Lab 1 Solutions

Stats 32: Introduction to R for Undergraduates

Harrison Li

April 2, 2024

Your solutions to this lab should be uploaded to Gradescope as a knitted .pdf file before 1:30 pm today. As this is the first lab, it will not count for credit, but you are highly encouraged to read the solutions when they are posted later today.

Note: The content of this lab is borrowed heavily from Kenneth Tay's course materials in the Autumn 2019 iteration of this course.

Installation

Before you do anything, you must first install R from `https://cran.r-project.org/`. Click the link at the top of the page that says "Download R for (Your Operating System)", and follow the instructions on that page. For Mac you will want to download the latest `.pkg` file compatible with your operating system. For Windows you can directly follow the link to the `base` distribution.

Next, you should install RStudio Desktop from `https://www.rstudio.com/products/rstudio/download/`, following instructions according to your operating system.

RStudio

When you open this file in RStudio, you will see 4 different windows along with a number of tabs:

Bottom-left

This is the **R console**, where you key in commands to be run in an interactive fashion. Type in your command and hit the Enter key. Once you hit the Enter key, R executes your command and prints the result, if any.

Top-right

- **Environment:** List of objects that we have created or have access to. We can also see the list of objects using the command `ls()`.
- **History:** List of commands that we have entered into the console.

Bottom-right

- **Files:** Allows you to navigate the directory structure on your computer.
- **Plots:** Any graphical output you make will be displayed here.
- **Packages:** List of packages that have been installed on your computer.
- **Help:** Documentation for `functionName` appears here when you type `?functionName` in the console.
- **Viewer:** For displaying local web content.

Top-left

This is a text editor where you can edit the contents of this file.

This file has an .Rmd (R Markdown) extension. R Markdown allows you to intersperse text with R code. There are some useful shortcuts (e.g. to enter bold or italic text, or insert an image) that can be easily searched up on the Internet.

Code needs to be written inside code chunks. To insert a code chunk at your cursor inside an .Rmd file, go to Code -> Insert Chunk. There is also a keyboard shortcut: Control+Alt+I on Windows or Control+Option+I on macOs.

You can run all the code in a code chunk by pressing the "play" button at the top right of the chunk. The button just to the left (a green rectangle below a downward facing triangle) runs all chunks above but not including that chunk.

Press the Knit button near the top of the screen to combine the text and code in your .Rmd file into a single PDF. Note that knitting will not work if there are any errors, which you will be notified of in the console.

R as a calculator

You can use R has a high-powered calculator. For example,

```r
1 + 2
```

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

```r
456 * 7
```

```
## [1] 3192
```

```r
5 / 2
```

```
## [1] 2.5
```

There are several math functions which come with R. For example, to evaluate $log(e^{25} - 2^{\sin(\sqrt{\pi})})$, we would type

```r
log(exp(25) - 2^(sin(sqrt(pi))))
```

```
## [1] 25
```

Variable assignment

Often, we want to store the result of a computation so that we can use it later. R allows us to do this by variable assignment. Variable names must start with a letter and can only contain letters, numbers, _ and ..

The following code assigns the value 2 to the variable x:

```r
x <- 2
```

Notice that no output was printed. This is because the act of variable assignment doesn't produce any output. If we want to see what x contains, simply key its name into the console:

```r
x
```

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

Types of variables

Apart from numbers, R supports a number of different data types. The most common types are numeric, character (i.e. strings), factor, and logical. We'll talk about factors later in the course.

We can check the type of a variable using the `typeof()` function:

1. Consider the following two variables. What are their types? Verify this with code.

```
x <- "1"
y <- TRUE
```

We can change the type of a variable to type x using the function `as.x()`. This process is called "coercion". For example, the following code changes the number `6507232300` to the string `"6507232300"`:

```
as.character(6507232300)
```

```
## [1] "6507232300"
```

```
typeof(6507232300)
```

```
## [1] "double"
```

```
typeof(as.character(6507232300))
```

```
## [1] "character"
```

We can also change character variables to numeric or logical variables.

```
as.numeric("123")
```

```
## [1] 123
```

```
as.logical(123)
```

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

```
as.logical(0)
```

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

Sometimes type conversion might not work:

```
as.numeric("def")
```

```
## Warning: NAs introduced by coercion
```

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

Other times, type conversion does not work as you might expect. Always check that the result is what you want by printing!

```
as.logical("123")
```

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

Note we can also use numeric variables in computations:

```
x <- 1
x^2 + 3*x
```

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

We can also reassign x to a different value:

```
x <- x+1
x
```

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

2. Explain what the value of x and y should be after I execute the following code. Verify this by printing it out.

```
y <- x
x <- x^2
```

Let's add a third variable:

```
z <- 3
```

Note that we now have 3 entries in our Environment tab: `x`, `y`, and `z`. To remove an object, use the `rm()` function:

```
rm(x)
```

To remove more than one object, separate them by commas:

```
rm(y, z)
```

Let's add the 3 variables back again:

```
x <- 1
y <- 2
z <- 3
```

To remove all objects at once, use the following code:

```
rm(list = ls())
```

Reasons you might want to remove objects include: - Not wanting old stored values of variables to interfere with your code - Saving memory (RAM)

Vectors

For data analysis, we often have to work with multiple values at the same time. There are a number of different R data structures which allow us to do this.

The **vector** is a 1-dimensional array whose entries are the same type. For example, the following code produces a vector containing the numbers 1,2 and 3:

```
vec <- c(1, 2, 3)
vec
```

```
## [1] 1 2 3
```

Typing out all the elements can be tedious. Sometimes there are shortcuts we can use. The following code assigns a vector containing the numbers 1 to 100 to the variable `vec`:

```
vec <- 1:100
vec
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

3. We can use the `c()` function to combine ("concatenate") several small vectors into one large vector. How many elements does the vector `z` below have? Explain.

```
z <- 1:5
z <- c(z, 3, z)
```

R allows us to access individual elements in a vector. Unlike many other programming languages, indexing begins at 1, not 0. For example, to return the first entry in the vector `z`, I would use the following code:

```r
z[1]
```

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

We can get multiple elements of a vector as well. The following code extracts the 2nd-4th elements of `z`.

```r
y <- z[2:4]
y
```

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

Matrices

Matrices are just the 2-dimensional analogs of vectors. We won't be talking about them a whole lot in this class. As with vectors, the entries of a matrix have to all be of the same type.

Use the `matrix()` command to change the built-in vector `LETTERS` into a matrix:

```r
A <- matrix(LETTERS, nrow = 2, byrow=TRUE)
A
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,] "A"  "B"  "C"  "D"  "E"  "F"  "G"  "H"  "I"  "J"   "K"   "L"   "M"
## [2,] "N"  "O"  "P"  "Q"  "R"  "S"  "T"  "U"  "V"  "W"   "X"   "Y"   "Z"
```

Notice that R takes the elements in the vector you give it and fills in the matrix column by column. If we want the elements to be filled in by row instead, we have to put in a `byrow = TRUE` argument:

```r
B <- matrix(letters, nrow = 2, byrow = TRUE)
```

To get the dimensions of the matrix, we can use the `dim()`, `nrow()` and `ncol()` functions.

```r
dim(B)
```

```
## [1]  2 13
```

```r
nrow(B)
```

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

```r
ncol(B)
```

```
## [1] 13
```

To access the element in the `i`th row and `j` column for the matrix `B`, use the index `i,j`:

```r
B[1, 2]  # for the element in the 1st row and 2nd column
```

```
## [1] "b"
```

  4. Try out the commands `A[2,]` and `A[,2]`, and explain what they do.

```r
A[2,]
```

```
##  [1] "N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```r
A[,2]
```

```
## [1] "B" "O"
```

  5. Write code to select the 5th, 7th, and 12th columns of A.

Lists

In all the data structures so far, the elements have to be of the same type. To have elements on different types in one data structure, we can use a list, which we can create with `list()`. The elements of a list can be *named*; for instance, in the code below, `person` is a 2-element list. The elements have names `name` and `age`, and values `"John Doe"` and `26`, respectively.

```
person <- list(name = "John Doe", age = 26)
person
```

```
## $name
## [1] "John Doe"
##
## $age
## [1] 26
```

There are two ways to access the `name` element in `person`:

```
person[["name"]]
```

```
## [1] "John Doe"
```

```
person$name
```

```
## [1] "John Doe"
```

6. What are the variable types of `person$name` and `person$age`?

The elements of a list can be anything, even another data structure (vector, matrix, etc.)! Let's add the names of John's children to the `person` object. The `str()` function (short for structure) can provide a useful summary of the overall structure and contents of a complex data structure.

```
person$children = c("Ross", "Robert")
str(person)
```

```
## List of 3
##  $ name    : chr "John Doe"
##  $ age     : num 26
##  $ children: chr [1:2] "Ross" "Robert"
```

We confirm `str` is a named list of length 3, with names `name`, `age`, and `children`.

To extract the names associated with a list, use the `names()` function:

```
names(person)
```

```
## [1] "name"     "age"      "children"
```

7. How can you extract the name of the first child in `person$children`?

8. What is the difference between `person[["children"]]` and `person["children"]`?

9. Explain, in words, what the following code does.

```
names(person) <- c("full_name", "age_years", "child_names")
```

10. Sadly, John has passed away. Remove the "person" object from the environment.