Intro to Statistical Computing HW 2

Sarah Lotspeich

20 September 2016

Grade: 54/50

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)

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

```
cancer.df <- data.frame(read.csv("https://raw.githubusercontent.com/fonnesbeck/Bios6301/master/datasets/
```

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

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

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

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

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

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

```
names(cancer.df)
```

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

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

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

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

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

```
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
```

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

```
cancer.df$incidence_rate <- (cancer.df$incidence/cancer.df$population)*100000
```

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

```
nrow(subset(cancer.df, cancer.df$incidence_rate == 0))
```

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

viii. 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

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)

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

ia. max(x)

```
max(x)
```

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

Since it was passed a vector of characters, the function max(x) returns the larget single-character value "7". If we had created vector x with integer values instead we would have received a resulting max(x) = 12. Therefore, even though this command did not display an Error message, the result was not as we expected.

ib. sort(x)

```
sort(x)
```

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

```
nchar(x[1])
```

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

```
nchar(x[2])
```

```
## [1] 2
```

```
nchar(x[3])
```

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

The sort function, when passed a vector of strings comprised of one- and two-character entries sorts first based on the leftmost character of each entry (hence, why 12 comes first since it begins with "1") and then increases. For clarification, please consider a different example vector:

```
y <- c("1", "2", "3", "4", "5", "10", "11", "12", "13", "14")
sort(y)
```

```
##  [1] "1"  "10" "11" "12" "13" "14" "2"  "3"  "4"  "5"
```

From this longer example we can see how the sort(y) command pulls all characters with a "1" in the left-most position to the front, sorts within all values containing a "1" in increasing order by the right-most position, and then continues to sort in this way. Again, this command does not prompt an Error message, but it does not return the same sorted vector as would be seen from a numeric vector.

ic. sum(x)

```
#sum(x)
```

This command returns an Error message warning about an invalid "type" (character) of the argument for function sum. The sum function is intended to be used with numeric, complex, or logical vectors only.

iia. y <- c("5",7,12)

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

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

Creating a vector in this way does not produce an error, but it should! Without any additional commands, the simple c(type1, type2) overrides the mix of character and numeric entries and forces the entire vector to become characters.

iib. y[2] + y[3]

```
#y[2] + y[3]
```

Now, as mentioned above, we cannot apply mathematical operations to the components of vector y because the entries were coerced to characters since the first value was designated a character. If we wanted to create vector y in this way, but maintain the mathematical operators, we could try this instead:

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

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

iii. For the next two commands, either explain their results, or why they should produce errors. (3 points)

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

Since the data.frame() function created a list called z, the specified data types of each entry z1, z2, z3 are preserved. The end result z is a dataframe with 1 row and 3 columns named z1, z2, and z3.

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

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

This command returns the sum of data.frame z row 1, column 2 + row 1, column 3 (i.e. 7 + 12).

3. Data structures

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

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

```
(vector1 <- c(seq(1,8),seq(7,1)))
```

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

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

```
(vector2 <- c(rep(1,1), rep(2,2), rep(3,3), rep(4,4), rep(5,5)))
```

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

iii. $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$

```
matrix1 <- matrix(rep(1,9),nrow=3,ncol=3)
diag(matrix1) <- 0
matrix1
```

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

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

```
(matrix2 <- matrix(c(seq(1, 4), seq(1, 4)^2, seq(1, 4)^3, seq(1, 4)^4, seq(1,
    4)^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
```

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Write an R program to perform the following calculations. (5 points)

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

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

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be $(1, 2, 3, 5, 8, 13, 21, 34, 55, 89)$. Write an R program to calculate the sum of the first 15 even-valued terms. (5 bonus points, euler2)

4. Basic programming

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)

```
h_xn <- function(x,n)
{
  returnValue <- 1
  for (i in 1:n)
  {
    returnValue <- returnValue + x^i
  }
  return(returnValue)
}
```

Check the function with an example! By hand, $h(x = 3, n = 4) = 1+3^1+3^2+3^3+3^4 = 1+3+9+27+81 = 121$. Now with the function defined above, we have that

```
h_xn(3,4)
```

```
## [1] 121
```

Tada!

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Write an R program to perform the following calculations. (5 points)

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

```
sumMult_3or5 <- 0
for (i in 1:1000)
{
  if ((i%%3==0)|(i%%5==0))
  {
    sumMult_3or5 <- sumMult_3or5 + i
  }
}
sumMult_3or5
```

```
## [1] 234168
```

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

```
sumMult_4or7 <- 0
for (i in 1:1000000)
{
  if ((i%%4==0)|(i%%7==0))
  {
    sumMult_4or7 <- sumMult_4or7 + i
  }
}
sumMult_4or7
```

```
## [1] 178572071431
```

iv. Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be $(1, 2, 3, 5, 8, 13, 21, 34, 55, 89)$. Write an R program to calculate the sum of the first 15 even-valued terms. (5 bonus points, euler2)

```
evenFib <- NULL
fib <- NULL
for (i in 1:46) #expand out the first 48 terms of the Fibonacci Sequence
{
  if (i > 2)
  {
    fib[i] = fib[i-1] + fib[i-2]
  }
  else
  {
      if (i == 1)
      {
        fib[i] <- 1
      }
      if (i == 2)
      {
        fib[i] <- 2
      }
  }
}
for (i in 1:length(fib))
```

```
{
  if (fib[i]%%2==0)
  {
    evenFib[length(evenFib)+1] <- fib[i]
  }
}
```

JC addition for clarity

```
evenFib
```

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

JC Bonus +4 (Also provide the sum of first 15 elements)