Bios 6301: Assignment 2

Hannah Weeks

(informally) Due Thursday, 17 September, 1:00 PM

50 points total.

This assignment won't be submitted until we've covered Rmarkdown. Create R chunks for each question and insert your R code appropriately. Check your output by using the `Knit PDF` button in RStudio.

1. **Working with data**
   In the `datasets` folder on the course GitHub repo, you will find a file called `cancer.csv`, which is a dataset in comma-separated values (csv) format. This is a large cancer incidence dataset that summarizes the incidence of different cancers for various subgroups. (18 points)

   1. Load the data set into R and make it a data frame called `cancer.df`. (2 points)

```
setwd("~/Documents/BIOS 6301/Homework")
cancer.df <- data.frame(read.csv('cancer.csv'))
head(cancer.df)
```

```
##   year                           site   state    sex     race mortality
## 1 1999 Brain and Other Nervous System alabama Female    Black      0.00
## 2 1999 Brain and Other Nervous System alabama Female Hispanic      0.00
## 3 1999 Brain and Other Nervous System alabama Female    White     83.67
## 4 1999 Brain and Other Nervous System alabama   Male    Black      0.00
## 5 1999 Brain and Other Nervous System alabama   Male Hispanic      0.00
## 6 1999 Brain and Other Nervous System alabama   Male    White    103.66
##   incidence population
## 1        19     623475
## 2         0      28101
## 3       110    1640665
## 4        18     539198
## 5         0      37082
## 6       145    1570643
```

```
2. Determine the number of rows and columns in the data frame. (2)
```

```
nrow(cancer.df)
```

```
## [1] 42120
```

```
ncol(cancer.df)
```

```
## [1] 8
```

```
3. Extract the names of the columns in `cancer.df`. (2)
```

```r
colnames(cancer.df)
```

```
## [1] "year"       "site"       "state"      "sex"        "race"
## [6] "mortality"  "incidence"  "population"
```

4. Report the value of the 3000th row in column 6. (2)

```r
cancer.df[3000,6]
```

```
## [1] 350.69
```

5. Report the contents of the 172nd row. (2)

```r
cancer.df[172,]
```

```
##     year                           site  state  sex  race mortality
## 172 1999 Brain and Other Nervous System nevada Male Black         0
##     incidence population
## 172         0      73172
```

6. Create a new column that is the incidence *rate* (per 100,000) for each row.(3)

```r
cancer.df$incidence.rate <- (cancer.df$incidence/cancer.df$population)*10^5
head(cancer.df)
```

```
##   year                           site   state    sex     race mortality
## 1 1999 Brain and Other Nervous System alabama Female    Black      0.00
## 2 1999 Brain and Other Nervous System alabama Female Hispanic      0.00
## 3 1999 Brain and Other Nervous System alabama Female    White     83.67
## 4 1999 Brain and Other Nervous System alabama   Male    Black      0.00
## 5 1999 Brain and Other Nervous System alabama   Male Hispanic      0.00
## 6 1999 Brain and Other Nervous System alabama   Male    White    103.66
##   incidence population incidence.rate
## 1        19     623475       3.047436
## 2         0      28101       0.000000
## 3       110    1640665       6.704598
## 4        18     539198       3.338291
## 5         0      37082       0.000000
## 6       145    1570643       9.231888
```

7. How many subgroups (rows) have a zero incidence rate? (2)

```r
nrow(cancer.df[cancer.df$incidence.rate == 0,])
```

```
## [1] 23191
```

8. Find the subgroup with the highest incidence rate.(3)

```
cancer.df[which.max(cancer.df$incidence.rate),]
```

```
##      year     site                state  sex  race mortality incidence
## 5797 1999 Prostate district of columbia Male Black     88.93       420
##      population incidence.rate
## 5797     160821       261.1599
```

2. **Data types** (10 points)

    1. Create the following vector: `x <- c("5","12","7")`. Which of the following commands will produce an error message? For each command, Either explain why they should be errors, or explain the non-erroneous result. (4 points)

        ```
        max(x)
        sort(x)
        sum(x)
        ```

```
x <- c("5","12","7")
```

The quotations used around each number when creating `x` indicate that the elements in `x` are characters, not numbers:

```
class(x)
```

```
## [1] "character"
```

The `sort()` function works on `x`, however it does not sort the numbers numerically from lowest to highest. Since we are dealing with character vectors, it sorts the elements "alphabetically." That is, the elements of `x` are sorted in order based on digit (starting with the left-most digit).

```
sort(x)
```

```
## [1] "12" "5"  "7"
```

Similarly, the `max()` function is still able to handle the character elements by picking the "largest" element (i.e. the one listed last when sorted). Since the characters would be arranged alphabetically rather than numerically, we see "7" as our maximum rather than "12".

```
max(x)
```

```
## [1] "7"
```

The `sum()` function requires that its arguments be numeric, complex, or logical. It fails here because it is trying to take the mathematical sum of character values. The exact error message is: *"Error in sum(x) : invalid 'type' (character) of argument"*

In order to correct this, we could force `x` to become a numeric vector:

```
sum(as.numeric(x))
```

```
## [1] 24
```

```
2. For the next two commands, either explain their results, or why they should produce errors. (3 point

        y <- c("5",7,12)
        y[2] + y[3]
```

The following expression will produce an error:

```r
y <- c("5",7,12)
#y[2] + y[3]
```

The exact error message is: *“Error in y[2] + y[3] : non-numeric argument to binary operator.”* The first element of `y` is stored as a character due to the use of quotation marks. The presence of a character element within the vector forces `y` to become a character vector:

```r
class(y)
```

```
## [1] "character"
```

Therefore, even though the second and third elements of y are entered as numeric, they are converted to character by the presence of $y[1] =$ “5”. Thus `y[2] + y[3]` is attempting to sum character values, producing an error.

```
3. For the next two commands, either explain their results, or why they should produce errors. (3 point

        z <- data.frame(z1="5",z2=7,z3=12)
        z[1,2] + z[1,3]
```

The following expression will not produce an error:

```r
z <- data.frame(z1="5",z2=7,z3=12)
z
```

```
##   z1 z2 z3
## 1  5  7 12
```

```r
z[1,2] + z[1,3]
```

```
## [1] 19
```

In question 2.2, we saw that multiple data types within a vector force coercion of the vector class to one type. The same does not hold true for vectors within a data frame. Since `z` is a data frame, each column of `z` can have its own class independent of the other columns:

```r
class(z)
```

```
## [1] "data.frame"
```

```
class(z$z1)
```

```
## [1] "factor"
```

```
class(z$z2)
```

```
## [1] "numeric"
```

```
class(z$z3)
```

```
## [1] "numeric"
```

Since both columns `z2` and `z3` are numeric, we can add elements from these columns without an error.

```
class(z[1,2])
```

```
## [1] "numeric"
```

```
class(z[1,3])
```

```
## [1] "numeric"
```

```
class(z[1,2] + z[1,3])
```

```
## [1] "numeric"
```

3. **Data structures** Give R expressions that return the following matrices and vectors (*i.e.* do not construct them manually). (3 points each, 12 total)

    1. $(1, 2, 3, 4, 5, 6, 7, 8, 7, 6, 5, 4, 3, 2, 1)$

```
c(1:8,7:1)
```

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

```
2. $(1,2,2,3,3,3,4,4,4,4,5,5,5,5,5)$
```

```
rep(1:5,1:5)
```

```
##  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

```
3. $\begin{pmatrix}
  0 & 1 & 1  \\
  1 & 0 & 1  \\
  1 & 1 & 0  \\
\end{pmatrix}$
```

```
1-diag(3)
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    1
## [3,]    1    1    0
```

```
4. $\begin{pmatrix}
  1 & 2 & 3 & 4 \\
  1 & 4 & 9 & 16 \\
  1 & 8 & 27 & 64  \\
  1 & 16 & 81 & 256 \\
  1 & 32 & 243 & 1024  \\
\end{pmatrix}$
```

```
x <- c(1:4)
matrix(data = c(x,x^2,x^3,x^4,x^5), nrow=5, ncol=4, byrow = TRUE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    1    4    9   16
## [3,]    1    8   27   64
## [4,]    1   16   81  256
## [5,]    1   32  243 1024
```

4. **Basic programming** (10 points)

    1. Let $h(x,n) = 1 + x + x^2 + \ldots + x^n = \sum_{i=0}^{n} x^i$. Write an R program to calculate $h(x,n)$ using a `for` loop. (5 points)

General idea:

```
$h \leftarrow 0$
$for(i \ in \ 0:n)\{$
$h \leftarrow h + x^i$
$\}$
$h$
```

In order for this to run without an error, `x` and `n` require specific values. Take, for example, `x=5` and `n=3`:

```
h <- 0
for(i in 0:3){
  h <- h + 5^i
}
h
```

```
## [1] 156
```

Alternatively, we can define a function `h` that takes arguments `x` and `n` and returns the sum $h(x,n)$ as above.

```
h <- function(x,n){
  sum <- 0
  for(i in 0:n){
    sum <- sum + x^i
  }
  return(sum)
}
```

Try it out:

```
h(5,3)
```

```
## [1] 156
```

```
h(2,4)
```

```
## [1] 31
```

```
h(4,9)
```

```
## [1] 349525
```

```
h(6,2)
```

```
## [1] 43
```

```
2. If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The s

    1. Find the sum of all the multiples of 3 or 5 below 1,000. (3, [euler1])
```

```
mult3or5 <- numeric(0)
for(i in 1:999){
  if(i %% 3 == 0 | i %% 5 == 0){
    mult3or5 <- c(mult3or5, i)
  }
}
sum(mult3or5)
```

```
## [1] 233168
```

```
    1. Find the sum of all the multiples of 4 or 7 below 1,000,000. (2)
```

Initially, I tried running a program analagous to the one for the previous question:
$mult4or7 \leftarrow numeric(0)$
$for(i\ in\ 1:(10^6-1))\{$
$if(i \mod 4\ ==\ 0\,|\,i \mod\ 7\ ==\ 0)\{$
$mult4or7 \leftarrow c(mult4or7, i)$
$\}$
$\}$
$sum(mult4or7)$

However, it took far too long to run so I revised to the following approach, which saw a huge increase in calculation speed:

```r
mult4or7 <- numeric(0)
sum4or7 <- sum(mult4or7)
for(i in 1:(10^6 - 1)){
  if(i %% 4 == 0 | i %% 7 == 0){
    sum4or7 <- sum4or7 +i
  }
}
sum4or7
```

```
## [1] 178571071431
```

```
1. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting wi
```

```r
fib <- c(1,2)

fibEven <- 2

while(length(fibEven) < 15){
  nextNum <- sum(tail(fib,2))
  fib <- c(fib,nextNum)
  if(nextNum %% 2 == 0) fibEven <- c(fibEven, nextNum)
}
fibEven
```

```
##  [1]          2          8         34        144        610       2584
##  [7]      10946      46368     196418     832040    3524578   14930352
## [13]   63245986  267914296 1134903170
```

```r
sum(fibEven)
```

```
## [1] 1485607536
```

```r
#first 15 odds just out of curiosity
fib <- c(1,2)

fibOdd <- 1

while(length(fibOdd) < 15){
  nextNum <- sum(tail(fib,2))
  fib <- c(fib,nextNum)
  if(nextNum %% 2 != 0) fibOdd <- c(fibOdd, nextNum)
}
fibOdd
```

```
##  [1]     1     3     5    13    21    55    89   233   377   987  1597
## [12]  4181  6765 17711 28657
```

```r
sum(fibOdd)
```

```
## [1] 60695
```

Some problems taken or inspired by projecteuler.