

Coding for Reproducible Research

Field Coordinator Training - R Track

Luiza Andrade, Leonardo Viotti & Rob Marty

June 2018



- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

Objective

Create an R Master Script.

Content

- 1 Comments
- 2 Using R Studio to create a code index
- 3 Indenting
- 4 File paths
- 5 If statements
- 6 Using functions within functions
- 7 Packages
- 8 Loops

- The exercises on this section will help us create a Master R script
- To do them, go to the *DataWork/Code* folder and open the file called *Lab 2 - Coding for Reproducible Research.R*

- Just like in Stata, a Master script must be a map of all data work
- It should be possible to follow all data work in the data folder, from raw data to analysis output, by following the master script
- It should also be possible to run all the code for the project by simply running the Master script
- Finally, in Stata, the Master Script is where you create globals. You can do the same in R by just creating new objects

Outline

- 1 Introduction
- 2 **Commenting**
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

- Let's take a look at the script we just opened
- You can see that the first few lines in the script are the header, but they're not commented out
- In R, errors will not always break your code, so you should still be able to run this script
- However, not commenting out comments is still bad practice, as it makes the code harder to read

- To comment a line, write # as its first character
- You can also add # half way through a line to comment whatever comes after it
- In Stata, you can use /* and */ to comment part of a line's code. That is not possible in R: whatever comes after # will be a comment
- To comment a selection of lines, press Ctrl + Shift + C

Exercise 1

Use the keyboard shortcut to comment the header of the script.

Exercise 2

Use the keyboard shortcut to comment the header of the script again.
What happened?

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

Creating a document outline in RStudio

- RStudio also allows you to create an interactive index for your scripts
- To add a section to your code, create a commented line with the title of your section and add at least 4 trailing dashes, pound signs or equal signs after it

Exercise 3

Open the script index and make PART 0 a section header. Do the same for PART 1.

Exercise 4

Note that once you create a section header, an arrow appears right next to it. Click on the arrows of parts 0 and 1 to see what happens.

Creating a document outline in RStudio

- The outline can be accessed by clicking on the button on the top right corner of the script window. You can use it to jump from one section to another
- You can also use the keyboard shortcuts `Alt + L` (`Cmd + Option + L` on Mac) and `Alt + Shift + L` to collapse and expand sections

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings**
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

- A Stata do-file typically starts with a few settings:

```
clear  
set maxvar 120000  
set more off
```

Initial Settings

- We don't need to set the memory or the maximum number of variables in R, and more is automatically selected
- However, if you saved the last RStudio session in `.Rhistory`, the objects that were in RStudio's memory last time you closed it will still be there whenever you open it again
- Therefore, it's good practice to always clean the memory when starting a new session
- You can see all the objects currently in you memory in the *Environment* pane

Exercise 5

- 1 Make sure the *Environment* window is open (it should be empty now)
- 2 Create an object called `foo` with any content you pick
- 3 Type `ls()` to print the names of the object in memory
- 4 Type `rm(foo)` to remove it
- 5 To remove all objects, use `rm(list=ls())`

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

Using packages

- Since there is a lot of people developing for R, it can have many different functionalities
- To make it simpler, these functionalities are bundled into packages
- A package is the fundamental unit of shareable code
- It may contain new functions, but also more complex functionalities, such as a Graphic User Interface (GUI) or settings for parallel processing (similar to Stata MP)
- They can be shared through R's official repository - CRAN (10,000+ packages reviewed and tested) and many other online sources
- There are many other online sources such as Github, but it's important to be careful, as these probably haven't gone through a review process as rigorous as those in CRAN

Using packages

- To install and use packages you can either do it with the user interface or by the command prompt.

```
# Installing a package  
install.packages("stargazer",  
                 dependencies = T)  
# the dependencies argument also installs all other packages  
# that it may depend upon
```

- You only have to install a package once, but you have to load it every new session. To load a package type:

```
library(stargazer)
```

Using packages

Once a package is loaded, you can use its features and functions. Here's a list of some useful and cool packages:

- Rcmdr - Easy to use GUI
- swirl - An interactive learning environment for R and statistics.
- ggplot2 - beautiful and versatile graphics (the syntax is a pain, though)
- stargazer - awesome latex regression and summary statistics tables
- foreign - reads dtas and other formats from inferior statistical software
- zoo - time series and panel data manipulation useful functions
- data.table - some functions to deal with huge data sets
- sp and rgeos - spatial analysis
- multiwayvcov and sandwich - clustered and robust standard errors
- RODBC, RMySQL, RPostgresSQL, RSQLite - For relational databases and using SQL in R.

Exercise 6

Install the `swirl` and `stargazer` packages, including packages necessary for them to run.

Using packages

```
# Install stargazer and swirl  
install.packages(c("stargazer", "swirl"),  
                 dependencies = TRUE)
```

Exercise 7

Call the packages you just installed. Note that the `library` function only accepts one argument, so you will need to load each of them separately.

Using packages

```
library(stargazer)
```

```
##
```

```
## Please cite as:
```

```
## Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics
```

```
## R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

```
library(swirl)
```

```
##
```

```
## | Hi! Type swirl() when you are ready to begin.
```

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting**
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping

- Indenting can be pretty different from what it looks like in Stata
- To indent a whole line, you can select that line and press `Tab`
- To unindent a whole line, you can select that line and press `Shift + Tab`
- However, this will not always work for different parts of a code in the same line
- In R, we typically don't introduce white space manually
- It's rather introduced by RStudio for us

Exercise 8

To see an example of how indenting works in RStudio, add a line between the two arguments of the `install.packages` function (the vector of package names and the `dependents` option). Then add a line between the two package names. Note that RStudio formats the different arguments of the function differently.

Indenting

```
# Same code, but this time easier to read  
install.packages(c("stargazer",  
                  "swirl"),  
                dependencies = TRUE)
```

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception**
- 8 Section switches
- 9 File paths
- 10 Looping

Functions inception

- In R, you can use the output of one function as the input of another, as long as they have the same format
- In fact, that's exactly what we just did when installing the packages
- To see that, select just the first argument of the `install.packages` function and press `Ctrl + Enter`
- The `c()` function, as we know, creates a vector with its arguments

```
c("stargazer", "swirl")
```

```
## [1] "stargazer" "swirl"
```

Functions inception

- The resulting vector is used as an input to the `install.packages` function
- We could also have stored this vector in the memory as an object and used that object as the input
- In fact, that's exactly what we are going to do next, so the code doesn't get too polluted as we add new packages

Functions inception

```
# Create packages object
packages <- c("stargazer",
              "swirl")

# Use it as an input to the install.packageS() function
install.packages(packages,
                 dependencies = TRUE)
```

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches**
- 9 File paths
- 10 Looping

- Now, installing packages can be time-consuming, especially as the number of packages grow, and each package only needs to be installed once
- What can we bring from our Stata Master do-files to avoid installing packages twice?

- In Stata, section switches would be saved as locals
- In R, the equivalent to that would be to create a new object

Exercise 10

Create a dummy scalar object called PACKAGES.

- TIP: Section switches can also be Boolean objects.

Section switches

- Now we need to create an if statement using this switch
- If statements in R look like this:

```
# Turn switch on
```

```
PACKAGES <- 1
```

```
# Install packages
```

```
if (PACKAGES == 1) {
```

```
  install.packages(packages,  
                   dependencies = TRUE)
```

```
}
```

Section switches

- Possible variations would include

```
# Turn switch on
```

```
PACKAGES <- TRUE
```

```
# Using a Boolean object
```

```
if (PACKAGES == TRUE) {
```

```
  install.packages(packages, dep = T)
```

```
}
```

```
# Which is the same as
```

```
if (PACKAGES) {
```

```
  install.packages(packages, dep = T)
```

```
}
```

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths**
- 10 Looping

- The next important part of a Master script are file paths
- We recommend always using **explicit** and **dynamic** file paths

- Implicit and static file path:

```
# Set working directory  
setwd("C:/Users/luiza/Documents/GitHub/R-Training/  
      DataWork/DataSets/Final")  
  
# Load data set  
read.csv("lwh_clean.csv",  
         header = T)
```

- Explicit and static file path:

```
# Load data set  
read.csv("C:/Users/luiza/Documents/GitHub/R-Training/  
DataWork/DataSets/Final/lwh_clean.csv",  
         header = T)
```

- Explicit and dynamic file path:

```
# Define dynamic file path
finalData <- "C:/Users/luiza/Documents/GitHub/R-Training/
             DataWork/DataSets/Final"

# Load data set
lwh <- read.csv(file.path(finalData,"lwh_clean.csv"),
                header = T)
```

- File paths in R, as in Stata, are basically just strings
- Note, however, that in R we can only use forward slashes (/) to separate folder names

Exercise 11

Let's start by adding the folder path to the training's folder in your computer to the beginning of PART 3.

- You can set file paths in your master using the `file.path()` function
- This function concatenates strings using `/` as a separator to create file paths

File paths

```
# Project folder
projectFolder <-
  "C:/Users/luiza/Documents/GitHub/R-Training"

# Data work folder
dataWorkFolder <- file.path(projectFolder, "DataWork")

# Print data work folder
dataWorkFolder
```

```
## [1] "C:/Users/luiza/Documents/GitHub/R-Training/DataWork"
```

Let's check if that worked, as we will need your Master script to be running smoothly for the other sessions.

Exercise 12

- 1 Turn off the switch that installs packages in your Master script
- 2 Run the whole script
- 3 Type the following code to open a data set using the file paths you just set:

```
# Load data set  
lwh <- read.csv(file.path(rawData, "lwh_panel.csv"),  
                header = T)
```

Outline

- 1 Introduction
- 2 Commenting
- 3 Creating a document outline in RStudio
- 4 Initial Settings
- 5 Using packages
- 6 Indenting
- 7 Functions inception
- 8 Section switches
- 9 File paths
- 10 Looping**

- We're almost at the end of this section
- But the DRY rule can still be applied to part of this code

The DRY rule:

DONT REPEAT YOURSELF

Looping

- In Stata, we'd usually use a `foreach` loop to go through a list of objects
- The equivalent to that in R would be to write a `for` loop like this

```
# A for loop in R  
for (number in c(1.2,2.5)) {  
  print(round(number))  
}
```

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

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

- R, however, has a whole function family that allows users to loop through an object in a more efficient way
- They're called `apply` and there are many of them, for different use cases.
- For the purpose of this training, we will only use two of them, `sapply` and `apply`
- If you look for the `apply` help file, you can see all of them

Looping

- `sapply(X, FUN, ...)`: applies a function to all elements of a vector or list and returns the result in a vector. Its arguments are
 - **X**: a matrix (or data frame) the function will be applied to
 - **FUN**: the function you want to apply
 - **...**: possible function options

```
# A much more elegant for loop in R  
sapply(c(1.2,2.5), round)
```

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

Exercise 13

Use the `sapply()` function to apply the `library()` function to all packages you have selected.

Looping

```
# Load all listed packages  
sapply(packages,  
       library, character.only = TRUE)
```

A more generic version is the `apply` function.

- `apply(X, MARGIN, FUN, ...)`: applies a function to all columns or rows of matrix. Its arguments are
 - **X**: a matrix (or data frame) the function will be applied to
 - **MARGIN**: 1 to apply the function to all rows or 2 to apply the function to all columns
 - **FUN**: the function you want to apply
 - **...**: possible function options

Looping

```
# Create a matrix
```

```
matrix <- matrix(c(1, 24, 9, 6, 9, 4, 2, 74, 2),  
                 nrow = 3)
```

```
# Look at the matrix
```

```
matrix
```

```
##      [,1] [,2] [,3]  
## [1,]    1    6    2  
## [2,]   24    9   74  
## [3,]    9    4    2
```


Looping

```
# Row means  
apply(matrix, 1, mean)
```

```
## [1] 3.00000 35.66667 5.00000
```

```
# Column means  
apply(matrix, 2, mean)
```

```
## [1] 11.333333 6.333333 26.000000
```

That's all, folks

- Now you have a template master script to use across this training's sessions
- Save the script that you created during this session in the *DataWork* folder. Call it *MASTER.R*
- You can run scripts from the Master script by using the `source()` function as you write scripts for future sessions, but we will do that on the next session

That's all, folks

Homework

In the next slide here's a list of all the packages we'll need to the next sessions. Installing them might take a while, so paste them to your Master script and run it again before the next session:

```
packages <- c("readstata13", "foreign",  
              "doBy", "broom",  
              "stargazer",  
              "ggplot2", "plotly", "ggrepel",  
              "RColorBrewer",  
              "sp", "rgdal", "rgeos", "raster", "velox",  
              "ggmap", "rasterVis", "leaflet")
```

Thank you!

Bonus exercise

- You can customize your loops in R by defining your own function
- This is done using a function conveniently called `function()`
- For example, if instead of just printing a number we want to print it's square, we could create a function that does both:

```
# A much more elegant for loop in R  
sapply(c(1,2), function(x) x^2)
```

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