Introduction to R

Ina Krapp
SAFE Research Datacenter*

Last Update: November 20, 2023

R is a programming language often used for Data Analysis.
Programming your analysis with R instead of doing it in, say, Excel, has a large number of advantages:

1. It can easily be rerun. You can change your input (for example, append more data) and simply rerun everything instead of having to click through the whole analysis again. That also makes it much more replicable: You never need to worry about forgetting what you did. Anyone else can replicate your analysis easily as well.

2. R is non-commercial. It is free, so you never have to worry about license fees!

3. Related to that, anyone can implement and customize anything in R. You never have to wait for a company to finally release a new version with that new function everyone waited for.

In this tutorial, we will also use RStudio. RStudio is a user interface which makes working with R much more convenient. This tutorial is intended to be run as a QuartoMarkdown file in RStudio, which allows you to execute the code yourself. To run it, download the QuartoMarkdown version here and open it in a recent version of RStudio.

Contents

*krapp@safe-frankfurt.de

1 Contact

If you encounter any difficulties or just want general information, do not hesitate to contact us.

SAFE Research Datacenter: datacenter@safe-frankfurt.de

More information about the SAFE Research Datacenter and further guides can be found here.

2 Prerequisite

R can be downloaded here.
We'll also use RStudio. You can get it here.

3 A first program

»

If you are completely new to programming, you may not know yet what the code below does (though you can probably guess it). Run it (by clicking on the green dart on the right) and see if you're right.

```
1 + 1
```

```
> 1 + 1
[1] 2
```

You can use R as a calculator. Change this code to calculate how much 2 and 3 are. In practice, you will often use code of other people and modify it. But of course, there's much more to R than that.

4 Vectors

When you have a dataset, you'll usually not just have one or two numbers. Assume you have data on a household. Two people are children and earn nothing, the father earns 1000 Euros a month, the mother 3000. There are different ways you could work with this data in R.
For example, you could put it into a vector. A vector can be created in R by typing c() and entering the values, separated by commas, between the brackets. The letter c is used because this is a column vector - you can imagine it as being the column of a table.

```
household_income <- c(0,0,1000,3000)
```

In the upper right of the monitor, in the field 'Environment', you can see the numbers now. The Environment now contains the value household_income, which is a vector with the four numbers I entered above. You can call it and look at it by typing its name - auto-complete will help with that.

```
household_income
```

```
> household_income
[1]    0    0 1000 3000
```

If you only want to look at individual values, you can call them by their position in the vector.

```
    household_income[4]
```

```
> household_income[4]
[1] 3000
```

You can calculate with them. A major advantage of R is that it is vectorized - you can calculate with an entire vector just like you would with a single value.
Assume each member of the family gets 100 Euro by the state. Then, you can simply calculate that with this code:

```
    household_income + 100
```

```
> household_income + 100
[1]  100  100 1100 3100
```

Note that the values in the environment are still the old ones. The result of the calculations is only shown, it is not saved. To save a value, you need to assign it to a variable name:

```
new_household_income = household_income + 100
```

The equal sign in many programming languages is used in that way. But most other mathematical symbols do the same as you know from school math.
The newly created variable is again visible in the 'Environment' pane. To see it here again, simply type its name.

```
    new_household_income
```

```
new_household_income
[1]  100  100 1100 3100
```

When you run such code, you can assign it to the same variable name you used before. Assume each family member gets another 200 Euro tax returns:

```
    new_household_income = new_household_income + 200
```

But be careful when running such code. Run it again and look at how the values (in the 'Environment' pane) are changing when you do so.
Each time you run the code, another 200 gets added. Running it more than once therefore gives you a wrong value. Click on the broom (above the 'Environment' pane) to clear out the wrong value. Don't be concerned about deleting everything: Remember that R is easily replicable. All data this pane contained can be recreated by simply running the complete code again. Click on the 'run all chunks above' button in the code block (this is the grew downward arrow to the left of the green arrow,) to recreate the environment.
You can also, for example, add or subtract vectors from each other. But you should only do that when they have the same length. Assume, for example, each family member spent some money: The children spent 10 and 20 Euros on sweets, the father was shopping for 500 Euros, the mother paid the rent for 1500 Euros. Then you can calculate how much money is left like that:

```
    new_household_income - c(10, 20, 500, 1500 )
```

```
> new_household_income - c(10, 20, 500, 1500 )
[1]  290  280  800 1800
```

Assume each member pays another 10 Euros when they all go to the cinema together. How can this be calculated?

```
new_household_income - 10
```

```
> new_household_income - 10
[1]  290  290 1290 3290
```

This gives the same result:

```
new_household_income - c(10, 10, 10, 10 )
```

```
new_household_income - 10
[1]  290  290 1290 3290
```

5 Functions

There are also functions available to make calculations simpler. Assume you'd want to know how much money the family owns on average. You could calculate it like that:

```
(new_household_income[1] + new_household_income[2] + new_household_
  income[3] + new_household_income[4])/4
```

```
> (new_household_income[1] + new_household_income[2] + new_household_income[3] +
[1] 1300
```

That is a perfectly valid calculation. It adds the money of all family members and divides it by the number of family members. But it took a while to type. R has a build-in function that does the job:

```
mean(new_household_income)
```

```
> mean(new_household_income)
[1] 1300
```

Likewise one can find the largest or smallest value in a vector with the functions sum, max and min, and do many other things. Assume you want to find a function but don't know if it exists. In that case, the 'Help' pane (below the 'Environment' pane) is your best friend.
Use it to find the function that sums up all elements in the vector and calculate how much all family members earn combined.

```
sum(new_household_income)
```

```
> sum(new_household_income)
[1] 5200
```

Most functions have several arguments, separated by commas. The first argument is usually the data the function should be applied to, and it is almost always required. But often, other arguments are optional. For example, functions like mean and sum include a na.rm argument. If you set it to TRUE, the function will ignore missing values. As default, it is set to false. Compare the results:

```r
household <- c(100, 100, NA, 200)
sum(household)
# False is the default, so this is the same as above:
sum(household, na.rm = FALSE)
# Setting it to TRUE ignores the missing value:
sum(household, na.rm = TRUE)
```

```
> household <- c(100, 100, NA, 200)
> sum(household)
[1] NA
> # False is the default, so this is the same as above:
> sum(household, na.rm = FALSE)
[1] NA
> # Setting it to TRUE ignores the missing value:
> sum(household, na.rm = TRUE)
[1] 400
```

6 Dataframes

There are different types of data structures in R. As mentioned above, you can imagine a vector also as column of a table. It does not necessarily have to contain numbers. The code below constructs a character vector:

```r
household_names = c('Anna', 'Benny', 'Christian', "Daniela")
```

But with such vectors, of course, you can't do many calculations.
More frequently, you'll encounter them as parts of dataframes. These are entire tables, which you can imagine as two or more columns side by side. You can create them by giving vectors of the same length into the data.frame function:

```r
household_data = data.frame(household_names, household_income)
```

Now, therés a new field in the Environment in the upper right corner. It is called Data and contains 'household_data'. You can click on the light blue button besides it to look at it into more detail.
This object is described as '4 obs. of 2 variables', in long form: '4 observations of 2 variables'. R always interprets dataframes in this way: It is assumed that each column in the table represents a specific variable (here, name and income of each family member) and each row represents an observation for the variables (here, we have 4 rows, one for each family member). Each column of a dataframe, being a variable, has a variable name. Since the dataframe was constructed from the named vectors household_income and household_names, the variables use these words as names. You can access them by writing the name of the dataframe, then a $-sign, then the name of the column (again, autocomplete will help and show which variables are in the dataframe).

```r
household_data$household_income
```

```
> household_data$household_income
[1]    0    0 1000 3000
```

This looks like a vector and behaves like a vector because it is - surprise - a vector. Again, you can simply take the mean, median, sum or use some other function that accepts a vector as input.
Calculate the median of the household_income from the household_data dataframe.

```r
median(household_data$household_income)
```

```
> median(household_data$household_income)
[1] 500
```

On dataframes, you can also conduct much more complex forms of analysis - from fixed effects regression to timeseries forecasting, R covers almost all modern forms of statistical analysis. This tutorial will not cover all that, but the second workshop will show those who are interested how to conduct a standard linear regression.
To finish this first part, we'll look at packages and what you can do with them.

7 Installing and activating a package

R is Open Source. A Core Team is working on it, but anyone can contribute. Contributions are typically packages that can be installed. They allow many, many things - creating maps, developing apps or animating small films, for example.
The package we'll install now serves a more mundane purpose: Data comes in a wide variety of formats. The standard Stata format (dta), Excel-files (xlsx) and csv-files are just a few examples. Packages allow to work with all of them.
We'll load a csv-file with the readr-package.
The library-function allows you to activate a package. But if you copy the code below, you'll receive an error. This is because the package is not installed yet. You can install a package by going to 'Packages' in the lower left pane, clicking on 'Install' and typing the name of the package which you want to install.
The run the library function and your package is activated. Now it can be used.

```r
library(readr)
```

We'll use the package to load the data from here: https://github.com/allisonhorst/palmerpenguins/blob/m
Download it and put it into the same folder that you have currently open in RStudio. A file named 'penguins.csv' should be visible in 'Files' now.
Now, there are two ways to load a file into R. One is to go to the 'Files' pane (another pane in the lower right) and search the csv file. Once you found it, click on it with the left mouse button and then click on 'Import Dataset'.
The other one is using code. It has the advantage that it can easily be replicated. For that reason, if you click on 'Import ', you get a code preview in the lower right corner. Copy it to the code block below.

```r
penguins <- read_csv("penguins.csv")
View(penguins)
```

We already loaded readr, so you can delete the library command before running the code.
The View function shows the dataset. But you can also look at it by clicking on the word 'penguins' in the 'Environment' pane.
In the workshop 'Linear Regression with R', we'll work with this dataset and look what it tells us about the penguins.

Finally, to convince yourself that R code is replicable, click on the broom in the environment pane to delete all objects. Then click on the Run-button above and on 'Run All'. You'll see the objects will be created again.