Introduction to R

Maja Kuzman

2019-10-25

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:

Add a new chunk to line 66.
Calculate the result of 13%/%3 and 13%%3.

We can even do better than that...

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, "word")
vvec
```

```
## [1] "1"    "5"    "word"
```

```
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
```

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

```
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

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

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

```
##  [1] 10 15 5  14 4  10 8  6  5  3  14 2  13 1  15 11 4  3  13 14
## Levels: 1 2 3 4 5 6 8 10 11 13 14 15
```

If you want a normal numeric vector:

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

```
##  [1]  8 12  5 11  4  8  7  6  5  3 11  2 10  1 12  9  4  3 10 11
```

you should do this...

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

```
##  [1] 10 15  5 14  4 10  8  6  5  3 14  2 13  1 15 11  4  3 13 14
```

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:

1) BY POSITION

Provide a vector of positions to look at:

```
myFirstVector
```

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

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

```
## [1] "some words" "word"
```

```
somePositions <- c(1,3)
somePositions
```

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

```
myFirstVector[somePositions]
```

```
## [1] "some words" "word"
```

Access multiple elements:

1) BY INCLUSION

Provide a LOGICAL vector :

```
myFirstVector
```

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

```
myFirstVector[c(TRUE, FALSE, TRUE,FALSE)]
```

```
## [1] "some words" "word"
```

```
someLogicalVector <- c(TRUE,TRUE, FALSE, FALSE)
someLogicalVector
```

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

```
myFirstVector[someLogicalVector]
```

```
## [1] "some words" "p"
```

Exercise!

1. Create a vector named myvector that contains numbers 15,16,17,18 and 20.
2. Get 2nd and 4th number in the vector by subsetting.
3. Get 2nd and 4th number in the vector by providing a logical vector.

Solution:

```
myvector <- c(15:18,20)
myvector[c(2,4)]
```

```
## [1] 16 18
```

or like this..

```
thosePositions <- c(4,2)
myvector[thosePositions]
```

```
## [1] 18 16
```

or like this:

```
myvector[c(FALSE, TRUE,FALSE, TRUE, FALSE)]
```

Think about it!

What happened here??:

```
someothervector <- c(1,0,1,0,1)
myFirstVector[someothervector]
myFirstVector[as.logical(someothervector)]
```

```
## [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)
   -> 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
```

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

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

...

WHAT?

Exercise: Multiplication, recycling and comparison.

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

```
myvector[ ??? ]
```

We want to get this:

```
## [1] 15 16
```

Exercise: Multiplication, recycling and comparison.

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

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

```
## [1] 15 16
```

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.
3. Return all the elements from your vector which are on EVEN positions (2,4,6,...)

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

Solution?

```
myvector[myvector%%3==0]
```

```
## [1] 15 18
```

Exercise: Subset by logical indexes

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 != 0 - not equal to

Solution:

```
myvector[(myvector%%3==0)| (myvector%%2!=0)]
```

```
## [1] 15 17 18
```

Return all the elements from your vector which are on EVEN positions (2,4,6,...)

```
myvector[c(F,T)]
```

```
## [1] 16 18
```

Functions

Some useful functions

length - Gives you the length of the vector:

```
length(someothervector)
```

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

unique -Gives you all unique elements in your vector:

```
unique(someothervector)
```

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

table : gives you list of all elements and counts them

```
table(someothervector)
```

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

Math:

```
sum(someothervector)
```

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

```
mean(someothervector)
```

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

```
sd(someothervector)
```

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

```
summary(someothervector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     1.0     0.6     1.0     1.0
```

more random math:

sample - gives you random numbers from a vector

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

```
##  [1] 68 14 34 58 56 38 85 10 54 83
```

rnorm - gives you 10 random numbers from normal distribution with mean=0 and sd=1

```
rnorm(10, mean = 0, sd = 1)
```

```
##  [1] -1.61674204 -0.25553117  0.44857984  0.55719096 -1.36979049
##  [6] -1.14230952  0.03393868  1.87759256  0.02449152 -0.14853960
```

analogously, rpois, runif

```
runif(n = 10, min=10, max=20)
```

Exercise!

What will the following produce?

```
x=seq(1,6, by=2)
x[x>4]
mean(x)
median(x)
x[c(T,F)]
rev(x)
```

Exercise!

Useful functions:

```
x=seq(1,6, by=1)
x[x>4]
mean(x)
median(x)
x[c(T,F)]
rev(x)
```

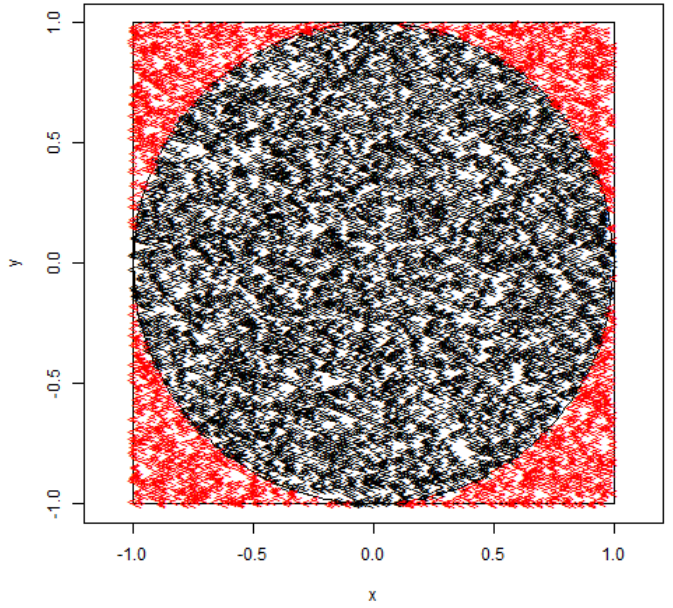
Create a sequence of numbers from 1 to 1000 and save it in a vector myseq.
Part 1: What is the mean of the elements which are larger then the median?
Part 2: What is the sum of every second element?
Part 3: sum all the elements pairwise by summing the first one with the last one, second one with second to last, and so on.

HELP!!??!!

HELP!!??!!

```
?runif
example(runif)
```

Exercise - estimate pi



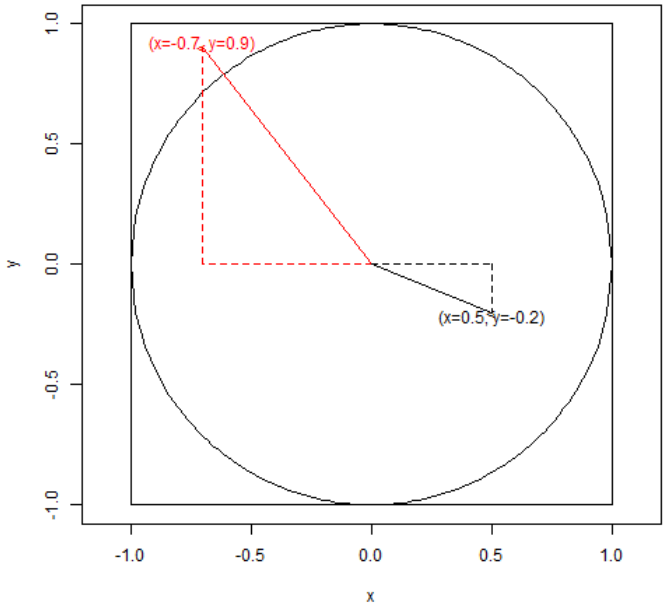
Exercise - estimate pi

Do the following:

Create random points in 2D:

1. Make a vector x with 1 000 000 random numbers uniformly distributed from -1 to 1.
2. Make another vector y with 1 000 000 random numbers uniformly distributed from -1 to 1.

Exercise - estimate pi



Exercise - estimate pi

Do the following:

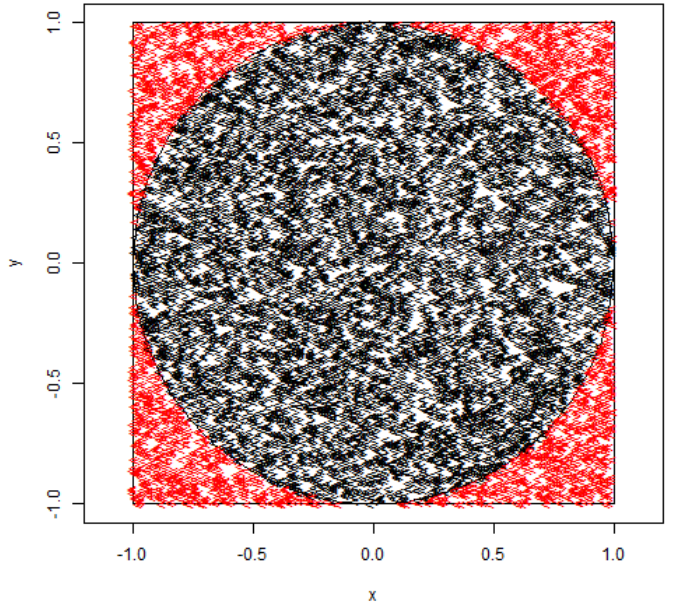
Calculate percentage of points that fall within the circle:

Calculate distance from center for each point!

```
3. Create a vector dist that will
calculate the result of x^2 + y^2

4. Create a logical vector
insideCircle that will check if
dist is smaller then 1^2
```

Exercise - estimate pi



Exercise - estimate pi

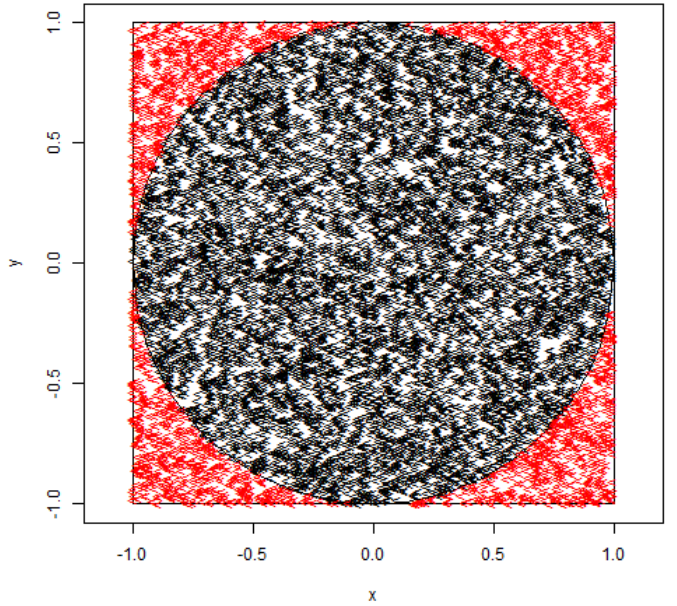
Do the following:

Calculate the ratio of points that fall into the circle.

```
Useful functions:

length(x) : vector length
as.numeric(x): conversion to num
mean(x): mean of all values in x
```

Exercise - estimate pi



And what now? :)

--

pi = 3.1415926535...