Stats-506 Problem Set #2

GitHub Repository: https://github.com/zkl2002/Stats-506/

Zhekai Liu

Problem 1 - Modified Random walk

(a).

Step function:

```
#' Sample walk increments
#' @return One time random return with values in {-3,-1,1,10}.
random_step <- function() {
  u1 <- runif(1)
  u2 <- runif(1)
  if (u1 < 0.5) {
    if (u2 < 0.20) -3L else -1L
  } else {
    if (u2 < 0.05) 10L else 1L
  }
}
```

Version 1: using a loop

```
#' Random walk (loop version)
#' @param n Number of steps (non-negative integer).
#' @return Final position of the walk.
random_walk1 <- function(n, seed=NULL) {
  # check input
  if (!(length(n) == 1L && is.numeric(n) && is.finite(n) &&
        n >= 0 && n == as.integer(n))) {
    stop("input must be one non-negative integer")
  }
  if (!is.null(seed)) set.seed(seed)
```

```
  n <- as.integer(n)
  if (n == 0L) return(0L)
  # for loop
  pos <- 0L
  for (i in seq_len(n)) pos <- pos + random_step()
  pos
}
```

Version 2: using built-in R vectorized function

```
#' Random walk version 2 - built-in R vectorized
#' @param n Number of steps.
#' @return Final position of the walk.
random_walk2 <- function(n, seed=NULL) {
  # check input
  if (!(length(n) == 1L && is.numeric(n) && is.finite(n) &&
        n >= 0 && n == as.integer(n))) {
    stop("input must be one non-negative integer")
  }
  if (!is.null(seed)) set.seed(seed)
  n <- as.integer(n)
  if (n == 0L) return(0L)
  # built-in R vectorzed
  u <- runif(2L * n)
  u1 <- u[c(TRUE, FALSE)]
  u2 <- u[c(FALSE, TRUE)]
  dir_pos <- u1 >= 0.5
  inc <- integer(n)
  inc[dir_pos] <- ifelse(u2[dir_pos] < 0.05, 10L, 1L)
  inc[!dir_pos] <- ifelse(u2[!dir_pos] < 0.20, -3L, -1L)
  sum(inc)
}
```

Version 3: using apply function

```
#' Random walk (inside vapply)
#' @param n Number of steps to take
#' @return Final position after n steps
random_walk3 <- function(n, seed=NULL) {
  # check input
  if (!(length(n) == 1L && is.numeric(n) && is.finite(n) &&
```

```r
        n >= 0 && n == as.integer(n))) {
    stop("input must be one non-negative integer")
  }
  # using vapply
  if (!is.null(seed)) set.seed(seed)
  n <- as.integer(n)
  if (n == 0L) return(0L)
  inc <- vapply(seq_len(n), function(i) random_step(), integer(1))
  sum(inc)
}
```

Show results:

```r
random_walk1(10)
```

```
[1] 7
```

```r
random_walk2(10)
```

```
[1] 2
```

```r
random_walk3(10)
```

```
[1] -6
```

```r
random_walk1(1000)
```

```
[1] 12
```

```r
random_walk2(1000)
```

```
[1] 171
```

```r
random_walk3(1000)
```

```
[1] 11
```

(b).

```r
c(random_walk1(10, seed = 506),
  random_walk2(10, seed = 506),
  random_walk3(10, seed = 506))
```

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

```r
c(random_walk1(1000, seed = 506),
  random_walk2(1000, seed = 506),
  random_walk3(1000, seed = 506))
```

```
[1] 73 73 73
```

With same random seed, the result of all three methods are same.

(c).

```r
library(microbenchmark)
## n = 1,000
bench_1000 <- microbenchmark(
  loop  = random_walk1(1000, seed = 506),
  vec   = random_walk2(1000, seed = 506),
  apply = random_walk3(1000, seed = 506)
)
bench_1000
```

```
Unit: microseconds
  expr    min      lq     mean  median      uq     max neval
  loop 2701.6 2788.75 3042.239 2819.85 2912.50  7619.4   100
   vec  126.1  150.30  181.712  170.60  200.80   394.8   100
 apply 3849.1 3915.00 4234.107 3940.00 4140.75 15110.3   100
```

```r
## n = 100,000
bench_10k <- microbenchmark(
  loop  = random_walk1(100000, seed = 506),
  vec   = random_walk2(100000, seed = 506),
  apply = random_walk3(100000, seed = 506)
)
bench_10k
```

```
Unit: milliseconds
  expr      min       lq       mean    median       uq      max neval
  loop 171.7818 177.1505 193.892952 180.39545 198.4828 308.8743   100
   vec   7.6471   8.1415   9.670794   9.39155  10.2754  17.9583   100
 apply 243.0993 250.8302 269.743757 253.48550 271.6712 464.0754   100
```

The fully vectorized implementation is consistently the fastest, then for-loop version comes next, and the apply method approach is the slowest. As the input size grows from 1k to 100k, the performance doesn't change.

(d).

```r
reps <- 10000
set.seed(506)
position_10 <- replicate(reps, random_walk2(10, seed = NULL))
cat("The probability that the random walk ends at 0 with 10 steps is:",
    mean(position_10 == 0))
```

```
The probability that the random walk ends at 0 with 10 steps is: 0.1331
```

```r
set.seed(506)
position_100 <- replicate(reps, random_walk2(100, seed = NULL))
cat("The probability that the random walk ends at 0 with 100 steps is:",
    mean(position_100 == 0))
```

```
The probability that the random walk ends at 0 with 100 steps is: 0.0184
```

```r
set.seed(506)
position_1000 <- replicate(reps, random_walk2(1000, seed = NULL))
cat("The probability that the random walk ends at 0 with 1000 steps is:",
    mean(position_1000 == 0))
```

```
The probability that the random walk ends at 0 with 1000 steps is: 0.0049
```

With Monte Carlo simulation, the probability of random walk ends at 0 is becoming smaller and closer to 0 with steps 10, 100 and 1000.

Problem 2 - Mean of Mixture of Distributions

```r
set.seed(506)
reps <- 10000

daily <- rpois(reps, 8)   +
         rnorm(reps, 120, sqrt(24)) +
         rpois(reps, 64)  +
         rpois(reps, 72)

mean(daily)
```

```
[1] 264.0015
```

By Monte Carlo simulation, the average number of cars pass this intersection daily is about 264.

Problem 3 - Linear Regression

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

(a).

```r
colnames(youtube)
```

```
 [1] "year"                      "brand"
 [3] "superbowl_ads_dot_com_url" "youtube_url"
 [5] "funny"                     "show_product_quickly"
 [7] "patriotic"                 "celebrity"
 [9] "danger"                    "animals"
[11] "use_sex"                   "id"
[13] "kind"                      "etag"
[15] "view_count"                "like_count"
[17] "dislike_count"             "favorite_count"
[19] "comment_count"             "published_at"
[21] "title"                     "description"
[23] "thumbnail"                 "channel_title"
[25] "category_id"
```

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

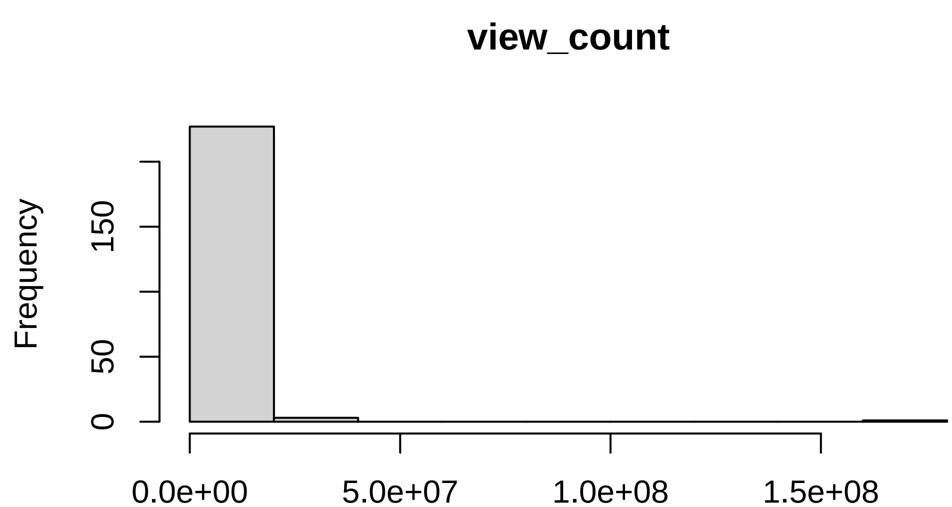
```
[1] 247  14
```

The dimension of the data after de-identify is 247 rows and 14 columns.

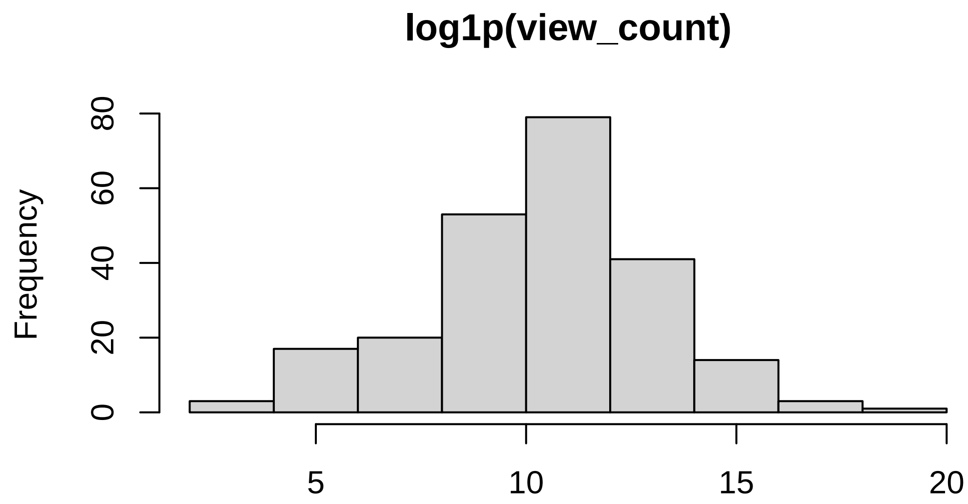
(b).

view_count:

```r
hist(youtube$view_count, main="view_count", xlab="")
```




```r
hist(log1p(youtube$view_count), main="log1p(view_count)", xlab="")
```



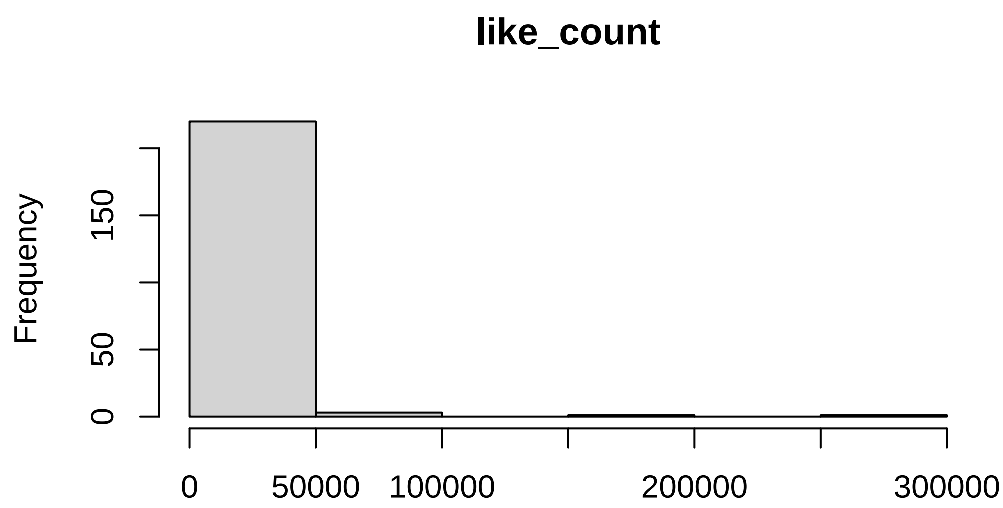

```r
youtube$view_count_log <- log1p(youtube$view_count)
```

view_count variable can use a transformation prior to being used as the outcome in a linear regression model.

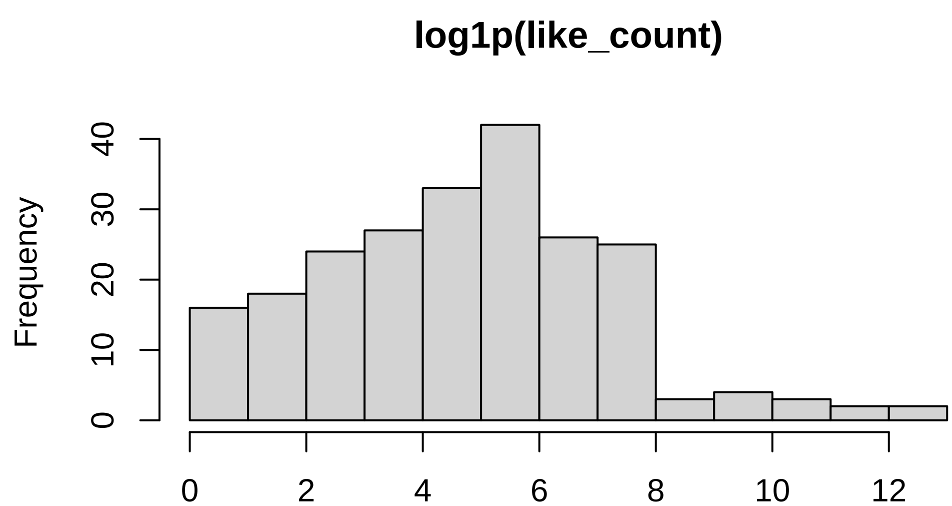
The view_count variable is extremely right-skewed, so we apply a $log(1 + x)$ transformation on it.

like_count:

```r
hist(youtube$like_count, main="like_count", xlab="")
```




```r
hist(log1p(youtube$like_count), main="log1p(like_count)", xlab="")
```



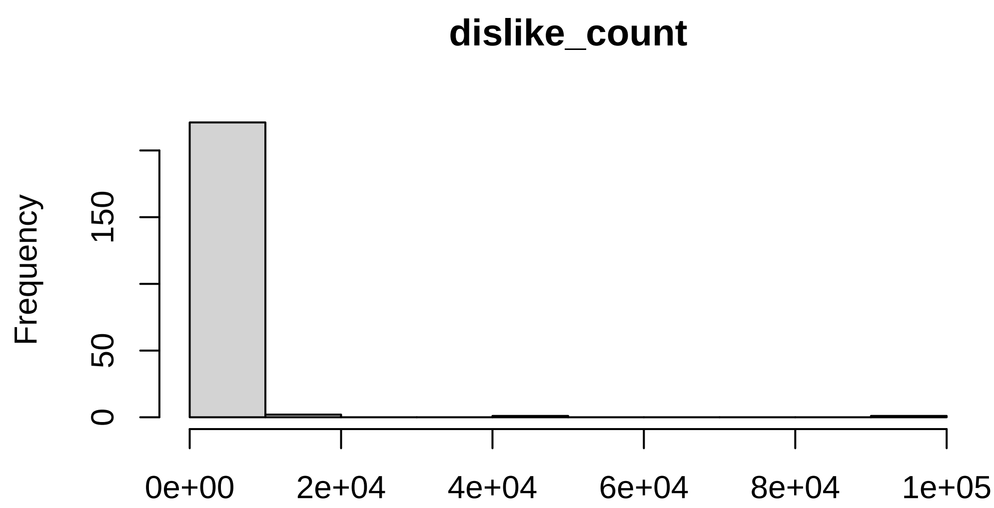
```r
youtube$like_count_log <- log1p(youtube$like_count)
```

like_count variable can use a transformation prior to being used as the outcome in a linear regression model.

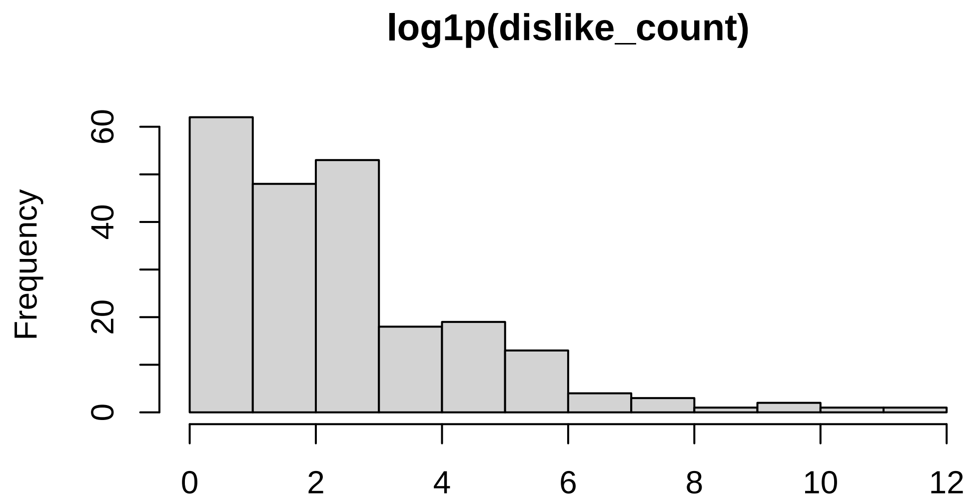
The like_count variable is extremely right-skewed, so we apply a $log(1 + x)$ transformation on it.

dislike_count:

```r
hist(youtube$dislike_count, main="dislike_count", xlab="")
```




```r
hist(log1p(youtube$dislike_count), main="log1p(dislike_count)", xlab="")
```




```
youtube$dislike_count_log <- log1p(youtube$dislike_count)
```

dislike_count variable can use a transformation prior to being used as the outcome in a linear regression model.

The dislike_count variable is extremely right-skewed, so we apply a $log(1 + x)$ transformation on it.

favorite_count:

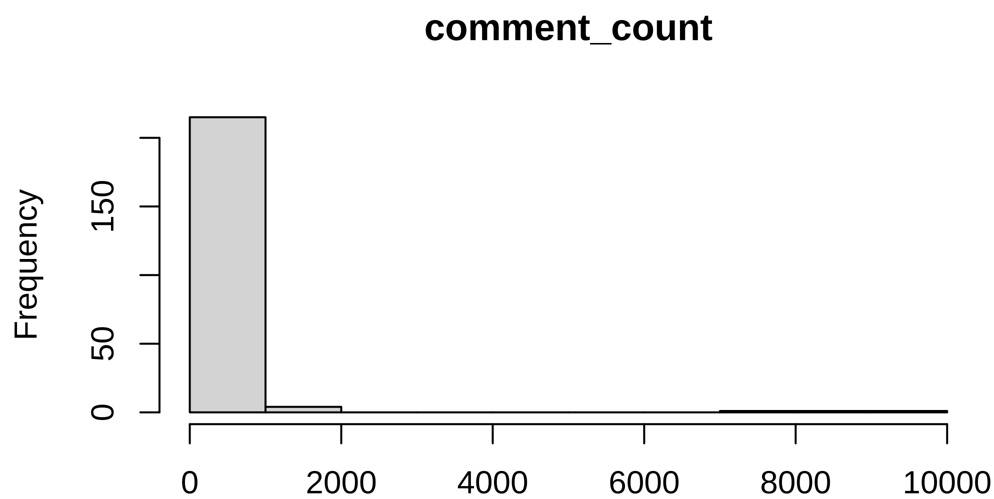
```
unique(youtube$favorite_count)
```

```
[1]  0 NA
```

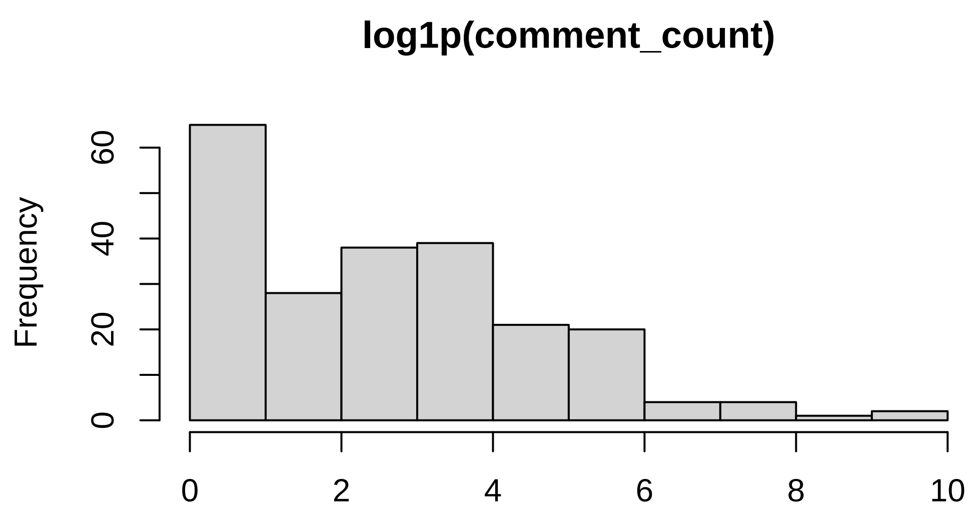
favorite_count only have 0 and null elements, so this variable would not be appropriate to use as the outcome in a linear regression model.

comment_count:

```
hist(youtube$comment_count, main="comment_count", xlab="")
```




```
hist(log1p(youtube$comment_count), main="log1p(comment_count)", xlab="")
```



```r
youtube$comment_count_log <- log1p(youtube$comment_count)
```

comment_count variable can use a transformation prior to being used as the outcome in a linear regression model.

The comment_count variable is extremely right-skewed, so we apply a $log(1+x)$ transformation on it.

(c).

view_count:

```r
fit_view <- lm(view_count_log ~ year + funny + show_product_quickly +
            patriotic + celebrity + danger + animals + use_sex,
          data = youtube)
summary(fit_view)
```

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

Residuals:
    Min      1Q  Median      3Q     Max
-7.7742 -1.6152  0.1311  1.7036  8.4481

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -31.55016   71.00538  -0.444    0.657
year                       0.02053    0.03531   0.582    0.561
funnyTRUE                  0.56492    0.46702   1.210    0.228
show_product_quicklyTRUE   0.21089    0.40530   0.520    0.603
patrioticTRUE              0.50699    0.53811   0.942    0.347
celebrityTRUE              0.03548    0.42228   0.084    0.933
dangerTRUE                 0.63131    0.41812   1.510    0.132
animalsTRUE               -0.31002    0.39348  -0.788    0.432
use_sexTRUE               -0.38671    0.44782  -0.864    0.389

Residual standard error: 2.787 on 222 degrees of freedom
  (   16     )
Multiple R-squared:  0.02694,   Adjusted R-squared:  -0.008122
F-statistic: 0.7684 on 8 and 222 DF,  p-value: 0.631
```

There's no statistically reliable association between ad attributes and view counts. The estimated directions are all not significant. So we could not provide any significant result.

like_count:

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

```

Call:
lm(formula = like_count_log ~ year + funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex, 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 *
year                       0.07685    0.03155   2.436   0.0157 *
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
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.467 on 216 degrees of freedom
  (    22     )
Multiple R-squared:  0.07313,   Adjusted R-squared:  0.03881
F-statistic:  2.13 on 8 and 216 DF,  p-value: 0.0342
```

For like_count, only year shows a positive and statistically significant association with like counts. Danger shows a positive tendency but doesn't reach conventional significance, and the other ad features show no statistically significant associations. As result, only like count numbers would increase with time.

dislike_count:

```r
fit_dislike <- lm(dislike_count_log ~ year + funny + show_product_quickly +
            patriotic + celebrity + danger + animals + use_sex,
          data = youtube)
summary(fit_dislike)
```

```

Call:
lm(formula = dislike_count_log ~ year + funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex, 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 ***
year                        0.09207    0.02653   3.471 0.000626 ***
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
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

For dislike_count, only year has a significant positive association with dislike counts. Patriotic shows a positive tendency but doesn't reach conventional significance, and the other ad features show no statistically significant associations. These also means with time increase, dislike count would also increase.

comment_count:

```r
fit_comment <- lm(comment_count_log ~ year + funny + show_product_quickly +
            patriotic + celebrity + danger + animals + use_sex,
          data = youtube)
summary(fit_comment)
```

```

Call:
lm(formula = comment_count_log ~ year + funny + show_product_quickly +
    patriotic + celebrity + danger + animals + use_sex, 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 .
year                       0.05034    0.02632   1.913   0.0571 .
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
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

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

For log comment counts, there are only marginal positive associations with year and patriotic, but they do not reach the conventional 0.05 significance level. The other ad features show no statistically reliable associations, and the model's explanatory power is low.

(d).

```r
vars <- c("view_count_log","year","funny","show_product_quickly",
          "patriotic","celebrity","danger","animals","use_sex")
dat <- na.omit(youtube[ , vars])

X <- model.matrix(view_count_log ~ year + funny + show_product_quickly +
                    patriotic + celebrity + danger + animals + use_sex,
                  data = dat)
y <- dat$view_count_log
beta_hat <- solve(t(X) %*% X, t(X) %*% y)
beta_hat
```

```
                                 [,1]
(Intercept)              -31.55015804
year                       0.02053399
funnyTRUE                  0.56492445
show_product_quicklyTRUE   0.21088918
patrioticTRUE              0.50699051
celebrityTRUE              0.03547862
dangerTRUE                 0.63131085
animalsTRUE               -0.31001838
use_sexTRUE               -0.38670726
```

The result is same with the lm model in part c.