HW1

Yunqi Song

Github link

https://github.com/SONG-Yunqi/STATS-506-HW1/

Problem 1

(a)

Below I load the data as data.frame and name the columns.

```
abalone = read.table('abalone.data',sep = ',')
names(abalone) = c('sex','length','diam','height','whole','shucked',
                   'viscera','shell','rings')
head(abalone)
```

```
  sex length  diam height  whole shucked viscera shell rings
1   M  0.455 0.365  0.095 0.5140  0.2245  0.1010 0.150    15
2   M  0.350 0.265  0.090 0.2255  0.0995  0.0485 0.070     7
3   F  0.530 0.420  0.135 0.6770  0.2565  0.1415 0.210     9
4   M  0.440 0.365  0.125 0.5160  0.2155  0.1140 0.155    10
5   I  0.330 0.255  0.080 0.2050  0.0895  0.0395 0.055     7
6   I  0.425 0.300  0.095 0.3515  0.1410  0.0775 0.120     8
```

(b)

```
table(abalone$sex)
```

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

The number of female is 1307. The number of infant is 1342. The number of male is 1528.

(c)

```
cor(abalone[5:9])
```

```
            whole   shucked   viscera     shell     rings
whole   1.0000000 0.9694055 0.9663751 0.9553554 0.5403897
shucked 0.9694055 1.0000000 0.9319613 0.8826171 0.4208837
viscera 0.9663751 0.9319613 1.0000000 0.9076563 0.5038192
shell   0.9553554 0.8826171 0.9076563 1.0000000 0.6275740
rings   0.5403897 0.4208837 0.5038192 0.6275740 1.0000000
```

We can see that shell weight has the highest correlation with rings.

```
for (s in c('F','I','M')){
  data = abalone[abalone$sex == s,]
  corr = cor(data[c(8,9)])[1,2]
  print(paste(s,':',corr))
}
```

```
[1] "F : 0.405907019837357"
[1] "I : 0.725435674284461"
[1] "M : 0.510996723795907"
```

Infant has the highest correlation.

```
abalone[which.max(abalone$rings),]
```

```
    sex length  diam height  whole shucked viscera shell rings
481   F    0.7 0.585  0.185 1.8075  0.7055  0.3215 0.475    29
```

For the abalone with the most rings, its whole weight, shucked weight, viscera weight and shell weight are 1.81, 0.71, 0.32, 0.48, respectively.

```
nrow(abalone[abalone$viscera > abalone$shell,])/nrow(abalone)
```

```
[1] 0.06511851
```

6.51% abalones have a viscera weight larger than shell weight.

(d)

```
female_cors = cor(abalone[abalone$sex == 'F',5:9])[5,1:4]
male_cors = cor(abalone[abalone$sex == 'M',5:9])[5,1:4]
infant_cors = cor(abalone[abalone$sex == 'I',5:9])[5,1:4]

cor_table = rbind(female_cors,male_cors,infant_cors)
rownames(cor_table) = c('female','male','infant')

cor_table
```

```
           whole    shucked   viscera     shell
female 0.2667585 0.09484802 0.2116154 0.4059070
male   0.3721966 0.22239382 0.3209535 0.5109967
infant 0.6963268 0.62024577 0.6732727 0.7254357
```

The table is shown above.

(e)

```
t.test(abalone[abalone$sex == 'F','rings'], abalone[abalone$sex == 'M','rings'])
```

```
    Welch Two Sample t-test

data:  abalone[abalone$sex == "F", "rings"] and abalone[abalone$sex == "M", "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
```

```
t.test(abalone[abalone$sex == 'F','rings'], abalone[abalone$sex == 'I','rings'])
```

```
    Welch Two Sample t-test

data:  abalone[abalone$sex == "F", "rings"] and abalone[abalone$sex == "I", "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
```

```
t.test(abalone[abalone$sex == 'I','rings'], abalone[abalone$sex == 'M','rings'])
```

```
    Welch Two Sample t-test

data:  abalone[abalone$sex == "I", "rings"] and abalone[abalone$sex == "M", "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
```

Above there are three t-test results. The first is about female and male, the second is about female and infant, and the third is about infant and male. All the three results give a p-value smaller than 0.05. This means that under the confidence level = 5%, the number of rings are significantly different across the three sexes.

Problem 2

(a)

```
food = read.csv('food_expenditure.csv')
```

The data has been loaded as a data.frame 'food'.

(b)

```
names(food) = c('id','age','individual','state','currency','total','grocery','dining','miscel
```

(c)

```
print(paste('before restriction:',nrow(food)))
```

```
[1] "before restriction: 262"
```

```
food = food[food$currency == 'USD',]
print(paste('after restriction:',nrow(food)))
```

```
[1] "after restriction: 230"
```

(d)

I will remove all the rows with age smaller than 18 and greater than 90 to avoid some extreme ages. Also, I will remove rows with age = null value.

```
food = food[(food$age >= 18) & (food$age <= 90) & (!is.na(food$age)),]
```

(e)

I will remove the rows with state = null value.

```
food = food[(food$state != '') & (!is.na(food$state)),]
```

(f)

First, I will remove the rows where expenditure value is null value or is not a number. Second, I will remove rows with non-positive expenditure value. Third, I will remove the rows where the total expenditure is less than the sum of other three expenditure.

```
food[6:9] = apply(food[6:9],c(1,2),function(x) suppressWarnings(as.numeric(x)))
food = food[apply(food[6:9],1,function(x) all((!is.na(x)) & (x != '') & (x > 0))),]
food = food[food$total >= food$grocery + food$dining + food$miscellaneous,]
```

(g)

I will remove the rows where dining time is 0 but the dining expenditure is not 0.

```
food = food[!((food$dining > 0) & (food$dining_time == 0)),]
```

(h)

```
nrow(food)
```

```
[1] 62
```

The final number of observations is 62.

Problem 3

(a)

Below is the required function.

```r
#' function to get the next number in a Collatz sequence
#' @param x a positive integer input
#' @return the next integer number in the Collatz sequence of x
nextCollatz = function(x){
  if ((x < 0) || (!is.numeric(x)) || (x %% 1 != 0)){
    stop('the input is not a positive integer')
  }

  if (x %% 2 == 0){
    return(x/2)
  } else{
    return(3*x+1)
  }
}
```

```r
print(nextCollatz(5))
```

```
[1] 16
```

```r
print(nextCollatz(16))
```

```
[1] 8
```

We see that the two examples are successfully reproduced.

(b)

Below is the required function.

```r
#' function to get the Collatz sequence
#' @param x a positive integer input
#' @return A list containing the vector of the entries in the Collatz sequence, beginning at
collatzSequence = function(x){
  if ((x < 0) || (!is.numeric(x)) || (x %% 1 != 0)){
    stop('the input is not a positive integer')
  }

  result = c()
  next_num = x
```

```
  while (next_num != 1){
    result = append(result,next_num)
    next_num = nextCollatz(next_num)
  }
  result = append(result,1)
  return(result)
}
```

```
print(collatzSequence(5))
```

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

```
print(collatzSequence(19))
```

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

We see that the two examples are successfully reproduced.

(c)

```
lens = sapply(100:500,function(x) length(collatzSequence(x)))
max_len_num = which.max(lens) + 99
min_len_num = which.min(lens) + 99
print(paste('lowest number giving the longest sequence:',max_len_num))
```

```
[1] "lowest number giving the longest sequence: 327"
```

```
print(paste('lowest number giving the shortest sequence length:',min_len_num))
```

```
[1] "lowest number giving the shortest sequence length: 128"
```

The lowest starting value giving the longest sequence is 327. The lowest starting value giving the shortest sequence is 128.