Introduction to R

Maja Kuzman

2019-05-06

Raise your hand!

Please interrupt me

when you stop following

when you have a question

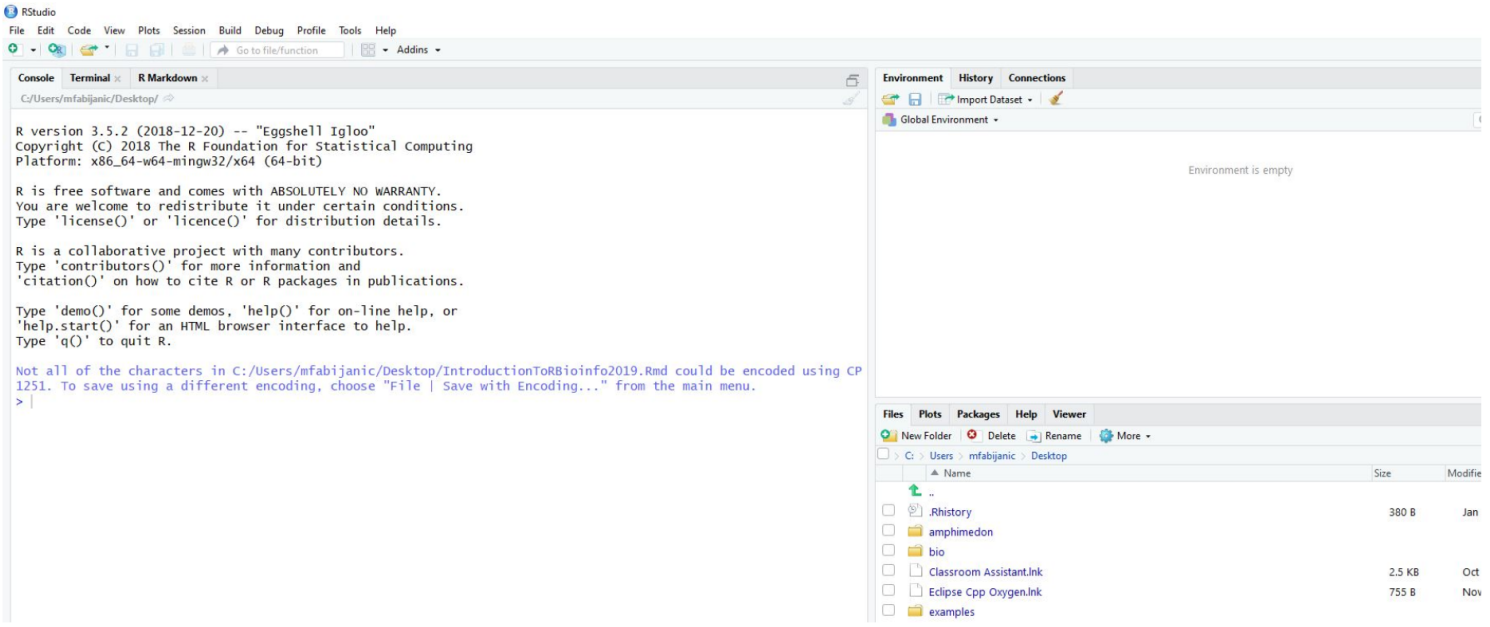
when you notice a mistake

...

1. Open RStudio

- If you don't have R:
  install R from:
  https://cloud.r-project.org/

- If you don't have Rstudio:
  install rstudio from:
  https://www.rstudio.com/products/rstudio/download/



if you are bored

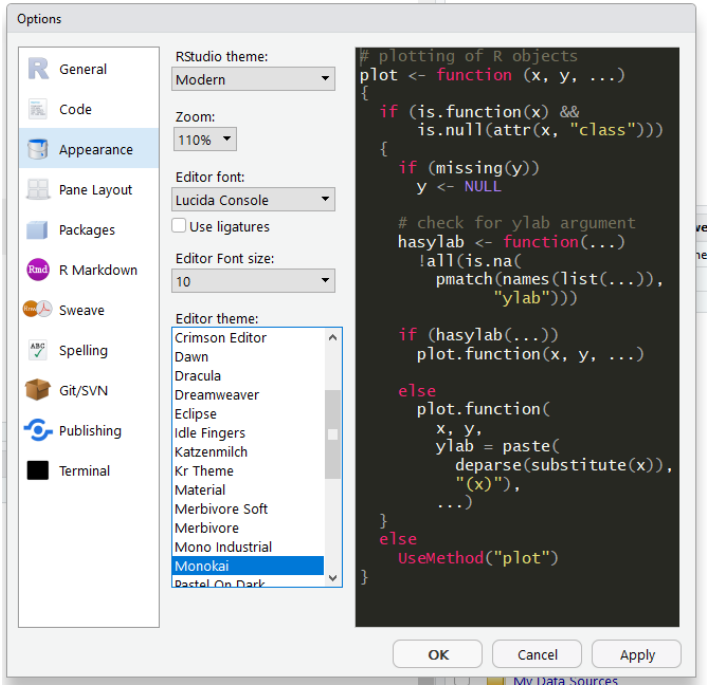
2. Customize your R studio:

Tools -> Global Options ... ->

1. General:
Allways save history - OFF

2. Code:
Tab width: 4

3. Appearance:
Editor theme: Monokai



How can we use R and RStudio?

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

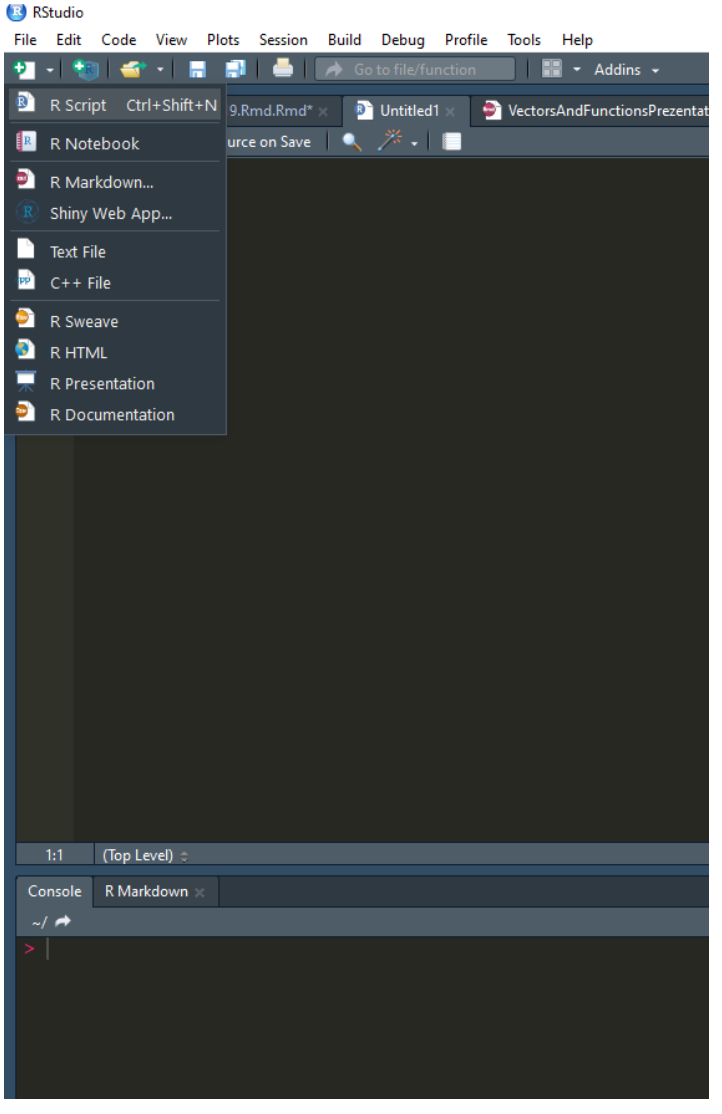
What can we do with R?

- We can use R as a calculator!

Try it in the console and in the script:

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, "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
```

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

```
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
```

Factors

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

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

```
##  [1] 12 4  4  8  8  1  8  7  2  14 8  2  9  13 6  8  5  12 12 3
## Levels: 1 2 3 4 5 6 7 8 9 12 13 14
```

If you want a normal numeric vector:

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

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

you should do this...

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

```
##  [1] 12  4  4  8  8  1  8  7  2 14  8  2  9 13  6  8  5 12 12  3
```

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(1,3)]
```

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

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

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

```
myFirstVector[somePositions]
```

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

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.

(line 222 in the document)

Solution:

```
myvector <- c(15:18,20)
myvector[c(1,3)]
```

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

or like this..

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

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

or like this:

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

(line ~ 289)

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[ ??? ]
```

Solution:

```
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.

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

Solution?
(around line 333)

```
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
```

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 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] 42 43 77 94 70  6 12 90 78 69
```

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

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

```
##  [1] -0.07847427  0.33455289 -0.73378659 -1.30828679 -0.52900796
##  [6]  0.18859914 -0.37604575  0.56672033  1.66108773 -0.10399628
```

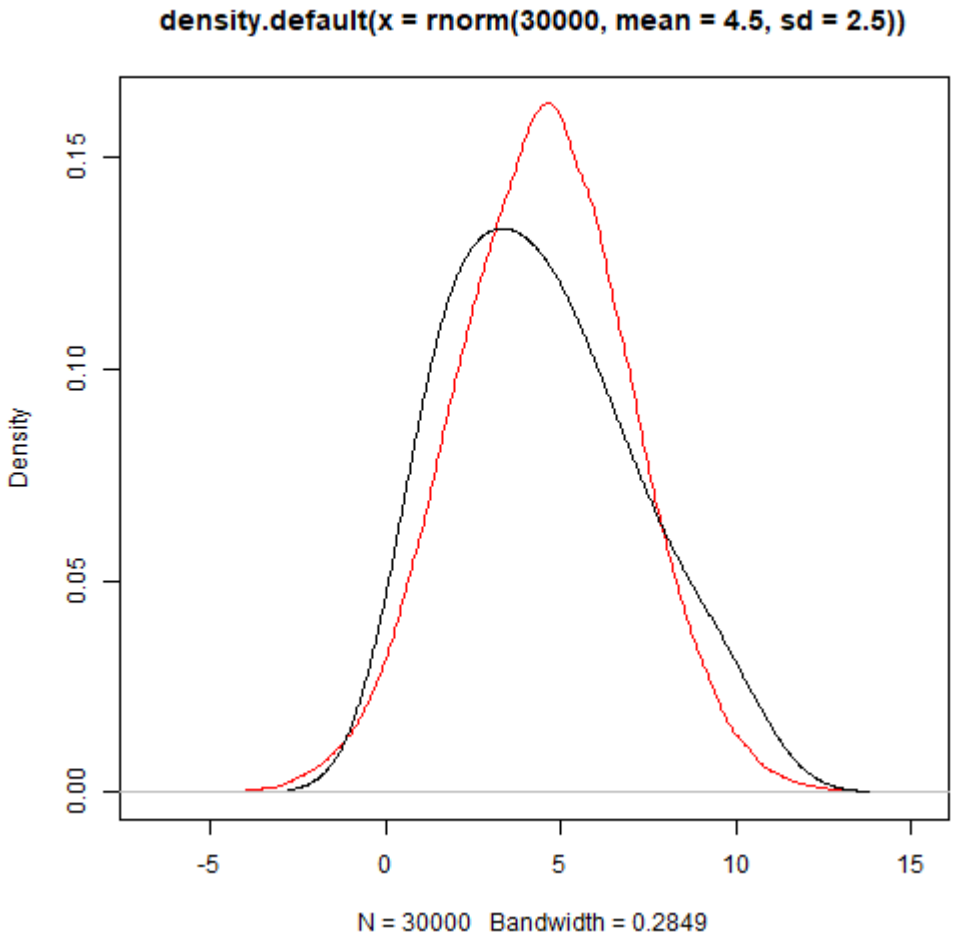
analogously, rpois, runif

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

Example usage: IS MY DATA NORMAL?

```
somedata <- c(1:10, 1:7, 1:5)
plot(density(somedata))
```

```
plot(density(rnorm(30000, mean=4.5, sd=2.5)), col=2)
lines(density(somedata))
```



HELP!!??!!

HELP!!??!!

```
?rnorm
example(rnorm)
```

Exercise - estimate pi