OSOS - Introduction to R

Nicole Stevens (Modifed from Danielle Walkup and STHDA)

Last compiled on 23 August, 2024, 10:58

Introduction to R and RStudio

Welcome to R! Today you will be learning some of the basics of using both R and RStudio, which will both have to be downloaded before progressing with this module. R is the software with the programming language itself, and RStudio is the user interface which makes programming more accessible.

Both R and RStudio can be downloaded here: https://posit.co/download/rstudio-desktop/

The RStudio Interface

1. Source (upper left): this is where you will write your code, in scripts (and any other file type you want). The scripts are a reusable record of your code that you can edit, check, and re-run.
2. Console (bottom left): runs the code and displays the results. This should be used to test code, run examples, and see results – your whole analysis shouldn't be run through here. Eventually you run out of display room!
3. Workspace/History (upper right): shows the objects you have created as well as the list of commands run.
4. Files/Plots/Packages/Help (bottom right):
    - Files – shows all the files in your working directory
    - Plots – shows the plots you have created during the session. Can also export and save them through this window
    - Packages – lists all the packages that come with R, as well as any packages you have installed
    - Help – lets you search R information

These windows can be re-arranged by clicking the squares to the left of Addins -> Pane Layout. The formatting (text size, text font, text color, and background color) can be changed using Tools -> Global Options -> Appearance.

Basic R Usage

You can type commands directly into the console at the bottom of your screen. This is a great way to get fast results, but your commands will not be easily accessible.

To save your work, you can create an R script by clicking on the icon that looks like a white square in the top left of your screen then click "R script." A new tab will open in your source screen where you can type your code. Note that your code will not run unless you tap the "Run" button or hit Control (or command for Mac) + Enter

Anything that you don't want R to run (e.g., notes) can be typed out after a #

Any scripts you make during this course should be saved to your working directory (see below)

The Working Directory

Your working directory is the folder on your computer where your R code is running. Any files you need to run your code should be in or accessible from your working directory (see Module 2). Any files you write will be in your working directory by default (often your documents folder)

```
## To check your current directory:
getwd()

## If you want to see the files currently in the working directory:
dir()

## If you need to change the working directory:
setwd("Path-To-The-Directory")

# the easiest way to do this is to open the folder and copy and paste the file
# path into the quotes.
## Note that on a pc, you need to use `/` in the path, `\` is used as an escape
#character in R, so if you copy-paste, make sure you change `\` to `/`.

## Let's double check it worked:
getwd()
```

If you open an R script from somewhere else in your computer (without R/RStudio being open), the working directory defaults to the folder where your R script was stored.

If setwd() just isn't working and you need to get on with your life, you can use the RStudio menu: Session -> Set Working Directory -> Choose Directory (OR ctrl-shift-H).

R as a Calculator

The R console can be used as a basic calculator where you can do basic addition, subtraction, multiplication, division, and exponents. Commands will follow the standard order of operations (PEMDAS)

Examples:

```
2+2
```

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

```
3*5
```

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

```
10/2 + 3
```

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

```
2^5 - 10
```

```
## [1] 22
```

Functions in R are written as the function_name(). Basic functions are built into R including:

Logarithms and exponentials: log(x) (natural log), log2(x), log10(x), exp(x)

```
log10(100)
```

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

Trigonometric functions: cos(x), sin(x), tan(x), acos(x), asin(x), atan(x)

```
tan(45)
```

```
## [1] 1.619775
```

Logical operators: >, <, =, <=, >=, ==, !=

```
10 > 5
```

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

```
10 < 5
```

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

```
10 == 10
```

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

Constants: pi, e

```
pi^2
```

```
## [1] 9.869604
```

Other mathematical functions: abs(x): absolute value; sqrt(x): square root.

```
sqrt(25)
```

```
## [1] 5
```

```
abs(-4)
```

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

If you ever have a question about what a function does, you can use the help() function and R will provide additional information.

```
help(abs)
```

Numeric Variables

Numeric variables contain numbers and can include decimals. You can save a variable by writing the name you want to call it (e.g., "a") followed by the operator "<-" and the number you'd like to assign to that variable.

Example:

```
a <- 2
alpha <- 3
```

To verify that they are numeric variables, you can test them with the class() function:

```
class(a)
```

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

```
class(alpha)
```

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

These variables will now show up in your Environment (upper right window) with their values. Now if you type alpha into your console, you will get its value, 3.

```
alpha
```

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

You can do some of the basic math we did before with these variables and even save that value as a new variable:

```
a+alpha
```

```
## [1] 5
```

```
sum_a_alpha<-a+alpha
sum_a_alpha
```

```
## [1] 5
```

Numeric Vectors and Functions

If you need to represent multiple variables at once, you can create a vector. The method to do so is similar to creating a simple numeric variable, except you put all objects in the vector within the c(), or concatenate function.

Example:

```
vector <- c(4,6,3,6,7,3,4,6,7,2)
```

Just like before, this variable will show up in your Environment. However, instead of just a single value, you can see all of the numbers. You can also see that it is a numeric variable (indicated by "num") and that it has 10 numbers ("[1:10]").

To see your vector in your console, type its name:

```
vector
```

```
##  [1] 4 6 3 6 7 3 4 6 7 2
```

You can also apply some of R's additional built-in functions to your new vector.

For example:

```
mean(vector) #Gives the average of values in the vector
```

```
## [1] 4.8
```

```
min(vector) #Gives smallest value
```

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

```
max(vector) #Gives maximum value
```

```
## [1] 7
```

```
median(vector)
```

```
## [1] 5
```

```
sd(vector) #Gives standard deviation
```

```
## [1] 1.813529
```

The summary() function will also provide some of this data:

```
summary(vector)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.25    5.00    4.80    6.00    7.00
```

Other useful functions to apply to your vector:

```
length(vector)  #shows how many objects are in the vector
```

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

```
head(vector)    #by default returns the 1st 6 things in the object
```

```
## [1] 4 6 3 6 7 3
```

```
head(vector, 2) #but we can tell it how many we actually want (> or < 6)
```

```
## [1] 4 6
```

```
tail(vector, 3)
```

```
## [1] 6 7 2
```

Other Variables Types in R

1. INTEGER: similar to numeric, but cannot contain decimals. For example:

```
alpha_int <- as.integer(alpha) #converting alpha from numeric to integer
alpha_int
```

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

```
class(alpha_int) #function to display class type
```

```
## [1] "integer"
```

2. CHARACTER: variables made of text (also known as strings). For example:

```
course_name <- "Open Source for Open Science"
course_name
```

```
## [1] "Open Source for Open Science"
```

```
course_topics <- c("R introduction", "calculations", "variables")
course_topics
```

```
## [1] "R introduction" "calculations"   "variables"
```

```
class(course_topics) #shows class type
```

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

Note that anything that you put into quotations will be considered text, regardless of what it is, and that spaces matter.

For example, numbers can be converted to strings:

```
vector_char<- as.character(vector)
vector_char #look how this appears differently in your environment than the original vector
```

```
##  [1] "4" "6" "3" "6" "7" "3" "4" "6" "7" "2"
```

```
a_char <- as.character(a)
alpha_char <- as.character(alpha)
#NOTE you will not be able to do numeric operations between a_char and alpha_char
#since they are now text and not numbers (e.g., a_char*alpha_char will give you an error)
```

If at least one item in a vector is a string, the whole thing will be a character:

```
char_num <- c("text", 1, 3.72, 4)
char_num
```

```
## [1] "text" "1"    "3.72" "4"
```

```
class(char_num)
```

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

Spaces can change variable meaning (not true for numerical data):

```
char_space <- "text "
char_nospace <- "text"
char_space == char_nospace # is char_space equal to char_nospace? NO
```

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

3. FACTOR: variables made of text that represents given categories. For example:

```
survey_responses <- factor(c("Agree", "Agree", "Disagree", "Neutral", "Disagree"))
survey_responses
```

```
## [1] Agree    Agree    Disagree Neutral  Disagree
## Levels: Agree Disagree Neutral
```

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock"))
music_genre
```

```
## [1] Jazz    Rock    Classic Classic Pop     Jazz    Rock
## Levels: Classic Jazz Pop Rock
```

To see the different levels represented within a factor, can use levels()

```
levels(survey_responses)
```

```
## [1] "Agree"    "Disagree" "Neutral"
```

```
levels(music_genre)
```

```
## [1] "Classic" "Jazz"    "Pop"     "Rock"
```

Character vectors can be converted to factors:

```
text <- c("test1", "test2", "test1", "test1") # create a character vector
class(text)
```

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

```
text_factor <- as.factor(text) # transform to factor
class(text_factor)
```

```
## [1] "factor"
```

4. LOGICAL: variables that can either be TRUE or FALSE. For example:

```
#is a greater than alpha?
greater <- a > alpha
greater
```

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

```
class(greater)
```

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

Logical variables can be converted to numerical variables, with FALSE equal to 0 and TRUE equal to 1

```
greater_num <- as.numeric(greater)
greater_num
```

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

Installing and Loading R Packages

Many R packages have been created to supply users with additional functions. When you first download R, you also download some of the base packages, but there are thousands more that exist! If you tried to download all of them, it would take up a LOT of space, so it is best to just download what you need. Available R packages can be used for things like spatial analyses, phylogenetic analyses, multivariate statistics, and so much more.

Many of the available R packages can be viewed by clicking on the "Packages" tab in the bottom right screen of RStudio. Those that you have in your system library will have a black checkmark (note the mark next to the "base" package).You can install additional packages by clicking the "Install" button in the top left of the window and typing in your desired package name. You can also use the install.packages() function. For example:

```
install.packages("readxl") #package allows you to read in data from Excel spreadsheets
```

Notice how you now have a black checkmark next to this package.

When you first open R and want to use a downloaded package, you need to first add it to your library or else you will not be able to use its functions. Whenever you start a new script, it is always a good idea to add relevant packages to your library to limit error messages. You will use the library() function a lot during OSOS.

```
library(readxl)
```

Happy coding!