STATS506HW2

```r
library(microbenchmark)
```

Problem 1.

```r
random_walk1 <- function(n, rand){
  position <- 0L

  for (i in 1:n){
    base_prob <- rand[2*i - 1]
    scale_prob <- rand[2*i]
    if (base_prob < 0.5) {
      if (scale_prob < 0.95) {
        step_size <- 1L
      }else {
        step_size <- 10L
      }
    }else {
      if (scale_prob < 0.8) {
        step_size <- -1L
      }else {
        step_size <- -3L
      }
    }
    position <- position + step_size
  }
  position
}

random_walk2 <- function(n, rand){
  base_prob <- rand[seq(1, 2*n, 2)]
  scale_prob <- rand[seq(2, 2*n, 2)]
```

```r
  step_size <- ifelse(base_prob < 0.5, ifelse(scale_prob < 0.95, 1L, 10L),
                      ifelse(scale_prob < 0.8, -1L, -3L))

  sum(step_size)
}

random_walk3 <- function(n, rand) {
  step_size <- sapply(1:n, function(i) {
    base_prob <- rand[2*i - 1]
    scale_prob <- rand[2*i]
    step_size <- ifelse(base_prob < 0.5, ifelse(scale_prob < 0.95, 1L, 10L),
                      ifelse(scale_prob < 0.8, -1L, -3L))
  })
  sum(step_size)
}

random_walk1(10, runif(2*10))
```

```
[1] -4
```

```r
random_walk2(10, runif(2*10))
```

```
[1] 0
```

```r
random_walk3(10, runif(2*10))
```

```
[1] -6
```

```r
random_walk1(1000, runif(2*1000))
```

```
[1] 34
```

```r
random_walk2(1000, runif(2*1000))
```

```
[1] 76
```

```r
random_walk3(1000, runif(2*1000))
```

```
[1] 3
```

(b)

```r
rand_prob <- function(n) {
  set.seed(123)
  runif(n * 2)
}

random_walk1(10, rand_prob(10))
```

```
[1] 7
```

```r
random_walk2(10, rand_prob(10))
```

```
[1] 7
```

```r
random_walk3(10, rand_prob(10))
```

```
[1] 7
```

```r
random_walk1(1000, rand_prob(1000))
```

```
[1] 78
```

```r
random_walk2(1000, rand_prob(1000))
```

```
[1] 78
```

```r
random_walk3(1000, rand_prob(1000))
```

```
[1] 78
```

All three functions work and we obtained the same result by using the same random sample for each function to ensure a consistent comparison.

(c)

```r
rand1000 <- rand_prob(1000)
rand100000 <- rand_prob(100000)

microbenchmark(
  rand_walk_loop_s = random_walk1(1000, runif(2*1000)),
  rand_walk_vector_s = random_walk2(1000, runif(2*1000)),
  rand_walk_apply_s = random_walk3(1000, runif(2*1000)),

  rand_walk_loop_l = random_walk1(100000, runif(2*100000)),
  rand_walk_vector_l = random_walk2(100000, runif(2*100000)),
  rand_walk_apply_l = random_walk3(100000, runif(2*100000))
)
```

```
Unit: microseconds
               expr      min        lq       mean    median        uq      max
   rand_walk_loop_s     70.5     77.40     82.429     81.10     85.55    128.0
 rand_walk_vector_s     91.7    121.05    158.781    154.30    190.80    246.8
  rand_walk_apply_s   1666.6   1831.65   2700.303   2660.45   3084.00  11446.0
   rand_walk_loop_l   6789.9   6961.60   7127.625   7068.40   7258.40   8116.3
 rand_walk_vector_l   4623.6   4930.15   6355.101   6187.85   7571.70   9667.1
  rand_walk_apply_l 172132.2 215088.80 246834.612 250435.40 275984.00 360668.0
 neval
   100
   100
   100
   100
   100
   100
```

The microbenchmark results show that for small input sizes (n = 1,000), the loop implementation actually runs faster than the vectorized version, while the apply version is the slowest. For large input sizes (n = 100,000), both the loop and vectorized versions perform similarly, with the loop being slightly faster in this case. Whereas the apply version remains the slowest.

(d)

```r
set.seed(123)
random_walk_0 <- function(n, rep = 100000) {
  ends <- replicate(rep, random_walk1(n, runif(2*n)))
  mean(ends == 0L)
}
```

```r
random_walk_0(10)
```

```
[1] 0.1323
```

```r
random_walk_0(100)
```

```
[1] 0.01921
```

```r
random_walk_0(1000)
```

```
[1] 0.00575
```

Using Monte Carlo simulation with 100,000 repetitions, we estimated the probability that the random walk ends at 0 fordifferent numbers of steps. The results show that the probability is approximately 13% for 10 steps, 1.9% for 100 steps, and 0.58% for 1000 steps. These values show the probability of ending at 0 decreases rapidly as the number of steps increases.

Problem 2.

```r
num_car <- function(rep = 10000){
  car <- replicate(rep, {
    c(rpois(7, 1),
      rpois(8, 8),
      rpois(7, 12),
      rnorm(1, mean = 12, sd = sqrt(12)),
      rnorm(1, mean = 12, sd = sqrt(12))
      )
  })
  total <- colSums(car)
  mean(total)
}

set.seed(123)
num_car()
```

```
[1] 179.2076
```

The Monte Carlo Simulation estimates that, on average, tabout 179 cars pass the intersection per day.

Problem 3.

(a)

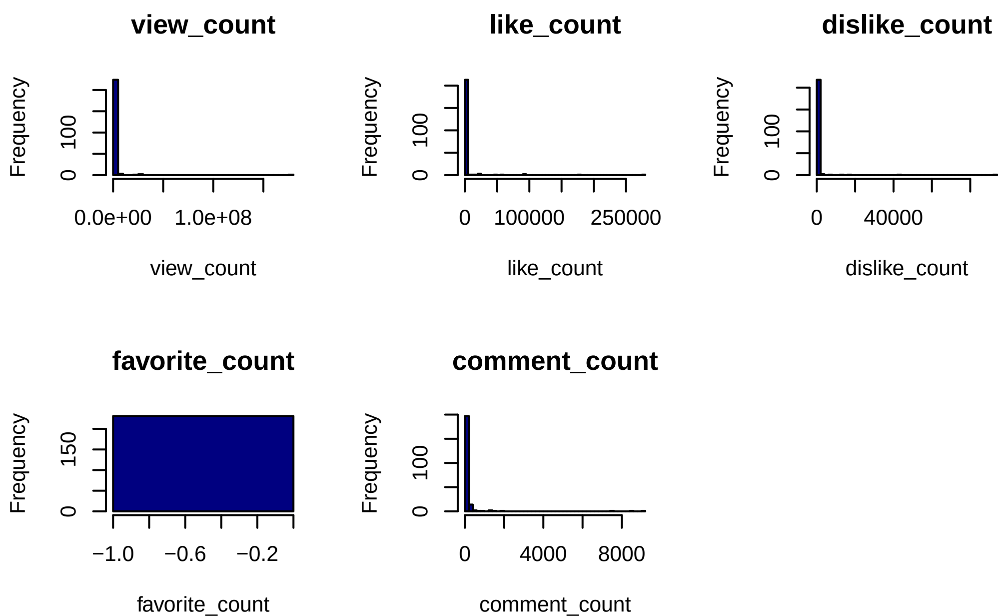
```
youtube <- read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/dat
column_id <- c("brand", "superbowl_ads_dot_com_url", "youtube_url", "id", "published_at", "cl

youtube <- youtube[, !(names(youtube) %in% column_id)]

dim(youtube)
```

```
[1] 247  19
```

The dimension of the data is 247 x 19 after removing the columns.

(b)

```
det_columns <- c("view_count", "like_count", "dislike_count", "favorite_count", "comment_cou
par(mfrow = c(2,3))
for (y in det_columns) {
  hist(youtube[[y]], main = y, xlab = y, col = "navyblue", breaks = 50)
}
youtube$log_view_count <- log1p(youtube$view_count)
youtube$log_like_count <- log1p(youtube$like_count)
youtube$log_dislike_count <- log1p(youtube$dislike_count)
youtube$log_comment_count <- log1p(youtube$comment_count)
```



View_count, like_count, dislike_count, and comment_count are heavily right-skewed and require a log transformation before being used, in order to approximate a normal distribution. Favorite_count is not appropriate to use, since practically all of its values are zero.

(c)

```
predictors <- c("funny","show_product_quickly","patriotic","celebrity",
                "danger","animals","use_sex")
response <- c("view_count","like_count","dislike_count","comment_count")

for (y in response) {
  df <- youtube[ , c(y, predictors, "year")]
  lm_temp <- as.formula(paste(y, "~", paste(c(predictors, "year"), collapse = " + ")))
  yt_lm <- lm(lm_temp, data = df)

  cat("\n------------------------------------------------------------------\n")
  cat("Model:", deparse(lm_temp), "\n")
  print(coef(yt_lm))
}
```

```
------------------------------------------------------------------
Model: view_count ~ funny + show_product_quickly + patriotic + celebrity +      danger + ani
```

```
        (Intercept)                funnyTRUE show_product_quicklyTRUE
       -310381925.3                1339891.1                 607353.8
        patrioticTRUE            celebrityTRUE               dangerTRUE
           447389.7               -1431593.3                -534695.4
        animalsTRUE               use_sexTRUE                     year
         -1589545.9               -1257640.7                 155152.7

----------------------------------------------------------------
Model: like_count ~ funny + show_product_quickly + patriotic + celebrity +      danger + anin
        (Intercept)                funnyTRUE show_product_quicklyTRUE
      -1.041191e+06             1.635838e+03             4.842927e+02
      patrioticTRUE            celebrityTRUE               dangerTRUE
       3.114939e+03            -6.015852e+01             1.557979e+03
        animalsTRUE              use_sexTRUE                     year
      -2.079466e+03            -3.498543e+03             5.196672e+02

----------------------------------------------------------------
Model: dislike_count ~ funny + show_product_quickly + patriotic + celebrity +      danger + a
        (Intercept)                funnyTRUE show_product_quicklyTRUE
      -179968.47451                -26.89093                557.56785
      patrioticTRUE            celebrityTRUE               dangerTRUE
         -291.90185               -992.40560                428.89811
        animalsTRUE              use_sexTRUE                     year
         -415.03866               -695.03426                 90.02907

----------------------------------------------------------------
Model: comment_count ~ funny + show_product_quickly + patriotic + celebrity +      danger + a
        (Intercept)                funnyTRUE show_product_quicklyTRUE
      -54132.529556               -27.159996              -166.758679
      patrioticTRUE            celebrityTRUE               dangerTRUE
         400.730202                -2.282163               241.663208
        animalsTRUE              use_sexTRUE                     year
        -106.706558               -68.605926                27.049129
```

Based on the summary of the linear regression models above, most ad features and year are not significantly related to view, like, or dislike counts. In the model for comment counts, however, ads with patriotic themes generate more comments, and newer ads also tend to receive more comments over time, as their p-values are below 0.05. This indicates that these predictors are statistically significant.

(d)

```r
df <- youtube[ , c("view_count", predictors, "year")]
df <- df[complete.cases(df), ]

x <- model.matrix(~ funny + show_product_quickly + patriotic + celebrity + danger + animals +
y <- df$view_count

beta <- solve(t(x) %*% x) %*% t(x) %*% y

view_count_lm <- lm(view_count ~ funny + show_product_quickly + patriotic + celebrity + dange

beta
```

```
                                [,1]
(Intercept)             -310381925.3
funnyTRUE                  1339891.1
show_product_quicklyTRUE    607353.8
patrioticTRUE               447389.7
celebrityTRUE             -1431593.3
dangerTRUE                 -534695.4
animalsTRUE               -1589545.9
use_sexTRUE               -1257640.7
year                        155152.7
```

```r
coef(view_count_lm)
```

```
             (Intercept)                funnyTRUE show_product_quicklyTRUE
            -310381925.3                1339891.1                 607353.8
           patrioticTRUE            celebrityTRUE               dangerTRUE
                447389.7               -1431593.3                -534695.4
             animalsTRUE              use_sexTRUE                     year
              -1589545.9               -1257640.7                 155152.7
```

The manual OLS calculation produced the same coefficients as the lm function, confirming that both methods give the identical results.