Problem Set 1

Brynn Woolley

Problem 1 - Abalone Data

a. Import the data into a `data.frame` in R. Use the information in the "abalone.names" file to give appropriate column names. (Note: Downloading and unzipping the file can take place outside of your submitted document, but importing the file should be in the submission.)

```
# get column names
raw_abalone_text <- read_lines("abalone/abalone.names")
# manually inspected the text to find the column names

# import abalone data
abalone <- read_csv("abalone/abalone.data",
                    col_names = c("Sex", "Length", "Diameter", "Height",
                                  "WholeWeight", "ShuckedWeight",
                                  "VisceraWeight", "ShellWeight", "Rings"))
head(abalone)
```

```
# A tibble: 6 x 9
  Sex   Length Diameter Height WholeWeight ShuckedWeight VisceraWeight
  <chr>  <dbl>    <dbl>  <dbl>       <dbl>         <dbl>         <dbl>
1 M      0.455    0.365  0.095       0.514        0.224         0.101
2 M      0.35     0.265  0.09        0.226        0.0995        0.0485
3 F      0.53     0.42   0.135       0.677        0.256         0.142
4 M      0.44     0.365  0.125       0.516        0.216         0.114
5 I      0.33     0.255  0.08        0.205        0.0895        0.0395
6 I      0.425    0.3    0.095       0.352        0.141         0.0775
# i 2 more variables: ShellWeight <dbl>, Rings <dbl>
```

b. The data contains information on three different sexes of abalone. Report the number of observations belonging to each sex.

```r
table(abalone$Sex)
```

```
   F    I    M
1307 1342 1528
```

Answer: The dataset contains 1,528 males, 1,307 females, and 1,342 infants.

c. Use the data to answer the following questions:

1. Which weight has the highest correlation with rings?

```r
weight_vars <- c("WholeWeight", "ShuckedWeight", "VisceraWeight", "ShellWeight")
cors <- sapply(abalone[weight_vars], function(x) cor(x, abalone$Rings))
cors
```

```
  WholeWeight ShuckedWeight VisceraWeight   ShellWeight
    0.5403897     0.4208837     0.5038192     0.6275740
```

```r
which.max(cors)
```

```
ShellWeight
          4
```

Answer: ShellWeight has the highest correlation with rings (~0.628).

2. For that weight, which sex has the highest correlation?

```r
best_weight <- names(which.max(cors))
abalone %>%
  group_by(Sex) %>%
  summarise(correlation = cor(.data[[best_weight]], Rings))
```

```
# A tibble: 3 x 2
  Sex   correlation
  <chr>       <dbl>
1 F           0.406
2 I           0.725
3 M           0.511
```

Answer: Infants have the highest correlation (0.725).

3. What are the weights of the abalone with the most rings?

```
max_rings <- max(abalone$Rings)
abalone %>%
  filter(Rings == max_rings) %>%
  select(Sex, all_of(weight_vars), Rings)
```

```
# A tibble: 1 x 6
  Sex   WholeWeight ShuckedWeight VisceraWeight ShellWeight Rings
  <chr>       <dbl>         <dbl>         <dbl>       <dbl> <dbl>
1 F            1.81         0.706         0.322       0.475    29
```

Answer: The abalone with the most rings (29) is female with weights Whole=1.81, Shucked=0.706, Viscera=0.322, Shell=0.475.

4. What percentage of abalones have a viscera weight larger than their shell weight?

```
mean(abalone$VisceraWeight > abalone$ShellWeight) * 100
```

```
[1] 6.511851
```

Answer: About 6.5% of abalones have viscera weight larger than shell weight.

d. Create a table of correlations between weights and rings, within each sex. The columns should be the four weights, and the rows should be the sexes. (This table does not need to be "fancy" but should clearly identify what each value represents.)

```
abalone %>%
  group_by(Sex) %>%
  summarise(across(c(WholeWeight, ShuckedWeight, VisceraWeight, ShellWeight),
                   ~cor(Rings, .)))
```

```
# A tibble: 3 x 5
  Sex   WholeWeight ShuckedWeight VisceraWeight ShellWeight
  <chr>       <dbl>         <dbl>         <dbl>       <dbl>
1 F           0.267        0.0948         0.212       0.406
2 I           0.696         0.620         0.673       0.725
3 M           0.372         0.222         0.321       0.511
```

Answer: Correlations vary by sex, with infants generally showing stronger relationships than males or females.

e. Carry out a series of t-tests to examine whether the number of rings differs across the three sexes. Present the R output and interpret the results. (You may use an existing R function to carry out the t-test, or for minor extra credit, manually write your own calculation of the t-test p-values.)

```
pairwise.t.test(abalone$Rings, abalone$Sex)
```

```
    Pairwise comparisons using t tests with pooled SD

data:  abalone$Rings and abalone$Sex

  F      I
I <2e-16 -
M 1e-04  <2e-16

P value adjustment method: holm
```

Answer: The differences in ring counts between sexes are statistically significant ($p < 0.001$ for most comparisons).

Problem 2 - Food Expenditure Data

a. Import the data into a `data.frame` in R. As with the abalone data, you may download the data outside of your submission, but importation should take place inside the problem set submission.

```
# import food expenditure data
foodexp <- read_csv("food_expenditure.csv")

head(foodexp)
```

```
# A tibble: 6 x 12
     ID `What is your age?` How many individuals live i~1 What state do you li~2
  <dbl>               <dbl>                        <dbl> <chr>
1     1                  68                            7 LA
2     2                  88                            5 WA
3     3                  82                            3 MS
```

```
4     4                  73                              8 AK
5     5                  89                              0 IN
6     6                  18                              6 WI
# i abbreviated names:
#   1: `How many individuals live in your household for which you are responsible for food ex
#   2: `What state do you live in?`
# i 8 more variables:
#   `What currency are you reporting your food expenditures in?` <chr>,
#   `What was your total food expenditure in the last week?` <chr>,
#   `What was your total food expenditures at grocery stores in the last week?` <dbl>, ...
```

b. Clean up the variable names. Simplify them.

```
# inspect original variable names
colnames(foodexp)
```

```
 [1] "ID"
 [2] "What is your age?"
 [3] "How many individuals live in your household for which you are responsible for food expe
 [4] "What state do you live in?"
 [5] "What currency are you reporting your food expenditures in?"
 [6] "What was your total food expenditure in the last week?"
 [7] "What was your total food expenditures at grocery stores in the last week?"
 [8] "What was your food expenditure while dining out in the last week?"
 [9] "What was your food expenditure (miscellaneous) in the last week?"
[10] "How many times did you dine out last week?"
[11] "Are you including alcohol in your food expenditures?"
[12] "What food assistance programs, if any, did you use for your food expenditures last week
```

```
# simplify names: lowercase, remove punctuation, replace spaces w/ underscores
foodexp <- foodexp %>%
  janitor::clean_names()

# simplify further
foodexp <- foodexp %>%
  rename(
    age              = what_is_your_age,
    household_size   = how_many_individuals_live_in_your_household_for_which_you_are_respons
    state            = what_state_do_you_live_in,
    currency         = what_currency_are_you_reporting_your_food_expenditures_in,
    expense_total    = what_was_your_total_food_expenditure_in_the_last_week,
    expense_grocery  = what_was_your_total_food_expenditures_at_grocery_stores_in_the_last_we
```

```
    expense_dining   = what_was_your_food_expenditure_while_dining_out_in_the_last_week,
    expense_misc     = what_was_your_food_expenditure_miscellaneous_in_the_last_week,
    count_dining_out = how_many_times_did_you_dine_out_last_week,
    alcohol_included = are_you_including_alcohol_in_your_food_expenditures,
    food_assistance  = what_food_assistance_programs_if_any_did_you_use_for_your_food_expendi
  )

# inspect column names
colnames(foodexp)
```

```
 [1] "id"               "age"              "household_size"   "state"
 [5] "currency"         "expense_total"    "expense_grocery"  "expense_dining"
 [9] "expense_misc"     "count_dining_out" "alcohol_included" "food_assistance"
```

c. Restrict the data to those paying in US dollars (USD). Show that it worked by confirming the number of observations before and after restricting the data.

```
# count observations before filtering
n_before <- nrow(foodexp)

# filter to paying in USD
foodexp_usd <- foodexp %>%
  filter(currency == "USD")

# count of observations after filtering
n_after <- nrow(foodexp_usd)

cat("Observations before filtering:", n_before, "\n")
```

```
Observations before filtering: 262
```

```
cat("Observations after filtering (USD only):", n_after, "\n")
```

```
Observations after filtering (USD only): 230
```

```
unique(foodexp_usd$currency)
```

```
[1] "USD"
```

Answer: 262 observations were in the dataset before filtering, and 230 remained after restricting to USD.

There are a number of issues with this data, likely due to the self-reported nature. For each of the following variables, clean them by removing any row with inappropriate data. For each variable, explain your rules for eliminating rows. For example, for the age variable, you might state "Excluded all minors under the age of 18". (Note that there is no "right" answer here, the goal is to i) choose reasonable rules and ii) carry out the corresponding code.)

d. The variable related to age.

```
# exclude minors (<18) and implausibly old ages (>115)
foodexp_usd <- foodexp_usd %>%
  filter(age >= 18, age <= 115)
```

e. The variable related to state.

```
# remove cases w/ state missing or reported as "XX" (invalid state responses)
foodexp_usd <- foodexp_usd %>%
  filter(!is.na(state), state != "XX")
```

f. The four variables related to food expenditures.

```
# drop rows w/ negative expenditures & exclude unlikely totals > $5000 per week
foodexp_usd <- foodexp_usd %>%
  filter(
    expense_total   >= 0,
    expense_grocery >= 0,
    expense_dining  >= 0,
    expense_misc    >= 0,
    expense_total   <= 5000
  )
```

g. The variable related to number of times dining out.

```
# keep only non-negative values & exclude more than 30 dining-out occasions in a week (more t
foodexp_usd <- foodexp_usd %>%
  filter(count_dining_out >= 0, count_dining_out <= 30)
```

h. Report your final number of observations after this cleaning.

```
cat("Final number of observations after cleaning:", nrow(foodexp_usd))
```

```
Final number of observations after cleaning: 82
```

Answer: After all cleaning, 82 observations remained.

Problem 3 - Collatz conjecture

The Collatz conjecture is an unsolved problem in mathematics that asks whether repeating two simple operations on a sequence of numbers will eventually reach 1 for positive integers. The operations are:

$$f(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases}$$

For example, for input 5, the sequence is: 5, 16, 8, 4, 2, 1. It has been shown that all integers up to (10^{21}) eventually reach 1.

a. Write function `nextCollatz` that given a positive integer, computes the next number in its Collatz sequence. Be sure to provide a reasonable error on an invalid input. Be sure to document your function (see instructions above).

   - Input: A positive integer
   - Output: A positive integer

   E.g.,
   > nextCollatz(5)
   [1] 16
   > nextCollatz(16)
   [1] 8

   Demonstrate it works by reproducing the examples.

```
#' Compute the next number in the Collatz sequence
#'
#' Given a positive integer n, this function returns the next number
#' in its Collatz sequence:
#'   - If n is even, the result is n / 2
#'   - If n is odd, the result is 3n + 1
#'
#' @param n A positive integer
#'
#' @return A positive integer, the next value in the Collatz sequence
```

```r
#'
#' @examples
#' nextCollatz(5)   # returns 16
#' nextCollatz(16)  # returns 8
nextCollatz <- function(n) {
  if (!is.numeric(n) || n <= 0) {
    stop("Input must be a positive integer.")
  }

  if (n %% 2 == 0) {
    n / 2
  } else {
    3 * n + 1
  }
}


# examples
nextCollatz(5)   # 16
```

```
[1] 16
```

```r
nextCollatz(16)  # 8
```

```
[1] 8
```

b. Create a function `collatzSequence` that returns the Collatz sequence for a given input. Use your `nextCollatz` function to perform the calculation. Be sure to provide a reasonable error on an invalid input. Be sure to document your function (see instructions above).

- Input: A positive integer
- Output: A list containing the vector of the entries in the Collatz sequence, beginning at the input and ending at 1; and the length of the Collatz sequence.

E.g.,

collatzSequence(5)
[1] 5 16 8 4 2 1
collatzSequence(19)
[1] 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Demonstrate it works by reproducing the examples.

```
#' Generate a Collatz sequence
#'
#' Given a positive integer n, this function computes the full Collatz sequence
#' starting at n and ending at 1. The sequence is generated using the
#' `nextCollatz` function iteratively.
#'
#' @param n A positive integer
#'
#' @return A numeric vector containing the full Collatz sequence from n to 1
#'
#' @examples
#' collatzSequence(5)   # returns c(5, 16, 8, 4, 2, 1)
#' collatzSequence(19)  # returns c(19, 58, 29, 88, 44, 22, 11, ..., 1)
collatzSequence <- function(n) {
  if (!is.numeric(n) || n <= 0) {
stop("Input must be a positive integer.")
  }

  seq_vals <- n
  while (tail(seq_vals, 1) != 1) {
seq_vals <- c(seq_vals, nextCollatz(tail(seq_vals, 1)))
  }

  seq_vals
}

# examples
collatzSequence(5)   # 5 16 8 4 2 1
```

```
[1]  5 16  8  4  2  1
```

```
collatzSequence(19)  # 19 58 29 88 44 22 11 ...
```

```
 [1] 19 58 29 88 44 22 11 34 17 52 26 13 40 20 10  5 16  8  4  2  1
```

c. Use these functions to find the shortest and longest Collatz sequence starting with values between 100 and 500, inclusive. In the case of ties, report the lowest starting value.

```
# calc lengths of collatz sequences for 100:500
seq_lengths <- sapply(100:500, function(n) length(collatzSequence(n)))

# find shortest & longest
```

```r
min_start <- which.min(seq_lengths) + 99  # +99 because index 1 = 100
max_start <- which.max(seq_lengths) + 99

cat("Shortest sequence starts at:", min_start,
    "with length", seq_lengths[min_start - 99], "\n")
```

```
Shortest sequence starts at: 128 with length 8
```

```r
cat("Longest sequence starts at:", max_start,
    "with length", seq_lengths[max_start - 99], "\n")
```

```
Longest sequence starts at: 327 with length 144
```

GitHub Link

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