HW\_1\_Template

\*\*\* Lina Cao , Thursdays 1:15PM \*\*\*

SDGB 7844; Prof. Nagaraja; Fall 2017

## Question 1

The vectors state.name, state.area, and state.region are pre-loaded in R and contain US state names, area (in square miles), and region respectively.

2a) Identify the data type for state.name, state.area, and state.region.

# "typeof" is a function to determine the type or storage mode of the required object  
  
typeof(state.name)

## [1] "character"

typeof(state.area)

## [1] "double"

typeof(state.region)

## [1] "integer"

# we can also use "mode" or "class" to get the results  
# as shown in "help", "integer" or "double" in "typeof()" will be placed as "numeric"  
  
mode(state.name)

## [1] "character"

mode(state.area)

## [1] "numeric"

mode(state.region)

## [1] "numeric"

class(state.name)

## [1] "character"

class(state.area)

## [1] "numeric"

class(state.region)

## [1] "factor"

# from the result and the commonsense, use "is.xxx" to get the exact type of the data  
# if the answer is TRUE, it is the correct type  
  
is.character(state.name)

## [1] TRUE

is.numeric(state.area)

## [1] TRUE

is.factor(state.region)

## [1] TRUE

**The results shows all true. The type of state.name is character. The type of state.area is numeric. The type of state.region is factor.**

2b) What is the longest state name (including spaces)? How long is it?

# COUNT HOW LONG IS THE LONGEST STATE NAME #  
# "nchar()" is used to count the number of characters (or bytes or Width), whose element is illustrated as "state.name[]". Since we know state.name belongs to characters, we have to use "nchar" instead of "length".  
# "max()" is to print the biggest one in the lengths of the state names  
  
max(nchar(state.name))

## [1] 14

# PRINT THE LONGEST STATE NAME #  
# we have the longest name width of the states, check where the state is in the vector state.name  
# use "==" to judge every element in state.name, whether they reach the requirement  
  
 ## nchar(state.name)==max(nchar(state.name))  
  
# a series of TRUE/FALSE will be returned  
# extract those element with TRUE in former criterion, use "state.name[]"  
  
state.name[nchar(state.name)==max(nchar(state.name))]

## [1] "North Carolina" "South Carolina"

**The longest state name have 14 characters (include space). There are two states have 14 characters. They are "North Carolina" and "South Carolina".**

2c) Compute the average area of the states which contain the word "New" at the start of the state name. Use the function substr().

# firstly, extract the first three letters from each state name  
  
 ## substr(state.name[],1,3)  
  
# we are required to find those start with "New", use "==" to check those reach the criterion  
  
 ## substr(state.name[],1,3)=="New"  
  
# third, sum all the areas from the states satisfied the condition, where "state.area[]" is used to extract area  
  
 ## sum(state.area[substr(state.name,1,3)=="New"])  
  
# then calculate the numbers that are summed up  
  
 ## length(state.area[substr(state.name,1,3)=="New"])  
  
# finally divide sum by length  
  
sum(state.area[substr(state.name,1,3)=="New"])/length(state.area[substr(state.name,1,3)=="New"])

## [1] 47095.5

**Use "substr()" to extract the first three letters of each state names, and choose those answer is “New”. After that, extract the area of the states, and add them up. The result of the states that meet the requirement is 47095.5 .**

2d) Use the function table() to determine how many states are in each region. Include the table in your solutions.

# simply use table() and list the results of how many states in each region  
   
table(state.region)

## state.region  
## Northeast South North Central West   
## 9 16 12 13

# but it is not consist with the required table format  
# copypaste the needed information to the following chart.

|  |  |
| --- | --- |
| **Region** | **Number of States** |
| **Northeast** | **9** |
| **South** | **16** |
| **North Central** | **12** |
| **West** | **13** |

## Question 2

Computing the geometric mean of a numerical vector x.

2a) Using a for loop, write code to compute the geometric mean of a numeric vector x. Make sure your code (i) removes any NA values and (ii) prints an error message if there are any non-positive values in x.

x <- c(NA,4,67,3)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number involveed in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 # use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 # print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 # print the final geometric mean when the elements are all positive  
} # end if/else

## [1] 9.298624

1b) Test your code on the following cases and show the output: (i) {4, 67, 3}, (ii) {NA, 4, 67, 3}, (iii) {0, NA}, 6}, (iv) {67, 3, infinity}, and (v) {-infinity, 67, 3}.

#### (i) {4,67,3} ####  
  
x <- c(4,67,3)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number included in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 ### use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 ### print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 ### print the final geometric mean when the elements are all positive  
} # end if/else

## [1] 9.298624

#### (ii) {NA,4,67,3} ####  
  
x <- c(NA,4,67,3)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number included in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 # use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 # print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 # print the final geometric mean when the elements are all positive  
} # end if/else

## [1] 9.298624

#### (iii) {0,NA,6} ####  
  
x <- c(0,NA,6)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number included in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 # use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 # print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 # print the final geometric mean when the elements are all positive  
} # end if/else

## [1] "Error!!There is non-positive value in x"

#### (IV) {67,3,Inf} ####  
  
x <- c(67,3,Inf)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number included in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 ### use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 ### print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 ### print the final geometric mean when the elements are all positive  
} # end if/else

## [1] Inf

#### (v) {-Inf,67,3} ####  
  
x <- c(-Inf,67,3)  
x <- x[!is.na(x)] # remove all the NA in vector x  
n=length(x) # count the number included in calculation which is used as radical exponent  
  
prod.x <- 1 # give product of x[i] the initial value 1  
for(i in 1:n){ # loop repeated n times  
   
 prod.x <- prod.x\*x[i] # multiply each new element to the former result  
   
} # end loop   
 # use "if statement"" to compute total product  
if (prod.x<=0){  
 print("Error!!There is non-positive value in x")   
 # print an error message if non-positive values exist  
}else{  
 print(prod.x^(1/n))   
 # print the final geometric mean when the elements are all positive  
} # end if/else

## [1] "Error!!There is non-positive value in x"

## Question 3

Perfect numbers are those where the sum of the proper divisors (i.e., divisors other than the number itself) add up to the number. For example, 6 is a perfect number because its divisors, 1, 2, and 3, when summed, equal 6.

3a) The following code was written to find the first 2 perfect numbers: 6 and 28; however, there are some errors in the code and the programmer forgot to add comments for readability. Debug and add comments to the following.

### the corrected code is   
  
num.perfect<-2 # set the number of perfect number that is required to output  
count<-1 # set the initial number of perfect number (the first one to be found)  
iter<-2 # start from 2 to find perfect number  
while(count<=num.perfect){  
divisor<-1  
 for(i in 2:(iter-1)){  
 if (iter%%i==0) {  
 divisor<-c(divisor,i)  
 } # end IF statement  
 }#endf FOR loop  
 if(sum(divisor)==iter){  
 print(paste(iter,"is","a","perfect","number",sep=" "))  
 count<-count+1  
 }#endif  
iter<-iter+1  
}#endwhileloop

## [1] "6 is a perfect number"  
## [1] "28 is a perfect number"

**EXPLANATION:**

**1.the 2nd line of the code should be "count <- 1", indicating the "1st number perfect number", rather than the "zero" one.**

**2.in the for loop, i should be given as the format of "i in a:b", there should be an "in" between "i" and "2:(iter-1)".**

**3.the format of "if statement" is wrong, which should appear a "{" after the condition part.**

**4.in equavalence judgment, double equal sign should be used instead of only one**

**5.when it is going to print the result, "paste()" can only separate characters with "sep=",but cannot separate a string.**

num.perfect<-2

count<-0 **### 1.it should be set as "1", indicating the "1st number "**

iter<-2

while(count<=num.perfect){

divisor<-1

for(i2:(iter-1)){ **### 2. i should be given as "i in a:b" in for loop"** if(iter%%i==0)divisor<-c(divisor,i) **### 3.format of "if statements" is wrong }#end for loop**

if(sum(divisor)=iter){ **### 4."==" should be used in equavalence judgment**

print(paste(iter,"is a perfect number",sep="") **### 5. paste() can only separate characters with "sep=" count<-count+1 }**#end if

**i**ter<-iter+1

}#end while loop

3b) Use the function date() at the start and at the end of your amended code. Then compute how long the program takes to run (you can do this subtraction by hand). Find the run time when you set num.perfect to 1, 2, 3, and 4. Make a table of your results. What are the first four perfect numbers?

date()

## [1] "Wed Sep 27 15:22:59 2017"

num.perfect<-4 # set the number of perfect number that is required to output  
count<-1 # set the initial number of perfect number (the first one to be found)  
iter<-2 # start from 2 to find perfect number  
while(count<=num.perfect){  
divisor<-1  
 for(i in 2:(iter-1)){  
 if (iter%%i==0) {  
 divisor<-c(divisor,i)  
 } # end IF statement  
 }#endf FOR loop  
 if(sum(divisor)==iter){  
 print(paste(iter,"is","the","perfect","number",sep=" "))  
 count<-count+1  
 }#endif  
iter<-iter+1  
}#endwhileloop)

## [1] "6 is the perfect number"  
## [1] "28 is the perfect number"  
## [1] "496 is the perfect number"  
## [1] "8128 is the perfect number"

date()

## [1] "Wed Sep 27 15:23:09 2017"

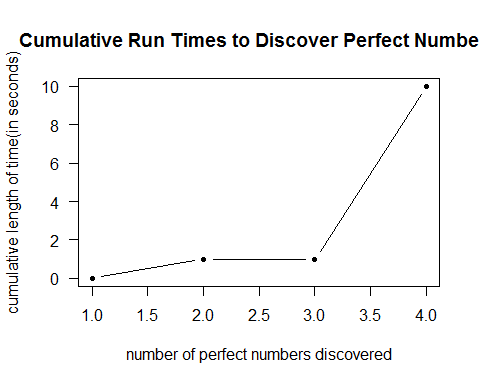
ENTER YOUR EXPLANATION FOR QUESTION 3b HERE.

|  |  |  |  |
| --- | --- | --- | --- |
| Count | Start Time | Stop Time | Elapsed Time (sec.) |
| 1 | 10:40:54 | 10:40:54 | almost os |
| 2 | 10:39:36 | 10:39:37 | 1s |
| 3 | 10:41:37 | 10:41:38 | 1s |
| 4 | 10:42:00 | 10:42:10 | 10s |

The first four perfect numbers are 6, 28, 496, 8128

3c) Let x <- 1:4 and define y to be the vector of run times. Plot y vs x using the code below. Is the relationship between the discovery of perfect numbers and run times on your computer linear? Justify your answer.

x<-1:4  
y<-c(0,1,1,10)   
plot(x,y,pch=20,type="b",  
 xlab="number of perfect numbers discovered",  
 ylab="cumulative length of time(in seconds)",  
 main="Cumulative Run Times to Discover Perfect Numbers",  
 las=TRUE)



I do not think the relationship is linear. As is illustrated in the plot, the time to compute one, two and three perfect numbers are almost the same, while it costs much longer time when it comes to get four perfect numbers.

I tried to compute five perfect numbers, however, the fifth perfect number did not show up even after 15 minutes run. Therefore, I can simply conclude that the time will balloon exponentially as the number of required perfect number increase from four to five.