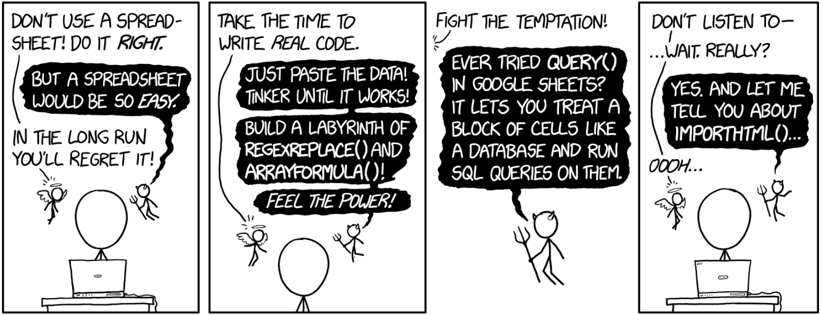
Introduction to data analysis in R

Getting started in R

Mahmoud Ahmed

April 2022

경상국립대학교
Gyeongsang National University


DON'T USE A SPREADSHEET! DO IT *RIGHT.*
BUT A SPREADSHEET WOULD BE SO *EASY.*
IN THE LONG RUN YOU'LL REGRET IT!
TAKE THE TIME TO WRITE *REAL* CODE.
JUST PASTE THE DATA! TINKER UNTIL IT WORKS!
BUILD A LABYRINTH OF REGEXREPLACE() AND ARRAYFORMULA()!
FEEL THE POWER!
FIGHT THE TEMPTATION!
EVER TRIED QUERY() IN GOOGLE SHEETS? IT LETS YOU TREAT A BLOCK OF CELLS LIKE A DATABASE AND RUN SQL QUERIES ON THEM.
DON'T LISTEN TO—
...WAIT. REALLY?
YES. AND LET ME TELL YOU ABOUT IMPORTHTML()...
OOOH...

https://xkcd.com/2180/

What is R?

You can think of R as a fancy calculator. One that can perform operations on not just numbers but all types of data.

Here are a few reasons why, learning to do data analysis in R can be useful to you.

- Store, manipulate, analyze and visualize data
- Scale data analysis
- Benefit from open-source tools and methods
- Reproducible analysis

How to get started in R?

- Once you get over the hurdle of expressing your experimental design in code, you will come to appreciate how easy it is to do data analysis.
- Only need a few things to know to start with.
- But there is much more that you will need to learn once you know what to look for.
- Most benefits can be accrued from many other tools.

Your fancy calculator

To drive home the calculator metaphor, look at the following line of code.

```
# compute 1 + 1
1 + 2

## [1] 3
```

The code above just adds one plus one, just like a calculator.

Note on the code (line by line):

1. Comments, proceeded by '#'
2. Code
3. Output, proceeded by '##'

Variables

The code below does the same, with an extra trick.

```
# assign x a value of 1
x <- 1

# add x plus 2
x + 2

## [1] 3
```

x here is a variable and it contains the value we assign to it using this symbol <-.

Using variables rather than typing the values is necessary for several reasons, here is just two of them

- Storing large datasets
- Reusing the same datasets

Functions

To assign multiple values to x you need a function called c, it stands for combine.
Here is a code that stores the values 1, 2, and 3 in a variable called x and prints it out.

```
# assign multiple values to x
x <- c(1, 2, 3)
x

## [1] 1 2 3
```

The function c which combines any number of values you put between parenthesis, (), separated by commas, ','.

Whenever you want to do anything in R, you should look at the function that does it. If there isn't one you can create it yourself.

More functions

The function sum takes a variable and returns its sum, The function length takes a variable and returns its length.

```r
# make a variable x
x <- c(1, 2, 3)

# call sum
sum(x)

## [1] 6

# call length
length(x)

## [1] 3
```

More functions

The function called mean takes a variable and returns its mean (average).

```r
# make a variable x
x <- c(154, 223, 309)

# call mean
mean(x)

## [1] 228.6667
```

Combining function outputs

Knowing that the mean of a variable it the sum divided by the length, we can compute it in a different way.

```r
# make a variable x
x <- c(154, 223, 309)

# compute the mean
sum(x) / length(x)

## [1] 228.6667
```

Your output should match the one we got using mean.
Notice the output of a function is itself a variable and functions can be combined in one line to produce other variables.

Functions

There is a function called function that is used to create a function.

takes as an input a variable ‘x‘ and computes its mean. It does that by dividing the sum by the length of the variable.

```r
# create a function to compute the mean
newfun <- function(x) sum(x) / length(x)

# call new fun
x <- c(154, 223, 309)
newfun(x)

## [1] 228.6667
```

sum calculates the sum of items in a variable and length counts in the number of items in a variable. / divides the first variable by the second.

Data types

The most basic data types are

1. Numeric: are real numbers such as 1.1, 1.2, or 1.3
2. Integer: are whole numbers such as 1, 2, or 3
3. Character: are strings of letters such as 'name', 'age', or 'gender'
4. Logical: are either TRUE or FALSE

Each of these types has a direct analogy in the real world data you will be analyzing.

They make life easier for you and your computer, but at cost of a bit of confusion such as the one you feel right now.

Creating variables of different types

The code creates variables of the four different types.

```r
# create a numeric, just numbers
num <- c(1.1, 1.2, 1.3)

# create an integer, proceeded by 'L'
int <- c(1L, 2L, 3L)

# create a character, between quotes
char <- c('name', 'age', "gender")

# create a logical, uppercase and no quotes
logi <- c(TRUE, TRUE, FALSE)
```

Finding variable type

To confirm what type a variable is, use the function class. Here is an example

```
# create a numeric, just numbers
num <- c(1.1, 1.2, 1.3)

# examine a variable class
class(num)

## [1] "numeric"
```

These variables also behave differently. You will come to appreciate them in time since they make things easier when it comes to data analysis.

Converting between classes

You can convert some data types into others. Often a function as.class exist to do just that.

This code converts a numeric to an integer and a character

```r
# create a numeric, just numbers
num <- c(1.1, 1.2, 1.3)

# call as.integer
as.integer(num)

## [1] 1 1 1

# call as.character
as.character(num)

## [1] "1.1" "1.2" "1.3"
```

Class in, class out

Often, the input to a function is of one type (class) and the output is of another. This can be confusing but very useful.

```
# which value in num is equal to 1.2
num == 1.2

## [1] FALSE  TRUE FALSE

# which value in num is bigger than 1.2
num > 1.2

## [1] FALSE FALSE  TRUE

# which value in num is smaller 1.2
num < 1.2

## [1]  TRUE FALSE FALSE
```

Although the inputs are all numeric, the output will be logical.

Data frames

Most of the time data comes in tables, and in mixed types: numbers, characters and all.

a data.frame is basically a table with rows and columns. Columns can have different types of data.

Here is an example

```
# make variables
x <- rep(c(1, 2), each = 3)
y <- rep(c('A', 'B'), each = 3)

# make a data.frame
d <- data.frame(x, y)
head(d, n = 3)

##   x y
## 1 1 A
## 2 1 A
## 3 1 A
```

Inspecting data structure

Some time the dataset is small, so you can print it in its entirety without a problem. However this is not always the case.

The function str can show you the most important info you need about the object without having to print it out on the screen.

```
# call str on the object d
str(d)

## 'data.frame': 6 obs. of  2 variables:
##  $ x: num  1 1 1 2 2 2
##  $ y: Factor w/ 2 levels "A","B": 1 1 1 2 2 2
```

Subsetting

Just like there is a way to build bigger objects, there is a way to subset (slice) them.

This code subsets the first column from the object d.

```
# subset first column
d[, 1]

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

To subset the first row of the object d, you do something similar.

```
# subset the first row
d[1,]

##   x y
## 1 1 A
```

Bigger slices

You can subset a data.frame by column and rows at once. You can also get more then one column or row at once.

Just substitute the index between [] with a vector of more then one index.

```
# subset rows and columns
d[c(1, 2), c(1,2)]

##   x y
## 1 1 A
## 2 1 A

# or
d[1:2, 1:2]

##   x y
## 1 1 A
## 2 1 A
```

Getting help

One of the benefits of using a statistical programming language is that you are never alone.

Pretty much any kind of analysis you would wish to do someone else has thought of. This also applies to learning the syntax of the language and solving the common issues that beginner face.

These are ways to get help

- Type a question mark ? followed by the name of a package, function, or class
- Copy and paste errors and warnings into your preferred search engine
- Trial and error

Summary

What you learned

- Variables
- Functions
- Data types
- Data frames
- Subsetting
- Getting help

What's next

- Practice (Link)
- Homework (Link)
- Module 2: Basic statistics in R (Link)