Lecture 1: Getting started with R

Katie Schuler

2023-08-29

1 Why R?

With many programming languages available for data science (e.g. R, Python, Julia, MATLAB), why use R?

- Built for stats, specifically
- Makes nice visualizations
- Lots of people are doing it, especially in academia
- Easier for beginners to understand
- Free and open source (though so are Python and Julia, MATLAB costs $)

If you are interested, here is a math professor's take on the differences between Python, Julia, and MATLAB. Note that although they're optimized for different things, **they're all great** and the technical skills and conceptual knowledge you gain in this course easily transfers to other languages.

2 Colab notebook basics

There are many ways to program with R. Some popular options include:

- R Studio
- Jupyter
- VS Code
- and even simply the command line/terminal

Google Colab is a cloud-based Jupyter notebook that allows you to write, execute, and share code like a google doc. We use Google Colab because it's simple and accessible to everyone. You can start programming right away, no setup required! Google Colab officially supports Python, but secretly supports R (and Julia, too!)

New R notebook:

- colab (r kernel) - use this link to start a new R notebook
- `File > New notebook` error, Python! `name 'x' is not defined`

Cell types:

- `+ Code` - write and execute code
- `+ Text` - write text blocks in markdown

Left sidebar:

- `Table of contents` - outline from text headings
- `Find and replace` - find and/or replace
- `Files` - upload files to cloud session

Frequently used menu options:

- `File > Locate in Drive` - where in your Google Drive?
- `File > Save` - saves
- `File > Revision history` - history of changes you made
- `File > Download > Download .ipynb` - used to submit assignments!
- `File > Print` - prints
- `Runtime > Run all` - run all cells
- `Runtime > Run before` - run all cells before current active cell
- `Runtime > Restart and run all` - restart runtime, then run all

Frequently used keyboard shortcuts:

- `Cmd/Ctrl+S` - save
- `Cmd/Ctrl+Enter` - run focused cell
- `Cmd/Ctrl+Shift+A` - select all cells
- `Cmd/Ctrl+/` - comment/uncomment selection
- `Cmd/Ctrl+]` - increase indent
- `Cmd/Ctrl+[` - decrease indent

3 R basics

Defining some basic concepts:

- **Expressions** are combination of values, variables, operators, and functions that can be evaluated to produce a result. Expressions can be as simple as a single value or more complex involving calculations, comparisons, and function calls. They are the fundamental building blocks of programming.
  - `10` - a simple value expression that evaluates to `10`.
  - `x + 10` - an expression that adds the value of `x` to `10`

  - `a <- x + 10` - an expression that adds the value of `x` to `10` and assigns the result to the variable `a`

- **Objects** in R allow us to store various types of data, such as numbers, text, vectors, matrices; and more complex structures like functions and data frames. Objects are created by assigning values to variable names with the assignment operator, `<-`. For example, in `x <- 10`, `x` is an object assigned to the value 10.
- **Names** that we assign to objects must include only letters, numbers, `.`, or `_`. Names must start with a letter (or `.` if not followed by a number).
- **Attributes** allow you to attach arbirary metadata to an object. For example, adding a `dim` (dimension) attribute to a vector allows it to behave like a matrix or n dimensional array.
- **Functions** (or commands) are reusable pieces of code that take some input, preform some task or computation, and return an output. Many functions are built-in to base R (see below!), others can be part of packages or even defined by you. Functions are objects!
- **Environment** is the collection of all the objects (functions, variables etc.) we defined in the current R session.
- **Packages** are collections of functions, data, and documentation bundled together in R. They enhance R's capabilities by introducing new functions and specialized data structures. Packages need to be installed and loaded before you can use their functions or data.
- **Comments** are notes you leave to yourself (within code blocks in colab) to document your code; comments are not evaluated.
- **Messages** are notes R leaves for you, after you run your code. Messages can be simply for-your-information, warnings that something unexpected might happen, or erros if R cannot evaluate your code.

Important functions to know:

- for objects:
  - `str(x)` - returns summary of object's structure
  - `typeof(x)` - returns object's data type
  - `length(x)` - returns object's length
  - `attributes(x)` - returns list of object's attributes
  - `is.*(x)` - test if object is data type (e.g. `is.double(x)`)
  - `as.*(x)` - coerce object to data type (e.g. `as.double(x)`)
- for environment:
  - `ls()` - list all variables in environment
  - `rm(x)` - remove x variable from environment
  - `rm(list = ls())` - remove all variables from environment
- for packages:

  - `install.packages()` to install packages
  - `library()` to load the package into your current R session.
  - `data()` to load data from package into environment
  - `sessionInfo()` - version information for current R session and packages

- for help:

  - `?mean` - get help with a function
  - `help.search('mean')` - search help files for word or phrase
  - `help(package='tidyverse')` - find help for a package

Ways to get help when coding in R:

- **Read packages docs** - packages usually come with extensive documentation and examples. Reading the docs is one of the best ways to figure things out. Here is an example from the dplyr package.
- **Read error messages** - read any error messages you receive while coding — they give clues about what is going wrong!
- **Ask R** - Use R's built-in functions to get help as you code
- **Ask on Ed**- ask questions on our class discussion board!
- **Ask Google/Stack Overflow** - It is a normal and important skill (not cheating) to google things while coding and learning to code! Use keywords and package names to ensure your solutions are course-relevant.
- **Ask ChatGPT** - You can similarly use ChatGPT or other LLMs as a resource. But keep in mind they may provide a solution that is wrong or not relevant to what we are learning in this course.

4 Vectors

One of the must fundamental data structures in R is the **vector**. There are two types:

- **atomic vector** - elements of the same data type
- **list** - elements refer to any object (even complex objects or other lists)

Atomic vectors can be one of six **data types**:

- `double` - real numbers, written in decimal (0.1234) or scientific notation (1.23e4)

  - numbers are double by default (3 is stored as 3.00)
  - three special doubles: `Inf`, `-Inf`, and `NaN` (not a number)

- `integer` - integers, numbers followed by `L` (3L or 1e3L)
- `character` - strings with single or double quotes ('hello world!' or "hello world!")
- `logical` - boolean written (`TRUE` or `FALSE`) or abbreviated (`T` or `F`)
- `complex` - complex numbers where `i` is imaginary (`5 + 3i`)

- `raw` - stores raw bytes

Some more complex data structures are **built from atomic vectors** by adding **attributes**:

- `matrix` - a vector with a `dim` attribute representing 2 dimensions
- `array` - a vector with a `dim` attribute representing n dimensions
- `factor` - an integer vector with two attributes: `class="factor"` and `levels`, which defines the set of allowed values (useful for categorical data)
- `date-time` - a double vector where the value is the number of seconds since Jan 01, 1970 and a `tzone` attribute representing the time zone
- `data.frame` - a named list of vectors (of equal length) with attributes for `names` (column names), `row.names`, and `class="data.frame"` (used to represent datasets)

To create atomic vectors:

- `c(2,4,6)` - c for combine, returns `2 4 6`
- `2:4` - vector of integers, returns `2 3 4`
- `seq(2:6, by=2)` - sequence by, returns `2 4 6`

To create more complex structures:

- `list(x=c(1,2,3), y=c('a','b'))` - create a list
- `matrix(x, nrow=2, ncol=2)` - create a matrix from x with nrow and ncol
- `array(x, dim=c(2,3,2))` - create an array from x with dimensions
- `factor(x, levels=unique(x))` - turn a vector into a factor
- `data.frame(x=c(1,2,3), y=c('a','b','c'))` - create a data frame

Missing elements and empty vectors:

- `NA`- used to represent missing or unknown elements in vectors. Note that `NA` is contageous: expressions including `NA` usually return `NA`
- `NULL` - used to represent an empty or absent vector of arbitrary type. `NULL` is its own special type and always has length zero and `NULL` attributes.

5 Subsetting

> Subsetting is a natural complement to str(). While str() shows you all the pieces of any object (its structure), subsetting allows you to pull out the pieces that you're interested in. ~ Hadley Wickham, Advanced R

There are three operators for subsetting objects:

- `[` - selects multiple elements
- `[[` and `$` - *extracts* a single element

There are six ways to **select multiple elements** from vectors with `[`:

- `x[c(1,2)]` - positive integers select elements at specified indexes
- `x[-c(1,2)]` - negative integers select all but elements at specified indexes
- `x[c("name", "name2")]` select elements by name, if elements are named
- `x[]` - nothing returns the original object
- `x[0]` - zero returns a zero-length vector
- `x[c(TRUE, TRUE)]` - select elements where corresponding logical value is `TRUE`

These also apply when selecting multiple elements from **higher dimensional objects** (matrix, array, data frame), but note that:

- indexes for different dimensions are separated by commas `[rows, columns, ...]`
- omitted dimensions return all values along that dimension
- the result is simplified to the lowest possible dimensions by default
- they can also be indexed like a vector (selects columns)

There are 3 ways to **extract a single element** from any data structure:

- `[[2]]` - a single positive integer (index)
- `[['name']]` - a single string
- `x$name` - the `$` operator is a useful shorthand for `[['name']]`

When extracting single elements, note that:

- `[[` is preferred for atomic vectors for clarity (though `[` also works)
- `$` does partial matching without warning; use `options(warnPartialMatchDollar=TRUE)`
- the behavior for invalid indexes is inconsistent: sometimes you'll get an error message, and sometimes it will return `NULL`

6 Operations

Arithmetic operators:

- `+` - add
- `-` - subtract
- `*` - multiply
- `/` - divide
- `^` - exponent

Comparison operators return true or false:

- `a == b` - equal to
- `a != b` - not equal to
- `a > b` - greater than

- `a < b` - less than
- `a >= b` - greater than or equal to
- `a <= b` - less than or equal to

Logical operators combine multiple true or false statements:

- `&` - and
- `|` - or
- `!` - not
- `any()` - returns true if any element meets condition
- `all()` - returns true if all elements meet condition
- `%in%` - returns true if any element is in the following vector

Most math operations (and many functions) are **vectorized** in R:

- they can work on entire vectors, without the need for explicit loops or iteration.
- this a powerful feature that allows you to write cleaner, more efficient code
- To illustrate, suppose `x <- c(1, 2, 3)`:
  - `x + 100` returns `[101 102 103]`
  - `x == 1` returns `[TRUE FALSE FALSE]`

7 Built-in functions

Note that you do not need to memorize these built-in functions to be successful on quizzes. Use this as a reference.

For basic math:

- `log(x)` - natural log
- `exp(x)` - exponential
- `sqrt(x)` - square root
- `abs(x)` - absolute value
- `max(x)` - largest element
- `min(x)` - smallest element
- `round(x, n)` - round to n decimal places
- `signif(x, n)` - round to n significant figures
- `sum(x)` - add all elements

For stats:

- `mean(x)` - mean
- `median(x)` - median
- `sd(x)` - standard deviation
- `var(x)` - variance

- `quantile(x)` - percentage quantiles
- `rank(x)` - rank of elements
- `cor(x, y)` - correlation
- `lm(x ~ y, data=df)` - fit a linear model
- `glm(x ~ y, data=df)` - fit a generalized linear model
- `summary(x)` - get more detailed information from a fitted model
- `aov(x)` - analysis of variance

For vectors:

- `sort(x)` - return sorted vector
- `table(x)` - see counts of values in a vector
- `rev(x)` - return reversed vector
- `unique(x)` - return unique values in a vector
- `dim(x)` - transform vector into n-dimensional array

For matrices:

- `t(m)` - transpose matrix
- `m %+% n` - matrix multiplication
- `solve(m, n)` - find x in m * x = n

For data frames:

- `view(df)` - see the full data frame
- `head(df)` - see the first 6 rows of data frame
- `nrow(df)` - number of rows in a data frame
- `ncol(df)` - number of columns in a data frame
- `dim(df)` - number of columns and rows in a data frame
- `cbind(df1, df2)` - bind columns
- `rbind(df1, df2)` - bind rows

For strings:

- `paste(x, y, sep=' ')` - join multiple vectors together
- `toupper(x)` - convert to uppercase
- `tolower(x)` - convert to lowercase
- `nchar(x)` - number of characters in a string

For simple plotting:

- `plot(x)` values of x in order
- `plot(x, y)` - values of x against y
- `hist(x)` - histogram of x

8 Programming in R

Writing **functions** and handling **flow control** are important aspects of learning to program in any language. For our purposes, some general conceptual knowledge on these topics is sufficient (see below). Those interested to learn more might enjoy the book Hands-On Programming with R.

- **Functions** are reusable pieces of code that take some input, perform some task or computation, and return an output.

```
function(inputs){
    # do something
    return(output)
}
```

- **Flow control** refers to managing the order in which expressions are executed in a program:
  - `if…else` - if something is true, do this; otherwise do that
  - `for` loops - repeat code a specific number of times
  - `while` loops - repeat code as long as certain conditions are true
  - `break` - exit a loop early
  - `next` - skip to next iteration in a loop

Further reading and references

Suggested further reading:

- Base R Cheat Sheet
- Getting Started with Data in R in ModernDive textbook
- R Nuts and Bolts in R Programming for Data Science by Roger Peng

Other references:

- Matlab vs. Julia vs. Python from blog post by Toby Driscoll
- Vectors in Advanced R by Hadley Wickham
- Subsetting in Advanced R by Hadley Wickham
- A field guide to base R in R for Data Science by Hadley Wickham