Lab 01: Getting started with R and R Studio

Mark Huber

Summary

In this lab you will be introduced to R and R Studio.

Installing R and R Studio

Before beginning this lab, you should install R and R Studio on your laptop or computer.

- You can download R from https://www.r-project.org/ (https://www.r-project.org/).
- You can download RStudio from https://www.rstudio.com/ (https://www.rstudio.com/), clicking on Products, then RStudio (**not** RStudio Server), then RStudio Desktop, then the button marked *Download RStudio Desktop*. Download the free version and install.

Setting up your script file

The solution to your labs will always be a *script* file. This file will assign values to 10 variables, named `answer01` through `answer10`.

Download the file `lab01.R` from the Sakai website. Open this file by double clicking on it. If R Studio has been properly installed, the file will open up automatically in R Studio.

Note that the ten variables `answer01` through `answer10` have been defined for you using the assignement operator `<-`.

Gradescope

The labs will be submitted and graded through Gradescope. You are allowed to submit as many times as you want before the lab is due. Try going to gradescope.com and submitting the file `lab01.R` as the answer for Lab 1.

Three of the answers have been filled in for you correctly in the file. The variables `answer01`, `answer02`, and `answer03` are all assigned correctly, so your submission should receive a score of 3 out of 10. To see why these first three are the correct answers, and to figure out the answers to the rest of the problems, read on!

Assigning a value to a variable in R

The word *variable* is used in two different ways in Data Science. In a dataset, a *variable* is something that can be measured, like the mass of an object.

In a computer language, a *variable* is a something that can store a value. The *assignment operator* `<-` is used to assign the thing on the right of the operator to the variable name on the left. For instance,

```
x <- -1.4
```

assigns the value -1.4 to the variable name `x`. Once assigned, when the variable name is used in an expression, -1.4 is automatically substituted. For instance,

```
x * (-2)
```

```
## [1] 2.8
```

returns 2.8, since $-1.4 \cdot (-2) = 2.8$. By the way, the `*` operator in R is multiplication. Other arithmetic operators include `/` for division, `+` for addition, and `-` for subtraction.

Problem 1

Assign the variable `answer01` the value 42.

The solution to this problem is

```
answer01 <- 42
```

which is the first line of code in the script file `lab01.R`.

Another useful arithmetic operator is *exponentiation*, which uses the circumflex symbol `^`. So `3^5 = 3 * 3 * 3 * 3 * 3`.

Problem 2

Given that taking the square root of a number is the same as raising it to the 1 / 2 power, assign `answer02` the square root of 42.

Normally, exponentiation comes before multiplication comes before addition, but using parentheses can force the order of operations you desire. For instance, another way to get `3^5` is to use:

```
3^(3 + 2)
```

```
## [1] 243
```

Data types

There are several different data types in R. Two common ones are the *integer* data type, abbreviated `int`, and the *string* data type, abbreviated `chr`.

- The `int` data type holds integers such as 47, 112, and 62.
- The `chr` data type holds strings of data such as `"Hello there!"`, `"Nice weather we are having!"`, and `"How're the kids doing?"` Note that string data must be surrounded either by double quotes `"` or single quotes `'` in R.

Problem 3

Consider the following dataset of Westbound exits from the I-10 interstate that contains both string and integer data.

name	number
Central Avenue, to Montclair	49
Monte Vista	48
Indian Hill Boulevard	47

What is the exit for Indian Hill Boulevard (`int` data type)?

The answer to the above problem can be written in the script as

```
answer03 <- 47
```

Note that there are no quotes around integer typed variable values. Now consider the next problem.

Problem 4

Continuing the last problem, what is the exit name for exit 48?

The answer is

```
answer04 <- "Monte Vista"
```

Note that the answer is in quotation marks. This is because it is a *string* type object.

One last data type that appears often is the *double*. These are for floating point numbers such as `3.4` or `-1.2222222`. The reason they are called double is that originally computers tended to use 32 bits to store these numbers. Later on, 64 bits were used, double the original amount of space.

Sequences

Sequence notation can be a fast way of getting numbers that are part of an arithmetic sequence, that is, a set of numbers whose difference is always the same number. The colon operator `:` gives a fast way to build simple sequences. For instance, to get the numbers $1, 2, \ldots, 10$, use:

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

A more powerful option is the seq function. This would look as follows.

```
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Note that the output of this function can be assigned to a variable.

```
seq1 <- seq(1, 10)
```

It can then be retrieved later.

```
seq1
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

To skip by a number different than 1, use the `by` parameter.

```
seq(1, 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

Problem 5

Assign to `answer05` the integers from 20 to 40, skipping by 3.

Vectors

The output from the seq command is called a *vector*, which in R is an ordered list of variable values. The `c` command (short for combine) can be used to make a vector with any components that you wish. For instance:

```
c(3, "a", 4, "b")
```

```
## [1] "3" "a" "4" "b"
```

Note that vectors need to have the same data type for all the components. Hence the first and third components were changed from integers to strings to make all of the components the same data type.

Problem 6

Assign to `answer06` a vector consisting of numbers `-11` , `12` , and `6` .

Statistics

There are several useful statistics commands that apply to vectors. The length function is probably the simplest, it counts how many components there are in a vector.

```
length(c(3, 4, -1))
```

```
## [1] 3
```

```
length(212)
```

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

```
length(c())
```

```
## [1] 0
```

When the components are integers or doubles, sum adds up all the numbers.

```
sum(c(3, 11, 7))
```

```
## [1] 21
```

The mean function takes the sample average of the numbers, which is the sum of the numbers divided by their length.

```
mean(c(3, 11, 7))
```

```
## [1] 7
```

The sd function is an estimate of the standard deviation of the numbers. Roughly speaking, this gives an idea of the spread in the numbers.

```
sd(c(3, 11, 7))
```

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

The next vector has the same average value of 7 as the last one.

```
mean(c(-27, 41, 7))
```

```
## [1] 7
```

But the spread away from 7 is much greater.

```
sd(c(-27, 41, 7))
```

```
## [1] 34
```

Problem 7

Assign to `answer07` the sample mean of the vector `(24, 3, 14, 19)`.

Problem 8

Assign to `answer08` the sample standard deviation of the vector `(24, 3, 14, 19)`.

Writing new functions

You are not limited to the built in functions in R! The way to write a new function is to assign it to a variable name using the keyword function.

For instance, consider a new function `f1` of the form $f_1(x, y) = x + 2y$. This could be constructed as follows.

```
f1 <- function(x, y) {
  return(x + 2 * y)
}
```

The function can now be used with any two inputs.

```
f1(3, 4)
```

```
## [1] 11
```

Problem 9

Assign to variable name `answer09` a function where $(x, y) \mapsto -x + 4y$.

Problem 10

Assign to variable name `answer10` a function where $(x, y) \mapsto xy$.

Note that by default, the arguments to `f1` are given in the same order that they are defined. So `x` first, then `y`. By explicitly naming the inputs, this order can be changed.

```
f1(y = 10, x = 4)
```

```
## [1] 24
```

When defining a function, the inputs can be given default values so that it is not necessary to always define them.

```
f2 <- function(x = 4, y = 10) return(x + 2 * y)
```

```
f2()
```

```
## [1] 24
```

```
f2(x = 6)
```

```
## [1] 26
```

```
f2(y = 5)
```

```
## [1] 14
```

Functions are an integral part of R, and form the heart of the code written for Data Science!