Intro to R I

Complexities in analyzing conflicts: Data wrangling and data management in R

Cosima Meyer

November 6, 2019

Introduction to R Studio

Throughout the course, we will use R Studio, as an interface to work with R.

This file shows you the essential (first) steps in R and introduces you to R Markdown. Markdown documents allow you to easily document your coding steps in R and to produce replicable research. You usually want to organize your workflow with several Markdown documents (see here for some inspiration) that includes several Markdown files that separate your data generation section, your analysis section from your robustness checks and/or visualization.

Most important shortcuts

The cheat sheet provides you with an incredible richness of shortcuts. I will find the shortcuts that I find most important below:

- *Execute code*:
  - Cmd + Enter (Mac)
  - Ctrl + Enter (Windows)
- *New chunks*:
  - Alt + Cmd + I (Mac)
  - Alt + Ctrl + I (Windows)
- *(Un)Comment (several) lines* (#):
  - Shift + Cmd + C (Mac)
  - Shift + Ctrl + C (Windows)
- *Pipe* (%>%):
  - Shift + Cmd + M (Mac)
  - Shift + Ctrl + M (Windows)
- *Format*:
  - Shift + Cmd + A (Mac)
  - Shift + Ctrl + A (Windows)

Introduction to R Markdown

A markdown document consists of a YAML header, plain text parts and so-called chunks. The header includes all essential information for compiling your document (here we chose a `pdf_document` but we could have also chosen a `html_document` or a presentation with `ioslides_presentation` for example).

This text is written in a plain text part where you can simply write your text. There is no need to set special comments like in Stata. Instead, R markdown follows a simple logic that allows us to include sections (with the hashtag `#`). The more hashtags you set, the more levels your section title gets. We can also write texts in **bold** or *italics*, include hyperlinks (e.g. to a specific R Markdown guide or a cheat sheet), and even lists:

- This is a
- unnumbered
- list.
- This list is also
- unnumbered.

1. This list
2. contains an
3. enumeration

The chunks are usually written the following way:

Chunks usually take commands. If you want to write something within a chunk as a plain text instead, you need to comment this text using a hashtag (`#`). We can easily customize chunks to allow them individually to be evaluated and to display certain messages. For this course, we do not need to customize our chunks; if you want to know more, please see here.

These basics provide you with the essential knowledge to manoeuver in the R Markdown documents for this course. If you wish to produce more sophisticated documents, please have a look here or here.

Introduction to R

If you are aready familiar with another (statistical) programming language such as Stata/Mata or Python, R should seem fairly familiar.

For former Python users: The biggest difference is that R does not require you to pay attention to your indents and that R is fairly good in easily producing statistical results. Most of the newest packages (in particular when it comes to text analysis) are often first published in Python.

For former Stata users: There are several differences in R. R works with objects (we will see this in a second), it allows you to read several data sets at once (we will talk about this more in-depth in the merging part of our seminar), it is easily customizable (this will become particular important when doing the "tricky merging parts" and talking about data visualization), and for free.

Here are a couple of good resources how to best translate your Stata knowledge into R:

- Stata to R Translation
- Turning your Stata Knowledge into R

First steps

These first steps show you how R works. If you need to look up functions, simply type in `? NAME` and replace `NAME` with the function name.

Data types

R knows several data types:

- **numeric**: decimal values, e.g. 4.3
- **integer**: natural numbers, e.g. 4 (integers are also numeric! $\rightarrow 4 = 4.0$)
- **logical**: `TRUE` and `FALSE`
- **character**: text or string

Vectors

As previously mentioned, R works with objects. We therefore produce a simple object.

```
year1 <- 1990
```

R users usually use arrows (`<-`) to assign objects. You might sometimes see equal signs as well (`=`). This, however, is bad practice because it does not allow you to say anything about the directions. If you want to know more about good practices in R, I recommend this manual.

We can now call this object by typing `year1`:

```
year1
```

```
## [1] 1990
```

What do we see? We assigned `year1` the value 1990.

This object is called a (numeric) *vector*. R can also take longer vectors. A vector is a set of values (one dimensional array).

```
year2 <- c(1995, 2000, 2005)
```

```
year2
```

```
## [1] 1995 2000 2005
```

We can also combine `year1` and `year2`.

```
years <- c(year1, year2)
```

```
years
```

```
## [1] 1990 1995 2000 2005
```

As we can see, we now added the 1990 of the `year1` vector to our `year2`.

The `years` vector is a *numeric vector*.

Vectors can also be strings, for example the ISO codes of countries.

```
country1 <- c("AGO", "RWA", "BDI")
```

```
country1
```

```
## [1] "AGO" "RWA" "BDI"
```

Similar to the numeric vectors, we can also combine several string vectors.

```
country2 <- "LSO"

countries <- c(country1, country2)
```

```
countries
```

```
## [1] "AGO" "RWA" "BDI" "LSO"
```

Single vectors usually only take one type of data (either numeric or character).

You may, sometimes, want to generate a vector that contains repeating values. This can be done with the `rep()` command.

```
repetition <- rep(2, 5)
```

As we can see, it contains the value of `2` five times.

```
repetition
```

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

Alternatively, we can also create a sequence.

```
sequence <- seq(1,5)
```

As we can see, the object `sequence` contains the values 1 to 5.

```
sequence
```

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

Hands-on exercise

1. Think of five cities. List them in a vector called `city`.
2. Now create a vector called `population` ranging between 10,000 and 100,000 (the distance between the numbers does not matter) that has the same length als your vector `city`. You could either do this by creating a vector manually. If you wish to do this automatically, you can use the `seq` function.

Matrices

If we want to combine our two resulting vectors (`countries` and `years`), we can do this with the `cbind()` command. This command combines **c**olumns whereas the `rbind()` command combines **r**ows. Both commands create *matrices*.

```
combined <- cbind(countries, years)
```

We can now have a look at our `combined` object:

```
combined
```

```
##      countries years
## [1,] "AGO"     "1990"
## [2,] "RWA"     "1995"
## [3,] "BDI"     "2000"
## [4,] "LSO"     "2005"
```

Alternatively, we can use `View()` to inspect our data. The `View()` command (note, capitalization in the beginning is important!) is similar to the `browse` command in Stata.

```
# View(combined)
```

A matrix is a two-dimensional array. Like vectors, it can take various data types: numeric, logical, character and, as we can see, all elements have the same data type.

Hands-on exercise

Repeat the same for your two vectors created above (`city` and `population`).

Data frames

Instead of creating matrices, we can also create a data frame.

```r
df_combined <- data.frame(countries, years)
```

We add `stringsAsFactors=FALSE` to avoid that R transforms our string variable `countries` in factors.

```r
df_combined <- data.frame(countries, years, stringsAsFactors = FALSE)
```

Hands-on exercise

Repeat the same for your two vectors created above (`city` and `population`).

Summary

- **Vectors**:
  - One-dimensional array
  - Can take various data types: numeric, logical, character
  - All elements have the same data type
- **Matrices**:
  - Two-dimensional array
  - Can take various data types: numeric, logical, character
  - All elements have the same data type
- **Data frames**:
  - Two-dimensional array
  - Can take various data types: numeric, logical, character
  - Within a column, all elements have the same data type; different columns can have different data types

How to structure a typical R document best

We will always come back to this structure, so please see it as a first disclaimer on how our documents will look like. I will also provide you with a template of this structure for your own research.

1. Preparation: Remove documents, install packages, load data sets

```r
# Remove all objects from R's memory
rm(list=ls())
```

```r
# We will use the following code to install all packages
packages <- c("ggplot2")

# Install uninstalled packages
lapply(packages[!(packages %in% installed.packages())], install.packages)
```

```
## list()
```

```r
# Load all packages to library
lapply(packages, library, character.only = TRUE)
```

```
## [[1]]
## [1] "ggplot2"   "stats"     "graphics"  "grDevices" "utils"     "datasets"
## [7] "methods"   "base"
```

```r
# Set working directory
setwd() # This command sets the working directory
getwd() # This command shows you the present working directory
dir() # This command shows you what's in your directory
```

2. Do the data management, analysis, visualization ...

```r
# Data management

# Data analysis

# Visualization
```

If you have more intensive data management and analysis projects, you might also consider saving the different sub-parts into single documents. For an illustrative example, see here and here.

Additional resources

- **How to set up a workflow (connecting R and github)**
- **R Markdown Reference Guide**