Inroduction to R

Maja Kuzman

2019-09-30

R, Rstudio, Rmd

How can we use R and RStudio?

- console
- script
- notebook:
  new chunk: ctrl + ALT + i

What can we do with R?

- We can use R as a calculator!

Try it:

Calculate the result of 13%/%3 and 13%%3.

```r
13%/%3
```

```
## [1] 4
```

```r
13%%3
```

Variables and data structures

Variables

```
myFirstNumber <- 0.1
myFirstVector <- c(2, 3, 7, 8)
```

-> Environment
-> call the variable by name
-> print() function
-> one line, part of code shaded and CTRL +ENTER

```
myFirstNumber
```

```
## [1] 0.1
```

```
print(myFirstNumber)
```

```
## [1] 0.1
```

Data structures in R

vectors
list matrix
data.frame
factors

Vectors

```
nVec <- c(1, 5, 7, 9, 12.5) # numeric vector
cVec <- c("a", "b", "some words") # character vector
lVec <- c(TRUE, FALSE, T, T, F) # logical vector

vvec <- c(1,5, "nesto")
vvec
```

```
## [1] "1"     "5"     "nesto"
```

```
nVec
```

```
## [1]  1.0  5.0  7.0  9.0 12.5
```

```
cVec
```

```
## [1] "a"          "b"          "some words"
```

```
lVec
```

```
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

```
nn <- c( "sth", 3)
```

Matrices

Matrices are tables that have rows and columns and store elements of same type.

```
y <- matrix(1:20, nrow=5,ncol=4)
y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

Data frames

```
mydf <- data.frame( firstColumn = c(1,5,7),
                    secondColumn = c("a","b","third Element"))
mydf
```

```
##   firstColumn  secondColumn
## 1           1             a
## 2           5             b
## 3           7 third Element
```

Lists

```r
l <- list(firstElement = c(1,5,44,6),
          second = c("a", 3),
          y)
l
```

```
## $firstElement
## [1]  1  5 44  6
##
## $second
## [1] "a" "3"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    6   11   16
## [2,]    2    7   12   17
## [3,]    3    8   13   18
## [4,]    4    9   14   19
## [5,]    5   10   15   20
```

Factors

```
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
gender
```

```
##  [1] male   male   male   male   male   male   male   male   male   male  
## [11] male   male   male   male   male   male   male   male   male   male  
## [21] female female female female female female female female female female
## [31] female female female female female female female female female female
## [41] female female female female female female female female female female
## Levels: female male
```

```
gender[2] <- "unknown"
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = "unknown"): invalid factor
## level, NA generated
```

```
gender
```

```
##  [1] male   <NA>   male   male   male   male   male   male   male   male  
## [11] male   male   male   male   male   male   male   male   male   male  
## [21] female female female female female female female female female female
## [31] female female female female female female female female female female
## [41] female female female female female female female female female female
## Levels: female male
```

```
as.numeric(gender)
```

Factors

If something is weird, factors are the usual suspects..

```
xx <- factor(sample(1:15,20, replace = T))
xx
```

```
##  [1] 11 13 15 7  1  6  13 6  5  9  4  14 15 9  5  8  13 14 7  1 
## Levels: 1 4 5 6 7 8 9 11 13 14 15
```

If you want a normal numeric vector:

```
as.numeric(xx)
```

```
##  [1]  8  9 11  5  1  4  9  4  3  7  2 10 11  7  3  6  9 10  5  1
```

you should do this...

```
as.numeric(as.character(xx))
```

```
##  [1] 11 13 15  7  1  6 13  6  5  9  4 14 15  9  5  8 13 14  7  1
```

Vectors

The basics

What we can do with vectors:

Make a vector c()

You can make a vector by using the function c() (concatenate). Here is an example of vectors myFirstvector, and myFirstSequence:

```
myFirstVector <- c("some words","p","word", "last one")
myFirstSequence <- 1:4
```

Subsetting [ ]

Print the whole vector

```
myFirstVector
```

```
## [1] "some words" "p"          "word"       "last one"
```

```
myFirstSequence
```

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

Print third element in a vector

```
myFirstVector[3]
```

```
## [1] "word"
```

```
myFirstSequence[3]
```

```
## [1] 3
```

Access multiple elements:

Provide a vector of positions to look at:

```
myFirstVector
```

```
## [1] "some words" "p"          "word"         "last one"
```

```
myFirstVector[c(2,3)]
```

```
## [1] "p"    "word"
```

```
somePositions <- c(2,3)
somePositions
```

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

```
myFirstVector[somePositions]
```

```
## [1] "p"    "word"
```

```
c(1:4,5:7)
```

```
## [1] 1 2 3 4 5 6 7
```

Exercise!

1. Create a vector named myvector that contains numbers 15,16,17,18 and 20.
2. Get first and third number in the vector by subsetting.

Solution:

```
myvector <- c(15:18,20)
```

a) subset directly the positions

```
myvector[c(1,3)]
```

```
## [1] 15 17
```

or like this.. b) (define a vector then subset by vector)

```
mypos <- c(1,3)
myvector[mypos[1]]
```

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

or like this: c) subset by logical indices

Think about it!

What happened here??:

```
## [1] "some words" "p"          "word"       "last one"
## [1] "some words" "some words" "some words"
## [1] "some words" "word"       NA
```

Basic operation on vectors

Same as on numbers:
+
-
/
*

Example:

Multiplication by constant

```
someothervector * 0.5
```

```
## [1] 0.5 0.0 0.5 0.0 0.5
```

Multiplication by other vector:

I) SAME size

```
someothervector
```

```
## [1] 1 0 1 0 1
```

```
someothervector * 1:5
```

```
## [1] 1 0 3 0 5
```

II) DIFFERENT size : recycling *because why not.*

```
someothervector*c(0.3,0.1)
```

```
## Warning in someothervector * c(0.3, 0.1): longer object length is not a
## multiple of shorter object length
```

```
## [1] 0.3 0.0 0.3 0.0 0.3
```

```
c(1,2,3,4,5,6) * 1:2 # doesnt produce a warning
```

```
## [1]  1  4  3  8  5 12
```

Basic comparisons

The following will return a logical vector for every compared position:

```
someothervector == 1
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE
```

```
someothervector == c(1,0,1,0,1)
```

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

```
someothervector>0
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE
```

what happened here?

```
someothervector[someothervector>0]
```

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

Exercise: Multiplication, recycling and comparison.

1. Multiply your myvector by c(0.1, 0.2) and save it to vector result
   -> what do you expect to get??
2. Check if you get what you expected by comparing it to vector you expect to get :)

Solution:

```
result <- myvector*c(0.1,0.2)
```

```
## Warning in myvector * c(0.1, 0.2): longer object length is not a multiple
## of shorter object length
```

```
round(result,1) == c(1.5, 3.2, 1.7, 3.6, 2.0)
```

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

```
as.numeric(as.character(result)) == c(1.5, 3.2, 1.7, 3.6, 2.0)
```

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

```
result
```

```
## [1] 1.5 3.2 1.7 3.6 2.0
```

Exercise: Multiplication, recycling and comparison.

1. Return only ELEMENTS in myvector that are smaller then 17. -> hint - use subsetting and comparison.

```
myvector[myvector<17 ]
```

BREAK AFTER YOU FINISH THOSE 4 EXERCISES!!!
(20 minutes from now break over!)

Everyone should be ok with those ways of subsetting! To practice:
Exercise subsetting:

```
set.seed(1)
x <- sample(1:20, 50, replace=T)
x
```

```
##  [1]  4  7  1  2 11 14 18 19  1 10 14 10  7  9 15  5  9 14  5  5  2 10 12
## [24] 15  1 20  3  6 10 10  6 15 20 20 12  6  8 12  6  7 19 10  6 14  2 13
## [47] 18 14  6  1
```

1) from vector x get only elements which are larger then 5.

```
x[x>5]
```

```
##  [1]  7 11 14 18 19 10 14 10  7  9 15  9 14 10 12 15 20  6 10 10  6 15 20
## [24] 20 12  6  8 12  6  7 19 10  6 14 13 18 14  6
```

2) Save positions from 1 to length(x) to vector y. Get only the elements from x which are in position which is larger then 5!

```
y <- 1:length(x)
x[y>5]
```

```
##  [1] 14 18 19  1 10 14 10  7  9 15  5  9 14  5  5  2 10 12 15  1 20  3  6
## [24] 10 10  6 15 20 20 12  6  8 12  6  7 19 10  6 14  2 13 18 14  6  1
```

3) from vector x get only elements which are divisable by 7.

```
x[(x%%7)==0]
```

```
## [1]  7 14 14  7 14  7 14 14
```

```
x[x%%7==0]
```

```
## [1]  7 14 14  7 14  7 14 14
```

4) Get only the elements from x which are in positions which are divisable by 7!

```
x[y%%7==0]
```

```
## [1] 18  9  2  6 12 10  6
```

Logical operators

AND: &

```
##   first_second TRUE. FALSE.
## 1         TRUE  TRUE  FALSE
## 2        FALSE FALSE  FALSE
```

OR: |

```
##   first_second TRUE. FALSE.
## 1         TRUE  TRUE   TRUE
## 2        FALSE  TRUE  FALSE
```

NOT: ! TRUE -> FALSE
FALSE -> TRUE

They are used in a following way:

```
firstLogical <- c(TRUE, TRUE, FALSE, FALSE)
secondLogical <- c(TRUE, FALSE, TRUE, FALSE)
firstLogical & secondLogical
firstLogical | secondLogical
! firstLogical
```

Exercise: Subset by logical indexes

1. Return all the elements from your vector that are divisable by 3.
2. Return all the elements from your vector that are divisable by 3 OR NOT divisable by 2.

remember:
x == 0 - where is x equal to 0
x%%5 -> gives you the modulo while dividing x by 5

Solution?

```
mm <- x%%3==0
nn <- x%%2!=0
x[mm|nn]
```

```
##  [1]  7  1 11 18 19  1  7  9 15  5  9  5  5 12 15  1  3  6  6 15 12  6 12
## [24]  6  7 19  6 13 18  6  1
```

Functions

Some useful functions

names

Assigns names to elements or returns names for elements.

```
names(someothervector)
```

```
## NULL
```

```
names(someothervector) <- c("one", "one","one", "zero","zero")
someothervector
```

```
##  one  one  one zero zero
##    1    0    1    0    1
```

Use it wisely

You can subset by names.. kind of.

```
someothervector["one"]
```

```
## one
##   1
```

length - Gives you the length of the vector:

```
length(someothervector)
```

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

unique -Gives you all distinct elements in your vector:

```
unique(someothervector)
```

```
## [1] 1 0
```

duplicated - logical is x duplicated or no?

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

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

```
duplicated(c(1,2,1,2,3,4))
```

```
## [1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

1) get all elements from first which appear in second.

```
first[first%in%second]
```

```
## [1] 3 5 3 5
```

```
first[first%in%second==TRUE]
```

```
## [1] 3 5 3 5
```

2) get all distinct elements from first which appear in second.

```
unique(first[first%in%second])
```

```
## [1] 3 5
```

3) get all elements from first which DON'T appear in second.

```
first[ !(first%in%second)]
```

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

```
first[first%in%second==FALSE]
```

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

names(s) give me al the elemrts that are called "one"

```
s <- 1:5
names(s) <- c("one", "one", "one", "none", "none")
s["one"]
```

```
## one
##   1
```

```
nms <- names(s)
s[nms=="one"]
```

```
## one one one
##   1   2   3
```

table : gives you list of all elements and counts them

```
table(someothervector)
```

```
## someothervector
## 0 1
## 2 3
```

Get all the elements that appear 4 times

```
tt <- table(x)
names(tt)
```

Math:

```
someothervector
```

```
##  one  one  one zero zero
##     1    0    1    0    1
```

```
sum(someothervector)
```

```
## [1] 3
```

```
mean(someothervector)
```

```
## [1] 0.6
```

```
sd(someothervector)
```

```
## [1] 0.5477226
```

```
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    1.00    6.00   10.00    9.68   14.00   20.00
```

sample - gives you random numbers from a vector

```
sample(1:100, 10)
```

```
##  [1]  70  87 100  75  81  13  40  89  48  93
```

HELP!!??!!

HELP!!??!!

```r
?unique
?`%in%`

#example(unique)
```

Matrix:

Matrix is defined by:

```
somedata <- sample(50)
m <- matrix(somedata, nrow=10, ncol=5)
m
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   23   21   30   32    6
##  [2,]   46   31   10   12   14
##  [3,]   20   38    7   42    4
##  [4,]   39   17   34    8   37
##  [5,]   29    9    2   24   27
##  [6,]   13   50   43    3    5
##  [7,]   22   19    1   40   36
##  [8,]   49   26   47   44   41
##  [9,]   28   16   15   35   18
## [10,]   33   11   25   48   45
```

Get the number of rows with nrow. Columns with ncol, dimensions with dim, works also with data.frames:

```
nrow(m)
```

```
## [1] 10
```

```
ncol(m)
```

To subset a matrix, give 2 coordinates: first one says which rows to subset, second one says which columns to subset! If one coordinate is empty, it means all rows. Example; to get the elements from first two rows and second column, do:

```
m[ 1:2, ] <- 3
m
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    3    3    3    3    3
##  [2,]    3    3    3    3    3
##  [3,]   20   38    7   42    4
##  [4,]   39   17   34    8   37
##  [5,]   29    9    2   24   27
##  [6,]   13   50   43    3    5
##  [7,]   22   19    1   40   36
##  [8,]   49   26   47   44   41
##  [9,]   28   16   15   35   18
## [10,]   33   11   25   48   45
```

To change those elements, just assign any values to them! for example, change those values to 15 and 16: (elements from first two rows and second column,)

```
m[1:2,2] <- c(15,16)
m
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    3   15    3    3    3
##  [2,]    3   16    3    3    3
##  [3,]   20   38    7   42    4
##  [4,]   39   17   34    8   37
##  [5,]   29    9    2   24   27
##  [6,]   13   50   43    3    5
##  [7,]   22   19    1   40   36
##  [8,]   49   26   47   44   41
##  [9,]   28   16   15   35   18
## [10,]   33   11   25   48   45
```

Exercise:

Calculate the sum of all values for the column 1, column 2, .. column 5 of the matrix.

```
ll <- m[,1]
sum(ll)
```

```
## [1] 239
```

```
sum(m[,2])
```

```
## [1] 217
```

```
sum(m[,3])
```

```
## [1] 180
```

```
sum(m[,4])
```

```
## [1] 250
```

you can do this more easily with colSums (also rowSums exists):

```
colSums(m)
```

1) Make one matrix mrows which will be the the same dimensions as m and each element will have the number of that row in the matrix. 2) Make a matrix mcols which will have elements with number of the column this element is in. 3) Change all the elements in matrix m for which column number is greater then row number to 0!

Data frames

There are some data frames already available in R, one of them is iris.
To see it you can use view(iris) or head(iris).

```
tail(iris)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
```

To add columns, do this:

```
iris$newColumn <- iris$Sepal.Length
```

To remove column, assign NULL to it.

```
iris$newColumn <- NULL
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
```

1. Save a Sepal.Length column from iris data frame to variable sl.
2. Divide this variable by its length and save the result in variable slscaled.
3. Calculate sum of squares for this vector (variable slscaled).

```
ssl <- iris$Sepal.Width
sl <- iris$Sepal.Length



myfunction <- function(sl){
    slscaled <- sl/length(sl)
    sum(slscaled^2)
}

myfunction(sl)
```

```
## [1] 0.2321711
```

```
myfunction(ssl)
```

```
## [1] 0.06357333
```

```
myfunction(iris$Sepal.Length)
```

```
## [1] 0.2321711
```

Lists

Make a list like this:

```
l <- list(1,2,3:5)
l
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3 4 5
```

Subset a list with [ ] or [[ ]] or $.

What is the difference? str, typeof or class functions will get you the structure of your variable.

```
str(x)
```

```
##  int [1:50] 4 7 1 2 11 14 18 19 1 10 ...
```

```
typeof(x)
```

Functions

Exercise:

1. Save a column Sepal.Length to variable sl.
2. Divide this variable by its length and save the result in variable slscaled.
3. Calculate sum of squares for this vector (variable slscaled).

To do this many times, you can define your own function that does this so there is no repetative code!

```r
myfunction <- function(sl){
    slscaled <- sl/length(sl)
    ssq <- sum(slscaled^2)
    ssq
}
```

Function format:

```
functionName <- function(arguments, argumentWithDefault=somedefaultValue){
    #we do some operations here
    someLocalVariable <- 2*argumentWithDefault
    returnedValueComesLast <- someLocalvariable/5
    returnedValueComesLast
}
```

Test this function on variable sl, on iris$Petal.Length, on iris$Sepal.Length

Write a function multipleOccurences that will take one argument (a vector), and return the number of elements that appear more then once! Use function table!
a)

a - more difficult) Add another argument to the function, a number n, which will tell you how many times an element needs to appear in order for it to be returned!

b) Use the function multipleOccurences to return all values of iris$Sepal.Width which appear more then once.
c) Use the operator %in% to return all rows in iris which have Sepal.Width that appears more then once. Example usage of %in%:

```
x <- 1:15
x%in% c(3,5,6)
```

```
##  [1] FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE
```