DS 5010: Assignment 1

Saurabh Yelne

9/12/2018

The book of R

Exercise 2.3

a. Create and store a sequence of values from 5 to −11 that progresses in steps of 0.3

```
x<-seq(5,-11,by=-0.3)
print(x)

##  [1]   5.0   4.7   4.4   4.1   3.8   3.5   3.2   2.9   2.6   2.3   2.0
## [12]   1.7   1.4   1.1   0.8   0.5   0.2  -0.1  -0.4  -0.7  -1.0  -1.3
## [23]  -1.6  -1.9  -2.2  -2.5  -2.8  -3.1  -3.4  -3.7  -4.0  -4.3  -4.6
## [34]  -4.9  -5.2  -5.5  -5.8  -6.1  -6.4  -6.7  -7.0  -7.3  -7.6  -7.9
## [45]  -8.2  -8.5  -8.8  -9.1  -9.4  -9.7 -10.0 -10.3 -10.6 -10.9
```

b. Overwrite the object from (a) using the same sequence with the order reversed

```
x<-sort(seq(5,-11,by=-0.3),decreasing = F)
# Alternate solution x <- x[length(x):1]
print(x)

##  [1] -10.9 -10.6 -10.3 -10.0  -9.7  -9.4  -9.1  -8.8  -8.5  -8.2  -7.9
## [12]  -7.6  -7.3  -7.0  -6.7  -6.4  -6.1  -5.8  -5.5  -5.2  -4.9  -4.6
## [23]  -4.3  -4.0  -3.7  -3.4  -3.1  -2.8  -2.5  -2.2  -1.9  -1.6  -1.3
## [34]  -1.0  -0.7  -0.4  -0.1   0.2   0.5   0.8   1.1   1.4   1.7   2.0
## [45]   2.3   2.6   2.9   3.2   3.5   3.8   4.1   4.4   4.7   5.0
```

c. Repeat the vector c(-1,3,-5,7,-9) twice, with each element repeated 10 times, and store the result. Display the result sorted from largest to smallest

```
vec<-sort(rep(c(-1,3,-5,7,-9),each=10, times=2),decreasing = T)
print(vec)

##   [1]  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  3  3  3
##  [24]  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3  3 -1 -1 -1 -1 -1 -1
##  [47] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -5 -5 -5 -5 -5 -5 -5 -5 -5
##  [70] -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -5 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9 -9
##  [93] -9 -9 -9 -9 -9 -9 -9 -9
```

d. Create and store a vector that contains, in any configuration, the following:

i. A sequence of integers from 6 to 12 (inclusive)
ii. A threefold repetition of the value 5.3
iii. The number −3
iv. A sequence of nine values starting at 102 and ending at the number that is the total length of the vector created in (c)

```
vec1<-c(6:12,rep(5.3,3),-3,seq(from=102, to= length(vec),length.out = 9))
print(vec1)

##  [1]   6.00   7.00   8.00   9.00  10.00  11.00  12.00   5.30   5.30   5.30
## [11]  -3.00 102.00 101.75 101.50 101.25 101.00 100.75 100.50 100.25 100.00
```

e. Confirm that the length of the vector created in (d) is 20

```
length(vec1)==20

## [1] TRUE
```

Exercise 2.4

a. Create and store a vector that contains the following, in this order:

– A sequence of length 5 from 3 to 6 (inclusive) – A twofold repetition of the vector c(2,-5.1,-33) – The value 7/42+2

```
vec2<-round(c(seq(3,6,length.out = 5),rep(c(2,-5.1,-33),2),7/42+2),2)
print(vec2)

##  [1]   3.00   3.75   4.50   5.25   6.00   2.00  -5.10 -33.00   2.00  -5.10
## [11] -33.00   2.17
```

b. Extract the first and last elements of your vector from (a), storing them as a new object

```
vec3<-vec2[c(1,length(x=vec2))]
print(vec3)

## [1] 3.00 2.17
```

c. Store as a third object the values returned by omitting the first and last values of your vector from (a)

```
vec4<-vec2[c(-1,-length(x=vec2))]
print(vec4)

##  [1]   3.75   4.50   5.25   6.00   2.00  -5.10 -33.00   2.00  -5.10 -33.00
```

d. Use only (b) and (c) to reconstruct (a).

```
a<-c(vec3[1],vec4,vec3[2])
a

##  [1]   3.00   3.75   4.50   5.25   6.00   2.00  -5.10 -33.00   2.00  -5.10
## [11] -33.00   2.17
```

e. Overwrite (a) with the same values sorted from smallest to largest

```
e<-sort(vec2,decreasing = F)
e

##  [1] -33.00 -33.00  -5.10  -5.10   2.00   2.00   2.17   3.00   3.75   4.50
## [11]   5.25   6.00
```

f. Use the colon operator as an index vector to reverse the order of (e), and confirm this is identical to using sort on (e) with decreasing=TRUE

```
e[12:1]

##  [1]   6.00   5.25   4.50   3.75   3.00   2.17   2.00   2.00  -5.10  -5.10
## [11] -33.00 -33.00

sort(e,decreasing = T)

##  [1]   6.00   5.25   4.50   3.75   3.00   2.17   2.00   2.00  -5.10  -5.10
## [11] -33.00 -33.00
```

g. Create a vector from (c) that repeats the third element of (c) three times, the sixth element four times, and the last element once

```
g<-c(rep(vec4[3],3), rep(vec4[6],4),vec4[10])
g

## [1]   5.25   5.25   5.25  -5.10  -5.10  -5.10  -5.10 -33.00
```

h. Create a new vector as a copy of (e) by assigning (e) as is to a newly named object. Using this new copy of (e), overwrite the first, the fifth to the seventh (inclusive), and the last element with the values 99 to 95 (inclusive), respectively

```
h<-e
h

##  [1] -33.00 -33.00  -5.10  -5.10   2.00   2.00   2.17   3.00   3.75   4.50
## [11]   5.25   6.00

h[c(1,5,6,7,12)]<-c(99:95)
h
```

```
## [1]  99.00 -33.00  -5.10  -5.10  98.00  97.00  96.00   3.00   3.75   4.50
## [11]   5.25  95.00
```

Exercise 2.5

a. Convert the vector c(2,0.5,1,2,0.5,1,2,0.5,1) to a vector of only 1s, using a vector of length 3

```
c(2,0.5,1,2,0.5,1,2,0.5,1) + c((-1),0.5,0) #vector of 3

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

b. The conversion from a temperature measurement in degrees Fahrenheit F to Celsius C is performed using the following equation:C = 5/9 (F – 32)

Use vector-oriented behavior in R to convert the temperatures 45, 77, 20, 19, 101, 120, and 212 in degrees Fahrenheit to degrees Celsius

```
f<-c(45,77,20,19,101,120,212)
5/9*(f-32)

## [1]   7.222222  25.000000  -6.666667  -7.222222  38.333333  48.888889
## [7] 100.000000
```

c. Use the vector c(2,4,6) and the vector c(1,2) in conjunction with rep and * to produce the vector c(2,4,6,4,8,12)

```
c1<-(rep(c(1,2),each=3))*c(2,4,6)
c1

## [1]  2  4  6  4  8 12
```

d. Overwrite the middle four elements of the resulting vector from (c) with the two recycled values -0.1 and -100, in that order

```
c1[2:5]<-c(-0.1, -100)
c1

## [1]    2.0   -0.1 -100.0   -0.1 -100.0   12.0
```

Exercise 3.1

a. Construct and store a 4 × 2 matrix that's filled row-wise with the values 4.3, 3.1, 8.2, 8.2, 3.2, 0.9, 1.6, and 6.5, in that order

```
mat_a<-matrix(data=c(4.3, 3.1, 8.2, 8.2, 3.2, 0.9, 1.6,
6.5),nrow=4,ncol=2,byrow=T)
mat_a

##      [,1] [,2]
## [1,]  4.3  3.1
## [2,]  8.2  8.2
## [3,]  3.2  0.9
## [4,]  1.6  6.5
```

b. Confirm the dimensions of the matrix from (a) are 3 × 2 if you remove any one row

```
mat_a.b<-mat_a[-4,]         # removing 4th row
dim(mat_a.b)

## [1] 3 2
```

c. Overwrite the second column of the matrix from (a) with that same column sorted from smallest to largest

```
mat_a.c<-mat_a
mat_a.c[,2]<-sort(mat_a.c[,2],decreasing = F)
mat_a.c

##      [,1] [,2]
## [1,]  4.3  0.9
## [2,]  8.2  3.1
## [3,]  3.2  6.5
## [4,]  1.6  8.2
```

d. What does R return if you delete the fourth row and the first column from (c)? Use matrix to ensure the result is a single-column matrix, rather than a vector

```
mat_a.d<-mat_a.c
matrix(data=mat_a.d[-4,-1])

##      [,1]
## [1,]  0.9
## [2,]  3.1
## [3,]  6.5
```

e. Store the bottom four elements of (c) as a new 2 × 2 matrix

```
mat_a.e<-mat_a.c
mat_a.e<-mat_a.e[3:4,1:2]
mat_a.e

##      [,1] [,2]
## [1,]  3.2  6.5
## [2,]  1.6  8.2
```

f. Overwrite, in this order, the elements of (c) at positions (4,2), (1,2), (4,1), and (1,1) with −1/2 of the two values on the diagonal of (e)

```
mat_a.c[c(4,1),c(2,1)] <- -1/2* (diag(mat_a.e))
mat_a.c

##      [,1] [,2]
## [1,] -4.1 -4.1
## [2,]  8.2  3.1
## [3,]  3.2  6.5
## [4,] -1.6 -1.6
```

Exercise 3.2

a. Calculate the following:

$$2/7 * \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 7 & 6 \end{bmatrix} - \begin{bmatrix} 10 & 20 \\ 30 & 40 \\ 50 & 60 \end{bmatrix}$$

```
matrix_a <- matrix(data=c(1,2,7,2,4,6),nrow = 3,ncol = 2)
matrix_a

##      [,1] [,2]
## [1,]    1    2
## [2,]    2    4
## [3,]    7    6

matrix_b <- matrix(data=c(10,20,30,40,50,60),nrow = 3,ncol = 2,byrow = T)
matrix_b

##      [,1] [,2]
## [1,]   10   20
## [2,]   30   40
## [3,]   50   60

2/7*((matrix_a-matrix_b))

##             [,1]      [,2]
## [1,]  -2.571429  -5.142857
```

```
## [2,]  -8.000000 -10.285714
## [3,] -12.285714 -15.428571
```

b. Store these two matrices:

$$A = \begin{bmatrix} 1 \\ 2 \\ 7 \end{bmatrix} B = \begin{bmatrix} 3 \\ 4 \\ 8 \end{bmatrix}$$

Which of the following multiplications are possible? For those that are, compute the result.
i. A·B

Ans. *Not possible because the total columns of 1st matrix(1) is not equal to the total rows of second matrix(3)*

```
A <- matrix(data=c(1,2,7),nrow = 3,ncol = 1)
B <- matrix(data=c(3,4,8),nrow = 3,ncol=1)
A

##      [,1]
## [1,]    1
## [2,]    2
## [3,]    7

B

##      [,1]
## [1,]    3
## [2,]    4
## [3,]    8
```

Here R gives an error as the both matrix cannot be multiplied

ii. AT·B

Ans. *The transpose of A matrix converts it into 1x3 matrix and B is 3x1 matrix so the multiplication is now possible*

```
t(A)%*%B

##      [,1]
## [1,]   67
```

iii. BT·(A·AT)

Ans. *Possible*

```
t(B)%*%(A%*%t(A))

##      [,1] [,2] [,3]
## [1,]   67  134  469
```

iv. (A·AT)·BT

Ans. *Not Possible*

```
#(A %*% t(A)) %*% t(B)
```

This is the error R is throwing 'Error in (A %% t(A)) %% t(B) : non-conformable arguments', which says that the dimensions of matrix are not correct for multiplication

v. [(B·B⊤)+(A·A⊤)−100I3]−1

Ans. *Possible*

```
((B%*%t(B))+(A%*%t(A))-100 * diag(3))-1
```

```
##      [,1] [,2] [,3]
## [1,]  -91   13   30
## [2,]   13  -81   45
## [3,]   30   45   12
```

c. For

$$A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

confirm that $A^{-1} \cdot A - I4$ provides a 4 × 4 matrix of zeros.

```
A<- matrix (data= c(2,0,0,0, 0,3,0,0, 0,0,5,0, 0,0,0,-1), nrow=4,ncol=4)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    0    0    0
## [2,]    0    3    0    0
## [3,]    0    0    5    0
## [4,]    0    0    0   -1
```

```
matrixcalc::svd.inverse(A) %*% A - diag(4) #using matrixcalc library. Also,
matlib library can be used.
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0    0
## [2,]    0    0    0    0
## [3,]    0    0    0    0
## [4,]    0    0    0    0
```

Thus we get a 4x4 matrix of zeros

R for Data Science

20.4.6 Exercises

1. What does mean(is.na(x)) tell you about a vector x? What about sum(!is.finite(x))?

Ans. *mean(is.na(x)) tells us the proportion of NA's in the vector x and sum(!is.finite(x)) tells us the elements which are not finite*

2. Carefully read the documentation of is.vector(). What does it actually test for? Why does is.atomic() not agree with the definition of atomic vectors above?

Ans. *is.vector returns TRUE if x is a vector of the specified mode having no attributes other than names. It returns FALSE otherwise. On the other hand atomic vector are linear vector of single type. They include logical, interger, double and character. For example*

is.vector(list())

[1] TRUE

is.vector(numeric())

[1] TRUE

is.atomic(list())

[1] FALSE

is.atomic(numeric())

[1] TRUE

Here we can see that while list is a recursive vector it is heterogeneous but it is a vector so is.vector returns true but list is not a atomic vector as it is not homogeneous so is.atomic returns false

3. Compare and contrast setNames() with purrr::set_names()

Ans. *Function setNames is from stats package and is written in camel case while purrr::set_names() is written in snake case. They basically do the same thing of setting a name of an object but purrr::set_names() has tweaked defaults, and stricter argument checking*

4. Create functions that take a vector as input and returns:

1.The last value. Should you use [or [[?

Ans. I will be using the double square brackets so that I can get the last value even if they are named

```
last_value <- function(v){
  v[[length(x=v)]]
}
 last_value(c(a=1,b=2,c=3)) # Using the function

## [1] 3

 last_value(c(1,2,3,4))

## [1] 4
```

2.The elements at even numbered positions

```
even_position <- function(v){
                  for (i in 1: length(v)){
                    if (i%%2==0) print (v[[i]])
                  }

}
even_position(4:7)

## [1] 5
## [1] 7
```

3.Every element except the last value

```
except_last_value <- function(v){
               v[-length(v)]
}
except_last_value(c(a=1,b=4,c=5))

## a b
## 1 4
```

4.Only even numbers (and no missing values)

```
even_number <- function(v){
  v[v%%2==0 & !is.na(v)]
}

even_number(c(1:7,NA,8,10,NA,NA,77,66,55,44))

## [1]  2  4  6  8 10 66 44
```

5. Why is x[-which(x > 0)] not the same as x[x <= 0]?

```
v1 <- c(1,2,3,4,5,-3,-4,-5,-6,NA,0,NA)
v1[-which(v1 > 0)]

## [1] -3 -4 -5 -6 NA  0 NA

v1[v1 <= 0]

## [1] -3 -4 -5 -6 NA  0 NA
```

As we can see the output from both the case is the same. Both gives elements which are less than equal to zero and also NA's

6. What happens when you subset with a positive integer that's bigger than the length of the vector? What happens when you subset with a name that doesn't exist?

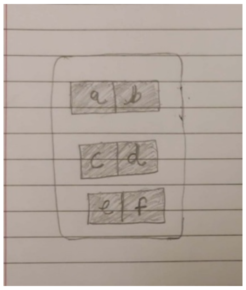
```
v1 <- c(1,2,3,4,5,-3,-4,-5,-6,NA,0,NA)
v2<-c(a=1,b=2,c=3)
length(v1)

## [1] 12

v1[13]

## [1] NA

v2["d"]

## <NA>
##   NA
```

In both the cases we get NA's for the values which are not present

20.5.4 Exercises

1. Draw the following lists as nested sets:

a. list(a, b, list(c, d), list(e, f))



b. list(list(list(list(list(list(a))))))