STATS 506 HW 2

Calder Moore

STATS 506 HW 2

Problem 1 - Modified Random Walk

a) (with loop)

```r
#' Random walk with loops. 50% chance of moving +-1, if +1 is chosen then a 5%
#' chance of moving +10 rather than just +1, if -1 is chosen then a 20% chance of
#' moving -3 rather than just -1.
#' @param count how many steps the random walk will take
#' @param override user can set to TRUE to provide a custom set of steps for testing
#' @param test_vec user can provide the vector that will be used to ensure it provides a cer
#' @returns the final value once the steps are completed
random_walk1 <- function(count, override = FALSE, test_vec){
  walk <- 0
  if(override == FALSE){
    for(i in 1:count){
      #take the initial sample for the +-1
      sample_1 <- sample(c(-1, 1), 1, replace = TRUE, prob = c(0.5, 0.5))
      #conditions for the extra possible +10 or -3 move
      if(sample_1 == 1){
        sample_2 <- sample(c(1, 10), size = 1, replace = TRUE, prob = c(0.95, 0.05))
        walk <- walk + sample_2
      }
      else{
        sample_3 <- sample(c(-1, -3), size = 1, replace = TRUE, prob = c(0.8, 0.2))
        walk <- walk + sample_3
      }
    }
  }
```

```
  else{
    sample_1 <- test_vec
    for(i in 1:length(sample_1)){
      #conditions for the extra possible +10 or -3 move
      if(sample_1[i] == 1){
        walk <- walk + sample_1[i]
      }
      else{
        walk <- walk + sample_1[i]
      }
    }
  }
  return(walk)
}
```

a) (vectorized)

```
#' Random walk vectorized. Same task as above, but vectorized rather than with loops.
#' Does all steps and once and sums them.
#' @param count how many steps the random walk will take
#' @returns the final value once the steps are completed
random_walk2 <- function(count, override = FALSE, test_vec){
  #we can do all the steps at once, add them up, and add the total to walk.
  #Need to include all possible outcomes and their respective probabilities.
  #Replace needs to be true now since we aren't looping over the function each
  #time anymore and need to be sampling from each of the 4 possibilities this time.
  if(override == FALSE){
    all_steps <- sample(c(-1, 1, -3, 10),
                        size = count,
                        replace = TRUE,
                        prob = c(4/10, 19/40, 1/10, 1/40))

    walk <- sum(all_steps)
  }
  else{
    walk<- sum(test_vec)
  }
  return(walk)
}
```

a) (with apply)

```r
#' Random walk with apply. Use apply to sample the steps for the random walk.
#' @param count how many steps the random walk will take
#' @returns the final value once the steps are completed
random_walk3 <- function(count, override = FALSE, test_vec){
  if(override == FALSE){
    steps_mat <- as.matrix(c(-1, 1, -3, 10))
    all_steps <- apply(steps_mat,
                       2,
                       sample,
                       size = count,
                       replace = TRUE,
                       prob = c(4/10, 19/40, 1/10, 1/40))

    walk <- sum(all_steps)
  }
  else{
    walk <- sum(test_vec)
  }
  return(walk)
}
```

```r
#Test
random_walk1(10)
```

```
[1] -4
```

```r
random_walk2(10)
```

```
[1] 36
```

```r
random_walk3(10)
```

```
[1] 4
```

```r
random_walk1(1000)
```

```
[1] 97
```

```r
random_walk2(1000)
```

```
[1] -33
```

```r
random_walk3(1000)
```

```
[1] 86
```

b)

```r
set.seed(123)
test_10 <- sample(c(-1, 1, -3, 10),
                  size = 10,
                  replace = TRUE,
                  prob = c(4/10, 19/40, 1/10, 1/40))
test_1000 <- sample(c(-1, 1, -3, 10),
                    size = 1000,
                    replace = TRUE,
                    prob = c(4/10, 19/40, 1/10, 1/40))
random_walk1(10, override = TRUE, test_10)
```

```
[1] -8
```

```r
random_walk2(10, override = TRUE, test_10)
```

```
[1] -8
```

```r
random_walk3(10, override = TRUE, test_10)
```

```
[1] -8
```

```r
random_walk1(10, override = TRUE, test_1000)
```

```
[1] 49
```

```r
random_walk2(10, override = TRUE, test_1000)
```

```
[1] 49
```

```r
random_walk3(10, override = TRUE, test_1000)
```

```
[1] 49
```

c)

```r
library(microbenchmark)

x <- 1000
y <- 100000
microbenchmark(random_walk1(x))
```

```
Unit: milliseconds
            expr      min       lq     mean   median       uq     max neval
 random_walk1(x) 8.479501 8.838001 9.421595 9.173301 9.904601 13.5358   100
```

```r
microbenchmark(random_walk2(x))
```

```
Unit: microseconds
            expr    min     lq     mean median     uq    max neval
 random_walk2(x) 20.101 21.001 24.09003 21.501 26.451 43.901   100
```

```r
microbenchmark(random_walk3(x))
```

```
Unit: microseconds
            expr    min     lq     mean median     uq     max neval
 random_walk3(x) 53.602 54.551 60.63696 54.951 56.001 176.101   100
```

```r
microbenchmark(random_walk1(y))
```

```
Unit: milliseconds
            expr      min       lq     mean  median       uq      max neval
 random_walk1(y) 920.9954 932.4283 947.7162 938.377 950.7217 1156.004   100
```

```r
microbenchmark(random_walk2(y))
```

```
Unit: milliseconds
            expr      min       lq     mean   median       uq      max neval
 random_walk2(y) 1.406201 1.515351 1.626478 1.574102 1.601901 4.506901   100
```

```r
microbenchmark(random_walk3(y))
```

```
Unit: milliseconds
            expr      min       lq     mean   median       uq    max neval
 random_walk3(y) 1.628201 1.654151 2.012522 1.914201 1.962201 5.8114   100
```

The vectorized function (random_walk2) is absolutely the fastest. The function using apply (random_walk3) seems to be the second fastest, and with the for loop (random_walk1) is the slowest.

d)

```r
prob_10 <- replicate(10000, random_walk2(10))
count_10 <- sum(prob_10 == 0)

count_10/length(prob_10)
```

```
[1] 0.1284
```

```r
prob_100 <- replicate(10000, random_walk2(100))
count_100 <- sum(prob_100 == 0)

count_100/length(prob_100)
```

```
[1] 0.0194
```

```r
prob_1000 <- replicate(10000, random_walk2(1000))
count_1000 <- sum(prob_1000 == 0)

count_1000/length(prob_1000)
```

```
[1] 0.0054
```

According to the Monte Carlo simulations, ~13% of random walks with 10 steps end at 0, ~2% of walks with 100 steps, and ~0.5% of walks with 1000 steps.

Problem 2 - Mean of Mixture of Distributions

```
intersection <- function(n){
  from_12_to_7 <- rpois(n, 1)
  from_9_to_4 <- rpois(n, 8)
  from_6_to_11 <- rpois(n, 12)
  rush_hour1 <- rnorm(n, 60, 12)
  rush_hour2 <- rnorm(n, 60, 12)

  car_count <- from_12_to_7 + from_9_to_4 + from_6_to_11 + rush_hour1 + rush_hour2
  return(mean(car_count))
}
```

Problem 3 - Linear Regression

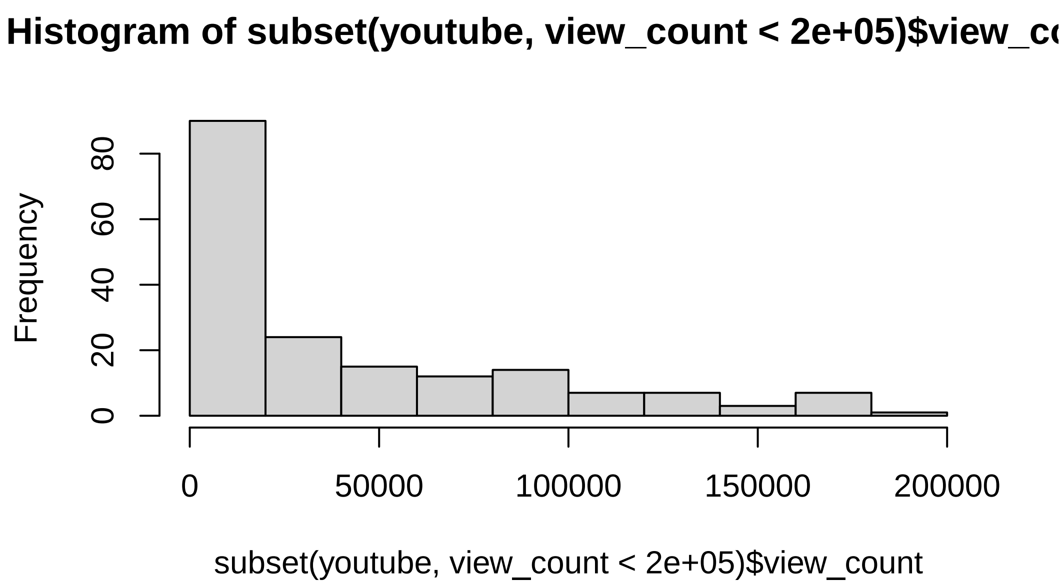
a)

```
youtube <- read.csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/da

youtube <- youtube[ , !(names(youtube) %in% c( "brand",
                                              "superbowl_ads_dot_com_url",
                                              "youtube_url",
                                              "id",
                                              "etag",
                                              "published_at",
                                              "title",
                                              "description",
                                              "thumbnail",
                                              "channel_title"))]

dim(youtube)
```

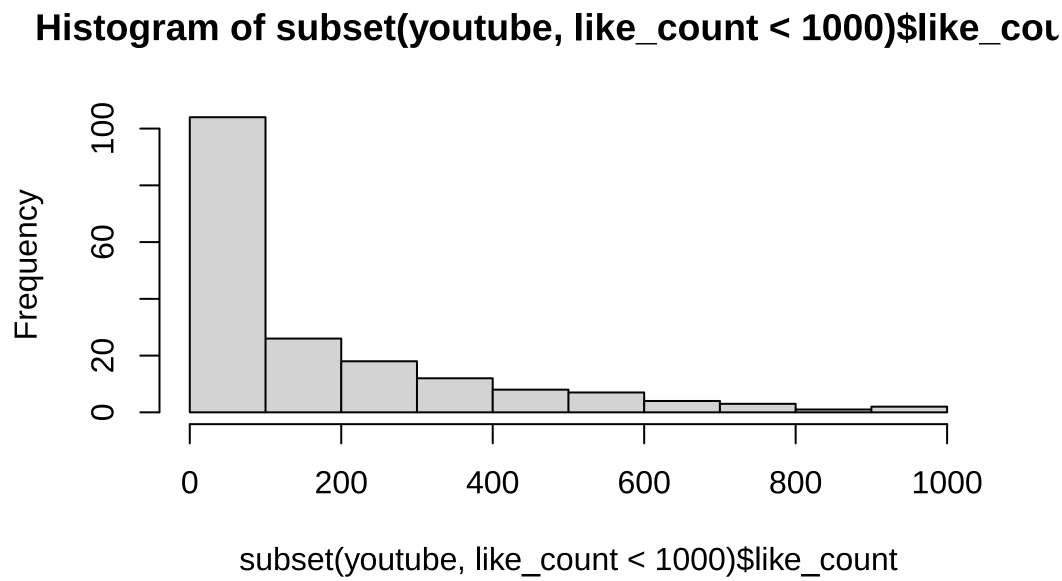
```
[1] 247  15
```

b)

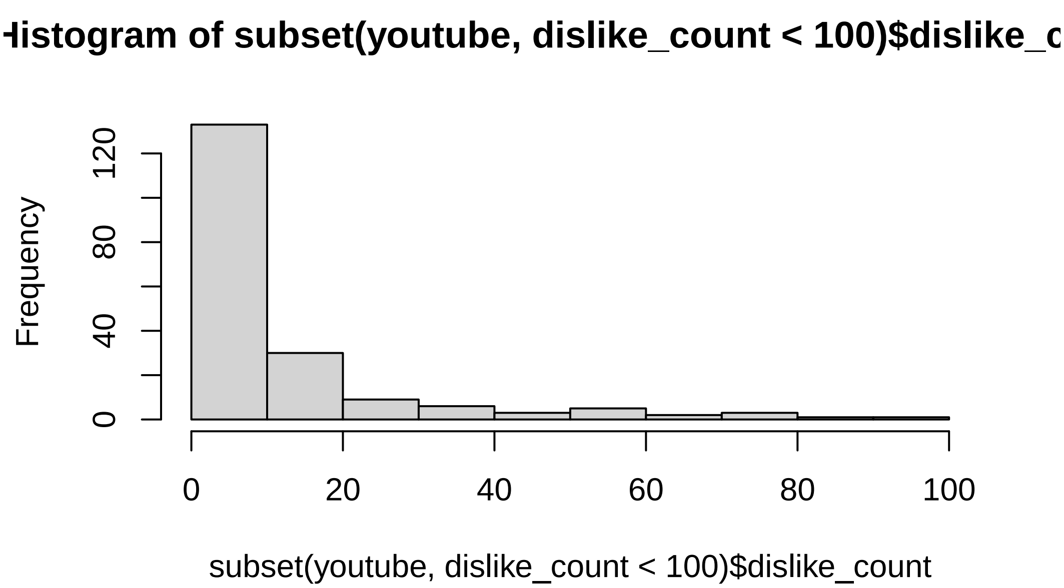
```
#Check distributions of each variable

hist(subset(youtube, view_count < 200000)$view_count)
```



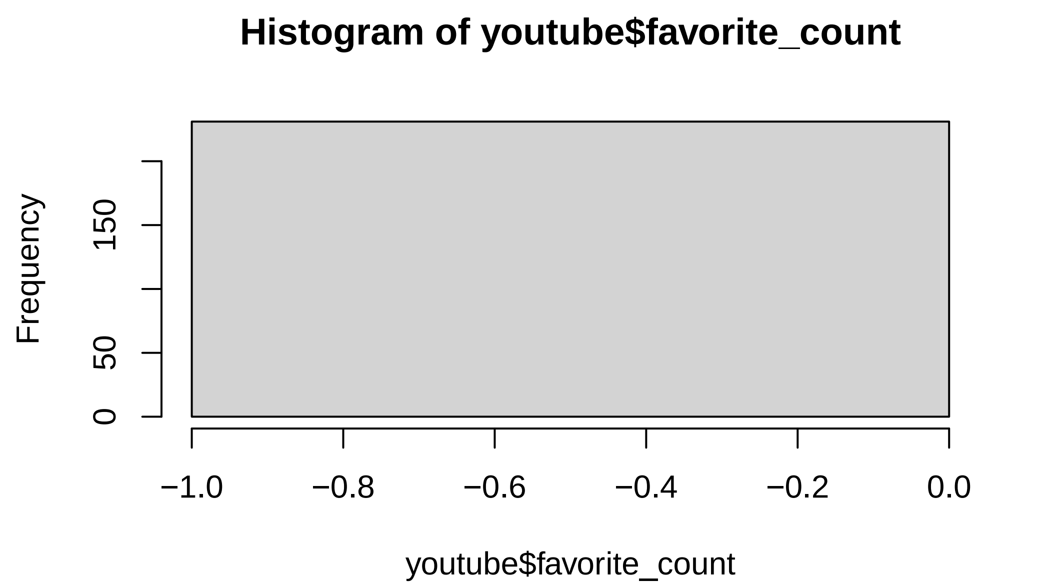

```
hist(subset(youtube, like_count < 1000)$like_count)
```



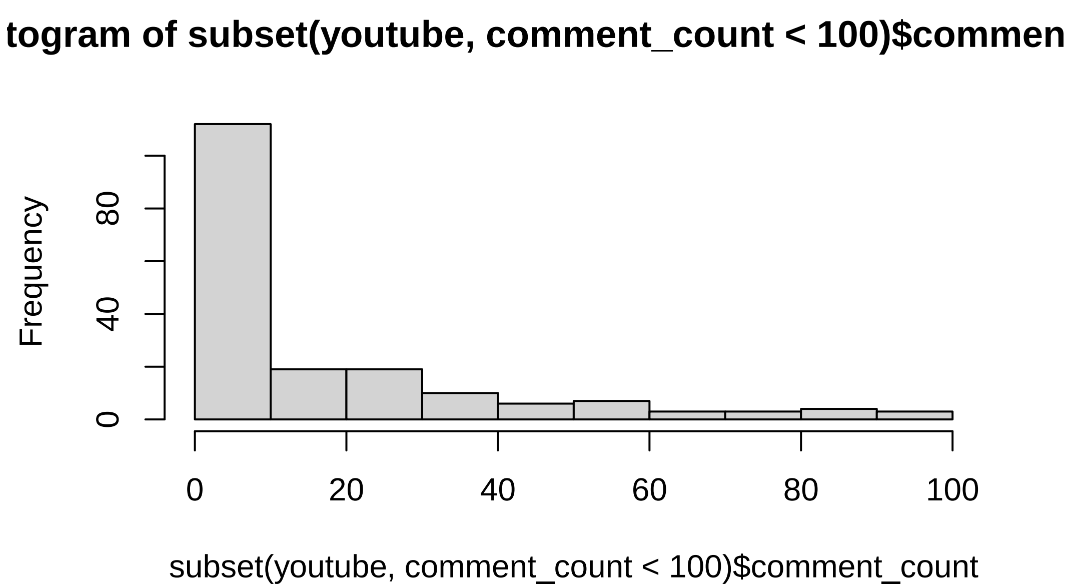

```
hist(subset(youtube, dislike_count < 100)$dislike_count)
```



```r
hist(youtube$favorite_count)
```




```r
hist(subset(youtube, comment_count < 100)$comment_count)
```



All but favorite_count seem suitable to use. Favorite_count is all 0s except for 16 NAs. The rest of the variables are suitable to use, but I will apply a log transformation to them so that they are more normally distributed.

```
youtube$log_view_count <- log(youtube$view_count)
youtube$log_like_count <- log(youtube$like_count + 1) #The +1 for these three
#variables is because they have entries of 0, which will cause problems when we make
#the log transformation
youtube$log_dislike_count <- log(youtube$dislike_count + 1)
youtube$log_comment_count <- log(youtube$comment_count + 1)
```

c)

```
lm_view <- lm(log_view_count ~ funny + show_product_quickly + patriotic +
                celebrity + danger + animals + use_sex + year, data = youtube)

lm_like <- lm(log_like_count ~ funny + show_product_quickly + patriotic +
                celebrity + danger + animals + use_sex + year, data = youtube)

lm_dislike <- lm(log_dislike_count ~ funny + show_product_quickly + patriotic +
                   celebrity + danger + animals + use_sex + year, data = youtube)

lm_comment <- lm(log_comment_count ~ funny + show_product_quickly + patriotic +
                   celebrity + danger + animals + use_sex + year, data = youtube)

summary(lm_view)
```

```

Call:
lm(formula = log_view_count ~ funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex + year,
    data = youtube)

Residuals:
    Min      1Q  Median      3Q     Max
-7.8648 -1.6138  0.1352  1.7048  8.4497

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -31.51445   71.08456  -0.443    0.658
```

```
funnyTRUE                 0.56534    0.46754   1.209    0.228
show_product_quicklyTRUE  0.21094    0.40575   0.520    0.604
patrioticTRUE             0.50512    0.53871   0.938    0.349
celebrityTRUE             0.03667    0.42275   0.087    0.931
dangerTRUE                0.63283    0.41859   1.512    0.132
animalsTRUE              -0.31078    0.39392  -0.789    0.431
use_sexTRUE              -0.38604    0.44832  -0.861    0.390
year                      0.02052    0.03535   0.580    0.562

Residual standard error: 2.791 on 222 degrees of freedom
  (16 observations deleted due to missingness)
Multiple R-squared:  0.02693,   Adjusted R-squared:  -0.008135
F-statistic: 0.768 on 8 and 222 DF,  p-value: 0.6313
```

```r
summary(lm_like)
```

```
Call:
lm(formula = log_like_count ~ funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex + year,
    data = youtube)

Residuals:
    Min      1Q  Median      3Q     Max
-5.2860 -1.6333  0.0868  1.4911  7.7431

Coefficients:
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)              -150.51357   63.45723  -2.372   0.0186 *
funnyTRUE                   0.47476    0.41816   1.135   0.2575
show_product_quicklyTRUE    0.20017    0.36391   0.550   0.5828
patrioticTRUE               0.80689    0.49791   1.621   0.1066
celebrityTRUE               0.41283    0.38212   1.080   0.2812
dangerTRUE                  0.63895    0.37350   1.711   0.0886 .
animalsTRUE                -0.05944    0.35298  -0.168   0.8664
use_sexTRUE                -0.42952    0.40064  -1.072   0.2849
year                        0.07685    0.03155   2.436   0.0157 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.467 on 216 degrees of freedom
  (22 observations deleted due to missingness)
```

```
Multiple R-squared:  0.07313,   Adjusted R-squared:  0.03881
F-statistic:  2.13 on 8 and 216 DF,  p-value: 0.0342
```

```r
summary(lm_dislike)
```

```
Call:
lm(formula = log_dislike_count ~ funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex + year,
    data = youtube)

Residuals:
    Min      1Q  Median      3Q     Max
-4.0292 -1.3299 -0.3192  0.8986  8.7219

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             -183.06813   53.34768  -3.432 0.000719 ***
funnyTRUE                  0.25949    0.35154   0.738 0.461224
show_product_quicklyTRUE   0.27511    0.30593   0.899 0.369515
patrioticTRUE              0.81407    0.41859   1.945 0.053095 .
celebrityTRUE             -0.20214    0.32125  -0.629 0.529852
dangerTRUE                 0.22184    0.31400   0.707 0.480630
animalsTRUE               -0.21192    0.29675  -0.714 0.475911
use_sexTRUE               -0.32980    0.33681  -0.979 0.328583
year                       0.09207    0.02653   3.471 0.000626 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.074 on 216 degrees of freedom
  (22 observations deleted due to missingness)
Multiple R-squared:  0.09753,   Adjusted R-squared:  0.06411
F-statistic: 2.918 on 8 and 216 DF,  p-value: 0.004115
```

```r
summary(lm_comment)
```

```
Call:
lm(formula = log_comment_count ~ funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex + year,
    data = youtube)
```

```
Residuals:
    Min      1Q  Median      3Q     Max 
-4.1372 -1.4665 -0.1427  1.4061  5.8468 

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)  
(Intercept)              -99.09835   52.92351  -1.872   0.0625 .
funnyTRUE                  0.21954    0.34528   0.636   0.5256  
show_product_quicklyTRUE   0.40939    0.30229   1.354   0.1771  
patrioticTRUE              0.66698    0.39902   1.672   0.0961 .
celebrityTRUE              0.29767    0.31541   0.944   0.3464  
dangerTRUE                 0.17784    0.31069   0.572   0.5677  
animalsTRUE               -0.26802    0.29347  -0.913   0.3621  
use_sexTRUE               -0.39323    0.33163  -1.186   0.2370  
year                       0.05034    0.02632   1.913   0.0571 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.039 on 213 degrees of freedom
  (25 observations deleted due to missingness)
Multiple R-squared:  0.06535,   Adjusted R-squared:  0.03025 
F-statistic: 1.862 on 8 and 213 DF,  p-value: 0.06748
```

In general, some patterns emerge. Use of animals and sex in the ads seem to be negatively associated with all four of our outcomes of views, likes, dislikes, and comments. It would seem that there is possibly just an association of lower interaction in general with ads that make use of animals and sex. Using a 95% confidence interval, there is only one significant variable, which is year in the function to predict the dislike count. The coefficient is positive, suggesting that there is an associated increase in the log dislike count of 0.09207 for each increase in the year (i.e. when an ad is newer).

d)

```
# Remove NAs and create design matrix
view_matrix <- na.omit(model.matrix(lm_view, data = youtube))
# Remove NAs and create the outcome vector
view_outcomes <- na.omit(matrix(youtube$log_view_count, nrow = length(youtube$log_view_count)
# Use formula for beta hat

beta_hat <- solve(t(view_matrix)%*%view_matrix)%*%t(view_matrix)%*%view_outcomes
```

```r
beta_hat
```

```
                                [,1]
(Intercept)             -31.51444975
funnyTRUE                 0.56534439
show_product_quicklyTRUE  0.21093609
patrioticTRUE             0.50512368
celebrityTRUE             0.03667410
dangerTRUE                0.63282761
animalsTRUE              -0.31077754
use_sexTRUE              -0.38604161
year                      0.02051521
```

```r
summary(lm_view)
```

```

Call:
lm(formula = log_view_count ~ funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex + year,
    data = youtube)

Residuals:
    Min      1Q  Median      3Q     Max
-7.8648 -1.6138  0.1352  1.7048  8.4497

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -31.51445   71.08456  -0.443    0.658
funnyTRUE                  0.56534    0.46754   1.209    0.228
show_product_quicklyTRUE   0.21094    0.40575   0.520    0.604
patrioticTRUE              0.50512    0.53871   0.938    0.349
celebrityTRUE              0.03667    0.42275   0.087    0.931
dangerTRUE                 0.63283    0.41859   1.512    0.132
animalsTRUE               -0.31078    0.39392  -0.789    0.431
use_sexTRUE               -0.38604    0.44832  -0.861    0.390
year                       0.02052    0.03535   0.580    0.562

Residual standard error: 2.791 on 222 degrees of freedom
  (16 observations deleted due to missingness)
Multiple R-squared:  0.02693,   Adjusted R-squared:  -0.008135
F-statistic: 0.768 on 8 and 222 DF,  p-value: 0.6313
```

The coefficients appear to match.