Problem Set 2

Brynn Woolley

Problem 1 - Modified Random walk

Consider a 1-dimensional random walk with the following rules:

1. Start at 0.
2. At each step, move +1 or -1 with 50/50 probability.
3. If +1 is chosen, 5% of the time move +10 instead.
4. If -1 is chosen, 20% of the time move -3 instead.
5. Repeat steps 2-4 (n) times.

(Note that if the +10 is chosen, it's not +1 then +10, it is just +10.)

Write a function to determine the end position of this random walk.

The input and output should be:
* Input: The number of steps
* Output: The final position of the walk

```
> random_walk(10)
 [1]  4

> random_walk(10)
 [1]  -11
```

We're going to implement this in different ways and compare them.

1. Implement the random walk in these three versions:
   - Version 1: using a loop.

- Version 2: using built-in R vectorized functions. (Using no loops.) (Hint: Does the order of steps matter?)

- Version 3: Implement the random walk using one of the "`apply`" functions.

```
# Helper Function
random_walk_fixed <- function(n, draws = NULL) {
  if (is.null(draws)) {
    draws <- runif(2 * n)
  }

  pos <- 0
  for (i in 1:n) {
    dir_draw <- draws[(2 * i - 1)]
    size_draw <- draws[(2 * i)]

    if (dir_draw < 0.5) {
      pos <- pos + ifelse(size_draw < 0.95, 1, 10)
    } else {
      pos <- pos + ifelse(size_draw < 0.80, -1, -3)
    }
  }
  pos
}


# Method 1
random_walk1 <- function(n, draws) {
  pos <- 0
  for (i in 1:n) {
    dir_draw <- draws[(2 * i - 1)]
    size_draw <- draws[(2 * i)]
    if (dir_draw < 0.5) {
      pos <- pos + ifelse(size_draw < 0.95, 1, 10)
    } else {
      pos <- pos + ifelse(size_draw < 0.80, -1, -3)
    }
  }
  pos
}
```

```
# Method 2
random_walk2 <- function(n, draws) {
  dirs <- draws[seq(1, 2 * n, by = 2)]
  sizes <- draws[seq(2, 2 * n, by = 2)]
  steps <- ifelse(
    dirs < 0.5,
    ifelse(sizes < 0.95, 1, 10),
    ifelse(sizes < 0.80, -1, -3)
  )
  sum(steps)
}

# Method 3
random_walk3 <- function(n, draws) {
  sum(sapply(1:n, function(i) {
    dir_draw <- draws[(2 * i - 1)]
    size_draw <- draws[(2 * i)]
    if (dir_draw < 0.5) {
      ifelse(size_draw < 0.95, 1, 10)
    } else {
      ifelse(size_draw < 0.80, -1, -3)
    }
  }))
}
```

```
Demonstrate that all versions work by running the following:

random_walk1(10)\
random_walk2(10)\
random_walk3(10)\
random_walk1(1000)\
random_walk2(1000)\
random_walk3(1000)
```

2. Demonstrate that the three versions can give the same result. Show this for both `n=10` and `n=1000`. (You will need to add a way to control the randomization.)

```
set.seed(123)
draws <- runif(2 * 10)
random_walk1(10, draws)
```

```
[1] 7
```

```r
random_walk2(10, draws)
```

```
[1] 7
```

```r
random_walk3(10, draws)
```

```
[1] 7
```

```r
set.seed(123)
draws <- runif(2 * 1000)
random_walk1(1000, draws)
```

```
[1] 78
```

```r
random_walk2(1000, draws)
```

```
[1] 78
```

```r
random_walk3(1000, draws)
```

```
[1] 78
```

3. Use the microbenchmark package to clearly demonstrate the speed of the implementations. Compare performance with a low input (1,000) and a large input (100,000). Discuss the results.

```r
library(microbenchmark)

# low input: 1,000 steps
set.seed(123)
draws_1k <- runif(2 * 1000)
bench_1k <- microbenchmark(
  loop = random_walk1(1000, draws_1k),
  vectorized = random_walk2(1000, draws_1k),
  apply = random_walk3(1000, draws_1k),
  times = 100
)
print(bench_1k)
```

```
Unit: microseconds
       expr    min     lq     mean  median      uq    max neval
       loop 1175.4 1215.4 1394.901 1272.65 1406.15 4497.8   100
 vectorized  123.5  142.0  176.394  166.75  198.45  302.9   100
      apply 1891.7 2008.1 2220.473 2073.20 2333.75 4932.9   100
```

```r
# large input: 100,000 steps
set.seed(123)
draws_100k <- runif(2 * 100000)
bench_100k <- microbenchmark(
  loop = random_walk1(100000, draws_100k),
  vectorized = random_walk2(100000, draws_100k),
  apply = random_walk3(100000, draws_100k),
  times = 10
)
print(bench_100k)
```

```
Unit: milliseconds
       expr      min       lq      mean   median       uq      max neval
       loop 127.9485 150.0513 166.61329 155.2871 178.7540 241.7026    10
 vectorized   7.4842   8.9625  11.22425  11.4613  12.6390  15.9955    10
      apply 216.8323 222.8684 256.80999 261.7614 270.1942 301.8428    10
```

Discussion: The benchmarking results show that the **vectorized implementation is the fastest**. For 1,000 steps, the vectorized method ran about 10 times faster than the loop and more than 10 times faster than the apply method. For 100,000 steps, the vectorized method was about 15 times faster than the loop and roughly 25 times faster than apply. The loop implementation performed reasonably well but was consistently slower than vectorized. The apply implementation was the slowest due to the overhead of repeated function calls.

In summary:

- Vectorized (fastest, most efficient for large n)

- Loop (moderate performance, simpler to read)

- Apply (slowest, avoid for performance)

4. What is the probability that the random walk ends at 0 if the number of steps is 10? 100? 1000? Defend your answers with evidence based upon a Monte Carlo simulation.

```r
set.seed(123)

# Monte Carlo simulation
simulate_prob_zero <- function(n, sims = 100000) {
  zeros <- replicate(sims, {
    draws <- runif(2 * n)
    random_walk2(n, draws)
  })
  mean(zeros == 0)
}

# Run for different step sizes
prob_10   <- simulate_prob_zero(10)
prob_100  <- simulate_prob_zero(100)
prob_1000 <- simulate_prob_zero(1000)

# Printing all three Monte Carlo probs properly
cat(
  "P(end=0 | n=10)  =", prob_10,  "\n",
  "P(end=0 | n=100) =", prob_100, "\n",
  "P(end=0 | n=1000)=", prob_1000, "\n"
)
```

```
P(end=0 | n=10)  = 0.1323
 P(end=0 | n=100) = 0.01921
 P(end=0 | n=1000)= 0.00575
```

The Monte Carlo simulation estimates the probability of ending at 0 as follows:

- 10 steps: ~13%
- 100 steps: ~2%
- 1,000 steps: ~0.6%

These results show that the probability decreases as the number of steps increases. With only 10 steps, there is still a reasonable chance of returning exactly to 0. By 100 steps, the chance has dropped to about 2%. By 1,000 steps, the probability is extremely small (less than 1%). This matches the intuition that as the walk grows longer, the random variability accumulates and the chance of ending exactly at 0 becomes increasingly small.

Problem 2 - Mean of Mixture of Distributions

The number of cars passing an intersection is a classic example of a Poisson distribution. At a particular intersection, Poisson is an appropriate distribution most of the time, but during rush hours (hours of 8am and 5pm) the distribution is really normally distributed with a much higher mean.

Using a Monte Carlo simulation, estimate the average number of cars that pass an intersection per day under the following assumptions:

- From midnight until 7 AM, the distribution of cars per hour is Poisson with mean 1.
- From 9am to 4pm, the distribution of cars per hour is Poisson with mean 8.
- From 6pm to 11pm, the distribution of cars per hour is Poisson with mean 12.
- During rush hours (8am and 5pm), the distribution of cars per hour is Normal with mean 60 and variance 12

Accomplish this without using any loops.

(Hint: This can be done with extremely minimal code.)

```r
set.seed(123)

simulate_cars_daily <- function(sims=100000) {

  # set hours
  hours <- 0:23

  rush_idx <- hours %in% c(8, 17)

  # poisson means/hour
  lam <- rep(NA_real_, 24)
  lam[hours %in% 0:7]   <- 1
  lam[hours %in% 9:16]  <- 8
  lam[hours %in% 18:23] <- 12
  lam[rush_idx] <- NA_real_

  # draws
    pois_mat <- sapply(lam, function(l) if (is.na(l)) rep(0, sims) else rpois(sims, l))

      rush_mat <- matrix(0, nrow = sims, ncol = 24)
  rush_mat[, rush_idx] <- replicate(sum(rush_idx), rnorm(sims, mean = 60, sd = sqrt(12)))
```

```r
    rowSums(pois_mat + rush_mat)

}
# estimate expected daily count
daily_totals <- simulate_cars_daily(100000)
mean_daily <- mean(daily_totals)
sd_daily   <- sd(daily_totals)

mean_daily; sd_daily
```

```
[1] 264.0163
```

```
[1] 13.01243
```

Problem 3 - Linear Regression

Use the following code to download the YouTube Superbowl commercials data:

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

Information about this data can be found at https://github.com/rfordatascience/tidytuesday/tree/main/data/20
03-02. The research question for this project is to decide which of several attributes, if any, is associated with increased YouTube engagement metrics.

1. Often in data analysis, we need to de-identify it. This is more important for studies of people, but let's carry it out here. Remove any column that might uniquely identify a commercial. This includes but isn't limited to things like brand, any URLs, the YouTube channel, or when it was published.

Report the dimensions of the data after removing these columns.

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

id_like_cols <- c(
  "brand","channel","title","description",
  "super_bowl_ads_dot_com_url","superbowl_ads_dot_com_url",
  "youtube_url","video_url","thumbnail",
  "id","video_id","embed","brand_url","brand_website","channel_url",
```

```
  "published_at","publishedAt","published"
)

keep <- setdiff(names(youtube), intersect(names(youtube), id_like_cols))
yt_deid <- youtube %>% dplyr::select(all_of(keep))

# report rows x cols
dim(yt_deid)
```

```
[1] 247  17
```

2. For each of the following variables, examine their distribution. Determine whether i) The variable could be used as is as the outcome in a linear regression model, ii) The variable can use a transformation prior to being used as the outcome in a linear regression model, or iii) The variable would not be appropriate to use as the outcome in a linear regression model.
   For each variable, report which category it falls in. If it requires a transformation, carry such a transformation out and use that transformation going forward.

```
* View counts
* Like counts
* Dislike counts
* Favorite counts
* Comment counts
```

(Hint: At least the majority of these variables are appropriate to use.)

```
count_vars <- c("view_count","like_count","dislike_count","favorite_count","comment_count")
avail <- intersect(count_vars, names(yt_deid))

# Flag constants (e.g., favorite_count often all zeros)
const_vars <- avail[sapply(avail, function(v) dplyr::n_distinct(yt_deid[[v]], na.rm = TRUE) <
usable     <- setdiff(avail, const_vars)

# Create log1p outcomes for usable variables
for (v in usable) {
  yt_deid[[paste0("log1p_", v)]] <- log1p(yt_deid[[v]])
}

list(usable_outcomes = usable, dropped_as_constant = const_vars)
```

```
$usable_outcomes
[1] "view_count"    "like_count"    "dislike_count" "comment_count"

$dropped_as_constant
[1] "favorite_count"
```

3. For each variable in part b. that are appropriate, fit a linear regression model predicting them based upon each of the seven binary flags for characteristics of the ads, such as whether it is funny. Control for year as a continuous covariate.

Discuss the results. Identify the direction of any statistically significant results.

```r
flag_names <- c("funny","show_product_quickly","patriotic","celebrity","danger","animals","us
flags_in   <- intersect(flag_names, names(yt_deid))

if ("year" %in% names(yt_deid)) yt_deid$year <- as.numeric(yt_deid$year)

outcomes_tr <- paste0("log1p_", usable)

mods <- lapply(outcomes_tr, function(y) {
  form <- as.formula(paste(y, "~ year +", paste(flags_in, collapse = " + ")))
  lm(form, data = yt_deid)
})
names(mods) <- outcomes_tr

model_summary <- purrr::map_df(names(mods), ~ broom::tidy(mods[[.x]]) %>% dplyr::mutate(outc
  dplyr::filter(term != "(Intercept)") %>%
  dplyr::arrange(outcome, p.value) %>%
  dplyr::select(outcome, term, estimate, std.error, statistic, p.value)

model_summary
```

```
# A tibble: 32 x 6
   outcome             term                 estimate std.error statistic p.value
   <chr>               <chr>                   <dbl>     <dbl>     <dbl>   <dbl>
 1 log1p_comment_count year                   0.0503    0.0263     1.91  5.71e-2
 2 log1p_comment_count patrioticTRUE          0.667     0.399      1.67  9.61e-2
 3 log1p_comment_count show_product_quickl~   0.409     0.302      1.35  1.77e-1
 4 log1p_comment_count use_sexTRUE           -0.393     0.332     -1.19  2.37e-1
 5 log1p_comment_count celebrityTRUE          0.298     0.315      0.944 3.46e-1
 6 log1p_comment_count animalsTRUE           -0.268     0.293     -0.913 3.62e-1
 7 log1p_comment_count funnyTRUE              0.220     0.345      0.636 5.26e-1
```

```
 8 log1p_comment_count dangerTRUE           0.178     0.311      0.572 5.68e-1
 9 log1p_dislike_count year                 0.0921    0.0265     3.47  6.26e-4
10 log1p_dislike_count patrioticTRUE        0.814     0.419      1.94  5.31e-2
# i 22 more rows
```

4. Consider only the outcome of view counts. Calculate $\hat{\beta}$ manually (without using `lm`) by first creating a proper design matrix, then using matrix algebra to estimate β. Confirm that you get the same result as `lm` did in part c.

```r
yvar <- "log1p_view_count"
stopifnot(yvar %in% names(yt_deid))

vars_needed <- c(yvar, "year", flags_in)
dat_fit <- yt_deid %>% dplyr::select(all_of(vars_needed)) %>% tidyr::drop_na()

X <- model.matrix(~ year + ., data = dat_fit %>% dplyr::select(-all_of(yvar)))
y <- dat_fit[[yvar]]

# OLS via (X'X)^{-1} X'y
beta_manual <- solve(t(X) %*% X, t(X) %*% y)

# Compare to lm fit using the same X
fit_lm <- lm(y ~ X - 1)   # X already contains intercept
coef_lm <- coef(fit_lm)

manual_vs_lm <- data.frame(
  term = colnames(X),
  beta_manual = as.numeric(beta_manual),
  beta_lm     = as.numeric(coef_lm),
  diff        = as.numeric(beta_manual - coef_lm)
)

manual_vs_lm
```

```
                     term  beta_manual      beta_lm          diff
1             (Intercept) -31.55015804 -31.55015804  2.208189e-10
2                    year   0.02053399   0.02053399 -1.097976e-13
3               funnyTRUE   0.56492445   0.56492445 -3.511635e-13
4 show_product_quicklyTRUE   0.21088918   0.21088918  5.848100e-14
5           patrioticTRUE   0.50699051   0.50699051  1.981748e-13
6           celebrityTRUE   0.03547862   0.03547862  3.074554e-13
7              dangerTRUE   0.63131085   0.63131085  2.398082e-14
```

```
8              animalsTRUE  -0.31001838  -0.31001838  6.838974e-14
9              use_sexTRUE  -0.38670726  -0.38670726 -2.345346e-13
```

For the regression models of `log1p(view_count)`, `log1p(like_count)`, `log1p(dislike_count)`, and `log1p(comment_count)`, the **year variable** showed a consistent **positive and statistically significant effect** for views, likes, and dislikes, indicating engagement has increased over time. None of the seven ad characteristic flags (funny, celebrity, animals, danger, use_sex, patriotic, show_product_quickly) were significant at the 0.05 level. For comments, no predictors reached significance, though year and patriotic were borderline positive effects. Overall, year is the only variable with a clear and consistent association with engagement outcomes.

GitHub Link

- Repo: https://github.com/brynnwoolley/STATS-506#
