PP4RS | R Module

Slot 1

Dora Simon

05.09.2018

Outline of the R-Module

Slot 1: Intro & Data Types

Slot 2: Conditionals and Functions & Loops

Slot 3: Read in Data

Slot 4: Data Manipulation

Slot 5: Regressions

Slot 6: Graphs

Slot 7: knitR

Today: Intro & Data Types

Setup of the Class

I will start each session with a presentation

Setup of the Class

I will start each session with a presentation

- There, you should only **listen**, not code.
- Of course you can **ask questions** any time.

Setup of the Class

I will start each session with a presentation

- There, you should only **listen**, not code.
- Of course you can **ask questions** any time.

After roughly 20 minutes, you will get an exercise

Setup of the Class

I will start each session with a presentation

- There, you should only **listen**, not code.
- Of course you can **ask questions** any time.

After roughly 20 minutes, you will get an exercise

- There, you should **code**.
- We will go around and help with problems.
- We will look at the solution together afterwards

Setup of the Class

I will start each session with a presentation

- There, you should only **listen**, not code.
- Of course you can **ask questions** any time.

After roughly 20 minutes, you will get an exercise

- There, you should **code**.
- We will go around and help with problems.
- We will look at the solution together afterwards

- I am **not** going to explain to you how to use certain commands beforehand

Setup of the Class

I will start each session with a presentation

- There, you should only **listen**, not code.
- Of course you can **ask questions** any time.

After roughly 20 minutes, you will get an exercise

- There, you should **code**.
- We will go around and help with problems.
- We will look at the solution together afterwards

- I am **not** going to explain to you how to use certain commands beforehand

After two sessions (one session = presentation and exercise), we will do a break.

Introduction

About R

> "R is a free software environment for statistical computing and graphics."[1]

- R is both a porgramming language[2] and a statistical environment
- Users can define new functions
- C, C++ and Fortran code can be linked
- Advanced users can write C code to manipulate R objects directly

[1] https://www.r-project.org/about.html

[2] It depends on how you define a porgramming language. But many people seem to agree that R can be called a computer language.

How to use R

Command-line Interface

- R runs in your shell!

Graphical front-ends

- Most of the time it is more convenient to work with a front-end
- We will use RStudio for that
- There are also other front-ends, but it is the most popular one

When to use R

- Why to use R rather than Stata
    - R is open source
    - R is becoming more and more popular
    - R has latex support (more on this on Day 4!)

When to use R

- Why to use R rather than Stata
  - R is open source
  - R is becoming more and more popular
  - R has latex support (more on this on Day 4!)
- Can you replace Matlab with R?
  - Yes.[1]

When to use R

- Why to use R rather than Stata
  - R is open source
  - R is becoming more and more popular
  - R has latex support (more on this on Day 4!)

- Can you replace Matlab with R?
  - Yes.[1]

- R or Python?

When to use R

- Why to use R rather than Stata
  - R is open source
  - R is becoming more and more popular
  - R has latex support (more on this on Day 4!)
- Can you replace Matlab with R?
  - Yes.[1]
- R or Python?

pro R	pro Python[2]
built for data analysis	general programming language
focus on user friendliness	focus on code readability
statistical tests are easy to use	novel things are easy to implement
graphs are great	integration with web apps great

[1] At least these people think so. [2] Source: DataCamp

Packages

> Packages are collections of R functions, data, and compiled code in a well-defined format.

Packages

> Packages are collections of R functions, data, and compiled code in a well-defined format.

- R comes with 14 base packages

Packages

> Packages are collections of R functions, data, and compiled code in a well-defined format.

- R comes with 14 base packages
- Many more are available through the CRAN (Comprehensive R Archive Network) family of Internet sites

Packages

> Packages are collections of R functions, data, and compiled code in a well-defined format.

- R comes with 14 base packages
- Many more are available through the CRAN (Comprehensive R Archive Network) family of Internet sites

> CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. [1]

[1] CRAN

Packages

If you want to install a package from CRAN that you do not have, this is the command:

```r
install.packages("package-name")
```

Packages

If you want to install a package from CRAN that you do not have, this is the command:

```r
install.packages("package-name")
```

If you want to use some functions out of a specific package that you already installed, you first need to load it.

Packages

If you want to install a package from CRAN that you do not have, this is the command:

```
install.packages("package-name")
```

If you want to use some functions out of a specific package that you already installed, you first need to load it.

There are two options:

```
library(package-name)
somefct()
```

```
package-name::somefct()
```

Writing Code in R

You can write code in the Console

- It will be executed, but not saved

Writing Code in R

You can write code in the Console

- It will be executed, but not saved

It is better to start a script and run (parts of) it

- Your scripts are saved in .R-files
- They can be opened in RStudio or in a text editor (also in Atom)
- If you open them in RStudio, you can immediately execute them
- If you want to run just a part of your code, highlight it and then press `Ctrl+Enter`
- If you edit them in Atom, you have to run the script in the shell to execute it

Help Files

If you do not know how a command works, you can consult the help file with the following two options:

```
help(somefct)
?somefct
```

The help file will open in a new window.

- In RStudio, it will open in the Help window on the right
- In the shell, a new window will open which you can leave by typing `q`

Exercise 1

Part A: Shell and RStudio

Let's get familiar with R in the shell and RStudio!

Command-line Interface

- R runs in your shell!
- Type 'R' in the shell
- Everything you enter from now on must be in 'R' language.
- In order to get back to the standard shell, type q().

> Show in class

Part A: Shell and RStudio

Let's get familiar with R in the shell and RStudio!

Command-line Interface

- R runs in your shell!
- Type 'R' in the shell
- Everything you enter from now on must be in 'R' language.
- In order to get back to the standard shell, type q().

> > Show in class

RStudio

- R has its own front-end called RStudio
- Open RStudio
  - You can do so by typing `rstudio` in your shell or by using the mouse
- Check out all windows you have

> Show in class
>
> Show example ex-a.R in class

Part B: Your turn!

- Create a folder for your R-Scripts of this class, anywhere you like using the shell (just not on Dropbox because we will use git!)
- Create a file called `ex1-b.R` using the shell. Store all of the following steps in your file `ex1-b.R` once you have figured out how to do them.
- Open RStudio
- Find out in which working directory you currently are (command: `getwd`)
- Change the working directory to the one of your newly created folder (command: `setwd`)
- Define two new variables `x` and `y` and assign them two arbitrary numerical values. (just try, it is straight forward)
- Try to see which variables are stored in your workspace (hint: the command is similar to the one you would use in the Shell)
- Delete the `x` variable (hint: the command is similar to the one you would use in the Shell). Again see which ones are left in the workspace.
- After saving all steps above in your .R-script, open your Shell and execute the script form there. (command: `Rscript`)

Part B: Solution

- Create a folder for your R-Scripts of this class, anywhere you like using the shell (just not on Dropbox because we will use git!)

Part B: Solution

- Create a folder for your R-Scripts of this class, anywhere you like using the shell (just not on Dropbox because we will use git!)

```
Solution:

Your paths will be different if your file is in a different folder.

In shell: mkdir ~/git/teaching/pp4rs/r
```

Part B: Solution

- Create a folder for your R-Scripts of this class, anywhere you like using the shell (just not on Dropbox because we will use git!)

```
Solution:

Your paths will be different if your file is in a different folder.

In shell: mkdir ~/git/teaching/pp4rs/r
```

- Create a file called `ex1-b.R` using the shell. Store all of the following steps in your file `ex1-b.R` once you have figured out how to do them.

Part B: Solution

- Create a folder for your R-Scripts of this class, anywhere you like using the shell (just not on Dropbox because we will use git!)

```
Solution:

Your paths will be different if your file is in a different folder.

In shell: mkdir ~/git/teaching/pp4rs/r
```

- Create a file called `ex1-b.R` using the shell. Store all of the following steps in your file `ex1-b.R` once you have figured out how to do them.

```
Solution:

In shell: touch "~/git/teaching/pp4rs/r/ex1-b.R"
```

Part B: Solution

- Open RStudio
- Find out in which working directory you currently are (command: `getwd`)

Part B: Solution

- Open RStudio
- Find out in which working directory you currently are (command: `getwd`)

```
Solution:
```

```
getwd()
```

```
[1] "/home/dsimon/git/teaching/pp4rs/r/day1"
```

Part B: Solution

- Change the working directory to the one of your newly created folder (command: `setwd`)

Part B: Solution

- Change the working directory to the one of your newly created folder (command: `setwd`)

Solution:

```r
setwd("~/git/teaching/pp4rs/r")
```

Part B: Solution

- Change the working directory to the one of your newly created folder (command: `setwd`)

`Solution:`

```
setwd("~/git/teaching/pp4rs/r")
```

- Define two new variables `x` and `y` and assign them two arbitrary numerical values (just try, it is straight forward).

Part B: Solution

- Change the working directory to the one of your newly created folder (command: `setwd`)

Solution:

```r
setwd("~/git/teaching/pp4rs/r")
```

- Define two new variables `x` and `y` and assign them two arbitrary numerical values (just try, it is straight forward).

Solution:

```r
x<-1
y<-3
```

Part B: Solution

- Try to see which variables are stored in your workspace (hint: the command is similar to the one you would use in the Shell)

Part B: Solution

- Try to see which variables are stored in your workspace (hint: the command is similar to the one you would use in the Shell)

`Solution:`

```
ls()
```

```
[1] "x" "y"
```

Part B: Solution

- Try to see which variables are stored in your workspace (hint: the command is similar to the one you would use in the Shell)

`Solution:`

```
ls()
```

```
[1] "x" "y"
```

- Delete the `x` variable (hint: the command is similar to the one you would use in the Shell). Again see which ones are left in the workspace.

Part B: Solution

- Try to see which variables are stored in your workspace (hint: the command is similar to the one you would use in the Shell)

Solution:

```
ls()
```

```
[1] "x" "y"
```

- Delete the `x` variable (hint: the command is similar to the one you would use in the Shell). Again see which ones are left in the workspace.

Solution:

```
rm(x)
ls()
```

```
[1] "y"
```

Part B: Solution

- After saving all steps above in your .R-script, open your Shell and execute the script form there. (command: `Rscript`)

```
Solution:

In shell: Rscript ./ex-1-b.R
```

> Show in class
>
> Why does it show those three entries and not more?

Variable Definition in R

You can define variables with the signs `<-`, `=`, and `->`.

```r
x<-2
x
```

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

```r
y=3
y
```

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

```r
5->z
z
```

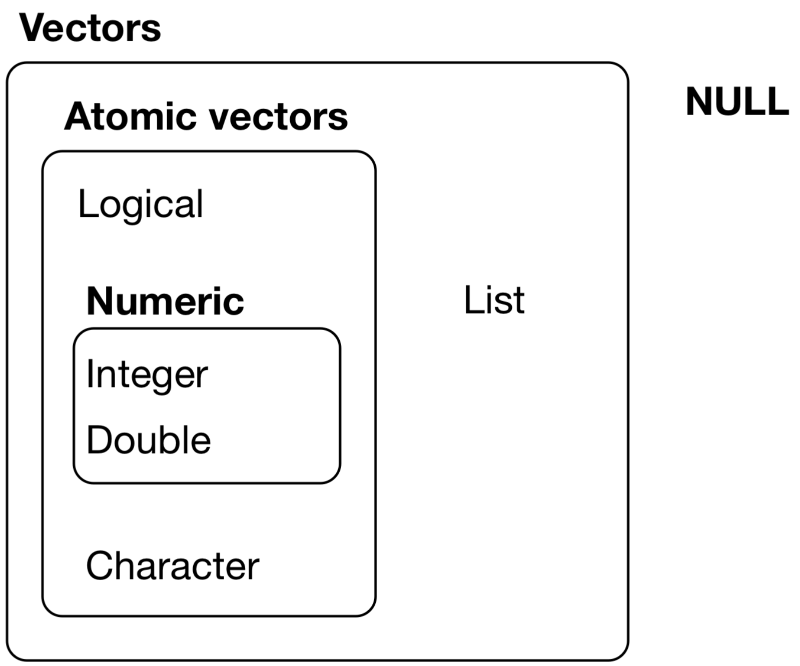
```
## [1] 5
```

Which one is the best?

Data Types in R

Which Data Types exist?

- Vectors
  - logical
  - character
  - numeric
  - lists
  - complex (not covered)
  - raw (not covered)
- Matrices
- Factors
- Data Frames
- Arrays



Vectors

Vectors can have one or more values.

```r
# Vector with one value
a<-1
a
```

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

```r
# Vector with several values
b<-c(1,2,3,4,5)
b
```

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

Logical Vectors

Logical vectors (or booleans) can have three different values in R:

- `TRUE`
- `FALSE`
- `NA` : not available

Example: What does this code do? Why is the last value not `FALSE`?

```
a<-TRUE
b<-TRUE
c<-NA

a==b
```

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

```
a==c
```

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

Numeric Vectors

- double: 2.0
- integer: 2

R stores every number as a double by default. This takes up a lot of memory. If you are sure you only need integers you can append the number with a ‘L’ to force the coercion to integer values.

```
typeof(2)
```

```
## [1] "double"
```

```
typeof(2L)
```

```
## [1] "integer"
```

```
longer_vector<-c(3.5, 6.7, NA)
longer_vector
```

```
## [1] 3.5 6.7  NA
```

Numeric Vectors & Missing Values

- double: 2.0, `NA`, `NaN`, `Inf`, `-Inf`
    - `NaN` : not a number
- integer: 2, `NA`

```
my_vector<-c(-1,0,1)

# Let's try to divide by 0

my_vector/0
```

```
## [1] -Inf  NaN  Inf
```

Character Vectors

Vector of one or more characters and entries, separated by either `'` or `"`.

```
w<-"d"
w
```

```
## [1] "d"
```

Character Vectors

Vector of one or more characters and entries, separated by either `'` or `"`.

```
w<-"d"
w
```

```
## [1] "d"
```

```
x<-'o'
x
```

```
## [1] "o"
```

Character Vectors

Vector of one or more characters and entries, separated by either `'` or `"`.

```
w<-"d"
w
```

```
## [1] "d"
```

```
x<-'o'
x
```

```
## [1] "o"
```

```
y<-'Ra'
y
```

```
## [1] "Ra"
```

Character Vectors

Vector of one or more characters and entries, separated by either `'` or `"`.

```
w<-"d"
w
```

```
## [1] "d"
```

```
x<-'o'
x
```

```
## [1] "o"
```

```
y<-'Ra'
y
```

```
## [1] "Ra"
```

```
z<-c("<3","programming", "really a lot")
z
```

```
## [1] "<3"           "programming"  "really a lot"
```

Lists

A list can contain many different types of elements inside it like vectors, functions and even another list

```
# List with 4 elements
first_list <- list(1, TRUE, "hello", 4)
str(first_list) #display internal structure of an element
```

```
## List of 4
##  $ : num 1
##  $ : logi TRUE
##  $ : chr "hello"
##  $ : num 4
```

Lists

```r
# Create a list with a list
list2 <- list(first_list,21.3,sin)
str(list2)
```

```
## List of 3
##  $ :List of 4
##   ..$ : num 1
##   ..$ : logi TRUE
##   ..$ : chr "hello"
##   ..$ : num 4
##  $ : num 21.3
##  $ :function (x)
```

Matrices

Let us create a matrix

```
M <- matrix( c('a','a','b','c','b','a'),
             nrow = 2,
             ncol = 3,
             byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "a"  "b" 
## [2,] "c"  "b"  "a" 
```

Data Frames

Data frames are tabular data objects.

- each column can contain different modes of data (unlike a matrix)
- it is a list of vectors of equal length
- similar to Stata's dataset or an Excel sheet

Data Frames

Data frames are tabular data objects.

- each column can contain different modes of data (unlike a matrix)
- it is a list of vectors of equal length
- similar to Stata's dataset or an Excel sheet

```r
# Create a data frame
BMI <-      data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
BMI
```

```
##   gender height weight Age
## 1   Male  152.0     81  42
## 2   Male  171.5     93  38
## 3 Female  165.0     78  26
```

Data Frames

If you want to access a certain column, you can do so with the `$`:

```
my_dataframe$my_column
```

Data Frames

If you want to access a certain column, you can do so with the `$`:

`my_dataframe$my_column`

In our example:

```r
# Create a data frame
BMI$Age
```

```
## [1] 42 38 26
```

Arrays

- Arrays can be of any number of dimensions (matrices have only 2)
- dim attribute creates the required number of dimensions

```r
# Create an array with two elements, each being a 3x3 matrix
a <- array(c('green','yellow'),dim = c(3,3,2))
a
```

```
## , , 1
##
##      [,1]     [,2]     [,3]    
## [1,] "green"  "yellow" "green" 
## [2,] "yellow" "green"  "yellow"
## [3,] "green"  "yellow" "green" 
##
## , , 2
##
##      [,1]     [,2]     [,3]    
## [1,] "yellow" "green"  "yellow"
## [2,] "green"  "yellow" "green" 
## [3,] "yellow" "green"  "yellow"
```

Factors

- categorical variables: fixed and known set of possible values
    - dummy variables
    - seasons
    - weekdays
- R maps the character strings to nominal values from [1… k]

Factors

- categorical variables: fixed and known set of possible values
  - dummy variables
  - seasons
  - weekdays
- R maps the character strings to nominal values from [1… k]

```r
# Create a character vector
apple_cols <- c('green','green','yellow','red','red','red','green')
```

Factors

- categorical variables: fixed and known set of possible values
  - dummy variables
  - seasons
  - weekdays
- R maps the character strings to nominal values from [1… k]

```r
# Create a character vector
apple_cols <- c('green','green','yellow','red','red','red','green')
```

```r
# Create a factor object
factor_apple <- factor(apple_cols)
```

Factors

- categorical variables: fixed and known set of possible values
  - dummy variables
  - seasons
  - weekdays
- R maps the character strings to nominal values from [1… k]

```r
# Create a character vector
apple_cols <- c('green','green','yellow','red','red','red','green')
```

```r
# Create a factor object
factor_apple <- factor(apple_cols)
```

```r
factor_apple #print factor
```

```
## [1] green  green  yellow red    red    red    green 
## Levels: green red yellow
```

```r
nlevels(factor_apple) #see how many levels it has
```

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

Exercise

- Explicit coercion: Define a variable `b` as the number 3. Convert `b` to a character value. (hint: use `as.character`)
- Implicit coercion: Add the values 3, TRUE and FALSE and call the result `d`. What happens?
- Check whether your previous result `d` is a logical value. (hint: `is.logical`)
- Check whether your previous result `d` is a numeric value.
- Define `e` as 3 divided by 0. Check if the result is finite, infinite or a missing value.
- Use the vector `V<-c('a','a','b','c','b','a')` and create a matrix N that looks like this:

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "b"
[2,] "a"  "c"  "a"
```

- Take the vectors w,x,y,z from the "Character Vectors" slide and print them.
- Using w,x,y and z, Create a vector with one entry which displays the sentence "doRa <3 programming really a lot". (hint: use `paste`. This is a bit trickier!)

Solution

- Explicit coercion: Define a variable `b` as the number 3. Convert `b` to a character value. (hint: use `as.character`)

Solution

- Explicit coercion: Define a variable `b` as the number 3. Convert `b` to a character value. (hint: use `as.character`)

```
b<-3
b
```

```
[1] 3
```

```
c<-as.character(b)
c
```

```
[1] "3"
```

Solution

- Explicit coercion: Define a variable `b` as the number 3. Convert `b` to a character value. (hint: use `as.character`)

```
b<-3
b
```

```
[1] 3
```

```
c<-as.character(b)
c
```

```
[1] "3"
```

- Implicit coercion: Add the values 3, TRUE and FALSE and call the result `d`. What happens?

Solution

- Explicit coercion: Define a variable `b` as the number 3. Convert `b` to a character value. (hint: use `as.character`)

```
b<-3
b
```

```
[1] 3
```

```
c<-as.character(b)
c
```

```
[1] "3"
```

- Implicit coercion: Add the values 3, TRUE and FALSE and call the result `d`. What happens?

```
d<-3+TRUE+FALSE
d
```

```
[1] 4
```

Solution

- Check whether your previous result `d` is a logical value. (hint: `is.logical`)

Solution

- Check whether your previous result `d` is a logical value. (hint: `is.logical`)

```r
is.logical(d)
```

```
[1] FALSE
```

Solution

- Check whether your previous result `d` is a logical value. (hint: `is.logical`)

```
is.logical(d)
```

```
[1] FALSE
```

- Check whether your previous result `d` is a numeric value.

Solution

- Check whether your previous result `d` is a logical value. (hint: `is.logical`)

```
is.logical(d)
```

```
[1] FALSE
```

- Check whether your previous result `d` is a numeric value.

```
is.numeric(d)
```

```
[1] TRUE
```

Solution

- Define `e` as 3 divided by 0. Check if the result is finite, infinite or a missing value.

```
e<-3/0
is.finite(e)
```

```
[1] FALSE
```

```
is.infinite(e)
```

```
[1] TRUE
```

```
is.na(e)
```

```
[1] FALSE
```

Solution

Use the vector `V<-c('a','a','b','c','b','a')` and create a matrix N that looks like this:

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "b" 
[2,] "a"  "c"  "a" 
```

```r
V<-c('a','a','b','c','b','a')
M <- matrix( V, nrow = 2, ncol = 3, byrow = FALSE)
M
```

```
     [,1] [,2] [,3]
[1,] "a"  "b"  "b" 
[2,] "a"  "c"  "a" 
```

Solution

- Take the vectors w,x,y,z from the "Character Vectors" slide and print them.

Solution

- Take the vectors w,x,y,z from the "Character Vectors" slide and print them.

```
w
```

```
[1] "d"
```

```
x
```

```
[1] "o"
```

```
y
```

```
[1] "Ra"
```

```
z
```

```
[1] "<3"           "programming"  "really a lot"
```

Solution

- Using w,x,y and z, Create a vector with one entry which displays the sentence "doRa <3 programming really a lot". (hint: use `paste`)

Solution

- Using w,x,y and z, Create a vector with one entry which displays the sentence "doRa <3 programming really a lot". (hint: use `paste`)

```
v<-paste(w,x,y, sep = "") #does not separate them
v
```

```
[1] "doRa"
```

```
u<-paste(z, collapse=' ') #makes one big string
t<-paste(v,u)
t
```

```
[1] "doRa <3 programming really a lot"
```