HW1 Solutions

Github link: https://github.com/atpabuser/STATS-506-repo/tree/main/Homework1

Problem 1:

1a

```
#1a. Import the data and give appropriate column names.

#read abalone data file
abalone_df <- read.table("C:/Users/hyperion/University of Michigan Dropbox/Deepan Islam/Fall

#Assign column names according to abalone.names
colnames(abalone_df) <- c("Sex", "Length", "Diameter", "Height", "Whole_Weight", "Shucked_Wei

head(abalone_df) #lets take a prelim look.
```

```
  Sex Length Diameter Height Whole_Weight Shucked_Weight Viscera_Weight
1   M  0.455    0.365  0.095       0.5140         0.2245         0.1010
2   M  0.350    0.265  0.090       0.2255         0.0995         0.0485
3   F  0.530    0.420  0.135       0.6770         0.2565         0.1415
4   M  0.440    0.365  0.125       0.5160         0.2155         0.1140
5   I  0.330    0.255  0.080       0.2050         0.0895         0.0395
6   I  0.425    0.300  0.095       0.3515         0.1410         0.0775
  Shell_Weight Rings
1        0.150    15
2        0.070     7
3        0.210     9
4        0.155    10
5        0.055     7
6        0.120     8
```

1b

```
sex_counts <-table(abalone_df$Sex)
sex_counts
```

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

```
males_count <-sum(abalone_df$Sex=="M")
females_count <-sum(abalone_df$Sex=="F")
infants_count <-sum(abalone_df$Sex=="I")

cat("\nThe number of males is:",males_count,
    "\nThe number of females is", females_count,
    "\nThe number of infants is", infants_count)
```

```

The number of males is: 1528 
The number of females is 1307 
The number of infants is 1342
```

1c (1)

Which weight has the highest correlation with rings?

```
#get weights from dataframe
weights <- names(abalone_df)[5:8]

#do correlations with ring
weight_correlations <-sapply(abalone_df[weights], function(x) cor(x, abalone_df$Rings))

#find which weight has the highest correlation
highest_corr_weight <-names(which.max(weight_correlations))

cat("The weight with the highest correlation to rings is:", highest_corr_weight, "and the cor
```

```
The weight with the highest correlation to rings is: Shell_Weight and the correlation value i
```

1c (2)

For that weight, which sex has the highest correlation?

```
#We know the weight with the highest correlation to rings is the Shell-Weight.

correlations_by_sex <- sapply(c("F", "I", "M"), function(s) {
  subset_data <- abalone_df[abalone_df$Sex == s, ]
  cor(subset_data$Shell_Weight, subset_data$Rings)
})

print(correlations_by_sex)
```

```
        F         I         M
0.4059070 0.7254357 0.5109967
```

```
highest_corr_sex <- names(which.max(correlations_by_sex))
max_corr <- max(correlations_by_sex)

cat("\nFor the shell weights,", highest_corr_sex, "has the highest correlation to rings with
```

```

For the shell weights, I has the highest correlation to rings with a value of  0.7254357
```

1c (3)

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

```
# Find max number of rings
max_rings <- max(abalone_df$Rings)
rows_max  <- which(abalone_df$Rings == max_rings)

cat("Max number of rings:", max_rings, "\n")
```

```
Max number of rings: 29
```

```
# Get all abalones with max rings
max_rings_data <- abalone_df[abalone_df$Rings == max_rings,]

# Display their weights
cat("Abalone(s) with most rings (", max_rings, " rings):\n", sep = "")
```

```
Abalone(s) with most rings (29 rings):
```

```r
print(abalone_df[rows_max, c("Sex", weights), drop = FALSE])
```

```
    Sex Whole_Weight Shucked_Weight Viscera_Weight Shell_Weight
481   F       1.8075         0.7055         0.3215        0.475
```

1c (4)

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

```r
viscera_weight_comp <- mean(abalone_df$Viscera_Weight>abalone_df$Shell_Weight)*100

cat("Percentage with viscera weight > shell weight:", round(viscera_weight_comp, 3), "%")
```

```
Percentage with viscera weight > shell weight: 6.512 %
```

1d

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.)

```r
# Create correlation table

sexes <-names(table(abalone_df$Sex))
#weights were defined before

corr_table <-matrix(nrow = length(sexes), ncol=length(weights))
rownames(corr_table) <- sexes
colnames(corr_table) <- weights

# populate the table
for(sx in sexes) {
  subset_data <- abalone_df[abalone_df$Sex == sx, ]
  for(weight in weights) {
    corr_table[sx, weight] <- cor(subset_data[[weight]], subset_data$Rings)
  }
}
```

```
# Display result
print(corr_table)
```

```
  Whole_Weight Shucked_Weight Viscera_Weight Shell_Weight
F    0.2667585     0.09484802      0.2116154    0.4059070
I    0.6963268     0.62024577      0.6732727    0.7254357
M    0.3721966     0.22239382      0.3209535    0.5109967
```

1e

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.)

So we need to do the following t-tests: 1. Female vs Infant 2. Female vs Male 3. Infant vs Male

```
#let's define the rings here so we don't need to retype over and over
female_rings<-abalone_df[abalone_df$Sex=="F","Rings"]
infant_rings<-abalone_df[abalone_df$Sex=="I","Rings"]
male_rings<-abalone_df[abalone_df$Sex=="M","Rings"]

#Test 1: Female vs Infant
t_test_FI <- t.test(female_rings, infant_rings)
print(t_test_FI)
```

```

    Welch Two Sample t-test

data:  female_rings and infant_rings
t = 29.477, df = 2508.9, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 3.023380 3.454304
sample estimates:
mean of x mean of y
11.129304  7.890462
```

```
#Test 2: Female vs Male
t_test_FM <- t.test(female_rings, male_rings)
print(t_test_FM)
```

```

    Welch Two Sample t-test

data:  female_rings and male_rings
t = 3.6657, df = 2742.4, p-value = 0.0002514
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1971045 0.6505082
sample estimates:
mean of x mean of y
  11.1293   10.7055
```

```
#Test 3: Infant vs Male
t_test_IM <- t.test(infant_rings,male_rings)
print(t_test_IM)
```

```

    Welch Two Sample t-test

data:  infant_rings and male_rings
t = -27.221, df = 2859, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.017808 -2.612263
sample estimates:
mean of x mean of y
 7.890462 10.705497
```

```
#analysis of p-value

#Case 1:
if (t_test_FI$p.value<0.05) {
  cat("There is a statistically significant difference between Female and Infant ring counts
} else {
  cat("There is no statistically significant difference between Female and Infant ring counts
}
```

```
There is a statistically significant difference between Female and Infant ring counts.
```

```
#Case 2:
if (t_test_FM$p.value<0.05) {
  cat("There is a statistically significant difference between Female and Male ring counts\n"
} else {
  cat("There is no statistically significant difference between Female and Male ring counts."
}
```

```
There is a statistically significant difference between Female and Male ring counts
```

```
#Case 3:
if (t_test_IM$p.value<0.05) {
  cat("There is a statistically significant difference between Infant and Male ring counts.\n
} else {
  cat("There is no statistically significant difference between Infant and Male ring counts."
}
```

```
There is a statistically significant difference between Infant and Male ring counts.
```

Problem 2

2a

```
#import dataset
food_data <- read.csv("C:/Users/hyperion/University of Michigan Dropbox/Deepan Islam/Fall 202

#(food_data)
```

2b

Clean up the variable names. Simplify them.

```
#cleanup the variable names

colnames(food_data)<- c("ID", "Age", "Household_Size", "State", "Currency",
                  "Total_Food_Exp", "Grocery_Exp", "Dining_Out_Exp","Misc_Food_Exp","Dining_Ou

head(food_data)
```

```
  ID Age Household_Size State Currency Total_Food_Exp Grocery_Exp
1  1  68              7    LA      USD         436.35      168.59
2  2  88              5    WA      USD                     452.10
3  3  82              3    MS      USD          279.1      301.66
4  4  73              8    AK      USD         -20.98      139.66
5  5  89              0    IN      USD         494.87          NA
6  6  18              6    WI      EUR         276.32      394.44
  Dining_Out_Exp Misc_Food_Exp Dining_Out_Count Alcohol_Included
1         140.71        109.77                4              Yes
2         192.94            NA                1          Unknown
3         239.84        103.94                9              Yes
4          69.19         44.84                2          Unknown
5         191.72        172.31                3              Yes
6         283.20        114.06                6          Unknown
  Food_Assistance
1            None
2            SNAP
3            None
4            None
5            None
6     Food Pantry
```

2c

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 before restriction
n_before <- nrow(food_data)
cat("Initial number of observations is:",n_before,"\n")
```

```
Initial number of observations is: 262
```

```
#restriction
food_data<-food_data[food_data$Currency=="USD",]
n_after <- nrow(food_data)
cat("After restricting to USD, number of observations is:",n_after,"\n")
```

```
After restricting to USD, number of observations is: 230
```

2d We will only consider people older than 18 but less than 100. BUT WE HAVE NA values. How do we deal with them? I googled this and found this reference: https://uc-r.github.io/missing_values

We can use `is.na()`.

```r
food_data <- food_data[food_data$Age >= 18 & food_data$Age <= 100 & !is.na(food_data$Age), ]
cat("After age cleaning, number of observations left is:", nrow(food_data), "\n")
```

```
After age cleaning, number of observations left is: 196
```

2e. The variable related to state.

Got to deal with missing states, invalid states and empty strings. Talking to gpt => R has a inbuilt state.abb. we can check `food_data$State` against it to make sure all state abbreviations are valid.

`%in%` inspiration: https://www.r-bloggers.com/2022/07/how-to-use-in-operator-in-r/

```r
valid_states <- state.abb
food_data <- food_data[ !is.na(food_data$State) &
                        food_data$State != "" &
                        food_data$State %in% valid_states, ]
cat("After state cleaning, number of observations left is:", nrow(food_data), "\n")
```

```
After state cleaning, number of observations left is: 176
```

2f. The four variables related to food expenditures.

Rules: Only consider positive values, discard missing values, discard any values above 3000/week for all 4 variables. Doesn't make sense to spend $3000 a week on any of them.

NOTE THAT *TOTAL_FOOD_EXP* is stored as a character! we got to make it numerical. Basically the R equivalent of casting.

```r
food_data$Total_Food_Exp   <- as.numeric(food_data$Total_Food_Exp)
```

```
Warning: NAs introduced by coercion
```

```
food_data$Grocery_Exp      <- as.numeric(food_data$Grocery_Exp)
food_data$Dining_Out_Exp   <- as.numeric(food_data$Dining_Out_Exp)
food_data$Misc_Food_Exp    <- as.numeric(food_data$Misc_Food_Exp)
```

```
#Cleaning time

# Total_Food_Exp
food_data <- food_data[ !is.na(food_data$Total_Food_Exp) &
                         food_data$Total_Food_Exp >= 0 &
                         food_data$Total_Food_Exp <= 3000, ]
cat("After Total_Food_Exp clean, n is:", nrow(food_data), "\n")
```

```
After Total_Food_Exp clean, n is: 157
```

```
# Grocery_Exp
food_data <- food_data[ !is.na(food_data$Grocery_Exp) &
                         food_data$Grocery_Exp >= 0 &
                         food_data$Grocery_Exp <= 3000, ]
cat("After Grocery_Exp clean, n is:", nrow(food_data), "\n")
```

```
After Grocery_Exp clean, n is: 154
```

```
# Dining_Out_Exp
food_data <- food_data[ !is.na(food_data$Dining_Out_Exp) &
                         food_data$Dining_Out_Exp >= 0 &
                         food_data$Dining_Out_Exp <= 3000, ]
cat("After Dining_Out_Exp clean, n is:", nrow(food_data), "\n")
```

```
After Dining_Out_Exp clean, n is: 151
```

```
food_data <- food_data[ !is.na(food_data$Misc_Food_Exp) &
                         food_data$Misc_Food_Exp >= 0 &
                         food_data$Misc_Food_Exp <= 3000, ]
cat("After Misc_Food_Exp clean. n is:", nrow(food_data), "\n")
```

```
After Misc_Food_Exp clean. n is: 122
```

2g

Dining out cleaning

Value has to be greater than 0 and less than or equal to 21 i.e. 3 meals a day *7 = 21

```
food_data <- food_data[ !is.na(food_data$Dining_Out_Count) &
                         food_data$Dining_Out_Count >= 0 &
                         food_data$Dining_Out_Count <= 21, ]

cat("After the dining out cleaning, n is:", nrow(food_data), "\n")
```

```
After the dining out cleaning, n is: 119
```

2h

The final number of observations is 119.

Problem 3: Collatz Conjecture

3a

Define the function `nextCollatz`

```
#' Function to compute the next number in its Collatz sequence
#' Collatz sequence: n/2 if n is even or 3*n+1 if n is odd
#'
#' @param n A single positive integer
#' @return A single positive integer, the next Collatz value.
#'

nextCollatz <- function(n) {
  if (!is.numeric(n) || length(n) != 1 || n <= 0 || n != floor(n)) {
    stop("n is not a single positive integer")
  } #If n is even divide by 2 or do 3n+1
  if (n %% 2 == 0) {
    return(n / 2)
  } else {
    return(3 * n + 1)
  }
```

```r
}

#Example
nextCollatz(5)
```

```
[1] 16
```

```r
nextCollatz(16)
```

```
[1] 8
```

3b

Create a function `collatzSequence` that returns the Collatz sequence for a given input.

```r
#' Function to return the Collatz sequence for a given input.
#' Applies nextCollatz() repeatedly to produce the full Collatz sequence from starting value
#' @param n A single positive integer
#' @return A list containing the vector of the entries in the Collatz sequence beginning at
#'
#' Example: collatzSequence(5)$sequence should return [5, 16, 8, 4, 2, 1]
#'          collatzSequence(5)$length should return 6

collatzSequence <- function(n) {
  # input validation (same as above)
  if (!is.numeric(n) || length(n) != 1 || n <= 0 || n != floor(n)) {
    stop("n is not a single positive integer")
  }
  seq_vals <- n
  # build the sequence from n to 1
  while (n != 1) {
    n <- nextCollatz(n)
    seq_vals <- c(seq_vals, n)
  }
  return(list(sequence = seq_vals, length = length(seq_vals)))
}

collatzSequence(5)
```

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

$length
[1] 6
```

```r
collatzSequence(19)
```

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

$length
[1] 21
```

3c

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.

```r
start_vals <- 100:500
lengths <- sapply(start_vals, function (n) collatzSequence(n)$length)

#Find shortest sequence length and the smallest n that produces this
min_length <- min(lengths)
min_n <- (start_vals)[which(lengths==min_length)[1]]

#Find longest sequence length and smallest n that produces this
max_length <- max(lengths)
max_n <- (start_vals)[which(lengths==max_length)[1]]

cat("The shortest sequence length is:",min_length, "and starts at ",min_n,"\n")
```

```
The shortest sequence length is: 8 and starts at  128
```

```r
cat("The longest sequence length is:", max_length, "and starts at", max_n, "\n")
```

```
The longest sequence length is: 144 and starts at 327
```