R Jargon

2022-06-26

Does a pesky word keep popping up that’s given you the sneaking suspicion that you have no idea what you’re doing?

I’ve got good news, but also some bad news.

**The bad:** The person writing this probably has no idea what *they’re* doing either. And unless you study computer engineering, there’s a good chance you will never really “know” what you are “doing”.

**The good:** A quick vocabulary lesson will help do away with that sneaking suspicion, at least temporarily.

## R Vocabulary List

Search this document using ctrl+f for a brief definition of that word or symbol and external links for further reading. Can’t find the word here? Try R Documentation or Google.

### $

Used to specify a specific variable within a list-like object (like a data.frame, a tibble, a model, an actual list created with list())

Example:

# this code uses head() to view the first six observations for the variable "cyl" in dataset "mtcars"  
  
head(mtcars$cyl)  
#> [1] 6 6 4 6 8 6

### #

In a chunk of code, # is used to comment out text. This tells the R console to ignore that line. This is useful for making comments in your code.

Outside of code chunks, a series of hashes on a new line, followed by a space and some text, indicate different levels for document subheadings, like this:

In R Markdown, type:

# one hash

## two hashes

### three hashes

#### four hashes

##### five hashes

Knitting in HTML renders as:

one hash

two hashes

three hashes

four hashes

five hashes

### argument

The various pieces of data necessary for a function to run. Many functions have arguments with default values.

For example, for many statistical tests of significance, the significance level is set to a default of 0.95. If you don’t include this argument in your function, it will refer to the default and use it.

In a function’s documentation, under “Usage”, you can tell when an argument has a default because the argument’s name is equated to a value.

As another example, the function head() takes two main arguments, an object, x, and n, the number of observations you’d like for head() to print.

Go to the help documentation for head() (?head) and find what its default is. Or maybe you already have noticed its default when you’ve run head() in your labs.

### assignment operator

The <- is called the ***Assignment Operator***. We can use it to assign names to objects in our coding environment:

We can use our assignment operator for characters, numbers, logical operators, etc.:

fruit <- c("oranges", "papayas", "apricots")  
  
number <- 99  
  
logical <- FALSE

Now that the above values are stored in our environment, we can use them in other functions or operations:

paste(fruit, "are orange", sep = " ")  
#> [1] "oranges are orange" "papayas are orange" "apricots are orange"  
  
number + 1  
#> [1] 100  
  
isTRUE(logical)  
#> [1] FALSE

### base R

These are functions that are a part of the original R programming language, and so do not require a call to a package using library().

[Go here](https://stat.ethz.ch/R-manual/R-devel/library/base/html/00Index.html) to see a complete list of functions that come with the R Base Package

### code chunk

In R Markdown, code chunks look like this:

```{r}

```

Anything written between these two lines can be sent to the R Console and run as code. Anything not bound within these lines is interpreted as text, and is printed as-is in a rendered document.

### console

In lieu of running code in your R Markdown document, you can type it directly into the window that says “Console”.

When you click “Run” on any R Markdown code, that code gets run in the Console.

### debugging

The process of identifying and fixing problems in your code. A debugger is a program that walks through your code, line-by-line, allowing you to inspect elements within the environment as the code runs.

This becomes more useful when you’re writing your own functions and are confused as to why they’re behaving a certain way.

### directory

[“The working directory of a process is a directory of a hierarchical file system”](https://en.wikipedia.org/wiki/Working_directory)

A **directory** is any file folder on your computer.

Your **root directory** is the top-most directory on your computer (in Windows, this is the folder calls “C:”, for Mac users, it’s usually labelled as “Macintosh HD”)

In R, your **working directory** is typically the folder that contains whatever [R Project](https://dghi-biostat.github.io/biostatlab/docs/help_projects.html) you have open.

Say you have a dataset called “data.rds” in your main working directory. You can import it using any number of functions by referring to its file path as simply “data.rds”.

But say you have that dataset in a series of folders within your working directory. The folders are organized like this, which each subsequent folder inside the last, and the data file in the folder titled “data”:

***working directory* > try1 > fullAnalysis > data**

The “file path” for referring to your data file from your working directory would be this:

“try1/fullAnalysis/data/data.rds”

### dplyr

A package that contains a set of functions that help solve “the most common data manipulation challenges”. Main functions include:

* mutate()
* select()
* filter()
* summarize() (or summarise())
* arrange()

[Here’s the chapter](https://r4ds.had.co.nz/transform.html) from R for Data Science

You may also find [this vignette](https://dplyr.tidyverse.org/articles/dplyr.html) helpful

### factor

A data type that is the preferred way to store categorical variables in R.

Using factor(), you can convert:

* **a character to a factor**: This takes all unique values of a character variable and assigns them an underlying number, or “levels”.

For example:

dumplings <- c("momos", "pop tarts", "raviolis", "momos", "empanadas", "pierogis")  
  
factor(dumplings)  
#> [1] momos pop tarts raviolis momos empanadas pierogis   
#> Levels: empanadas momos pierogis pop tarts raviolis

* **an integer to a factor**: This takes integers and makes the lowest number the base level, and the next highest 1, the next highest 2, etc. You can assign a “label” to each respective level with “labels = c()”, with a vector, c(), with the same length as the number of levels.

integers <- c(1, 2, 3, 2, 3, 2, 1)  
  
factor(integers,  
 labels = c("chicken", "egg", "rooster"))  
#> [1] chicken egg rooster egg rooster egg chicken  
#> Levels: chicken egg rooster

Understanding factors mostly takes time. If you want to speed that up, [here’s the R for Data Science chapter on factors](https://r4ds.had.co.nz/factors.html)

In it, they reference a few articles for further reading:

* [Wrangling categorical data in R](https://peerj.com/preprints/3163/)
* [stringsAsFactors =](https://notstatschat.tumblr.com/post/124987394001/stringsasfactors-sigh)  – this one is kinda funny

### function

A “self-contained” piece of code that takes a predefined type of input data (arguments), operates on it, and returns an output or result.

### ggplot2

The “gg” in ggplot2 stands for **“grammar of graphics”**.

To get a high-level overview of ggplot2 basics, I highly recommend [this introduction](https://r4ds.had.co.nz/data-visualisation.html) to data visualization.

If you’re really trying to nerd out, you can access the free full text of [***ggplot2: Elegant Graphics for Data Analysis***](https://www.springer.com/gp/book/9780387981413) by Hadley Wickham by looking it up using [the Duke Library search engine](https://quicksearch.library.duke.edu/?q=ggplot2) and logging in with your Duke credentials

### git

You may have heard of [GitHub](https://github.com/), a popular website for version control, collaboration, and code sharing.

Git is the underlying file management software being run on GitHub. It’s free and open source. You won’t be expected to use a Git repository for your projects in this class, but it’s nice to know what’s out there.

[This massive book](https://git-scm.com/book/en/v2) is available for those who would like to learn more. I think the first and second sections, “Getting Started” and “Git Basics”, are good places to start. John Little also teaches a [helpful workshop](https://rfun.library.duke.edu/portfolio/git_workshop/) that might help you get oriented to the Git paradigm.

### HTML

Stands for HyperText Markup Language. It’s the standard language used for documents that are meant to be displayed in a web browser, and is highly customizable. When R Markdown renders to HTML, it does almost all of the heavy lifting for you.

### index

Not particularly important for Fall semester, but is an important concept to understand when working with data.

In programming, an index is a numerical representation of an item’s position in a sequence.

In R, indexes start at number 1 (as opposed to other languages that start at 0).

You can refer to an item’s index with [ ].

LETTERS gives us a character vector of every letter of the alphabet

LETTERS  
#> [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"  
#> [16] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

We can refer to individual letters by calling their index:

LETTERS[1]  
#> [1] "A"  
  
LETTERS[5]  
#> [1] "E"  
  
LETTERS[26]  
#> [1] "Z"

Data Frames and Tibbles can be indexed using syntax [row, column]. So data[1,1] would call the value in the first row of the first column, data[2,1] would call the value in the second row of the first column, and so on.

### knit

The button at the top of your R Markdown document that instructs the document to render as its designated output. The standard outputs for R Markdown are HTML, Word, and PDF. But there are an ever expanding set of R Markdown outputs available to R users. This entire website was created using a document output type called [“Distill”](https://rstudio.github.io/distill/)

### library/packages

I’ll quote from [an answer in StackOverflow](https://stackoverflow.com/questions/26900228/what-is-the-difference-between-a-library-and-a-package-in-r) for this one:

“In R, a package is a collection of R functions, data and compiled code. The location where the packages are stored is called the library.”

### magrittr/pipe/%>%

Chapter from [R for Data Science](https://r4ds.had.co.nz/pipes.html) gives a nice intro.

For you nerds, [here’s a history of the pipe operator in R](http://adolfoalvarez.cl/blog/2021-09-16-plumbers-chains-and-famous-painters-the-history-of-the-pipe-operator-in-r/)

### parameter

Used interchangeably with “argument”

### reproducible examples (reprex)

When you encounter a problem in your code that you just can’t figure out, it’s often best to create a [**reproducible example**](http://adv-r.had.co.nz/Reproducibility.html). This allows whoever is helping you to recreate your problem in their own R console.

Creating a “reprex” often entails trimming your code to the bare essentials and isolating whatever step in your code is causing it to hit an error.

It might also be the case that you need to create toy data. To do this, you can use the following tools:

* tibble() to build a dataset from vectors of equal length
* runif() to create a vector of n length of random floating point values
* seq() to create a vector of n length in a defined sequence
* sample() to draw n random samples, with replace = TRUE/FALSE, from a pre-existing vector
* x:y – use a colon to generate a series of integers from x to y
* c() with comma separated values to designate a vector

### string/character

Strings are created with either single quotes or double quotes. It indicates that a value is meant to be read as-is, rather than transformed according to R’s computational rules for numbers and logical values.

For example, writing the logical value, TRUE as a string makes it unrecognizable to R as logical:

a <- TRUE  
isTRUE(a)  
#> [1] TRUE  
  
b <- "TRUE"  
isTRUE(b)  
#> [1] FALSE

As simple as it sounds, strings are a topic of mind-numbing complexity, as the underlying encoding of text strings governs the way that code is able to interact with data as well as other code.

R for Data Science gives [a nice introduction to the mechanics of strings here](https://r4ds.had.co.nz/strings.html#strings), but hints at their wider implications in [its chapter on data importation](https://r4ds.had.co.nz/data-import.html).

Regular expressions (not to be confused with reproducible examples), or regex, are their own beast, [and may help you understand how databases and search engines work](https://www.regular-expressions.info/)

### tibble

Like a data.frame object, but with enhanced “printing”.

Read [this section on tibbles](https://r4ds.had.co.nz/tibbles.html#tibbles-vs.-data.frame) from R for Data Science to learn more.

### tidy data

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” - Hadley Wickham

The first few sections in [this chapter](https://r4ds.had.co.nz/tidy-data.html) from R for Data Science gives a nice introduction.

The basic ideas behind tidy data are defined by these three rules:

1. Each variable must have its own column
2. Each observation must have its own row
3. Each value must have its own cell

You can read more about the underlying theory in [this article](../files/TidyData.pdf) that was published in the Journal of Statistical Software.

### tidyverse

“[The tidyverse](https://www.tidyverse.org/) is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.”

Familiar packages include:

* dplyr
* ggplot2
* tibble

### traceback

If you hit an error, use traceback() to print a summary of how your program/code arrived at that error. In simple terms, it’s tracing your steps prior to your code hitting an error.

### tribble

Short for “transposed tibble”, it’s a function that allows you to create a tibble by hand, with the syntax and subsequent output:

tribble(  
 ~colA, ~colB, ~colC, ~colD,  
 "a", 1, "Square", "orange",  
 "b", 2, "Circle", "maracuya",  
 "c", 3, "Rhombus", "cashew"  
)

| colA | colB | colC | colD |
| --- | --- | --- | --- |
| a | 1 | Square | orange |
| b | 2 | Circle | maracuya |
| c | 3 | Rhombus | cashew |

?tribble in the R Console for more details

### vector

A vector is a list of values, all of the same type.

We use c() to create a vector, separating items with commas when we specify them individually.

The following are all valid vectors:

# a vector of numbers 1, 2, 3:  
c(1, 2, 3)  
#> [1] 1 2 3  
  
# a vector of numbers 1 through 12, and then 20:  
c(1:12, 20)  
#> [1] 1 2 3 4 5 6 7 8 9 10 11 12 20  
  
# a vector of logical values:  
c(TRUE, FALSE, NA, TRUE)  
#> [1] "TRUE" "FALSE" "TRUE"   
  
# a vector of 5 values randomly drawn from a uniform distribution with min 0 and max 1:  
runif(5)  
#> [1] 0.97420369 0.04387835 0.68818071 0.13420150 0.30322080

Most operations on vectors apply to each value individually:

x <- c(1:5)  
  
x + 2  
#> [1] 3 4 5 6 7  
  
x \* 2  
#> [1] 2 4 6 8 10

As such, a data frame is just a ***list of vectors*** of all the same length. That ***list*** is what’s known formally as a “recursive vector”. Lists can contain other lists.

This is a somewhat complex topic. If you’re really hungry for more info, [this chapter](https://r4ds.had.co.nz/vectors.html) in R for Data Science is highly informative, but may be confusing at first for those without any programming background.

### vignette

A vignette is a long-form guide to a package. It highlights a package’s main functions and their usage. Learning from vignettes is one of the best ways to self-teach yourself a skill in R.

### Wickham, Hadley

A kiwi and a statistician who probably authored 80% of the links on this page. He is known for the tidyverse, the book R for Data Science, and [his twitter](https://twitter.com/hadleywickham?lang=en).

### YAML header

A short blob of text at the top of your R Markdown document specifying things like the document’s title, time and date stamp, and the document’s output type.

The YAML header is part of what makes R Markdown such a flexible document. There are many ways to customize your [R Markdown output](https://bookdown.org/yihui/rmarkdown/). We won’t get into those.

Just know that at the top of your R Markdown document that says output: html\_document is what instructs your it to automatically knit as an HTML file.