Homework 2

Molly Olson

September 29, 2015

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 csv format. This is a large cancer incidence dataset that summarizes the incidence of different cancers for various subgroups. **i. Load the data set into R and make it a data frame called `cancer.df`.**

```
setwd("/Users/mollyolson/documents/Vanderbilt_2015_Fall/Statistical_Computing/Bios6301-master/datasets")
cancer.df <- read.csv("cancer.csv",header=TRUE)
```

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

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

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

```
#8 columns
nrow(cancer.df)
```

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

```
#42,120 rows
```

iii. Extract the names of the columns in `cancer.df`

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

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

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

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

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

v. Report the contents of the 172nd row

```
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.

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

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

```
length(which(cancer.df$IncidenceRate == 0.00))
```

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

```
#23,191 rows have 0 incidence rates
```

viii. Find the subgroup with the highest incidence rate

```
which.max(cancer.df$IncidenceRate)
```

```
## [1] 5797
```

```
#row 5797
```

2. Data Types

i. 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.

```
x <- c("5","12","7")
#y <- c("apple","banana")
#z <- c(T,F)
```

sum(x) will give an error.

max(x) gives the maximum value, regardless of the data type.
sort(x) will sort the data, regardless of type. It sorts in increasing pattern. It sorts 12 first, because it is looking at the "10's" position, which is equal to 1. Then it will sort 5, since 1<5<7, and then 7.
sum(x) produces an error because the vector is not numerical, where sum works on numberical vectors.

```
max(x)
```

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

```
sort(x)
```

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

```
#sum(x)
```

ii. For the next two commands, either explain their results, or why they should produce errors.

The first command: you can have different "types" of elements into a vector, but a vector can only be one type, so it will pick the least flexible type. So defining y will work using this command, but the vector type will be character, not a combination of characters and integers. So, if we call class(y), we will get character.

```
y <- c("3",7,12)
#class(y)
```

The second command: You can not have a vector be more than one data type, so it will choose the least flexible data type in the vector to be the vector type. In this case, character is the least flexible, so 7 and 12 also become characters. You can not add character values together, so it will produce an error that says "non-numerica argument to binary operator"

```
y[2]
```

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

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

For the first command, we can define the dataframe this way because each column of the data frame is considered to be it's own part. So if we call class on z, we will get back "data.frame"

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

For the second command, since each column of the data frame is it's own part, if we call class(zz2), *wewillget*"*numeric*", *andlikeu*
So, since z2 and z3 are both numberic, we can perfrom additon on them.

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

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

3. Data Structures.

Give R expressions that return the following matrices and vectors. (ie do not construct them manually)

i. (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
```

ii. (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
```

iii.

```
matrix(rep(1,9),nrow=3) - diag(3)
```

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

iv.

```
x <- c(1:4)
t(matrix(c(x,x^2,x^3,x^4),ncol=4))
```

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

Basic Programming

i. 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.

```
#h will be a function that takes x, a number, and n, the degree of the polynomial defined above.
h <- function(x,n){
  #sumVector is a vector with length of the degree, it will hold all of the x^i's
  sumVector <- numeric(n)
  #for loop, from 0 to n
  for(i in 0:n){
    #since we are starting our loop at index 0, we will look at the index i+1 of the sumVector and assi
    sumVector[i+1] <- x^i
  }
  #now we have all of the values of x^i in the sumVector, now we just take the sum.
  sum(sumVector)
}
```

```
#test
h(2,3) #expect 15
```

```
## [1] 15
```

```
h(5,5) #expect 3906
```

```
## [1] 3906
```

```
h(0,0) #expect 1
```

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

```r
h(0,1) #expect 1
```

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

```r
h(2,0) #expect 1
```

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

ii. 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:

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

```r
#n is the number of values we want to be below
n <- 1000
#fiveVec and threeVec are vectors that will hold all of the multiples of five and three respectively
fiveVec <- numeric(n/5)
threeVec <- numeric(n/3)
#initializing counters for 3 and 5 vectors. These will count the index we are currently at in the vecto
counter3 <- 0
counter5 <- 0

#for loop
for(i in 1:n-1){
  #if the number is a multiple of 3, add that value to the next index in the threeVec, and incrememnt t
  if(i %% 3 == 0){
    threeVec[counter3] <- i
    counter3 <- counter3 + 1
  }
  #if the number is a multiple of 5, add that vlue to the next index of fiveVec, and increment the coun
  if(i %% 5 == 0){
    fiveVec[counter5] <- i
    counter5 <- counter5 + 1
  }
}
#combine threeVec and fiveVec into one vector named threeFiveVec
threeFiveVec <- c(threeVec,fiveVec)
#since threeVec and fiveVec may have values that are both multiples of 3 and 5, sum the unique values o
sum(unique(threeFiveVec))
```

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

b) Find the sum of all the multiples of 4 or 7 below 1,000,0000

```r
n <- 1000000
#fourVec and sevenVec are vectors that will hold all of the multiples of four ond seven respectively
fourVec <- numeric(n/4)
sevenVec <- numeric(n/7)
#initialize counters that will keep track of the index of fourVec and sevenVec
counter4 <- 0
counter7 <- 0
```

```r
#for loop
for(i in 1:n-1){
  #if the number is a multiple of 4, add that value to the next index in the fourVec, and incrememnt th
  if(i %% 4 == 0){
    fourVec[counter4] <- i
    counter4 <- counter4 + 1
  }
  #if the number is a multiple of 7, add that value to the next index in the sevenVec, and incrememnt t
  if(i %% 7 == 0){
    sevenVec[counter7] <- i
    counter7 <- counter7 + 1
  }
}
#sum the unique values
fourSevenVec <- c(fourVec,sevenVec)
sum(unique(fourSevenVec))
```

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

iii. 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.

```r
#fibVec will hold the values in the Fibonacci sequence
n <- 50
fibVec <- numeric(n)
#j, k are place holders to do recursion
j <- 1
k <- 2
#we know the first two values of the sequence are 1 and 2, so I put those values in the first two indic
fibVec[1] <- 1
fibVec[2] <- 2

#for loop, starting at 3 since we already know the first two values of the sequence, and up until the l
for(i in 3:length(fibVec)){
  #the next value will be j+k, by definition
  fibVec[i] <- j + k
  #recursion
  nxt <- j + k
  j <- k
  k <- nxt
}

#now that we have the first n values of the sequene, we only care about the first 15 even values.
#evenFib will hold the first 15 even values of the Fibonacci sequence
evenFib <- numeric(15)
#evenFibCounter will keep track of what index we are at in evenFib
evenFibCounter <- 0
#for loop
for(i in 1:length(fibVec)){
  #if the ith value in the fibVec is even, let's add it to evenFib vector and increment the counter by
  if(fibVec[i] %% 2 == 0){
```

```
      evenFibCounter <- evenFibCounter + 1
      evenFib[evenFibCounter] <- fibVec[i]
  }
  if(evenFibCounter == 15){
        break
      }
}
#sum of the first 15 even Fibonacci values
sum(evenFib)
```

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

Note: the x-axis is incremented by multiples of 100. This is because of how the function handles the gradation of p in the output. So the x-axis represents the values p*1000; e.g. 200 represents the value of p=0.2.