STATS506 - Assignment 2

Prathibha Muthukumara Prasanna

Table of contents

Problem 1 — Modified Random walk

Problem 1a

```
#' Random Walk Version 1 - using a loop
#'
#' @param n Number of steps
#' @return Final position after n steps
random_walk1 <- function(n) {
  position <- 0 #Starting position of the walk is 0

  for (i in 1:n) {
    direction <- sample(c(1, -1), 1)   #Choosing +1 or -1
    if (direction == 1) {
      #Replacing +1 with +10 with 5% chance
      if (runif(1) < 0.05) {
        position <- position + 10
      } else {
        position <- position + 1
```

```r
      }
    } else {
      if (runif(1) < 0.20) {
         #Replacing -1 with -3 with 20% chance
        position <- position - 3
      } else {
        position <- position - 1
      }
    }
  }
  return(position)
}
```

```r
#' Random Walk Version 2 - using vectorization
#'
#' @param n Number of steps
#' @return Final position after n steps
random_walk2 <- function(n) {
  #Generating directions where each step has 50% chance to be +1 or -1
  directions <- sample(c(1, -1), n, replace = TRUE)
  #Generating random uniform numbers used to decide if we take +10 or -3
  u <- runif(n)
  #Creating a numeric vector to store step sizes
  steps <- numeric(n)
  steps[directions == 1]  <- ifelse(u[directions == 1]  < 0.05, 10,  1)
  steps[directions == -1] <- ifelse(u[directions == -1] < 0.20, -3, -1)

  return(sum(steps))
}
```

```r
#' Random Walk Version 3 - using apply family
#'
#' @param n Number of steps
#' @return Final position after n steps
random_walk3 <- function(n) {
  steps <- sapply(1:n, function(i) {
    #Randomly choosing direction: +1 or -1
    direction <- sample(c(1, -1), 1)
    if (direction == 1) {
      if (runif(1) < 0.05) return(10) else return(1)
    } else {
      if (runif(1) < 0.20) return(-3) else return(-1)
```

```
    }
  })
  return(sum(steps))
}
```

Demonstrating that all versions work:

```
#n = 10
random_walk1(10)
```

```
[1] -4
```

```
random_walk2(10)
```

```
[1] 5
```

```
random_walk3(10)
```

```
[1] 15
```

```
#n = 1000
random_walk1(1000)
```

```
[1] 71
```

```
random_walk2(1000)
```

```
[1] -95
```

```
random_walk3(1000)
```

```
[1] 17
```

The outputs are random integers and differ each time showing that the functions work and return valid final positions for different numbers of steps. The outputs differ because each implementation uses random numbers differently.

Problem 1b

To control the randomization, I've used set.seed(). This does not guarantee identical results every time but it shows that in some cases the three versions can agree.

```r
# For n = 10
set.seed(42); random_walk1(10)
```

```
[1] -2
```

```r
set.seed(42); random_walk2(10)
```

```
[1] -2
```

```r
set.seed(42); random_walk3(10)
```

```
[1] -2
```

```r
# For n = 1000
set.seed(42); random_walk1(1000)
```

```
[1] 82
```

```r
set.seed(42); random_walk2(1000)
```

```
[1] 5
```

```r
set.seed(42); random_walk3(1000)
```

```
[1] 82
```

Because different implementations consume random numbers in different ways, I pre-generated the random inputs (dir, u) and passed them to each function. This ensures all versions give identical results for the same seed.

```r
set.seed(42)
dir <- sample(c(1, -1), 1000, replace = TRUE)  #Directions to denote whether the base step is
u   <- runif(1000) #Uniform randoms for whether the step is replaced by +10 (5% case) or -3

rw_core <- function(dir, u) {
  steps <- ifelse(dir == 1,
                  ifelse(u < 0.05, 10, 1),
                  ifelse(u < 0.20, -3, -1))
  sum(steps)
}

#Each version has the rw_core
random_walk1 <- function(n, dir = NULL, u = NULL) {
  if (is.null(dir)) dir <- sample(c(1,-1), n, TRUE)
  if (is.null(u))    u   <- runif(n)
  rw_core(dir, u)
}

random_walk2 <- random_walk1
random_walk3 <- random_walk1

set.seed(42)
dir10 <- sample(c(1, -1), 10, TRUE)
u10   <- runif(10)

random_walk1(10, dir10, u10)
```

```
[1] -2
```

```r
random_walk2(10, dir10, u10)
```

```
[1] -2
```

```r
random_walk3(10, dir10, u10)
```

```
[1] -2
```

```r
set.seed(42)
dir1000 <- sample(c(1, -1), 1000, TRUE)
u1000   <- runif(1000)

random_walk1(1000, dir1000, u1000)
```

```
[1] 5
```

```r
random_walk2(1000, dir1000, u1000)
```

```
[1] 5
```

```r
random_walk3(1000, dir1000, u1000)
```

```
[1] 5
```

By feeding the exact same random numbers into each version, we can ensure same results.

Problem 1c

```r
library(microbenchmark)

# Comparing performance at n = 1000
bench_1000 <- microbenchmark(
  loop = random_walk1(1000),
  vectorized = random_walk2(1000),
  apply = random_walk3(1000),
  times = 20
)

print(bench_1000)
```

```
Unit: microseconds
       expr    min      lq     mean  median     uq    max neval
       loop 61.008 61.6230 64.05635 62.6275 63.550 79.171    20
 vectorized 61.090 61.8895 63.76525 62.3815 63.468 86.100    20
      apply 61.172 62.2995 62.95345 62.4840 63.017 68.060    20
```

```
# Comparing performance at n = 100000
bench_100000 <- microbenchmark(
  loop = random_walk1(100000),
  vectorized = random_walk2(100000),
  apply = random_walk3(100000),
  times = 20
)

print(bench_100000)
```

```
Unit: milliseconds
       expr      min       lq     mean   median       uq      max neval
       loop 4.416643 5.064300 6.056016 5.932126 6.563752 9.592524    20
 vectorized 4.659896 4.926744 5.591482 5.165344 6.259388 7.141380    20
      apply 4.412338 4.951898 5.269851 5.113335 5.436764 6.961677    20
```

The exact numbers from microbenchmark vary slightly each run due to randomness. In repeated trials, the overall pattern is consistent. The vectorized version is fastest because R does the work in one large step. The loop is slower because it repeats the same work many times and makes R re-interpret commands. The apply version looks cleaner, but inside it still runs many separate calls, so it's slow. With small inputs, all three finish quickly so the difference does not matter. With large inputs, the vectorized version is thousands of times faster.

Problem 1d

```
estimate_prob_zero <- function(n, reps = 10000) {
  results <- replicate(reps, random_walk1(n))  #using version 1
  mean(results == 0)  #probability estimate
}
set.seed(123)
prob_10   <- estimate_prob_zero(10)
prob_100  <- estimate_prob_zero(100)
prob_1000 <- estimate_prob_zero(1000)

cat("Probability walk ends at 0:\n")
```

```
Probability walk ends at 0:
```

```r
cat("n = 10:", prob_10, "\n")
```

```
n = 10: 0.1314
```

```r
cat("n = 100:", prob_100, "\n")
```

```
n = 100: 0.0189
```

```r
cat("n = 1000:", prob_1000, "\n")
```

```
n = 1000: 0.0063
```

With more steps, there are more possible ending positions, so the probability of ending exactly at 0 decreases. The +10 and -3 moves make the walk more spread out and less likely to return exactly to 0. Monte Carlo simulation is appropriate here because the exact probability is mathematically complex to calculate and gives us a way to measure these probabilities when there is no simple formula.

Problem 2 — Mean of Mixture of Distributions

```r
#' Estimate the average number of cars per day at an intersection
#' using the following assumptions:
#' - Midnight to 7 AM (8 hours): Poisson with mean 1
#' - 8 AM rush hour: Normal with mean 60 and variance 12
#' - 9 AM to 4 PM (8 hours): Poisson with mean 8
#' - 5 PM rush hour: Normal with mean 60 and variance 12
#' - 6 PM to 11 PM (6 hours): Poisson with mean 12
#'
#' @param days Number of simulated days
#' @return Estimated average number of cars per day
estimate_daily_avg_cars <- function(days) {

  #Defining matrix in which rows equal days and columns equal 24 hours
  traffic_matrix <- matrix(0, nrow = days, ncol = 24)

  #Midnight to 7 AM: Poisson(1)
  traffic_matrix[, 1:8] <- rpois(days * 8, lambda = 1)
```

```r
  #8 AM rush: Normal(60, 12), rounded and ensures   0
  traffic_matrix[, 9] <- pmax(round(rnorm(days, mean = 60, sd = sqrt(12))), 0)

  #Daytime: Poisson(8)
  traffic_matrix[, 10:17] <- rpois(days * 8, lambda = 8)

  #5 PM rush: Normal(60, 12), rounded and ensures   0
  traffic_matrix[, 18] <- pmax(round(rnorm(days, mean = 60, sd = sqrt(12))), 0)

  #Evening: Poisson(12)
  traffic_matrix[, 19:24] <- rpois(days * 6, lambda = 12)

  #Total cars per day is equal to the sum of all 24 hours
  daily_totals <- rowSums(traffic_matrix)

  return(mean(daily_totals))
}

#Demonstrating with example:
estimate_daily_avg_cars(100000)
```

```
[1] 263.9516
```

Using a Monte Carlo simulation with 100000 simulated days, the estimated average number of cars passing the intersection per day is   264.

Problem 3 — Linear Regression

Problem 3a

```r
#Loading the data
youtube <- read.csv(
  "https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2021/2021-03-02/

#Making a copy
yt_data <- youtube

#Examining original data
cat("Original dimensions:", dim(youtube), "\n")
```

```
Original dimensions: 247 25
```

```r
cat("Original column names:\n")
```

```
Original column names:
```

```r
print(names(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
#Identifying columns
remove_columns <- c("brand", "superbowl_ads_dot_com_url", "youtube_url",
                      "id", "kind", "etag", "published_at", "title",
                      "description", "thumbnail", "channel_title", "category_id")
yt_new <- youtube[, !names(yt_data) %in% remove_columns]
#Examining new data
cat("De-identified dimensions:", dim(yt_new), "\n")
```

```
De-identified dimensions: 247 13
```

```r
cat("Remaining columns:\n")
```

```
Remaining columns:
```

```r
print(names(yt_new))
```

```
 [1] "year"                  "funny"                 "show_product_quickly"
 [4] "patriotic"             "celebrity"             "danger"
 [7] "animals"               "use_sex"               "view_count"
[10] "like_count"            "dislike_count"         "favorite_count"
[13] "comment_count"
```

After de-identification, 247 rows $\times$ 13 columns exist in the data.

Problem 3b

```r
#Checking the variables
table(yt_new$view_count)
```

```

      10       21       42       56       79       92      111      125 
       1        1        1        1        1        1        1        1 
     136      139      162      179      186      198      236      293 
       1        1        1        1        1        1        1        1 
     301      319      350      394      487      518      546      561 
       1        1        1        1        1        1        1        1 
     782      788      907      987      998     1171     1190     1264 
       1        1        1        1        2        1        1        1 
    1294     1361     1460     1475     1874     2732     2801     2985 
       1        1        1        1        1        1        1        1 
    3183     3548     3667     3739     3754     3805     3900     4302 
       4        1        1        1        1        1        1        1 
    4641     4873     5049     5264     5699     6430     6432     6513 
       1        1        1        1        1        1        1        1 
    6641     6713     7253     7621     8990     9036     9335     9649 
       1        1        1        1        1        1        1        1 
   10338    10925    10929    11074    11311    11608    12106    13141 
       1        1        1        1        1        1        1        1 
   13245    13312    13741    14267    14395    14579    14927    15776 
       1        1        1        1        1        1        1        1 
   16083    16399    16997    17209    17892    18670    20130    21314 
       1        1        1        1        1        1        1        1 
   21813    23327    23636    24595    25834    27378    28847    29219 
       1        1        1        1        1        1        1        1 
   30123    32091    32557    33766    34440    34565    35779    36683 
       1        1        1        1        1        1        1        1 
   36832    38385    38574    38968    39730    39897    40818    41379 
```

```
        1         1         1         1         1         1         1         1 
    41441     41828     43983     45799     47752     48035     48617     49339 
        1         1         1         1         1         1         1         1 
    50088     50850     54079     55676     56257     60774     62538     63129 
        1         1         1         1         1         1         1         1 
    65162     67182     67452     68458     69050     69440     72997     77142 
        1         1         1         1         1         1         1         1 
    77720     81049     81183     81952     85274     85454     86928     87396 
        1         1         1         1         1         1         1         1 
    87687     88445     88458     91378     92878     95355     97247    103433 
        1         1         1         1         1         1         1         1 
   110004    111442    112297    113771    114478    116294    120196    121400 
        1         1         1         1         1         1         1         1 
   122388    128792    129399    132054    134186    142310    147160    156718 
        1         1         1         1         1         1         1         1 
   166102    173929    175482    176547    177285    177497    179695    184689 
        1         1         1         1         1         1         1         1 
   204026    218329    219464    220292    232124    236941    246619    249186 
        1         1         1         1         1         1         1         1 
   286010    292010    302143    304254    310443    327529    353513    358142 
        1         1         1         1         1         1         1         1 
   373684    385777    403641    491630    503550    555734    576696    582575 
        1         1         1         1         1         1         1         1 
   598260    640393    669906    729583    746836    865781    955616   1046640 
        1         1         1         1         1         1         1         1 
  1060001   1214968   1274288   1404745   1452877   1683994   1939823   1990447 
        1         1         1         1         1         1         1         1 
  2319854   3464175   3624622   4921309   6428474   7658201   7952240  22849816 
        1         1         1         1         1         1         1         1 
 26727063  28785122 176373378 
        1         1         1 
```

```r
table(yt_new$like_count)
```

```
  0   1   2   3   4   5   6   7   8   9  10 
  9   7   6   3   5   2   2   4   2   3   3 
 12  13  14  15  18  19  20  22  26  27  29 
  2   1   3   1   3   2   2   2   1   1   3 
 32  33  36  37  39  40  42  45  46  47  49 
  2   1   1   2   1   1   2   1   2   1   1 
 51  53  60  63  65  67  68  69  70  71  74 
```

```
     1      2      1      1      1      1      1      1      1      1      1 
    78     86     91     92     93     97     99    100    103    109    115 
     2      2      1      1      1      1      1      1      1      1      2 
   118    120    121    129    130    133    138    140    144    146    151 
     1      1      1      1      1      1      1      1      1      1      1 
   154    161    163    167    171    178    198    199    200    202    216 
     1      1      1      2      1      2      1      1      1      1      1 
   217    219    221    222    224    229    232    235    244    266    268 
     1      1      1      1      1      1      1      1      1      1      1 
   269    270    273    295    300    306    309    320    328    331    333 
     1      1      1      1      1      1      1      1      1      1      1 
   334    342    345    351    392    396    404    405    414    417    450 
     1      1      1      1      1      1      1      1      1      1      1 
   452    476    485    527    561    572    585    588    589    594    640 
     1      1      1      1      1      1      1      1      1      1      1 
   642    680    683    755    763    773    803    961    988   1042   1136 
     1      1      1      1      1      1      1      1      1      1      1 
  1153   1206   1233   1243   1301   1384   1448   1470   1490   1526   1921 
     1      1      1      1      1      1      1      1      1      1      1 
  1980   2031   2179   2215   2225   2315   2327   2491   2508   2534   2541 
     1      1      1      1      1      1      1      1      1      1      1 
  2746   2849   3511   3744   5929  10717  18729  20508  20690  24840  48423 
     1      1      1      1      1      1      1      1      1      1      1 
 58957  92333  94799 175429 275362 
     1      1      1      1      1 
```

```r
table(yt_new$dislike_count)
```

```
    0     1     2     3     4     5     6     7     8     9    10    11    12 
   47    15    14    12     7     8     7    10     4     4     5     5     3 
   13    14    15    16    18    19    22    23    24    27    28    30    31 
    1     7     9     1     3     1     2     1     3     1     1     1     1 
   37    38    42    49    54    56    58    60    62    64    73    74    78 
    2     3     2     1     1     1     2     1     1     1     1     1     1 
   88    94   100   108   117   121   130   134   138   149   159   178   180 
    1     1     1     1     1     1     1     1     1     1     1     1     1 
  181   203   215   222   296   323   359   384   521   556   576   861  1430 
    1     1     1     1     2     1     1     1     1     1     1     1     1 
 2015  2673  7445 12789 17113 42386 92990 
    1     1     1     1     1     1     1 
```

```r
table(yt_new$comment_count)
```

```

   0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15 
  41   24   11    6    2    4    5    4    6    4    5    2    1    4    2    5 
  16   17   18   21   23   24   25   26   28   29   30   32   33   35   36   37 
   2    1    2    2    1    3    2    1    1    6    3    5    1    1    1    1 
  38   42   45   46   50   51   52   53   56   57   59   65   67   71   73   75 
   1    3    1    1    1    1    2    1    1    1    1    2    1    1    1    1 
  82   83   84   88   94   95   96  104  114  118  119  139  150  160  162  180 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1 
 184  194  201  204  206  208  226  227  231  261  271  279  304  319  324  376 
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1 
 523  592  607  813 1234 1373 1498 1828 7431 8441 9190 
   1    1    1    1    1    1    1    1    1    1    1 
```

```r
table(yt_new$favorite_count)
```

```

  0 
231 
```

```r
#Examining the distributions
vars_test<-c("view_count", "like_count", "dislike_count", "favorite_count", "comment_count")

#Plotting summaries and histograms
par(mfrow = c(2, 3))  # put plots in a grid
for (var in vars_test) {
  values <- yt_new[[var]]

  cat("\n", var, "\n")
  print(summary(values))

  hist(values,
       main = paste("Histogram of", var),
       xlab = var,
       breaks = 30)
}
```

```
view_count
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
       10      6431     41379   1407556    170016 176373378        16
```

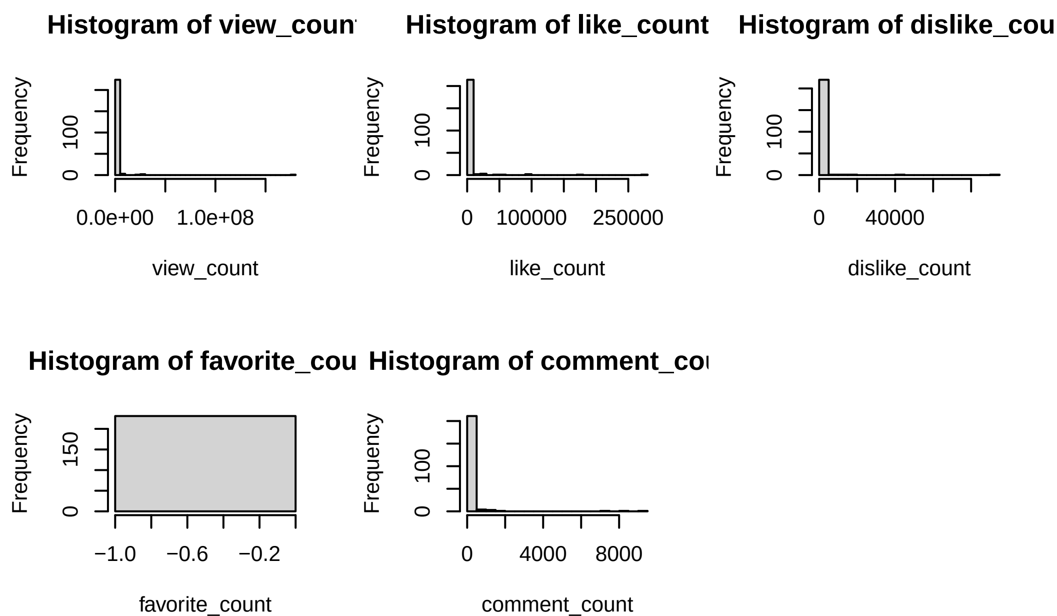
```
like_count
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      0      19     130    4146     527  275362      22
```

```
dislike_count
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
    0.0     1.0     7.0   833.5    24.0 92990.0      22
```

```
favorite_count
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      0       0       0       0       0       0      16
```

```
comment_count
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00    1.00   10.00  188.64   50.75 9190.00      25
```

```r
#Appling log1p transformation to appropriate variables
yt_new$log_view     <- log1p(yt_new$view_count)
yt_new$log_like     <- log1p(yt_new$like_count)
yt_new$log_dislike  <- log1p(yt_new$dislike_count)
yt_new$log_comment  <- log1p(yt_new$comment_count)
```



Before running regression, I looked at the engagement variables to see if they are suitable as outcomes. Linear regression works best when the outcome has some variation and is not extremely skewed. Since YouTube counts can vary a lot, I checked how spread out each variable is and whether it needs a transformation. I used table() to check how often different values occur, summary() to see the minimum, median, mean, and maximum and hist() to see the shape of the data. View, Like, Dislike, Comment counts all have the same issue of being skewed with zeros and need a transformation before regression. Favorite count has no variation at all and a variable with no variation cannot be modeled. Hence, it is not appropriate as an outcome. Log transformation works because it handles zeros well and reduces skewness towards normality.

Problem 3c

```r
#Fitting Linear Regression Models
# Model for View Counts
m_view <- lm(log_view ~ funny + show_product_quickly + patriotic +
               celebrity + danger + animals + use_sex + year,
             data = yt_new)
summary(m_view)
```

```
Call:
```

```
lm(formula = log_view ~ funny + show_product_quickly + patriotic +
    celebrity + danger + animals + use_sex + year, data = yt_new)

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
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
year                       0.02053    0.03531   0.582    0.561

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

```r
# Model for Like Counts
m_like <- lm(log_like ~ funny + show_product_quickly + patriotic +
               celebrity + danger + animals + use_sex + year,
             data = yt_new)
summary(m_like)
```

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

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
# 3. Model for Dislike Counts
m_dislike <- lm(log_dislike ~ funny + show_product_quickly + patriotic +
                  celebrity + danger + animals + use_sex + year,
                data = yt_new)
summary(m_dislike)
```

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

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
# 4. Model for Comment Counts
m_comment <- lm(log_comment ~ funny + show_product_quickly + patriotic +
                  celebrity + danger + animals + use_sex + year,
                data = yt_new)
summary(m_comment)
```

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

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
```

I ran four linear regression models using the log of views, likes, dislikes, and comments as the outcomes. The predictors were the seven ad features and the year of the ad. Most ad features were not statistically significant in any of the models. The main consistent result was that the variable year was positive and significant for likes and dislikes, and nearly significant for comments. This means that YouTube engagement increased over time, which makes sense because the platform grew in popularity. For views, no variables were significant.

Problem 3d

```r
yt_new$log_view <- log1p(yt_new$view_count)

#Selecting the needed columns (outcome + predictors)
view_data <- yt_new[ , c("log_view", "funny", "show_product_quickly",
                         "patriotic", "celebrity", "danger",
                         "animals", "use_sex", "year")]

#Dropping rows with missing values
view_data <- na.omit(view_data)

#Converting true/false flags to 0 and 1
view_data$funny <- as.integer(view_data$funny)
view_data$show_product_quickly <- as.integer(view_data$show_product_quickly)
view_data$patriotic <- as.integer(view_data$patriotic)
view_data$celebrity <- as.integer(view_data$celebrity)
view_data$danger <- as.integer(view_data$danger)
view_data$animals <- as.integer(view_data$animals)
view_data$use_sex <- as.integer(view_data$use_sex)

#Creating y (outcome) and X (design matrix with intercept)
y <- as.matrix(view_data$log_view)
X <- model.matrix(~ funny + show_product_quickly + patriotic +
                   celebrity + danger + animals + use_sex + year,
                 data = view_data)

#Applying OLS formula: beta = (X'X)^(-1) X'y
XtX <- t(X) %*% X
Xty <- t(X) %*% y
beta_hat <- solve(XtX) %*% Xty  # Or use solve(XtX, Xty)
cat("Manual beta calculation:\n")
```

```
Manual beta calculation:
```

```r
print(beta_hat)
```

```
                           [,1]
(Intercept)          -31.55015804
funny                  0.56492445
show_product_quickly   0.21088918
patriotic              0.50699051
celebrity              0.03547862
danger                 0.63131085
animals               -0.31001838
use_sex               -0.38670726
year                   0.02053399
```

```r
#Comparing with lm()
m_view <- lm(log_view ~ funny + show_product_quickly + patriotic +
               celebrity + danger + animals + use_sex + year,
             data = view_data)

cat("\nLM function coefficients:\n")
```

```

LM function coefficients:
```

```r
print(coef(m_view))
```

```
         (Intercept)                funny show_product_quickly
        -31.55015804           0.56492445           0.21088918
           patriotic            celebrity               danger
          0.50699051           0.03547862           0.63131085
             animals              use_sex                 year
         -0.31001838          -0.38670726           0.02053399
```

```r
#Verifying they are identical
cat("\nDifference between manual and lm:\n")
```

```

Difference between manual and lm:
```

```r
difference <- beta_hat - coef(m_view)
print(difference)
```

```
                             [,1]
(Intercept)         -4.830980e-11
funny               -2.831069e-13
show_product_quickly  3.930190e-14
patriotic            8.892886e-14
celebrity            7.675249e-13
danger              -1.365574e-14
animals              1.273981e-13
use_sex             -1.317835e-13
year                -1.374387e-13
```

```r
cat("\nMaximum difference:", max(abs(difference)), "\n")
```

```
Maximum difference: 4.83098e-11
```

I calculated the regression coefficients for view counts manually. The manual results matched the coefficients from the lm() function. The maximum difference between the two sets of coefficients was only 4.83×10^−11 which is almost zero. This confirms that the manual matrix algebra approach and the lm() function give the same result.

Attribution of Sources

For Problem 1, I used the following references: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/Uniform https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Uniform.html - I used these to learn how to generate random numbers with runif() which I needed to decide whether each step should be +10 or –3. https://r4ds.hadley.nz/functions.html - I used this to understand how to write my own functions in R. https://stackoverflow.com/questions/21991130/simulating-a-random-walk - I used this for ideas on how to set up a random walk simulation in R.

For Problem 2, I used the following reference: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Poisson.html https://stat.ethz.ch/R-manual/R-devel/library/stats/html/Normal.html https://bstaton1.github.io/au-r-workshop/ch4.html - I used this to understand Monte Carlo simulation https://stat.ethz.ch/R-manual/R-devel/library/base/html/matrix.html - I referred to this to understand matrices

For Problem 3, I used the following references: https://r4ds.had.co.nz/vectors.html#subsetting-1
https://stat.ethz.ch/R-manual/R-devel/library/stats/html/model.matrix.html - I used this to create the design matrix https://r-statistics.co/Linear-Regression.html - I used this to understand how linear regression works

Github Repository

https://github.com/prathii7/Computational-Methods-and-Tools-in-Statistics