AMS597_HW1_Solution_Spring2024

Question 1(a)

```
hsb2 <- read.csv("http://www.ams.sunysb.edu/~pfkuan/Teaching/AMS597/Data/hsb2.csv")
df_subset <- subset(hsb2, race == 1 | race == 4)
head(df_subset)
```

```
##    id female race ses schtyp prog read write math science socst
## 1  70      0    4   1      1    1   57    52   41      47    57
## 2 121      1    4   2      1    3   68    59   53      63    61
## 3  86      0    4   3      1    1   44    33   54      58    31
## 4 141      0    4   3      1    3   63    44   47      53    56
## 5 172      0    4   2      1    2   47    52   57      53    61
## 6 113      0    4   2      1    2   44    52   51      63    61
```

Question 1(b)

```
df_ordered <- hsb2[order(hsb2$id, hsb2$race), ]
head(df_ordered)
```

```
##     id female race ses schtyp prog read write math science socst
## 99   1      1    1   1      1    3   34    44   40      39    41
## 139  2      1    1   2      1    3   39    41   33      42    41
## 84   3      0    1   1      1    2   63    65   48      63    56
## 112  4      1    1   1      1    2   44    50   41      39    51
## 76   5      0    1   1      1    2   47    40   43      45    31
## 149  6      1    1   1      1    2   47    41   46      40    41
```

Question 1(c)

```
df_mean <- colMeans(hsb2[, c("read", "write", "math", "science", "socst")])
df_new <- hsb2
for (col in c("read", "write", "math", "science", "socst")) {
  df_new[, col][df_new[, col] < df_mean[col]] <- "Low"
  df_new[, col][df_new[, col] >= df_mean[col]] <- "High"
}
head(df_new)
```

```
##    id female race ses schtyp prog read write math science socst
## 1  70      0    4   1      1    1 High  High High    High  High
## 2 121      1    4   2      1    3 High  High High    High  High
## 3  86      0    4   3      1    1 High  High High    High  High
## 4 141      0    4   3      1    3 High  High High    High  High
## 5 172      0    4   2      1    2 High  High High    High  High
## 6 113      0    4   2      1    2 High  High High    High  High
```

Question 1(d)

```
df_long <- data.frame(id=rep(hsb2$id, 5), female = rep(hsb2$female, 5), race= rep(hsb2$race, 5),
                      ses = rep(hsb2$ses, 5), schtyp = rep(hsb2$schtyp, 5), prog = rep(hsb2$prog),
                      subject=rep(c("read", "write", "math", "science", "socst"), each = nrow(hsb2)),
                      score = c(hsb2$read, hsb2$write, hsb2$math, hsb2$science, hsb2$socst))
df_long <- df_long[order(df_long$id), ]
head(df_long)
```

```
##     id female race ses schtyp prog subject score
## 99   1      1    1   1      1    3    read    34
## 299  1      1    1   1      1    3   write    44
## 499  1      1    1   1      1    3    math    40
## 699  1      1    1   1      1    3 science    39
## 899  1      1    1   1      1    3   socst    41
## 139  2      1    1   2      1    3    read    39
```

```
## From the documentation
l <- reshape(hsb2,
  varying = c("read", "write", "math", "science", "socst"),
  v.names = "score",
  timevar = "subj",
  times = c("read", "write", "math", "science", "socst"),
  new.row.names = 1:1000,
  direction = "long")

l.sort <- l[order(l$id),]
l.sort[1:10,]
```

```
##     id female race ses schtyp prog    subj score
## 99   1      1    1   1      1    3    read    34
## 299  1      1    1   1      1    3   write    44
## 499  1      1    1   1      1    3    math    40
## 699  1      1    1   1      1    3 science    39
## 899  1      1    1   1      1    3   socst    41
## 139  2      1    1   2      1    3    read    39
## 339  2      1    1   2      1    3   write    41
## 539  2      1    1   2      1    3    math    33
## 739  2      1    1   2      1    3 science    42
## 939  2      1    1   2      1    3   socst    41
```

```
## Testing if our answer without reshape is the same as using reshape
## df_long == l.sort TRUE
```

Question 2

```
generate_Matrix <- function(r, c) {
  if (r%%1 == 0 & c%%1 == 0 & r >= 2 & c >= 2) {
    M <- matrix(1:(r*c),nrow=r,byrow=T)
    print(M)

    y1 <- M[, 1] + M[, c]

    product <- t(M)%*%M
    y2 <- sum(product)

    return(list(y1 = y1, y2 = y2))

  } else {
    return("Both arguments must be integers greater than or equal to 2.")
  }
}

generate_Matrix(1,4)
```

```
## [1] "Both arguments must be integers greater than or equal to 2."
```

```
generate_Matrix(2.5,3.5)
```

```
## [1] "Both arguments must be integers greater than or equal to 2."
```

```
generate_Matrix(3,3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9

## $y1
## [1]  4 10 16
##
## $y2
## [1] 837
```

Question 3

According to the help operator, ?identical, we can test objects for exact equality. It will return TRUE if the two vectors are the same and FALSE otherwise. By default, the identical() function considers missing (NA) values as equal.

```r
a1 <- c(1, 2, 3, NA)
a2 <- c(1, 2, 3, NA)
identical(a1, a2)
```

```
## [1] TRUE
```

```r
b1 <- c(1, 2, 3, NA)
b2 <- c(1, 2, NA, 3)
identical(b1, b2)
```

```
## [1] FALSE
```

Question 4

By default, R chooses the baseline from the first category that appears alphabetically or numerically. In this case the levels are shown below.

y[x] is a vector that has the same length as x. There is a one-to-one correspondence of the levels of x and the vector y such that y[x] is the vector x but refactored in terms of the vector y.

```r
x <- factor(c("D", "B", "C", "D", "A", "C"))
levels(x)
```

```
## [1] "A" "B" "C" "D"
```

```r
y1 <- c(1, 2, 3, 4)
y2 <- c(2, 1, 3, 4)
y1[x]
```

```
## [1] 4 2 3 4 1 3
```

```r
y2[x]
```

```
## [1] 4 1 3 4 2 3
```

Question 5

```r
myMed.cal <- function(x) {
  ## Finding the median by sorting
  MED <- function(k) {
    k_sorted <- sort(k)
    n <- length(k)
    if (n %% 2 == 0) {
      med <- (k_sorted[n%/%2] + k_sorted[n%/%2+1]) / 2
    } else {
      med <- k_sorted[n%/%2+1]
```

```
    }
    return(med)
  }

  ## Finding the median of the absolute deviation
  med_vector <- rep(MED(x), length(x))
  abs_dev <- abs(x - med_vector)
  return(list(median = MED(x), mad = MED(abs_dev)))
}

myMed.cal(c(5,3,4,2,88))
```

```
## $median
## [1] 4
##
## $mad
## [1] 1
```

Question 6(a)

```
set.seed(123)
mydna <- paste(sample(c('a','t','c','g'),1000,replace=T),collapse='')

count_cg_ta <- function(dna_sequence){

  cg_count <- lengths(gregexpr("cg", dna_sequence))
  ta_count <- lengths(gregexpr("ta", dna_sequence))

  new_dna_sequence <- gsub("cg", "XY", dna_sequence)
  new_dna_sequence <- gsub("ta", "AB", new_dna_sequence)

  return(list(cg_count=cg_count, ta_count=ta_count, new_dna_sequence=new_dna_sequence))
}

result <- count_cg_ta(mydna)

cat("Number of CG:", result$cg_count, "\n")
```

```
## Number of CG: 61
```

```
cat("Number of TA:", result$ta_count, "\n")
```

```
## Number of TA: 71
```

```
cat("New DNA Sequence:", result$new_dna_sequence, "\n")
```

```
## New DNA Sequence: ccctctttcagtABtXYaccagaaaXYtcABtXYABccagXYABcaatcXYacaXYABABaggcaABacaABcactgXYgtt
```

Question 6(b)

```
library(stringr)
```

```
## Warning: package 'stringr' was built under R version 4.2.3
```

```
count_cg_ta_stringr <- function(dna_sequence){

  cg_count <- str_count(dna_sequence, "cg")
  ta_count <- str_count(dna_sequence, "ta")

  new_dna_sequence <- str_replace_all(dna_sequence, "cg", "XY")
  new_dna_sequence <- str_replace_all(new_dna_sequence, "ta", "AB")

  return(list(cg_count=cg_count, ta_count=ta_count, new_dna_sequence=new_dna_sequence))
}

result_stringr <- count_cg_ta_stringr(mydna)

cat("Number of CG:", result_stringr$cg_count, "\n")
```

```
## Number of CG: 61
```

```
cat("Number of TA:", result_stringr$ta_count, "\n")
```

```
## Number of TA: 71
```

```
cat("New DNA Sequence:", result_stringr$new_dna_sequence, "\n")
```

```
## New DNA Sequence: ccctctttcagtABtXYaccagaaaXYtcABtXYABccagXYABcaatcXYacaXYABABaggcaABacaABcactgXYgtt
```

```
## Test replaced string
result$new_dna_sequence == result_stringr$new_dna_sequence
```

```
## [1] TRUE
```

Question 7

Note we use readLines to read each line as a separate element.

```
process_phone_numbers <- function(file){
  rawdata <- readLines(file)
  pattern = paste('(^\\d{3}-\\d{3}-\\d{4}$)',
  '(^\\(\\d{3}\\) \\d{3} -\\d{4}$)',
  '(^\\d{3} \\d{3} \\d{4}$)',
  '(^\\d{3}\\.\\d{3}\\.\\d{4}$)', sep='|')
  return(grep(pattern, rawdata, value = TRUE, perl = T))
}
process_phone_numbers("http://www.ams.sunysb.edu/~pfkuan/Teaching/AMS597/Data/PhoneNumber.txt")
```

```
## [1] "631-631-1234" "773 687 3241" "888.888.9900" "777 687 3241" "123-631-1233"
## [6] "666.666.1234"
```

Question 8

Note that we are supposed to use package dplyr, and pipe %>%.

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.3.2
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
df <- read.table("http://www.ams.sunysb.edu/~pfkuan/Teaching/AMS597/Data/d_logret_6stocks.txt",
header = T, sep = "", dec = ".")
head(df)
```

```
##       Date       Pfizer       Intel   Citigroup      AmerExp        Exxon
## 1 1-Aug-00 -0.001438612  0.04998126  0.04427510  0.017410003  0.010224894
## 2 1-Sep-00  0.017489274 -0.25561927 -0.03353650  0.012656982  0.037989020
## 3 2-Oct-00 -0.017046116  0.03454674 -0.01164558 -0.004897625  0.000330555
## 4 1-Nov-00  0.012012934 -0.07255067 -0.02267479 -0.038275870 -0.003650020
## 5 1-Dec-00  0.016278701 -0.10249787  0.01070831  0.000000000 -0.005252049
## 6 2-Jan-01 -0.008063083  0.09022312  0.03990062 -0.066129678 -0.014169243
##      GenMotor
## 1  0.09329402
## 2 -0.03220924
## 3 -0.01960217
## 4 -0.09489160
## 5  0.01246125
## 6  0.02297158
```

```r
df_tibble <- as_tibble(df)
head(df_tibble)
```

```
## # A tibble: 6 x 7
##   Date       Pfizer   Intel Citigroup  AmerExp     Exxon GenMotor
##   <chr>       <dbl>   <dbl>     <dbl>    <dbl>     <dbl>    <dbl>
## 1 1-Aug-00 -0.00144  0.0500    0.0443  0.0174   0.0102     0.0933
## 2 1-Sep-00  0.0175  -0.256    -0.0335  0.0127   0.0380    -0.0322
## 3 2-Oct-00 -0.0170   0.0345   -0.0116 -0.00490  0.000331  -0.0196
## 4 1-Nov-00  0.0120  -0.0726   -0.0227 -0.0383  -0.00365   -0.0949
## 5 1-Dec-00  0.0163  -0.102     0.0107  0       -0.00525    0.0125
## 6 2-Jan-01 -0.00806  0.0902    0.0399 -0.0661  -0.0142     0.0230
```

```r
res <- df %>%
  ## Step 1: generate another tibble which subset rows corresponding to months Apr, May or June;
  mutate(Date = as.Date(Date, format = "%d-%b-%y")) %>%
  filter(month(Date) %in% c(4, 5, 6)) %>%
  ## Step 2: add a new column ExxonGenMotor;
  mutate(ExxonGenMotor = rowMeans(select(., Exxon:GenMotor))) %>%
  ## Step 3: computing the median of ExxonGenMotor groupby year.
  group_by(year(Date)) %>%
  summarize(medianExxonGenMotor = median(ExxonGenMotor))
res
```

```
## # A tibble: 5 x 2
##   `year(Date)` medianExxonGenMotor
##          <dbl>               <dbl>
## 1         2001             0.0233
## 2         2002            -0.00603
## 3         2003             0.00743
## 4         2004             0.00574
## 5         2005             0.0212
```