars to the right, as well as an invisible bar (likely due to a large outlier), we can deduce that this data is right-skewed. We can apply a log transformation to mitigate skewness, since it will reduce the impact of larger values and spread out smaller values.

```r
df$HighwayMPG <- log(df$HighwayMPG)

hist(df$HighwayMPG, main = "Log Transformed Highway MPG",
     xlab = "Log Highway MPG", col = "lightgreen", breaks = 30)
```



d) Fit a linear regression model predicting MPG on the highway. The predictor of interest is torque. Control for:

The horsepower of the engine All three dimensions of the car The year the car was released, as a categorical variable.

```
# Fit a linear model with torque as variable of interest
model <- lm(HighwayMPG ~ Torque + Horsepower + Length
            + Width + Height + Year, data = df)

# output a summary of model
summary(model)
```

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

Residuals:
     Min       1Q   Median       3Q      Max
-0.54207 -0.09294 -0.00562  0.09919  2.41249

Coefficients:
```

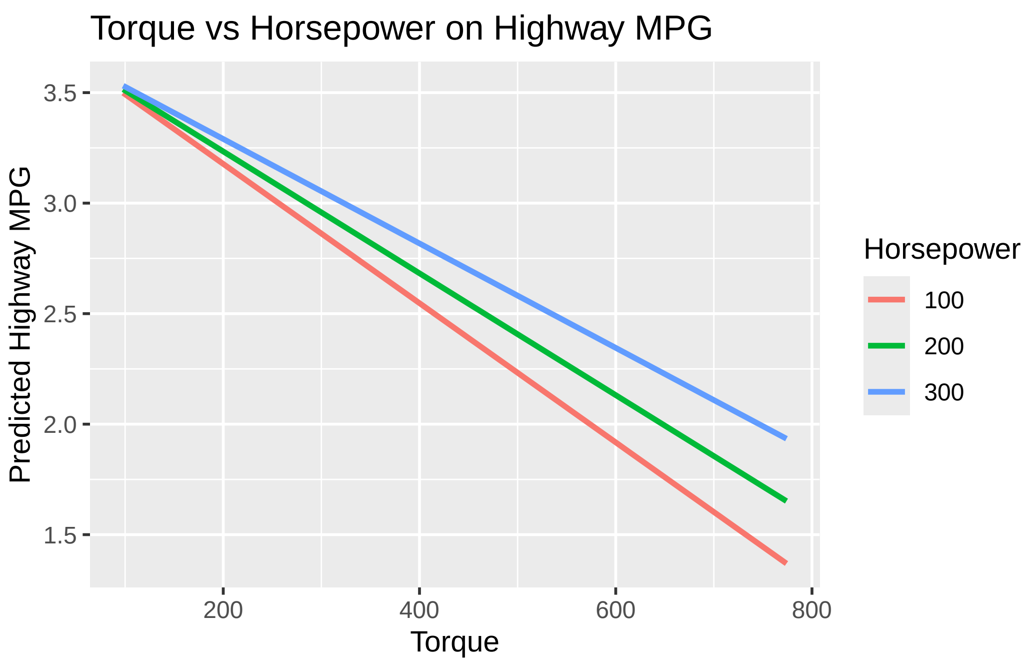
```
              Estimate Std. Error t value Pr(>|t|)    
(Intercept) -5.340e+01  5.334e+00 -10.010  < 2e-16 ***
Torque      -2.306e-03  6.758e-05 -34.119  < 2e-16 ***
Horsepower   9.329e-04  6.988e-05  13.348  < 2e-16 ***
Length       2.990e-05  2.711e-05   1.103 0.270066    
Width       -9.178e-05  2.775e-05  -3.307 0.000949 ***
Height       4.020e-04  3.448e-05  11.657  < 2e-16 ***
Year         2.830e-02  2.653e-03  10.667  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1413 on 4584 degrees of freedom
Multiple R-squared:  0.5625,    Adjusted R-squared:  0.5619 
F-statistic: 982.1 on 6 and 4584 DF,  p-value: < 2.2e-16
```

Briefly discuss the estimated relationship between torque and highway MPG. Be precise about the interpretation of the estimated coefficient.

When holding all other variables constant, for every additional unit of torque the highway MPG will decrease by about 0.0023 units. Because this effect is very small, small changes in torque may have minimal effects on the highway fuel economy. The extremely low p-value (($p < 2e-16$)) indicates statistical significance, meaning that torque is related to highway MPG within this dataset.

e) It seems reasonable that there may be an interaction between torque and horsepower. Refit the model (with lm) and generate an interaction plot, showing how the relationship between torque and MPG changes as horsepower changes. Choose reasonable values of torque, and show lines for three different reasonable values of horsepower.

```r
# Model Torque vs Horsepower, specifying interaction between vars using *
model_TvH <- lm(HighwayMPG ~ Torque * Horsepower
                + Length + Width + Height + Year, data = df)

# Find reasonable values of torque for prediction
torque_values <- seq(min(df$Torque), max(df$Torque), length.out = 100)

# Try just 3 reasonable differing horsepower values
horsepower_values <- c(100, 200, 300)

#create all possible combinations of model inputs using expand.grid
#while holding control variables constant
prediction_data <- expand.grid(Torque = torque_values,
                               Horsepower = horsepower_values,
                              Length = mean(df$Length), #constant
                              Width = mean(df$Width), #constant
                              Height = mean(df$Height), #constant
                              Year = (df$Year)[1]) # Use a fixed year

# Use predict function to predict MPG for each combination of torque and horsepower
prediction_data$PredictedMPG <- predict(model_TvH, newdata = prediction_data)

# Load ggplot2 for plotting
library(ggplot2)

# Create interaction plot
ggplot(prediction_data, aes(x = Torque, y = PredictedMPG,
                           color = as.factor(Horsepower))) +
  geom_line(linewidth = 1) +
  labs(title = "Torque vs Horsepower on Highway MPG",
       x = "Torque",
       y = "Predicted Highway MPG",
       color = "Horsepower")
```



f) Calculate $\hat{\beta}$ from d. manually (without using lm) by first creating a proper design matrix, then using matrix algebra to estimate $\beta$. Confirm that you get the same result as lm did prior.

```r
# Design matrix with intercept
X <- model.matrix(~ Torque + Horsepower + Length + Width +
                    Height + Year, data = df)

 # Create response vector
y <- df$HighwayMPG

# Calculate beta manually using matrix algebra
beta_hat_manual <- solve(t(X) %*% X) %*% t(X) %*% y

# Compare with lm coefficients
model_lm <- lm(HighwayMPG ~ Torque + Horsepower + Length
               + Width + Height + Year, data = df)
beta_hat_lm <- coef(model_lm)

# Display manual beta_hat and lm beta_hat
print(beta_hat_manual)
```

```
                     [,1]
(Intercept) -5.339592e+01
Torque      -2.305897e-03
Horsepower   9.328547e-04
Length       2.989852e-05
Width       -9.177856e-05
Height       4.019629e-04
Year         2.829937e-02
```

```r
print(beta_hat_lm)
```

```
  (Intercept)        Torque    Horsepower        Length         Width
-5.339592e+01 -2.305897e-03  9.328547e-04  2.989852e-05 -9.177856e-05
       Height          Year
 4.019629e-04  2.829937e-02
```

As we can see, the computed beta_hat_manual matches the output of the beta_hat_ml function.