Intro to R II

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

Cosima Meyer

November 8, 2019

Recap: What did we learn last session?

- **R Universe**: R, R Studio, R Markdown
- Code needs to be written in chuncks in R Markdown (shortcut for chuncks: Alt + Cmd + I (Mac), Alt + Ctrl + I (Windows))
- R is **object-based**; that means we create and work with objects. Objects can be vectors, matrices, or data frames for example.

- **Vectors**:
  - One-dimensional array
  - Can take various data types: numeric, logical, character
  - All elements have the same data type

An example for vectors:

```
countries <- c("RWA", "AGO", "BEN") # This vector is consists of characters
countries # This command displays the vector
```

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

```
years <- c(1990, 1990, 1991) # This vector is numeric
years # This command displays the vector
```

```
## [1] 1990 1990 1991
```

- **Matrices**:
  - Two-dimensional array
  - Can take various data types: numeric, logical, character
  - All elements have the same data type

An example for a matrix:

```
mat_combined <- cbind(countries, years) # This command generates the matrix mat_combined
mat_combined
```

```
##      countries years
## [1,] "RWA"     "1990"
## [2,] "AGO"     "1990"
## [3,] "BEN"     "1991"
```

How to access values in a matrix?

To access a single value in a matrix, we need to think in a matrix. We need to call the matrix object first with `mat_combined` and then add with square brackets the rows, columns, or cells. The basic syntax is as follows `[row, column]`. If we just want to access the third row, we can write:

```
mat_combined[3,] # We access the third row
```

```
## countries     years
##     "BEN"    "1991"
```

If we want to access the second column we need to write respectively:

```
mat_combined[,2] # We access the second column
```

```
## [1] "1990" "1990" "1991"
```

How do we then access the cell in the first column and the first row?

Data frames

Instead of creating matrices, we can also create a data frame. Data frames are a standard way of handling social science problems in R. We generate a data.frame using `data.frame()`.

```
df_combined_test <- data.frame(countries, years)
```

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

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

Hands-on exercise

Remember what you generated last session? I'll provide you with two vectors for `city` and `population`.

```
# Vector for city
city <- c("Heidelberg", "Frankfurt am Main", "Hamburg", "Berlin", "Schwerin")

# Vector for population
population <- seq(10000, 100000, length.out = 5)
```

Repeat the same for your two vectors created above (`city` and `population`) and generate a data frame that does not transform strings into factors.

How to access parts of a data frame?

As in matrices, we can also access variables in data frames– that are usually listed in the columns of a data frame – with the dollar sign (`$`). Let's assume we want to access the column that contains information on countries in the data frame `df_combined`:

```
df_combined$countries
```

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

Accessing single columns seems to be fairly simple. But what if you want to access the very first observation of a column? (This might be useful for later data wrangling exercises.) We will use functions from the `dplyr` package here.

```
# Load package
install.packages("dplyr")
library(dplyr)
```

To access the first observation, we use `first()`.

```
first(df_combined$countries)
```

```
## [1] "RWA"
```

If we want to access the last observation, we use `last()`.

```
last(df_combined$countries)
```

```
## [1] "BEN"
```

Or, if we want to access the nth observation, we simply use `nth()`. Note, we need to specify which observation we want to display!

```
nth(df_combined$countries, 2)
```

```
## [1] "AGO"
```

Bonus: What is the difference between a `data.frame` and a `data.table`?

A data.table is an advanced version (a different format of) a data.frame that allows you to work with larger data sets. The syntax is fairly similar to SQL and base R. I hardly ever work with data.tables – I personally prefer the tidyverse because I find it more intuitive. We will learn more about this in our next session.

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

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

```
# We will use the following code to install all packages
# This code makes particularly sense if you require many packages.

packages <- c("ggplot2")

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

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

```
# Set working directory
setwd() # This command sets the working directory
# (not required if you are working in a project as we do right now)
getwd() # This command shows you the present working directory
dir() # This command shows you what's in your directory
```

2. Load the data

```
# Load data
```

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

```
# 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.