STATS 506 Problem Set #2

Haiming Li

Dice Game

a. Here are different implementations of the function:

```
#' simulation version 1: loop implementation
#' @param n number of plays to make
#' @param seed seed to control random
#' @return final payoff
play_dice1 <- function(n, seed=NULL) {
  # input sanitation
  if (n < 1) {
    return(0)
  }

  res <- -2 * n
  set.seed(seed)
  rolls <- sample(1:6, n, replace=TRUE)
  for (roll in rolls) {
    if (roll == 3 | roll == 5) {
      res <- res + 2 * roll
    }
  }
  return(res)
}

#' simulation version 2: vectorized implementation
#' @param n number of plays to make
#' @param seed seed to control random
#' @return final payoff
play_dice2 <- function(n, seed=NULL) {
  # input sanitation
  if (n < 1) {
```

```r
    return(0)
  }

  set.seed(seed)
  rolls <- sample(1:6, n, replace=TRUE)
  # replace payoff of all loss with 0
  rolls[which(!(rolls == 3 | rolls == 5))] <- 0
  return(2*sum(rolls) - 2*n)
}

#' simulation version 3: table implementation
#' @param n number of plays to make
#' @param seed seed to control random
#' @return final payoff
play_dice3 <- function(n, seed=NULL) {
  # input sanitation
  if (n < 1) {
    return(0)
  }

  # construct table with factor (predetermined levels)
  set.seed(seed)
  rolls <- table(factor(sample(1:6, n, replace=TRUE), 1:6))
  # calculate final payoff & remove name of vector
  res <- 2*(rolls[3]*3 + rolls[5]*5) - 2*n
  names(res) <- NULL
  return(res)
}

#' simulation version 4: table implementation
#' @param n number of plays to make
#' @param seed seed to control random
#' @return final payoff
play_dice4 <- function(n, seed=NULL) {
  # input sanitation
  if (n < 1) {
    return(0)
  }

  set.seed(seed)
  rolls <- sample(1:6, n, replace=TRUE)
  # apply a function that return the winning value of a given roll
```

```r
  res <- vapply(rolls, function(roll) {
    if (roll == 3 | roll == 5) {
      return(2 * roll)
    }
    return(0)
  }, numeric(1))
  return(sum(res) - 2*n)
}
```

b. Here are some demonstrations:

```r
cat("Functions with input n=3\n")
cat("play_dice1:", play_dice1(3), '\n')
cat("play_dice2:", play_dice2(3), '\n')
cat("play_dice3:", play_dice3(3), '\n')
cat("play_dice4:", play_dice4(3), '\n\n')
cat("Functions with input n=3000\n")
cat("play_dice1:", play_dice1(3000), '\n')
cat("play_dice2:", play_dice2(3000), '\n')
cat("play_dice3:", play_dice3(3000), '\n')
cat("play_dice4:", play_dice4(3000), '\n')
```

```
Functions with input n=3
play_dice1: 16
play_dice2: 20
play_dice3: 0
play_dice4: -6

Functions with input n=3000
play_dice1: 2244
play_dice2: 2072
play_dice3: 1864
play_dice4: 2504
```

c. Here are some demonstrations with seed 123:

```r
cat("Functions with input n=3\n")
cat("play_dice1:", play_dice1(3, 123), '\n')
cat("play_dice2:", play_dice2(3, 123), '\n')
cat("play_dice3:", play_dice3(3, 123), '\n')
cat("play_dice4:", play_dice4(3, 123), '\n\n')
cat("Functions with input n=3000\n")
```

```r
cat("play_dice1:", play_dice1(3000, 123), '\n')
cat("play_dice2:", play_dice2(3000, 123), '\n')
cat("play_dice3:", play_dice3(3000, 123), '\n')
cat("play_dice4:", play_dice4(3000, 123), '\n')
```

```
Functions with input n=3
play_dice1: 6
play_dice2: 6
play_dice3: 6
play_dice4: 6

Functions with input n=3000
play_dice1: 2174
play_dice2: 2174
play_dice3: 2174
play_dice4: 2174
```

d. Here are speed comparisons. It seems that the implementation with apply is the slowest, the explicit loop implementation is the second slowest. This make sense because apply is loop hiding, and by passing in a function it creates extra overhead compared to explicit loop. The vectorized implementation is the fastest, and the table implementation is the second fastest. This also makes sense, it both of them leverage the speed of C, while the vectorized implementation have less part that need to run in R.

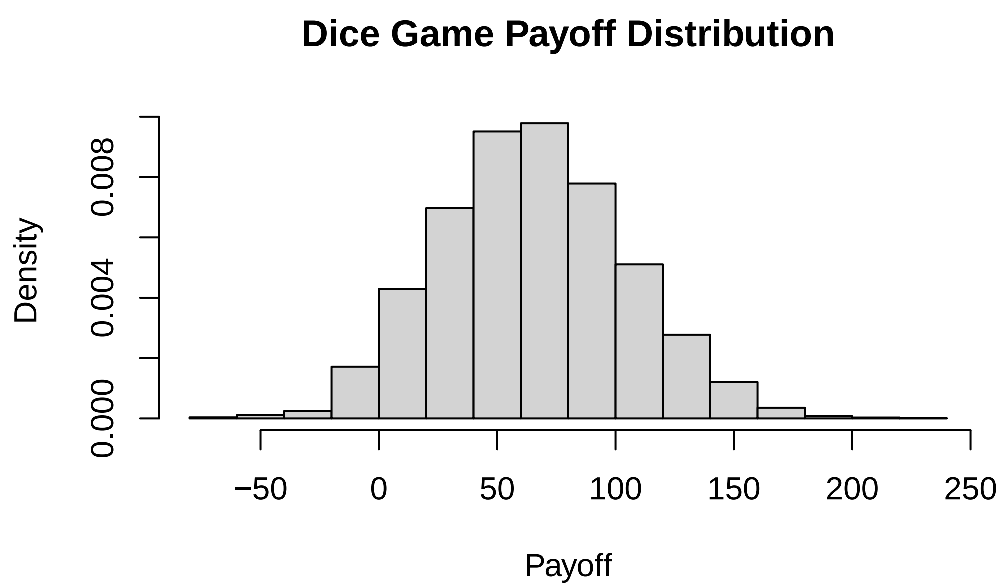
```r
library(microbenchmark)

microbenchmark(
  play_dice1 = play_dice1(1000, 123),
  play_dice2 = play_dice2(1000, 123),
  play_dice3 = play_dice3(1000, 123),
  play_dice4 = play_dice4(1000, 123)
)

microbenchmark(
  play_dice1 = play_dice1(100000, 123),
  play_dice2 = play_dice2(100000, 123),
  play_dice3 = play_dice3(100000, 123),
  play_dice4 = play_dice4(100000, 123)
)
```

```
Unit: microseconds
```

```
        expr     min        lq      mean   median       uq      max neval
 play_dice1  89.995  97.2725 103.14739 100.9010 103.8735  154.775   100
 play_dice2  34.563  38.2120  41.41164  39.0935  41.6560   91.799   100
 play_dice3  75.030  80.9955  93.60833  84.8905  91.7375  214.266   100
 play_dice4 296.389 328.8815 392.35565 340.2590 357.3560 2432.612   100
Unit: milliseconds
        expr       min        lq      mean    median        uq       max neval
 play_dice1  8.442105  8.702065  9.460167  9.294680  9.723990 14.340488   100
 play_dice2  3.212883  3.323071  3.496758  3.373849  3.502651  6.723426   100
 play_dice3  4.827463  5.033508  5.278128  5.101261  5.362431  7.606443   100
 play_dice4 30.327454 31.480968 34.600066 32.434095 34.618924 72.889595   100
```

e. It looks like the game is not fair, as the histogram is not centered around 0. This makes sense, as the expected payoff for each toss is $\frac{6+10}{6} - 2 = \frac{2}{3}$. The player is expected to gain.

```r
res <- c()
for (i in 1:10000) {
  res <- append(res, play_dice2(100))
}
hist(res, main='Dice Game Payoff Distribution', xlab='Payoff', freq=FALSE)
```



Linear Regression

a. Here's the dataset with shortened column name.

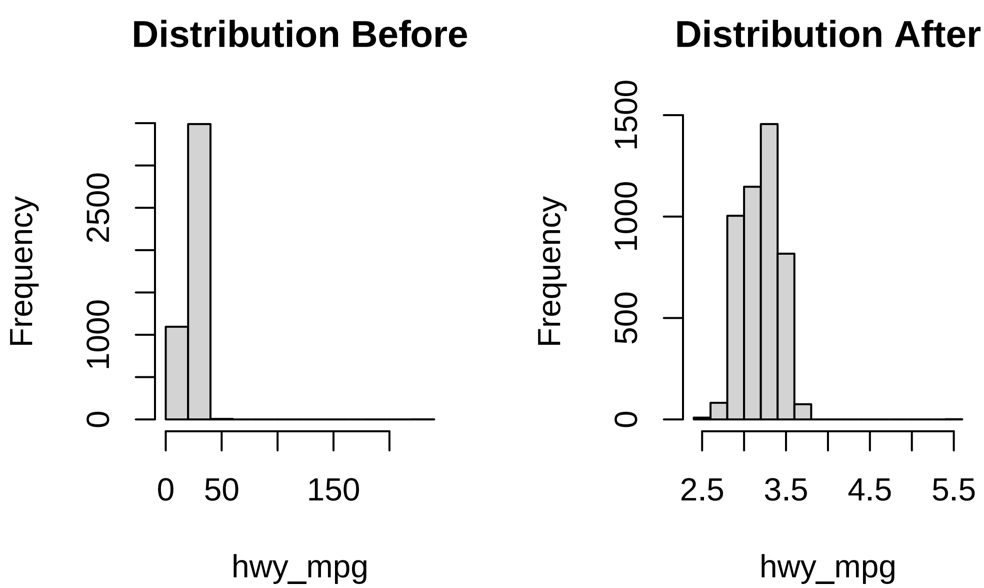
```
cars <- read.csv('cars.csv')
names(cars) <- c(
  "height", "length", "width", "driveline", "engine_type", "hybrid",
  "gears_cnt", "transmission", "city_mpg", "fuel_type", "hwy_mpg",
  "class", "id", "make", "model", "year", "horsepower", "torque"
)
```

b. Here's the filtered dataset.

```
cars <- subset(cars, fuel_type == 'Gasoline')
```

c. There's an extreme value in highway mpg. Without removing it, the best course of action is to normalize it via box-cox transformation. This would specifically benefit linear regression.

```
par(mfrow= c(1,2))
hist(cars$hwy_mpg, main='Distribution Before', xlab='hwy_mpg')
cars$hwy_mpg <- log(cars$hwy_mpg)
hist(cars$hwy_mpg, main='Distribution After', xlab='hwy_mpg')
```



d. It seems that, while holding all else constant, a unit increase in torque would corresponds to 0.002294 decrease in highway mpg on average.

```
cars$year <- as.factor(cars$year)
model_fit <- lm(hwy_mpg ~ torque + horsepower + height +
                  length + width + year,
                data = cars)
summary(model_fit)
```

```
Call:
lm(formula = hwy_mpg ~ torque + horsepower + height + length +
    width + 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 **
year2010    -2.181e-02  2.076e-02  -1.051  0.29342
year2011    -2.430e-03  2.072e-02  -0.117  0.90665
year2012     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
```

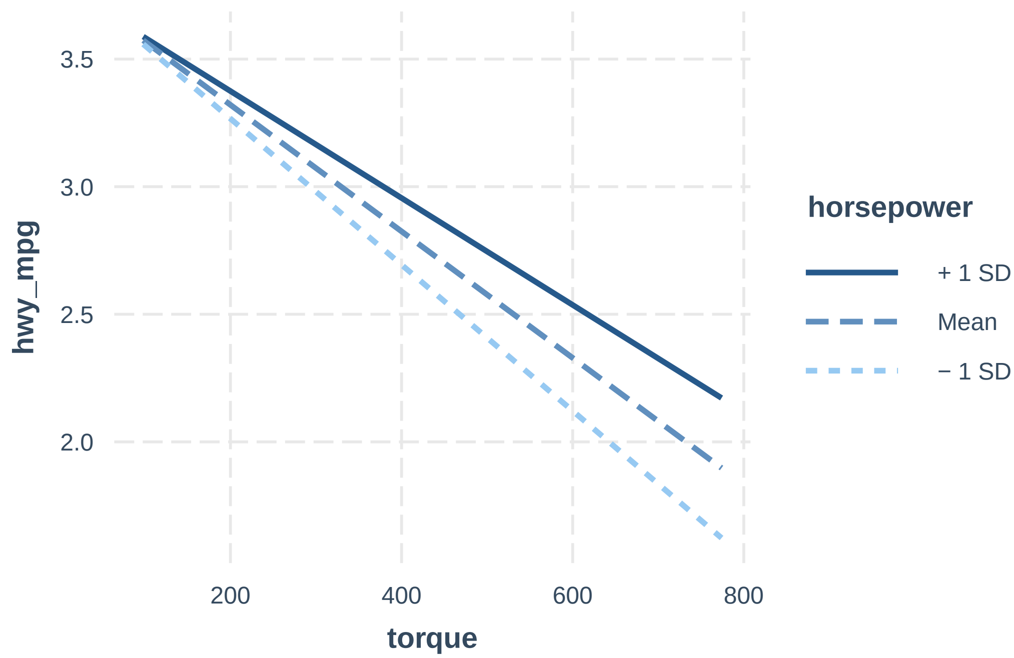
e. As shown, year 2011 have most data. Thus, I will use 2011 in the interaction plot.

```
table(cars$year)
```

```
2009 2010 2011 2012
  48 1633 1794 1116
```

Here's the interaction plot with year 2011.

```r
library(interactions)
model_fit <- lm(hwy_mpg ~ torque * horsepower + height +
                  length + width + year,
                data = cars)
interact_plot(model_fit, pred = torque, modx = horsepower,
              at = list(year = factor(2011, levels(cars$year))))
```



f. For OLS, we have $\hat{\beta} = (X^T X)^{-1} X^T Y$.

```r
X <- model.matrix(hwy_mpg ~ torque + horsepower + height +
                    length + width + year,
                  data = cars)
Y <- cars$hwy_mpg
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
year2010    -2.181247e-02
year2011    -2.430359e-03
year2012     4.011528e-02
```

Citation & Github Link

- Use of interaction_plot
- GitHub Repo of this Pset