Using R and RStudio: Basic Engagement

Ashley I Naimi

Oct 2022

Contents

1 Basic Engagement with R

1.1 Running Code in R

To run a line of code in the R programming language, place your cursor at the end of a line, and press:

- COMMAND + RETURN (Mac)
- CTRL + ENTER (Windows)

```
2 * 2 * 2
```

```
## [1] 8
```

Alternatively, highlight a single or multiple lines with your cursor, and press the same keys

1.2 R as a calculator

Most basically, R is a very advanced calculator:

```
2 + 2  # add numbers
2 * pi  # multiply by a constant
3^4  # powers
runif(5)  # random number generation
sqrt(4^2)  # functions
log(10)  # natural log (i.e., base e)
log(100, base = 10)  # log base 10
23%/%2  # integer division
23%%2  # modulus operator

# scientific notation
5e+09 * 1000
5e+09 * 1000
```

More operators can be found here: Quick-R

1.3 Assigning values to R objects

R is "object oriented". A basic task in R is to assign values to objects and perform functions on them:

```
a <- 10
a
```

```
## [1] 10
```

```
a/100
```

```
## [1] 0.1
```

```
a + 10
```

```
## [1] 20
```

```
# R is case sensitive!!!
A <- 15
print(c(a, A))
```

```
## [1] 10 15
```

The left arrow assignment operator is the most common one used, but there are other ways to do it as well[1]:

[1] (https://www.roelpeters.be/
the-difference-between-arrow-and-equals-assignme
//www.roelpeters.be/
the-difference-between-arrow-and-equals-assignme

```
(x <- 3)  # Prefix notation
```

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

```
x <- 3  # Leftwards assignment
3 -> x  # Rightwards assignment
x = 3  # Equal sign
```

1.4 Vectors

```
## Basic functional unit in R is a
## vector: numeric vector
nums <- c(1.1, 3, -5.7)
nums
```

```
## [1]  1.1  3.0 -5.7
```

```
nums <- rep(nums, 2)
nums
```

```
## [1]  1.1  3.0 -5.7  1.1  3.0 -5.7
```

```
# integer vector
ints <- c(1L, 5L, -3L)  # force storage as integer not decimal number
# 'L' is for 'long integer'
# (historical)

# sample nums with replacement
new_nums <- sample(nums, 8, replace = TRUE)
new_nums
```

```
## [1] -5.7  1.1  3.0  3.0  1.1 -5.7  1.1  3.0
```

```
# logical (i.e., Boolean) vector
bools <- c(TRUE, FALSE, TRUE, FALSE, T, T,
    F, F)
bools
```

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

```
# character vector
chars <- c("epidemiology is", "the study",
```

```
    "of the", "distribution", "and determinants",
    "of disease", "in", "a population")
chars
```

```
## [1] "epidemiology is"  "the study"        "of the"           "distribution"
## [5] "and determinants" "of disease"       "in"               "a population"
```

1.5 Data Frames

Vectors can be combined into data frames (the basic data unit in R):

```
A <- data.frame(new_nums, bools, chars)
A
```

```
##   new_nums bools            chars
## 1     -5.7  TRUE  epidemiology is
## 2      1.1 FALSE        the study
## 3      3.0  TRUE           of the
## 4      3.0 FALSE     distribution
## 5      1.1  TRUE and determinants
## 6     -5.7  TRUE       of disease
## 7      1.1 FALSE               in
## 8      3.0 FALSE     a population
```

1.6 Lists

And pretty much anything (vectors, data frames) can be combined into lists:

```
basic_list <- list(rep(1:3, 5), "what do you think of R so far?",
    A)
basic_list[[1]]
```

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

```
basic_list[[2]]
```

```
## [1] "what do you think of R so far?"
```

```
head(basic_list[[3]])
```

```
##   new_nums bools             chars
## 1     -5.7  TRUE  epidemiology is
## 2      1.1 FALSE         the study
## 3      3.0  TRUE            of the
## 4      3.0 FALSE      distribution
## 5      1.1  TRUE and determinants
## 6     -5.7  TRUE        of disease
```

1.7 Subsetting

```
vals <- seq(2, 12, by = 2)
vals
```

```
## [1]  2  4  6  8 10 12
```

```
vals[3]
```

```
## [1] 6
```

```
vals[3:5]
```

```
## [1]  6  8 10
```

```
vals[c(1, 3, 6)]
```

```
## [1]  2  6 12
```

```
vals[-c(1, 3, 6)]
```

```
## [1]  4  8 10
```

```
vals[c(rep(TRUE, 3), rep(FALSE, 2), TRUE)]
```

```
## [1]  2  4  6 12
```

1.8 Subsetting Data Frames

```
A[3, ]
```

```
##   new_nums bools  chars
## 3        3  TRUE of the
```

```
A[, 3]
```

```
## [1] "epidemiology is"  "the study"        "of the"           "distribution"
## [5] "and determinants" "of disease"       "in"               "a population"
```

```
A[2:3, ]
```

```
##   new_nums bools     chars
## 2      1.1 FALSE the study
## 3      3.0  TRUE    of the
```

```
A[, 2:3]
```

```
##   bools            chars
## 1  TRUE  epidemiology is
## 2 FALSE        the study
## 3  TRUE           of the
## 4 FALSE     distribution
## 5  TRUE and determinants
```

```
## 6  TRUE       of disease
## 7 FALSE               in
## 8 FALSE     a population
```

```
subset(A, bools == F, select = -bools)
```

```
##   new_nums        chars
## 2      1.1    the study
## 4      3.0 distribution
## 7      1.1           in
## 8      3.0 a population
```

1.9 R Functions: Getting Help

```
# HELP!
`?`(median)

help.search("linear regression")

help(package = "ggplot2")
```

1.10 (Base) R Functions: Object Structure

`iris` is a flower dataset included with R. The `str()` command gives the structure of the iris dataset:

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

The `class()` command tells us what kind of object this is:

```
class(iris)
```

```
## [1] "data.frame"
```

2 R Packages

R remains cutting edge through a network of users/maintainers who contribute **packages**. Packages are functions that are not part of base R. Without these packages, R would be much less useful.

For example:

- `VIM` is a package for the VIsualisation of Missing data
- `boot` is a package to get bootstrap CIs and standard errors
- `splines` is a package for including flexible regression splines in linear models
- `data.table` is a package for fast manipulation of data frames
- The `tidyverse` is a collection of packages that facilitate the practice of "tidy" data science.

> **CRAN Packages and Development Packages**:
> The reason that R is such a useful tool for statistical analysis is that there is a large community of users and developers who contribute **packages** that can be deployed in R. Packages are tools that enable the wider community to implement statistical methods. For example, if you were interested in using quantile regression, you would install and load the `quantreg` package. If you wanted to use generalized additive models, you could install and load the `mgcv` package. For anything related to survival analysis, you would install and load the `survival` package, and so on.
> Generally, there are two places where packages are stored. The first is CRAN. To install packages from CRAN, you would simply use the code presented below (i.e., `install.packages()`). However, there are countless packages that are not on CRAN, and are considered **development** packages. These packages can be hosted anywhere, but are usually found on GitHub. There are ways to install packages from GitHub directly into R. For example, using the `install_github()` function in the `remotes` package (which can be installed from CRAN).

2.1 Installing and loading packages

Let's install the tidyverse, and some other packages that are important for basic data visualization.

If this is your first time installing packages in R, you'll have to choose a CRAN mirror. This is done with the "repos =" (repository) argument (but can be done other ways too).

```
install.packages("tidyverse", repos = "http://lib.stat.cmu.edu/R/CRAN")
```

```
##
## The downloaded binary packages are in
##  /var/folders/z_/cty0tpg97wz_x1d1zgdhwllr0000gs/T//RtmpB6XoKO/downloaded_packages
```

```
library(tidyverse)
```

You should get a warning and other messages that I excluded here.

Let's also install and load a package for the VIsualisation of Missing data:

```
install.packages("VIM", repos = "http://lib.stat.cmu.edu/R/CRAN")
```

```
##
## The downloaded binary packages are in
##  /var/folders/z_/cty0tpg97wz_x1d1zgdhwllr0000gs/T//RtmpB6XoKO/downloaded_packages
```

```
library(VIM)
```

For some projects, you will need to install and load several packages, and it may not be good practice to keep repeating the `install.packages` and `library` commands for every single package needed.[2] Instead of writing these functions over and over again, we can create a for loop that installs and loads the packages we need. For example:

[2] There is a principle in data science we refer to as DRY: Don't Repeat Yourself. When you find yourself copying and pasting code over an over again, there is usually a better solution (and that solution usually comes in the form of a loop or function).

```
packages <- c("data.table", "tidyverse",
    "here")
```

```
for (package in packages) {
    if (!require(package, character.only = T,
        quietly = T)) {
        install.packages(package, repos = "http://lib.stat.cmu.edu/R/CRAN")
    }
}

for (package in packages) {
    library(package, character.only = T)
}
```

2.2 Importing data into R

We can now use functions from the `tidyverse` and the `here` packages[3] to load our NHEFS data:

[3] We will learn a lot more about `here` in a subsequent section.

```
library(here)
nhefs <- read_csv(here("data", "nhefs.csv"))
```

```
## Rows: 1394 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (11): seqn, qsmk, sex, age, income, sbp, dbp, price71, tax71, race, wt82_71
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We can also import data directly from online:

```
nhefs <- read_csv(url("https://tinyurl.com/2s432xv6"))
```

```
## Rows: 1629 Columns: 64
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (64): seqn, qsmk, death, yrdth, modth, dadth, sbp, dbp, sex, age, race, ...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Using the `tidyverse` package (in this case, the `read_csv` function) to import data (as opposed to base R options, such as `read.csv`) creates a tibble, which is an augmented data frame.

```
class(nhefs)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

More options for importing data: R Studio Data Import Cheat Sheet

2.3 Exploring Data

Let's examine the structure of our NEHFS data:

```
dim(nhefs)
```

```
## [1] 1629   64
```

There are 1629 observations, and 64 columns in the nhefs tibble.

Let's select only specific columns from this tibble. We can do this using functions in the `dplyr` package, which is part of the `tidyvverse`:

```
nhefs <- nhefs %>%
    select(seqn, qsmk, sex, age, income,
        sbp, dbp, price71, tax71, race, wt82_71)
```

We'll learn more about the %>% (pipe) operator later. We've just re-written the nhefs object to include only the 11 variables in the `select()` function.

This is what the selected columns look like:

```
head(nhefs)
```

```
## # A tibble: 6 x 11
```

```
##    seqn  qsmk   sex   age income   sbp   dbp price71 tax71  race wt82_71
##   <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl>   <dbl>
## 1   233     0     0    42     19   175    96    2.18 1.10      1   -10.1
## 2   235     0     0    36     18   123    80    2.35 1.36      0    2.60
## 3   244     0     1    56     15   115    75    1.57 0.551     1    9.41
## 4   245     0     0    68     15   148    78    1.51 0.525     1    4.99
## 5   252     0     0    40     18   118    77    2.35 1.36      0    4.99
## 6   257     0     1    43     11   141    83    2.21 1.15      1    4.42
```

```
# can also use 'tail' to see the end of
# the file tail(nhefs)
```

2.4 Functions and for loops

Functions are pieces of code written to accomplish specific tasks. Suppose we wanted to evaluate the proportion of missing data in each column in `nhefs`. We could do this by writing a function:

```
propMissing <- function(x) {
    mean(is.na(x))
}
propMissing(nhefs[, 1])
```

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

```
propMissing(nhefs[, 2])
```

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

In the above code, `mean()` takes the sample average. In R, missing values are coded as `NA`, and `is.na()` is a base R function that returns a Boolean (true/false) value for each element in `x` that is missing. Thus, `mean(is.na(x))` returns the proportion of `x` that is missing.

Instead of copying and pasting the function over and over, we can put it in a `for` loop:

```r
for (i in 1:ncol(nhefs)) {
    output <- propMissing(nhefs[, i])
    print(output)
}
```

```
## [1] 0
## [1] 0
## [1] 0
## [1] 0
## [1] 0.03806016
## [1] 0.04726826
## [1] 0.04972376
## [1] 0.05647637
## [1] 0.05647637
## [1] 0
## [1] 0.03867403
```

Instead of a `for` loop, we can use the `apply` family of functions, which presents things in a way that is more informative. For example:

```r
apply(nhefs, 2, propMissing)
```

```
##       seqn       qsmk        sex        age     income        sbp        dbp
## 0.00000000 0.00000000 0.00000000 0.00000000 0.03806016 0.04726826 0.04972376
##    price71      tax71       race    wt82_71
## 0.05647637 0.05647637 0.00000000 0.03867403
```

More information on the apply family: Apply tutorial

We can also make the above much more presentable and easier to read:

```r
round(apply(nhefs, 2, propMissing), 3) *
    100
```

```
##    seqn    qsmk     sex     age  income     sbp     dbp price71   tax71    race
##     0.0     0.0     0.0     0.0     3.8     4.7     5.0     5.6     5.6     0.0
## wt82_71
##     3.9
```

2.5 R & RStudio: Diving Deeper

Resources for further learning in R / Rstudio are endless:

- Chris Paciorek (UC Berkeley Bootcamp on youtube)
- R for Data Science (e-book)
- swirl
- Udacity Data Analysis with R
- Roger Peng's Coursera (advanced)
- r-bloggers