R Tutorial #2 - Getting started with R [COM520]

Benjamin Mako Hill

January 2, 2021

Contents

This second screencast focuses on building basic skills with R. It can/will be far more interactive. The rest of the R Markdown script is intentionally short and is basically just an outline of the topics that will be covered. Please run these commands and experiment with R yourself in parallel as you watch/listen.

Using R as a calculator

R is a very fast calculator. You can enter simple arithmetic operations (addition, subtraction, multiplication, division, exponentiation) directly into the console or via your scripts, e.g.:

```r
2 + 2
```

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

```r
6/3
```

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

```r
10^5
```

```
## [1] 1e+05
```

Try entering some others at the console yourself.

Variables

In R, you can use variables to do many things. The basic idea is that a variable allows you to 'assign' a value or set of values to a name. You indicate assignment by typing `<-` (keyboard shortcut: 'Alt–') or `=`. Here's an example:

```r
x <- 2
x
```

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

In the first line, I assigned a value of '2' to be called 'x'. In the second line, I just type 'x', which tells R to print the value for x. Surprise, surprise, it prints '2'. (More on why it also prints `[1]` in a moment…)

Try this out yourself at the R console. Then try assigning another value to 'x' and ask R to print x again.

For the most part, you can assign any value or set of values to any variable name and you can then use the variable name instead of the value(s):

```r
cups.of.coffee <- 3
cups.of.coffee + 1
```

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

```r
cups.of.coffee*3
```

```
## [1] 9
```

Some variable names and words are 'special', however, in that R has pre-assigned values to them or pre-assigned functions. We will encounter many of these. For one example of a pre-assigned variable, try typing `pi` at the console and press 'Enter'.

One other special value a variable may take is `NA` (no quotes!) which means it is missing. If a value is missing, you may not be able to do mathematical operations with it:

```r
cups.of.coffee <- NA
cups.of.coffee-1
```

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

Types (also known as classes)

Every variable has a 'type' or 'class'. For example, we've already created a few variables which are 'numeric'. These can be whole integers or have decimals. If you ever want to know what a variable's type is, you can ask R to tell you using the `class()` function like this:

```r
class(x)
```

```
## [1] "numeric"
```

We'll come back to functions in a moment. In the meantime, other important types of variables are are 'characters' and 'logical':

```r
my.name <- "Mako"
class(my.name)
```

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

```r
my.answer <- TRUE ## Note the capitalization!
class(my.answer)
```

```
## [1] "logical"
```

It is often important to know what class a variable is because R lets you perform some operations on certain kinds of variables, but not on others.

Functions

In R, you use functions to do just about everything (e.g., inquire about the class or type of a variable as we did above). Every function takes some input (called an argument) usually in parentheses and provides some output (sometimes called the return value). Some functions take multiple inputs and return multiple outputs. You can also write your own functions and edit existing functions. This is part of what makes R so powerful and flexible.

Arguably the most important function is `help()`. The help function will retrieve the documentation for any function. To learn more about help, try entering `help(help)` at the console.

Another useful function allows you to delete a variable: `rm()` or `remove()`. Try creating a variable and removing it.

There are many built in functions. Some are common mathematical operations like `sqrt()`, `log()`, or `log1p()`. Others help you manage your workspace like `ls()`.

Check your reference card for many, many more examples.

Vectors

You can think of a vector as a set of things that are all the same type. In R, all variables are vectors even though they may have just one thing in them! That's why the R Console prints out `[1]` next to the value of a variable with just one value:

```r
my.name
```

```
## [1] "Mako"
```

You can make vectors with a special function `c()`:

```r
ages <- c(36, 50, 38)
ages
```

```
## [1] 36 50 38
```

Vectors can be of any type but they can have only one type:

```r
class(ages)
```

```
## [1] "numeric"
```

```r
painters <- c("frida", "diego", "georgia")
class(painters)
```

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

If you mix types vectors together, they will be "coerced" to a single type. The results be surprising (and sometimes annoying).

```r
class(c(ages, painters)) ## Notice that you can "nest" functions within each other.
```

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

Indexing

You can index the elements in a vector using square brackets and a number like this:

```
painters[2]
```

```
## [1] "diego"
```

You can also use indexing to refer to multiple elements in a vector

```
painters[2:3] ## A sequence of the second and third elements
```

```
## [1] "diego"   "georgia"
```

You can even assign new values to an item (or add items) in a vector using indexing:

```
ages[2] <- 52
ages
```

```
## [1] 36 52 38
```

Recycling

Mathematical operations are "recycled" when applied to a vector. R just performs the same operation on each item in the vector and gives you the output:

```
ages*2
```

```
## [1]  72 104  76
```

```
ages/2
```

```
## [1] 18 26 19
```

Naming items

You can apply a name to any item in a vector

```
names(ages)
```

```
## NULL
```

```
names(ages) <- c("Wilma", "Fred", "Barney")
names(ages)
```

```
## [1] "Wilma"  "Fred"   "Barney"
```

Now you can index into 'ages' using the name of each item:

```
ages["Barney"]
```

```
## Barney
##     38
```

Working with vectors with multiple elements

Some functions are very handy for working with vectors that have multiple elements:

```
length(ages)
```

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

```r
sum(ages)
```

```
## [1] 126
```

```r
mean(ages)
```

```
## [1] 42
```

```r
sd(ages) ## Standard deviation. More on that later.
```

```
## [1] 8.717798
```

```r
sort(ages)
```

```
##  Wilma Barney   Fred 
##     36     38     52
```

```r
range(ages)
```

```
## [1] 36 52
```

```r
summary(ages)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##      36      37      38      42      45      52
```

```r
table(ages)
```

```
## ages
## 36 38 52 
##  1  1  1
```

You can also construct new vectors by performing logical comparisons on an existing vector:

```r
ages < 39
```

```
##  Wilma   Fred Barney 
##   TRUE  FALSE   TRUE
```

```r
ages != 38
```

```
##  Wilma   Fred Barney 
##   TRUE   TRUE  FALSE
```

```r
painters == "Diego"
```

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

```r
painters == "diego"
```

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

```r
painters != "frida"
```

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

This is very useful for indexing and recoding a variable. In this case I'll use the built-in variable 'rivers' which is the lengths in miles of 141 major North American rivers (type `help(rivers)` to learn more) :

```r
rivers
```

```
##   [1]  735  320  325  392  524  450 1459  135  465  600  330  336  280  315  870
##  [16]  906  202  329  290 1000  600  505 1450  840 1243  890  350  407  286  280
##  [31]  525  720  390  250  327  230  265  850  210  630  260  230  360  730  600
##  [46]  306  390  420  291  710  340  217  281  352  259  250  470  680  570  350
##  [61]  300  560  900  625  332 2348 1171 3710 2315 2533  780  280  410  460  260
##  [76]  255  431  350  760  618  338  981 1306  500  696  605  250  411 1054  735
##  [91]  233  435  490  310  460  383  375 1270  545  445 1885  380  300  380  377
## [106]  425  276  210  800  420  350  360  538 1100 1205  314  237  610  360  540
## [121] 1038  424  310  300  444  301  268  620  215  652  900  525  246  360  529
## [136]  500  720  270  430  671 1770
```

```
head(rivers) ## 'head()' shows you the first five values of a vector
```

```
## [1] 735 320 325 392 524 450
```

```
rivers < 300 ## Recycles the comparison and returns TRUE or FALSE for each river
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [13]  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [25] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE
##  [37]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
##  [49]  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
##  [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
##  [73] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [85] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [97] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
## [121] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE
## [133]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
rivers[rivers < 300] ## A subset of the data
```

```
##  [1] 135 280 202 290 286 280 250 230 265 210 260 230 291 217 281 259 250 280 260
## [20] 255 250 233 276 210 237 268 215 246 270
```

```
little.rivers <- rivers[rivers < 300]
big.rivers <- rivers; big.rivers[big.rivers < 300] <- NA ## Two commands, one line. Recodes the short r
```

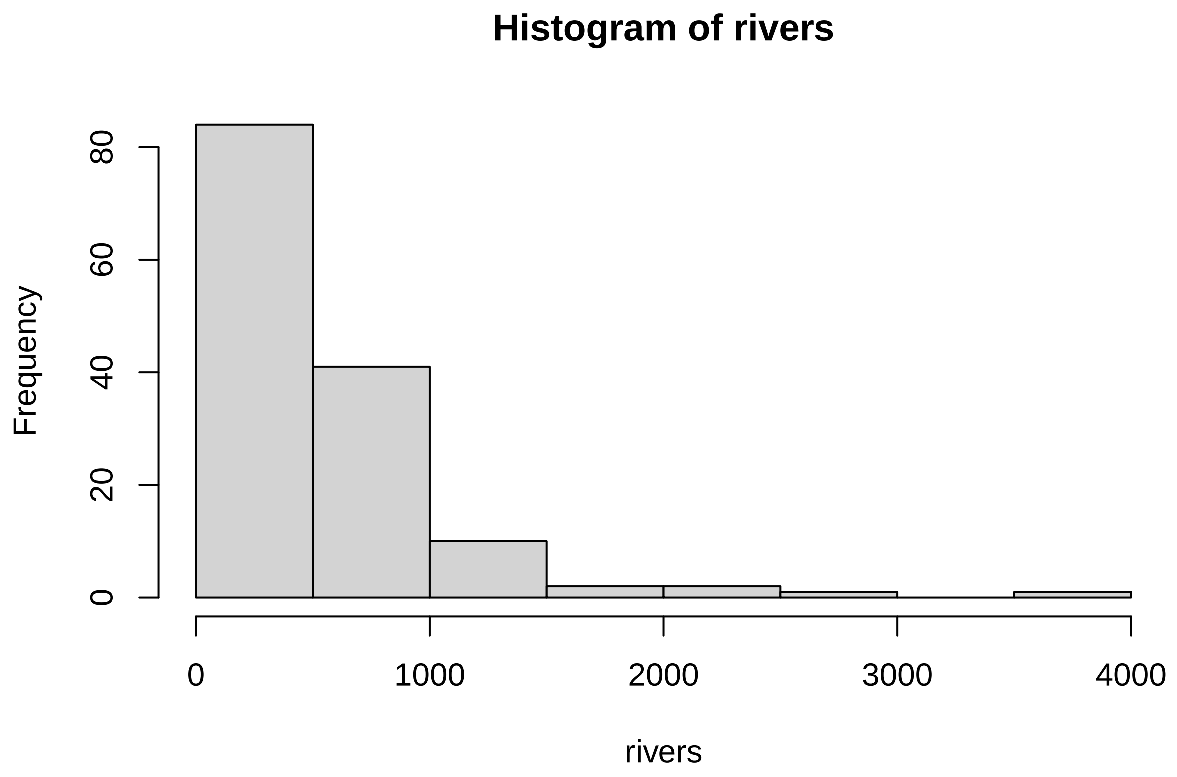
Basic plotting and visualizations

Visualizations can help you explore data and interpret results. Use them often!

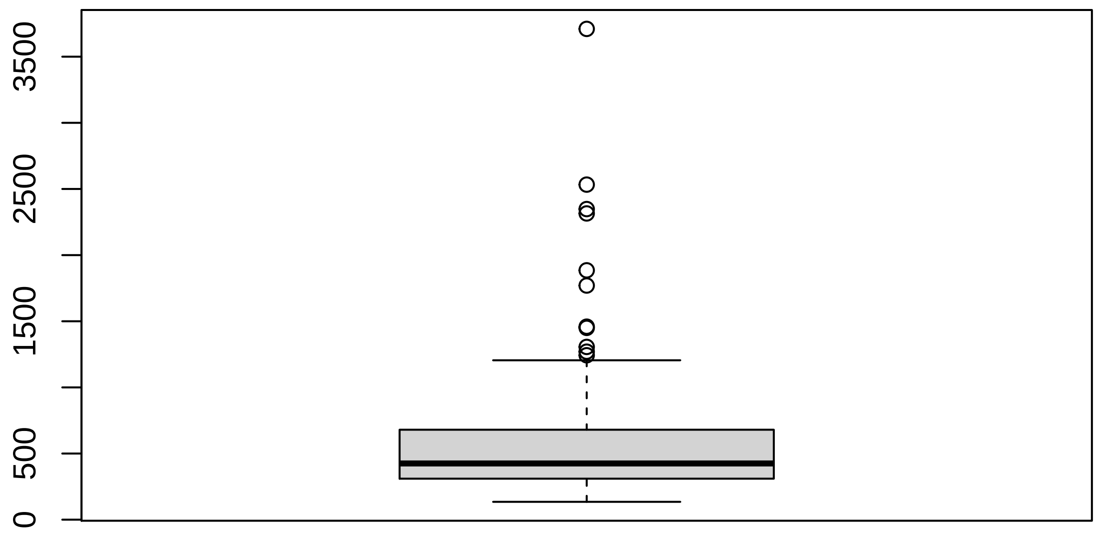
```
table(rivers>300)
```

```
## 
## FALSE  TRUE 
##    32   109
```

```
hist(rivers)
```



```
boxplot(rivers)
```



Packages

By default, R has many built-in functions and example datasets. However, many people have extended R by creating additional functions. Often these additional functions are collected together and distributed as "packages" or "libraries" that may also include additional datasets. Rstudio gives you a couple of ways to work with these. The traditional method is via the following commands

```
install.packages("UsingR") ## note the quotation marks. This package accompanies the Verzani book.
install.packages("openintro") ## This package goes along with our textbook.
```

```r
## Then you can load the package this way:
library(UsingR) ##  No quotes!
library(openintro)
```

Run these commands on your system. Use the 'Packages' tab to explore the documentation of the functions and datasets available through the `openintro` package.

Note that I have told R not to evaluate this last chunk of code because it generates a bunch of output and you only need to install any given package once. To skip evaluating a code chunk you can include the `eval=FALSE` in the header to that chunk in your R Markdown file. This tells R not to execute the code when it knits the file. Take a look at the .Rmd file to learn more.

Loading datasets

Often datasets will be located online or locally on your computer and you'll want to load them directly. For '.Rdata' files you can do this using the `load()` command. For others you may want to use commands like `read.csv()`, `read.table()`, or `read.foreign()` (that last one requires the 'foreign' package, so you'll need to load it first). RStudio also has a drop-down menu item ('File' → 'Import dataset') that can help you load a local file.

Environment and History

By default, R Studio allows you to see all the variables or 'objects' currently available to you in a particular session. Find the window/tab called "Environment" and take a look at what's there.

There's another tab (likely in the same window) called "History" that contains all the commands you have run in the current session. This can be super helpful when you're trying to piece together what you did a few moments ago or why that command you just ran worked and the one you tried a before did not.

Getting help

As mentioned earlier, the `help()` command is your friend. RStudio also has a 'Help' tab in one of the default windows. You can also use the RStudio cheatsheets, StackOverflow, the Verzani textbook, the Quick-R tutorials, and/or many, many other resources on the internet, including the rseek search engine (which just searches the web for R-related resources).