Introduction to R

Computational Statistics

September 13, 2023

Generating long vectors

1a. The colon operator creates a sequence of integers via `start:end`. In R there is a more general sequence generator `seq()`. Use the help command `?seq` to learn about the function. Use `seq()` to create a sequence of numbers from `1` to `1001` in increments of `20`.

Ans: Sequence of numbers from 1 to 1001 with an interval of 20 are as follows,

```
seq(from=1, to=1001, by=20)
```

```
##  [1]    1   21   41   61   81  101  121  141  161  181  201  221  241  261  281
## [16]  301  321  341  361  381  401  421  441  461  481  501  521  541  561  581
## [31]  601  621  641  661  681  701  721  741  761  781  801  821  841  861  881
## [46]  901  921  941  961  981 1001
```

1b. The function `rep()` replicates the values in its input vector. Take a look at its help page. Explain the difference between `rep(1:4, times = 2)` and `rep(1:4, each = 2)`

Ans:`rep(1:4, times=2)` replicates the given sequence of 1 to 4 *as a whole* for two times as shown below,

```
rep(1:4, times=2)
```

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

`rep(1:4, each=2)` replicates the **every element** of the sequence for 2 times i.e every integer in the sequence is repeated twice before printing the next integer.

```
rep(1:4, each=2)
```

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

Functions and conditional statements

2a. Write a function `rescale01` that takes as input a numeric vector, and returns a rescaled vector whose components lie in [0, 1]. Linear scaling is applied to the finite numbers in the input vector, where the maximum and minimum are scaled to 1 and 0, respectively. Unexpected values should be handled as well. After the scaling, `NA` values should be unchanged, and positive infinity and negative infinity are set to 1 and 0, respectively. **Hint:** Use `is.infinite()`.

Ans:Let us consider a vector `v` with values as below,

```
v<-c(9,NA,5,3,Inf,-Inf,12:15)
v
```

```
##  [1]    9   NA    5    3  Inf -Inf   12   13   14   15
```

Rescaling this vector while maintaining NA values will result in the following output,

```
v<-c(9,NA,5,3,Inf,-Inf,12:15)
rescale01<-function(l)
{
  RNG <- range(l,na.rm=TRUE, finite = TRUE) #excludes infinite values and ignores NA values
  lm<-(l-RNG[1])/(RNG[2]-RNG[1]) #rescaling using range function
  lm[lm == -Inf]<-0
  lm[lm == Inf]<-1
  lm
}
rescale01(v)
```

```
##  [1] 0.5000000        NA 0.1666667 0.0000000 1.0000000 0.0000000 0.7500000
##  [8] 0.8333333 0.9166667 1.0000000
```

The positive and negative infinity values can also be set to 1 and 0 respectively as mentioned above **or** can be changed after rescaling as below,

```
v<-c(9,NA,5,3,Inf,-Inf,12:15)
rescale01<-function(l)
{
  RNG <- range(l,na.rm=TRUE, finite = TRUE) #excludes infinite values and ignores NA values
  lm<-(l-RNG[1])/(RNG[2]-RNG[1])
  lm
}
p<-rescale01(v)
p[p == -Inf]<-0 #Replacing the negative infinity to 0
p[p == Inf]<-1 #Replacing the positive infinity to 1
p
```

```
##  [1] 0.5000000        NA 0.1666667 0.0000000 1.0000000 0.0000000 0.7500000
##  [8] 0.8333333 0.9166667 1.0000000
```

2b. With the `rescale01()` function, write a `for` loop to rescale every **numeric** column in the data frame `df`. **Hint:** Use `if` statement with `is.numeric()`.

```
set.seed(2023 - 9 - 13)
df <- data.frame(
    a = rnorm(5),
    b = 1:5,
    c = letters[6:10],
    d = c(-1, NA, Inf, -Inf, 1)
)
```

Ans: The given data frame will look as below,

```
set.seed(2023 - 9 - 13)
df <- data.frame(
  a = rnorm(5),
  b = 1:5,
  c = letters[6:10],
  d = c(-1, NA, Inf, -Inf, 1)
)
df
```

```
##           a b c    d
## 1  0.7019977 1 f   -1
## 2 -0.7778131 2 g   NA
## 3 -1.0240757 3 h  Inf
## 4  0.3027668 4 i -Inf
## 5  0.3511394 5 j    1
```

Rescaling only the numeric columns in the above data frame can be achieved as below

```
set.seed(2023 - 9 - 13)
df <- data.frame(
  a = rnorm(5),
  b = 1:5,
  c = letters[6:10],
  d = c(-1, NA, Inf, -Inf, 1)
)
df
```

```
##           a b c    d
```

```
## 1  0.7019977 1 f   -1
## 2 -0.7778131 2 g   NA
## 3 -1.0240757 3 h  Inf
## 4  0.3027668 4 i -Inf
## 5  0.3511394 5 j    1
```

```
rescale01<-function(l)
{
  RNG <- range(l,na.rm=TRUE, finite = TRUE) #excludes infinite values and ignores NA values
  lm<-(l-RNG[1])/(RNG[2]-RNG[1])
  lm[lm == -Inf]<-0
  lm[lm == Inf]<-1
  lm
}
outputdf<-data.frame(df) #preallocating the output space
for(i in seq_along(outputdf))
{
  if(is.numeric(outputdf[[i]])) #check if column is numeric
  {
    outputdf[[i]]<-rescale01(outputdf[[i]])
  }
}
outputdf
```

```
##           a    b c  d
## 1 1.0000000 0.00 f  0
## 2 0.1426722 0.25 g NA
## 3 0.0000000 0.50 h  1
## 4 0.7687058 0.75 i  0
## 5 0.7967304 1.00 j  1
```

Note: If we observe, we set the values to positive and negative infinity to 1 and 0 respectively.

2c. Write a function `csum` that takes as input a finite positive integer n and computes the sum of cubes $S_n = 1^3 + 2^3 + \cdots + n^3$. The function should include conditional checks. If an input is invalid (e.g., unexpected length or type, infinite, non-positive, not a number, etc.), the function should stop the execution with informative error message. Test your function with `csum(NA)`, `csum(NaN)`, `csum(Inf)`, `csum(1:3)`, `csum("two")`, `csum(0)`, `csum(10)`, `csum(-5L)`, `csum(1L)`, `csum(2L)`, `csum(3L)`.

Ans: A function that takes a finite positive integer and computes the sum of cubes is defined below,

```
csum <- function(n) {
  # Check if n is missing
  if (missing(n)) {
```

```
    stop("Input is missing (NA).")
  }
  #Check if the input is vector with length>1
  if ((length(n)>1) && (is.vector(n))) {
    stop("The input value is a vector and length is greater than 1")
  }
  # Check if n is NaN
  if (is.nan(n)) {
    stop("Input is not a number (NaN).")
  }
  # Check if n is infinite
  if (is.infinite(n)) {
    stop("Input is infinite.")
  }
  # Check if n is not a finite positive integer
  if (!is.numeric(n) || !is.finite(n) || n <= 0 || n != as.integer(n)) {
    stop("Input is not a finite positive integer.")
  }

  # Computing the sum of cubes
  Sn <- sum((1:n)^3)

  return(Sn)
}
```

When executed for multiple values of 'n' we get the below

```
  csum(NA) #Testing the function for multiple values
```

```
## Error in csum(NA): Input is not a finite positive integer.
```

```
  csum(NaN)
```

```
## Error in csum(NaN): Input is not a number (NaN).
```

```
  csum(Inf)
```

```
## Error in csum(Inf): Input is infinite.
```

```
  csum(1:2)
```

```
## Error in csum(1:2): The input value is a vector and length is greater than 1
```

```
  csum("two")
```

```
## Error in csum("two"): Input is not a finite positive integer.
```

```
  csum(0)
```

```
## Error in csum(0): Input is not a finite positive integer.
```

```
  csum(10)
```

```
## [1] 3025
```

```
  csum(-5L)
```

```
## Error in csum(-5L): Input is not a finite positive integer.
```

```
  csum(1L)
```

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

```
  csum(2L)
```

```
## [1] 9
```

```
  csum(3L)
```

```
## [1] 36
```

2d. Write a `while` loop to print the value of S_n for $n = 1, 2, \ldots$ until $S_n > 2000$. At each iteration, you can print the sum of cubes using an R command similar to this: `print(paste0("n = ", n, ", sum of cubes is ", csum(n)))`.

Ans: Let us initialize the sum of cubes to be zero. Now, printing the sum of cubes until the sum reaches 2000 is defined via the below code and will result in the following output.

```
csum<-function(num){
  sum((1:num)^3)
}
num<-1
while(csum(num)<=2000){
  print(paste0("n= ", num,", sum of cubes is ", csum(num)))
  num<-num+1
}
```

```
## [1] "n= 1, sum of cubes is 1"
## [1] "n= 2, sum of cubes is 9"
## [1] "n= 3, sum of cubes is 36"
## [1] "n= 4, sum of cubes is 100"
## [1] "n= 5, sum of cubes is 225"
## [1] "n= 6, sum of cubes is 441"
## [1] "n= 7, sum of cubes is 784"
## [1] "n= 8, sum of cubes is 1296"
```

Conversion of abbreviations into full words

A character vector `fall_abbr` stores information of fall months as `sept`, `oct` and `nov`:

```
fall_abbr <- sample(c("sept", "oct", "nov"), size = 30, replace = TRUE)
fall_abbr
```

```
##  [1] "sept" "sept" "sept" "nov"  "sept" "nov"  "nov"  "nov"  "sept" "sept"
## [11] "oct"  "oct"  "sept" "nov"  "sept" "sept" "oct"  "nov"  "sept" "oct"
## [21] "nov"  "oct"  "oct"  "nov"  "nov"  "oct"  "oct"  "nov"  "sept" "sept"
```

We will use different ways to convert the abbreviations into full words: `"sept"` → `"September"`, `"oct"` → `"October"`, and `"nov"` → `"November"`.

3a. Use a `for` loop and the `switch` statement to convert all the elements of `fall_abbr` into full words, and store the results in a vector named `fall_full`.

Ans: After the conversion, the `fall_full` will result in,

```
fall_full<-character(length(fall_abbr))
for(i in seq_along(fall_abbr)){
  abbr<-fall_abbr[i]
  full_name<-switch(abbr,
         sept="September",
         oct="October",
         nov="November",
         stop("Invalid value")
  )
  fall_full[i]<-full_name
}
fall_full
```

```
##  [1] "September" "September" "September" "November"  "September" "November"
##  [7] "November"  "November"  "September" "September" "October"   "October"
## [13] "September" "November"  "September" "September" "October"   "November"
## [19] "September" "October"   "November"  "October"   "October"   "November"
## [25] "November"  "October"   "October"   "November"  "September" "September"
```

3b. Carry out the conversion by creating a lookup table and subsetting the lookup table with characters.

Ans: Creating a lookup table and referencing this data to the `fall_abbr` table will give us the desired result.

```
lookup_table<-c(sept="September", oct="October", nov="November")
fall_full<-lookup_table[fall_abbr]
fall_full
```

```
##        sept        sept        sept         nov        sept         nov
## "September" "September" "September"  "November" "September"  "November"
##         nov         nov        sept        sept         oct         oct
##  "November"  "November" "September" "September"   "October"   "October"
##        sept         nov        sept        sept         oct         nov
## "September"  "November" "September" "September"   "October"  "November"
##        sept         oct         nov         oct         oct         nov
## "September"   "October"  "November"   "October"   "October"  "November"
##         nov         oct         oct         nov        sept        sept
##  "November"   "October"   "October"  "November" "September" "September"
```

3c. The function `ifelse()` is a vectorized version of the `if-else` statements. Use the function to convert all the elements of `fall_abbr`.

Ans: **Vectorized version**

```
fall_full <- ifelse(fall_abbr %in% names(lookup_table), lookup_table[fall_abbr], "Unknown Fall M
fall_full
```

```
##  [1] "September" "September" "September" "November"  "September" "November"
##  [7] "November"  "November"  "September" "September" "October"   "October"
## [13] "September" "November"  "September" "September" "October"   "November"
## [19] "September" "October"   "November"  "October"   "October"   "November"
## [25] "November"  "October"   "October"   "November"  "September" "September"
```