STATS-506 HW2

Zekai Xu

2025-09-13

Github Repo

Question 1

Part a

```r
#' Compute random walk using explicit loops
#' @parameter n, a positive integer, number of steps
#' @return an integer, position after random walk
random_walk1 <- function(n)
{
  current <- 0
  for (i in 1:n)
  {
    direction <- runif(1)
    if (direction <= 0.5 ) # Left
    {
      boost <- runif(1)
      if (boost <= 0.8) # No boost
      {
        current <- current - 1
      }
      else # Boost
      {
        current <- current - 3
      }
    }
    else
```

```r
    {
      boost <- runif(1)
      if (boost <= 0.95) # No boost
      {
        current <- current + 1
      }
      else # Boost
      {
        current <- current + 10
      }
    }
  }
  return (current)
}
```

```r
#' Compute random walk using built-in R vectorized functions
#' @param n, a positive integer, number of steps
#' @return an integer, position after random walk
random_walk2 <- function(n)
{
  current <- 0

  rand_nums <- runif(2 * n)

  direction <- rand_nums[c(TRUE, FALSE)]
  direction <- as.integer(direction > 0.5)

  boost <- rand_nums[c(FALSE, TRUE)]
  boost <- as.integer((direction == 0 & boost > 0.8) | (direction == 1 & boost > 0.95))

  steps <- ifelse(direction == 1, 1 + 9 * boost, -1 - 2 * boost)
  return (sum(steps))
}
```

```r
#' Compute random walk using one of the `apply` functions
#' @param n, a positive integer, number of steps
#' @return an integer, position after random walk
random_walk3 <- function(n)
{
  steps <- sapply(1:n, \(i){
    direction <- runif(1)
    if (direction <= 0.5) # Left
```

```r
    {
      boost <- runif(1)
      if (boost <= 0.8)
        return (-1)
      else
        return (-3)
    }
    else # Right
    {
      boost <- runif(1)
      if (boost <= 0.95)
        return (1)
      else
        return (10)
    }
  })
  return (sum(steps))
}
```

```r
random_walk1(10)
```

```
[1] -4
```

```r
random_walk2(10)
```

```
[1] 4
```

```r
random_walk3(10)
```

```
[1] -10
```

```r
random_walk1(1000)
```

```
[1] 120
```

```r
random_walk2(1000)
```

```
[1] 45
```

```r
random_walk3(1000)
```

```
[1] -65
```

Part b

```r
SEED <- 43

set.seed(SEED)
random_walk1(10)
```

```
[1] -10
```

```r
random_walk1(1000)
```

```
[1] 150
```

```r
set.seed(SEED)
random_walk2(10)
```

```
[1] -10
```

```r
random_walk2(1000)
```

```
[1] 150
```

```r
set.seed(SEED)
random_walk3(10)
```

```
[1] -10
```

```r
random_walk3(1000)
```

```
[1] 150
```

Part c

```r
TIMES <- 10
N1 <- 1000
N2 <- 100000

result_1k <- microbenchmark(random_walk1(N1), random_walk2(N1),
                            random_walk3(N1), times = TIMES)
```

```
Warning in microbenchmark(random_walk1(N1), random_walk2(N1), random_walk3(N1),
: less accurate nanosecond times to avoid potential integer overflows
```

```r
result_100k <- microbenchmark(random_walk1(N2), random_walk2(N2),
                            random_walk3(N2), times = TIMES)

print(result_1k)
```

```
Unit: microseconds
             expr     min       lq      mean   median       uq      max neval
 random_walk1(N1) 730.907  755.466  792.1938  793.514  819.754  859.483    10
 random_walk2(N1)  41.492   46.699   50.9917   51.086   56.621   58.753    10
 random_walk3(N1) 993.512 1038.694 1067.7876 1071.761 1098.021 1130.493    10
```

```r
print(result_100k)
```

```
Unit: milliseconds
             expr        min         lq       mean    median         uq
 random_walk1(N2)  81.440227  82.138867  83.382664  82.58774  84.503829
 random_walk2(N2)   3.860519   3.923536   4.339395   4.16191   4.324926
 random_walk3(N2) 112.537538 115.168385 128.515472 118.74971 121.815592
        max neval
  88.636629    10
   6.476524    10
 188.091067    10
```

Based on the benchmark results, the vectorized implementation (random_walk2) is consistently the fastest, being nearly twenty times quicker than the explicit loop (random_walk1) and over twenty-five times faster than the apply-family version (random_walk3) for both small and large input sizes. The explicit loop performs moderately well, faster than the apply approach but still much slower than vectorization, while the apply family is the slowest due to the overhead of repeated function calls. Overall, vectorization clearly outperforms the other methods, especially as the problem size grows.

Part d

The random walk is $Y = \sum_{i=1}^{n} X_i$, where X_i i.i.d follows discrete pdf:

$$P(X_i = -3) = 0.1, \quad P(X_i = -1) = 0.4, \quad P(X_i = 1) = 0.475, \quad P(X_i = 10) = 0.025$$

Note X_i has finite variance, therefore we could apply central limit theorem:

$$\frac{\bar{X} - E[X_i]}{\sqrt{Var(X_i)}/\sqrt{n}} \xrightarrow{d} N(0,1) \Rightarrow Y \xrightarrow{d} N(nE[X_i], nVar(X_i))$$

To compute the probability, we adopt **local** central limit theorem, and the approximate probability of $Y = 0$ can be calculated as follow:

$$P(Y = 0) \approx \frac{h}{\sqrt{2\pi n Var(X_i)}} exp\{-\frac{1}{2nVar(X_i)}(0 - nE[X_i])^2\}$$

where h is the lattice span (the greatest common divisor of all pairwise differences between possible values of X_i), i.e. the spacing of the grid on which the distribution is supported. In this question, $h = 1$.

The expectation and variance of X_i are:

$$\begin{aligned} E[X_i] &= 0.025 \\ Var(X_i) &= E[X_i^2] - E^2[X_i] = 4.274375 \end{aligned}$$

Now we could compute the asymptotic probability with number of steps 10, 100 and 1000:

$$\begin{aligned} P(Y = 0; n = 10) &= 0.06097562 \\ P(Y = 0; n = 100) &= 0.01915573 \\ P(Y = 0; n = 1000) &= 0.00567182 \end{aligned}$$

Monte Carlo Simulation

```r
#' Monte Carlo Simulation of random walk back to 0 after n steps
#' @param n, a positive integer, number of steps
#' @param N, a positive integer, number of simulations
#' @param seed, a positive integer, random number seed
#' @return frequency of random walk back to 0 after n steps
simul <- function(n, N = 1e4, seed = SEED)
{
  vals <- c(-3, -1, 1, 10)
  probs <- c(0.1, 0.4, 0.475, 0.025)
```

```r
  set.seed(seed)
  steps <- matrix(sample(vals, size = N * n, replace = TRUE, prob = probs), nrow = N)
  sums <- rowSums(steps)

  return (sum(sums == 0) / N)
}
simul(10)
```

```
[1] 0.1388
```

```r
simul(100)
```

```
[1] 0.0195
```

```r
simul(1000)
```

```
[1] 0.0055
```

Notice that for $n = 100$ and $n = 1000$, the theoretical asymptotic probability obtained from the local CLT is very close to the empirical frequency from the Monte Carlo simulation, which supports the validity of the local CLT approximation in large samples. However, when n is small (e.g., $n = 10$), the discrepancy between the theoretical probability and the simulation result is substantial, reflecting the inaccuracy of the asymptotic approximation in the small-sample regime.

Question 2

```r
set.seed(SEED)
(
  sum(rpois(7, 1)) # Midnight - 7AM
+ sum(rnorm(2, 60, sqrt(12))) # 7AM - 9AM
+ sum(rpois(7, 7)) # 9AM - 4PM
+ sum(rnorm(2, 60, sqrt(12))) # 4PM - 6PM
+ sum(rpois(6, 12)) # 6PM - Midnight
) / 24
```

```
[1] 15.11011
```

Question 3

Part a

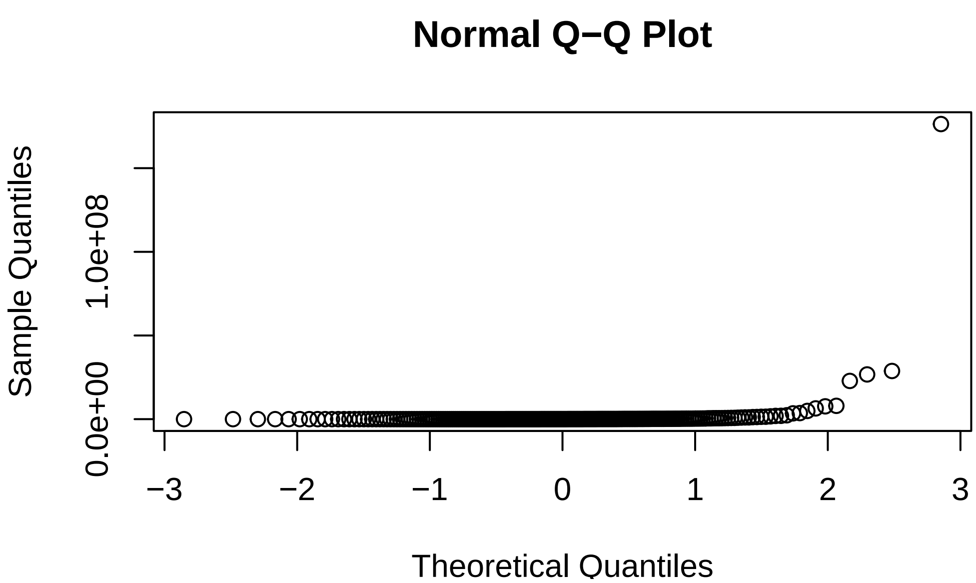
```r
columns2drop <- c("brand", "superbowl_ads_dot_com_url", "youtube_url",
                  "id", "etag", "published_at", "title", "description",
                  "thumbnail", "channel_title", "kind")
youtube_deid <- youtube %>%
  select(-all_of(columns2drop))
dim(youtube_deid)
```

```
[1] 247  14
```

Part b

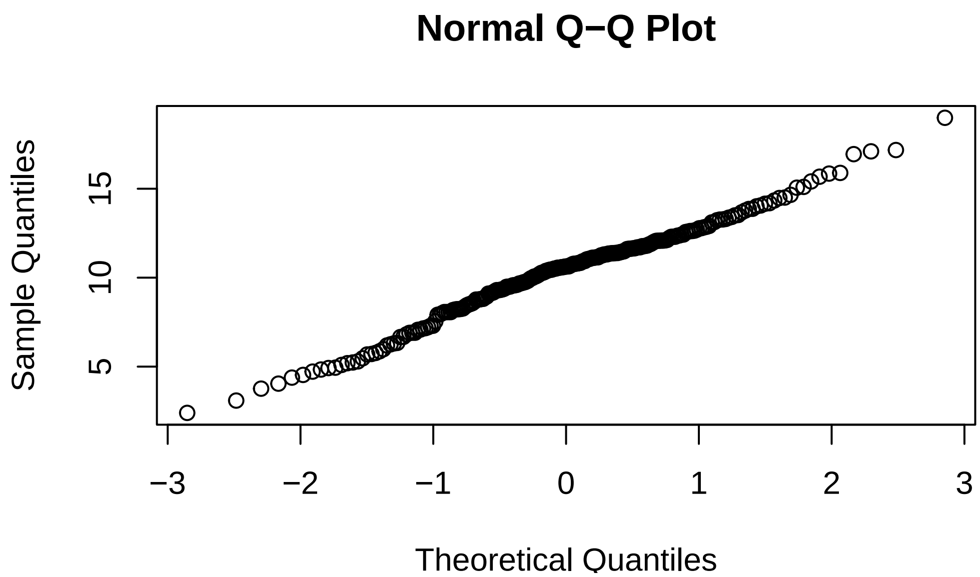
View counts: category 2

```r
qqnorm(youtube_deid$view_count)
```



The View counts columns is heavily right-skewed, and is therefore not suitable to serve as outcome variable directly

```r
qqnorm(log(1 + youtube_deid$view_count))
```

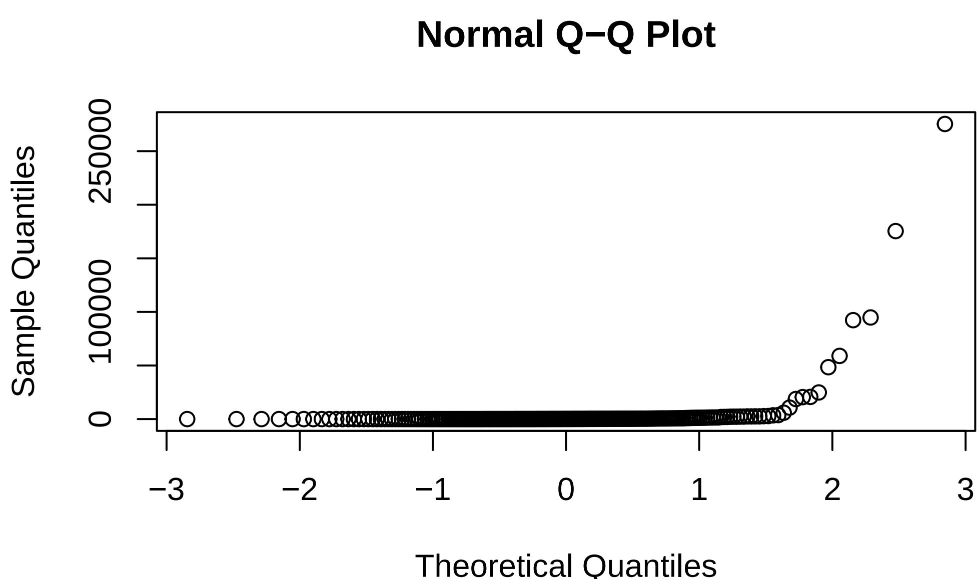


Through transformation log(1 + x), the data is approximately normal, and is therefore appropriate to serve as outcome variable

```r
youtube_deid$view_count <- log(1 + youtube_deid$view_count)
```

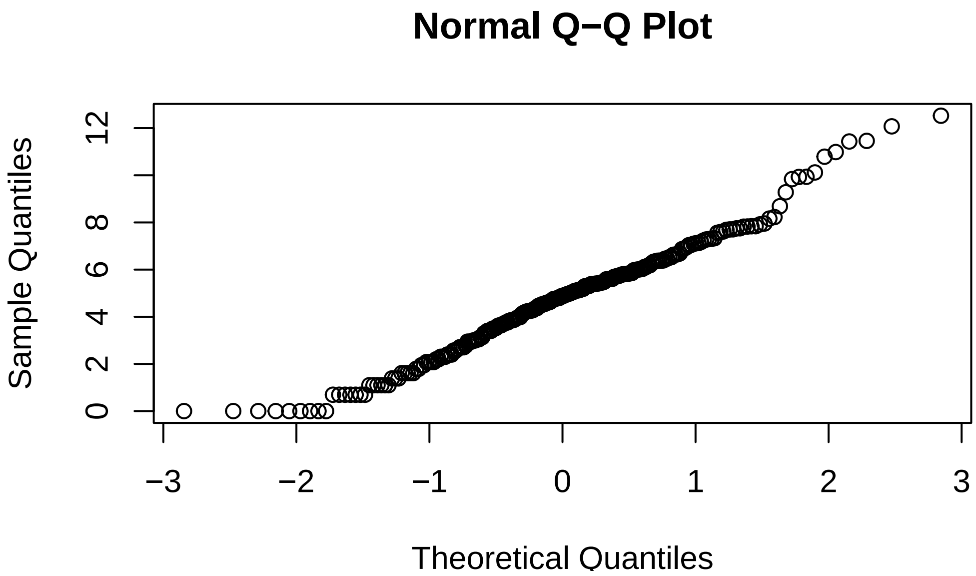
Like counts: category 2

```r
qqnorm(youtube_deid$like_count)
```



The Like counts columns is heavily right-skewed, and is therefore not suitable to serve as outcome variable directly

```
qqnorm(log(1 + youtube_deid$like_count))
```

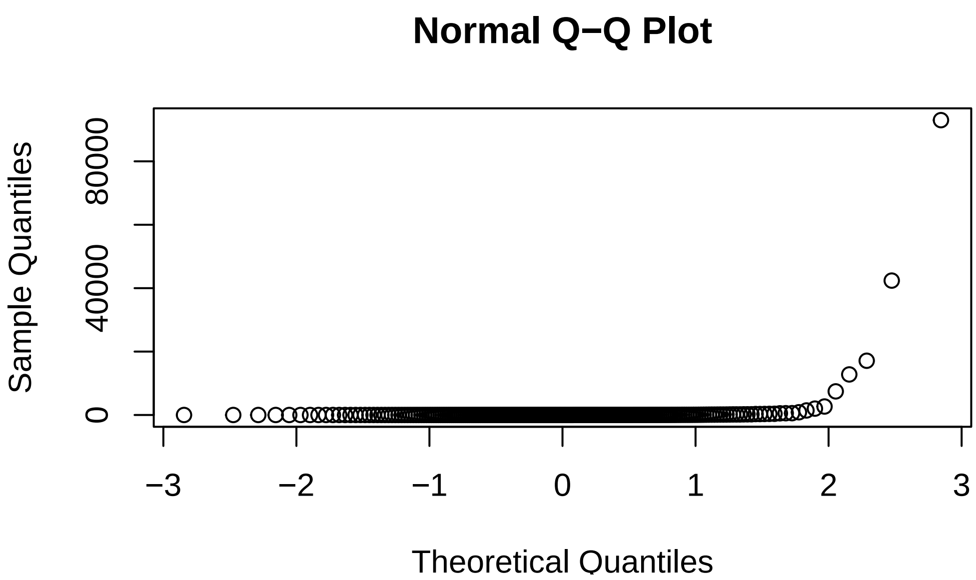


Through transformation log(1 + x), the data is approximately normal, and is therefore appropriate to serve as outcome variable

```
youtube_deid$like_count <- log(1 + youtube_deid$like_count)
```

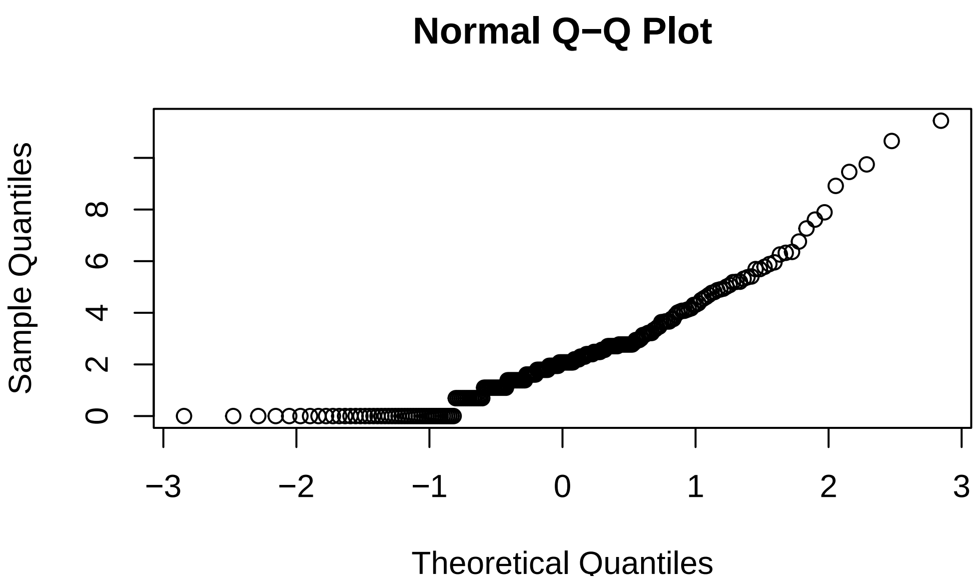
Dislike counts: category 2

```
qqnorm(youtube_deid$dislike_count)
```



The Dislike counts columns is heavily right-skewed, and is therefore not suitable to serve as outcome variable directly

```r
qqnorm(log(1 + youtube_deid$dislike_count))
```



Through transformation log(1 + x), the data is approximately normal, and is therefore appropriate to serve as outcome variable

```r
youtube_deid$dislike_count <- log(1 + youtube_deid$dislike_count)
```

Favorite counts: category 3

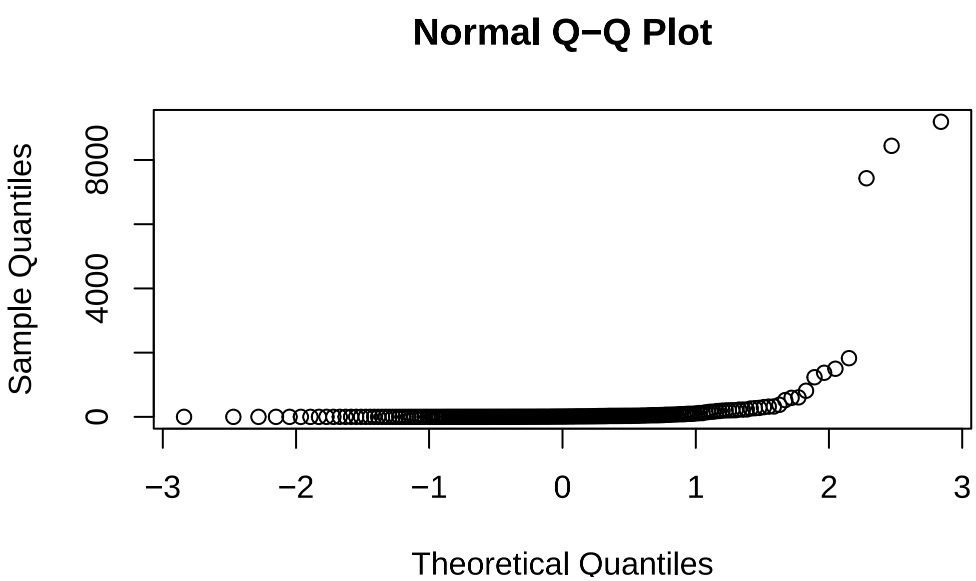
```r
unique(youtube_deid$favorite_count)
```

```
[1]  0 NA
```

The Favorite counts only has 0 and NA values, which is categorical, and this column is therefore not suitable to seve as outcome variable.

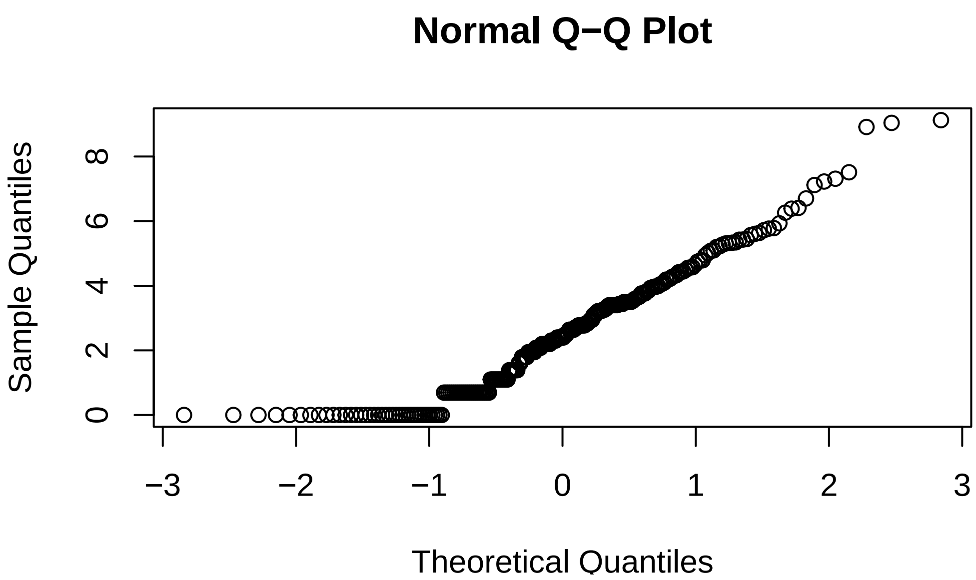
Comment counts: category 2

```r
qqnorm(youtube_deid$comment_count)
```



The Comment counts columns is heavily right-skewed, and is therefore not suitable to serve as outcome variable directly

```r
qqnorm(log(1 + youtube_deid$comment_count))
```



Throught transformation log(1 + x), the data is approximately normal, and is therefore appropriate to serve as outcome variable

```
youtube_deid$comment_count <- log(1 + youtube_deid$comment_count)
```

Part c

```
outcomes <- c("view_count", "like_count", "dislike_count", "comment_count")
predictors <- "funny + show_product_quickly + patriotic +
              celebrity + danger + animals + use_sex + year"
models <- lapply(outcomes, function(y)
{
  formula <- as.formula(paste(y, "~", predictors))
  lm(formula, data = youtube_deid)
})

lapply(models, summary)
```

```
[[1]]

Call:
lm(formula = formula, data = youtube_deid)

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


[[2]]

Call:
lm(formula = formula, data = youtube_deid)

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


[[3]]

Call:
lm(formula = formula, data = youtube_deid)

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


[[4]]

Call:
lm(formula = formula, data = youtube_deid)

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
```

Across the four linear regression models, the advertising characteristics showed little consistent association with YouTube engagement metrics, as most binary flags were statistically insignificant. The only notable effects were that ads featuring danger tended to receive more likes, and patriotic ads showed some evidence of attracting more dislikes and comments, though these results were only marginally significant. In contrast, year was a consistent predictor: newer ads were associated with significantly higher numbers of likes and dislikes, and a marginally higher number of comments, reflecting the overall growth of YouTube engagement over time. However, the explanatory power of all models was low (R^2 values below 0.1), indicating that engagement is likely driven by other unobserved factors such as brand influence, ad content quality, or promotion strategy.

Part d

```r
vars <- c("view_count", "funny","show_product_quickly","patriotic",
          "celebrity","danger","animals","use_sex","year")
# Filter na
dat <- youtube_deid[, vars]
dat <- dat[complete.cases(dat),]
```

```r
form <- ~ funny + show_product_quickly + patriotic + celebrity + danger + animals + use_sex +
X <- model.matrix(form, data = dat)
y <- dat$view_count

XtX   <- crossprod(X)
XtX_i <- solve(XtX)
Xty   <- crossprod(X, y)
beta_hat <- XtX_i %*% Xty

beta_hat
```

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

```r
summary(models[[1]])
```

```
Call:
lm(formula = formula, data = youtube_deid)

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
```