The R Book

Chapter 2: Essentials of the R Language
Session 4

Questions last week

Rounding floating point values

```
> x<-4.654583732637648594387372632
> round(x,digits=5)
[1] 4.65458
> round(x,3)
[1] 4.655
```

Practical situation of a list … I have no real example but :

Data frames are lists as well, but they have a few restrictions:

- you can't use the same name for two different variables
- all elements of a data frame are vectors
- all elements of a data frame have an equal length.

lists are suitable for executing functions to global datasets, without the need of using iterations or loops.

-powerful, fast but should be used with care. At our level, we will use dataframes.

Addresses within Vectors, length(), seq()

```
> y<-c(8,3,5,7,6,6,8,9,2,3,9,4,10,4,11)
> length(y)
[1] 15
> length(y[y>5])
[1] 9
```

Extract every nth element from a vector with seq()

> xv<-rnorm(1000,100,10) random numbers, length 1000

from, to (try 1000) by

mean 100

standard deviation 10

> xv[seq(25,length(xv),25)]

```
 [1]  78.55738 102.43888  98.35262  90.35767 108.45821 105.87016 103.74105
 [8]  88.86585 107.17449 107.34596  90.55796 106.00437 119.99036 101.28521
[15]  92.50904  92.89058 105.71306  84.45659 105.17407 100.62878 110.44917
[22]  85.21301 103.60246  85.07047  98.14852  99.96667  78.37900 115.25886
[29] 109.51843  98.88434  83.30707 105.62302  97.21630  90.99227 106.60842
[36]  74.92409  92.88968  90.89482  96.53100  91.82529
```

every 25th value

Finding Closest Values

find the value of xv that is closest to 108.0:

logical argument

```
which(abs(xv-108)==min(abs(xv-108)))
```

absolute minimum

```
[1] 813
```

- |abs|
- which() tests a logical operation

```
> xv[813]
[1] 107.9925
```

write a function : closest value to a specified value sv

```
> closest<-function(xv,cb){
+ xv[which(abs(xv-cb)==min(abs(xv-cb)))] }

> closest(xv,108)
[1] 107.9925
```

Finding Closest Values

find the value of xv that is closest to 108.0:

logical argument

```
which(abs(xv-108)==min(abs(xv-108)))
```

absolute minimum

```
[1] 813
```

- |abs|
- which() tests a logical operation

```
> xv[813]
[1] 107.9925
```

write a function : closest value to a specified value sv

```
> which.closest<-function(xv,sv){
+ which(abs(xv-sv)==min(abs(xv-sv))) }

> which.closest(xv,108)
[1] 813
```

Trimming Vectors Using Negative Subscripts

```
> x<- c(5,8,6,7,1,5,3)
> x
[1] 5 8 6 7 1 5 3
```

Take out first element

```
> z<-x[-1]
> z
[1] 8 6 7 1 5 3
```

Take out smallest and largest value

- sort → remove first element : x[-1]
  → remove last element : x[-length(x)]
- concatenation of both: -c(1,length(x))

```
> sort(x)
[1] 1 3 5 5 6 7 8

> sort(x)[-c(1,length(x))]
[1] 3 5 5 6 7
```

Trimming Vectors Using Negative Subscripts

```
> x<- c(5,8,6,7,1,5,3)
> x
[1] 5 8 6 7 1 5 3
```

Take out first element

```
> z<-x[-1]
> z
[1] 8 6 7 1 5 3
```

Take out smallest and largest value

```
> sort(x)[-c(1,length(x))]
[1] 3 5 5 6 7
```

Calculate mean of trimmed vector

```
> trim.mean <- function(x) mean(sort(x)[-c(1,length(x))])
> trim.mean(x)
[1] 5.2
```

Taking out multiple values

omitting all the multiples of seven (7, 14, 21, etc.)

```
> vec<-1:50
> (multiples<-floor(length(vec)/7))
[1] 7

> (subscripts<-7*(1:multiples))
[1]  7 14 21 28 35 42 49

> vec[-subscripts]
 [1]  1  2  3  4  5  6  8  9 10 11 12 13 15 16 17 18 19 20 22 23 24 25 26 27 29
[26] 30 31 32 33 34 36 37 38 39 40 41 43 44 45 46 47 48 50
```

Taking out multiple values
omitting all the multiples of seven (7, 14, 21, etc.)

```
vec<-1:50
vec[-(1:50*(1:50%%7==0))] or use the modulo

> vec[-(1:50)]
Integer(0)

> 1:50%%7==0
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
[13] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
[25] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
[37] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[49]  TRUE FALSE

> 1:50*(1:50%%7==0)
 [1]  0  0  0  0  0  0  7  0  0  0  0  0  0 14  0  0  0  0  0  0 21  0  0  0  0
[26]  0  0 28  0  0  0  0  0  0 35  0  0  0  0  0  0 42  0  0  0  0  0  0 49  0

> vec[-(1:50*(1:50%%7==0))]
 [1]  1  2  3  4  5  6  8  9 10 11 12 13 15 16 17 18 19 20 22 23 24 25 26 27 29
[26] 30 31 32 33 34 36 37 38 39 40 41 43 44 45 46 47 48 50
```

Logical Arithmetic

logical expressions evaluate to either true or false. R can coerce TRUE or FALSE into numerical values: 1 or 0

```
x<-0:6
x<4
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE
```

logical functions « all » and « any »

```
all(x>0)
[1] FALSE

any(x<0)
[1] FALSE

sum(x<4)
[1] 4
```

random number (0-1)

```
(x<4)*runif(7)
[1] 0.9433433 0.9382651 0.6248691 0.9786844 0.0000000 0.0000000 0.0000000
```

Logical Arithmetic

Useful to generate simplified factor levels

```
(treatment<-letters[1:5])
[1] "a" "b" "c" "d" "e"

(t2<-factor(1+(treatment=="b")+2*(treatment=="c")+2*(treatment=="d")))
[1] 1 2 3 3 1

Levels: 1 2 3
```

x <- y x is assigned the value of y (x gets the values of y);

x=y in a function or a list x is set to y unless you specify otherwise;

x == y produces TRUE if x is exactly equal to y and FALSE otherwise.

Table 2.3. Logical operations.

Symbol	Meaning
!	logical NOT
&	logical AND
|	logical OR
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	logical equals (double =)
!=	not equal
&&	AND with IF
||	OR with IF
xor(x,y)	exclusive OR
isTRUE(x)	an abbreviation of identical(TRUE,x)

Evaluation of combinations of TRUE and FALSE

x <- c(NA, FALSE, TRUE)
names(x) <- as.character(x) to assign the values as

```
> outer(x, x, "&")
        <NA> FALSE  TRUE
<NA>      NA FALSE    NA
FALSE FALSE FALSE FALSE
TRUE      NA FALSE  TRUE
```

Only TRUE & TRUE evaluates to TRUE.

```
> outer(x, x, "|")
       <NA> FALSE TRUE
<NA>     NA    NA TRUE
FALSE    NA FALSE TRUE
TRUE   TRUE  TRUE TRUE
```

Only FALSE & FALSE evaluates to FALSE.

Repeats

```
> rep(9,5)
[1] 9 9 9 9 9

> rep(1:4,2)
[1] 1 2 3 4 1 2 3 4

> rep(1:4,each=2)
[1] 1 1 2 2 3 3 4 4

> rep(1:4,1:4)
 [1] 1 2 2 3 3 3 4 4 4 4

> rep(1:4,c(4,1,4,2))
 [1] 1 1 1 1 2 3 3 3 3 4 4
```

Generate Factor Levels

gl(‘up to’, ‘with repeats of’, ‘to total length’)

```
> gl(4,3)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
Levels: 1 2 3 4

> gl(4,3,24)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
Levels: 1 2 3 4

> gl(4,3,20)
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3
Levels: 1 2 3 4

> gl(4,1,20)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
Levels: 1 2 3 4

> gl(3,2,24,labels=c("A","B","C"))
 [1] A A B B C C A A B B C C A A B B C C A A B B C C
Levels: A B C
```

Generating Regular Sequences of Numbers

```
> 10:18
[1] 10 11 12 13 14 15 16 17 18

> 18:10
[1] 18 17 16 15 14 13 12 11 10

> -0.5:8.5
 [1] -0.5  0.5  1.5  2.5  3.5  4.5  5.5  6.5  7.5  8.5

> seq(0,1.5,0.2)
[1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 not > 1.5

> seq(1.5,0,-0.2)
[1] 1.5 1.3 1.1 0.9 0.7 0.5 0.3 0.1 not < 0
```

To generate an array of values between min(x) and max(x)

```
x.values<-seq(min(x),max(x),(max(x)-min(x))/100)
```

Generating Regular Sequences of Numbers

To generate an array of values between min(x) and max(x)
x.values<-seq(min(x),max(x),(max(x)-min(x))/100)

Generate a sequence of values with the same length as vector x

```
> x<-rnorm(18,10,2)
> seq(88,50,along=x)
 [1] 88.00000 85.76471 83.52941 81.29412 79.05882 76.82353 74.58824 72.35294
 [9] 70.11765 67.88235 65.64706 63.41176 61.17647 58.94118 56.70588 54.47059
[17] 52.23529 50.00000
```

sequence(x), to produce a vector consisting of sequences

```
> sequence(5)
[1] 1 2 3 4 5

> sequence(5:1)
 [1] 1 2 3 4 5 1 2 3 4 1 2 3 1 2 1

> sequence(c(2,4,1))
[1] 1 2 1 2 3 4 1
```

Variable Names

- Case-sensitive, x is not X.
- Should not begin with numbers (e.g. 1x) or symbols (e.g. %x).
- Should not contain blank spaces: back.pay (not back pay).

Sorting, Ranking and Ordering

```
> houses<-read.table("/home/.../houses.txt",header=T)
> attach(houses)
> ranks<-rank(Price)
> sorted<-sort(Price)
> ordered<-order(Price)
> view<-data.frame(Price,ranks,sorted,ordered)
> view
```

	Price	ranks	sorted	ordered
1	325	12.0	95	9
2	201	10.0	101	6
3	157	5.0	117	10
4	162	6.0	121	12
5	164	7.0	157	3
6	101	2.0	162	4
7	211	11.0	164	5
8	188	8.5	188	8
9	95	1.0	188	11
10	117	3.0	201	2
11	188	8.5	211	7
12	121	4.0	325	1

Sorting, Ranking and Ordering

- Using order with subscripts is a much safer option than using sort !

- Values of the response variable and the explanatory variables could be uncoupled with sort

```
> names(houses)
[1] "Location" "Price"
> Location[order(Price)]
 [1] Reading     Staines     Winkfield   Newbury     Bracknell   Camberley
 [7] Bagshot     Maidenhead  Warfield    Sunninghill Windsor     Ascot


> Location[rev(order(Price))]
 [1] Ascot       Windsor     Sunninghill Warfield    Maidenhead  Bagshot
 [7] Camberley   Bracknell   Newbury     Winkfield   Staines     Reading
```