EPID674 Epidemiologic Data Analysis using R

Getting Started with R

Kelly Bakulski, Lauren Middleton

Last compiled on October 18, 2022

## Programming language: R

This course will introduce the R statistical programming language for epidemiologic analyses. R statistical software is a freely available, versatile, and powerful program for statistical computing and graphics (<https://www.r-project.org/>). A helpful interface for R is provided by RStudio (<http://www.rstudio.com/>). For a shared educational environment in this class, we will use the online version of R and RStudio called RStudio Cloud (<https://rstudio.cloud/>).

## Authoring Software: Quarto

This is a quarto markdown document. Markdown is a simple formatting programming language for authoring HTML, PDF, and Microsoft Word documents. For more details on using quarto Markdown see <https://quarto.org/docs/authoring/markdown-basics.html>.

When you click the **-->Render** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

# Getting Started with R

## Explaining commands/output

An R command (we’ll also call it code or a code chunk) will be grey and look like this. The text in this section will be talking to R.

print("I'm code")

[1] "I'm code"

Directly after the code chunk will be the output of the code. The code chunk is print("I'm code") and the output is [1] “I’m code”.

## Make a new code chunk

You can embed an R code chunk like this: Press the green +C Insert button at the top of this window next to the Run button

## R as a calculator

The R console is a full calculator. Try to play around with it: + is add, - is subtract, / is divide, \* is multiply. ^ or \*\* is power parentheses ( ) work with order of operations, meaning the contents of the inner pair will be calculated first.

2 + 2

[1] 4

2 \* 4

[1] 8

2 ^ 3

[1] 8

2 + (2 \* 3)^2

[1] 38

(1 + 3) / 2 + 45

[1] 47

Note, when you type your command, R inherently thinks you want to print the result.

## Test your comprehension:

Make an R code chunk and try evaluating the following:  
2 plus 2 times 3 divided by 4 minus 3 2 times 3 divided by 4 times 2 2 to the 4th power minus 1

## Make your first R objects

* You can create objects (variables in your dataset) within the R environment and from files on your computer
* R uses “<-”or “=” to assign values to a variable name
* Variable names are case-sensitive, i.e. A and a are different
* Name variables with descriptive names so you remember what they are for

# num\_apples is assigned the value of 5, i.e. there are 5 apples  
num\_apples <- 5  
# See what num\_apples is:  
num\_apples

[1] 5

# The cost of one apple is $0.55  
cost\_apple <- 0.55  
  
# Tax is 7%  
tax <- 0.07  
  
# What is the total cost of the apples without tax? Total cost is assigned to variable: cost\_no\_tax  
cost\_no\_tax <- num\_apples \* cost\_apple  
cost\_no\_tax

[1] 2.75

# What is the total cost with tax?  
cost\_no\_tax \* (1 + tax)

[1] 2.9425

# Assign a vector of integers from 1 to 4 to the object num\_apples, using multiple different functions

* Here we introduce “1 dimensional” object classes; referred to as ‘vectors’
* The function c() collects/combines/joins single R objects into a vector of R objects. It is mostly used for creating vectors of numbers, character strings, and other data types.
* The function length(): Get or set the length of vectors (including lists) and factors, and of any other R object for which a method has been defined.

length(num\_apples) # length is a function for checking how long an object is

[1] 1

# This writes over the previous num\_apples object. No warning or error messages!  
num\_apples <- 1:4 # Create a vector of numbers 1 to 4 by 1  
num\_apples

[1] 1 2 3 4

num\_apples <- c(1, 2, 3, 4) # Combine elements into a vector  
num\_apples

[1] 1 2 3 4

num\_apples <- c(2, 3, 4, 1) # Order matters  
num\_apples

[1] 2 3 4 1

num\_apples <- seq(from = 1, to = 4, by = 1) # Create a vector counting from 1 to 4 by 1  
num\_apples

[1] 1 2 3 4

length(num\_apples) # How long is the vector now?

[1] 4

## Perform some calculations with “num\_apples” and observe the results

You can perform functions on vectors of numbers very easily

num\_apples + 4

[1] 5 6 7 8

num\_apples - 3

[1] -2 -1 0 1

num\_apples \* 7

[1] 7 14 21 28

num\_apples / 10

[1] 0.1 0.2 0.3 0.4

num\_apples \* num\_apples

[1] 1 4 9 16

num\_apples^2

[1] 1 4 9 16

num\_apples / num\_apples

[1] 1 1 1 1

num\_apples == num\_apples

[1] TRUE TRUE TRUE TRUE

## Make a character vector

Objects can contain character information (words, letters, or combinations). Each character element should be surrounded by quotes ” ”

#|: label: make-character-vector  
  
fruit\_farms <- c("Tantre Farm",  
 "Kapnick Orchards",  
 "Wasem Fruit Farm",  
 "Erie Orchard and Cider Mill")  
fruit\_farms

[1] "Tantre Farm" "Kapnick Orchards"   
[3] "Wasem Fruit Farm" "Erie Orchard and Cider Mill"

short\_names <- c("Tantre",  
 "Kapnick",  
 "Wasem",  
 "Erie")  
short\_names

[1] "Tantre" "Kapnick" "Wasem" "Erie"

## R object classes

* Vectors can have multiple observations, but each observation has to be the same class.
* The functions class() and str() can help you learn about the class and structure of your objects

class(num\_apples)

[1] "numeric"

class(fruit\_farms)

[1] "character"

# Can get more information using str()  
str(fruit\_farms)

chr [1:4] "Tantre Farm" "Kapnick Orchards" "Wasem Fruit Farm" ...

# chr is the data type (character)  
# [1:4] is the length (the vector has 4 elements)  
# "Tantre Farm"... is a snapshot of the elements in the vector

## What are the differences between ( ) and [ ] ?

length(fruit\_farms) # Rounded parentheses are for functions

[1] 4

fruit\_farms[3] # Square brackets are for looking in objects, this is also called "indexing"

[1] "Wasem Fruit Farm"

fruit\_farms[2] # Find value based on position

[1] "Kapnick Orchards"

fruit\_farms[1:3]

[1] "Tantre Farm" "Kapnick Orchards" "Wasem Fruit Farm"

# Find positions meeting criteria  
num\_apples < 4 # Provides True/False for whether meets the criteria

[1] TRUE TRUE TRUE FALSE

which(num\_apples < 4) # Finds the positions of the Trues

[1] 1 2 3

# Find values meeting criteria  
num\_apples[num\_apples < 4]

[1] 1 2 3

# Make your first data frame

* The most comfortable and familiar class/data type for many of you will be data.frame
* You can think of these as essentially Excel spreadsheets with rows (usually subjects or observations) and columns (usually variables)

# Combine vectors by column into a data frame and assign it to an object called "apple\_farms"  
apple\_farms <- data.frame(num\_apples, short\_names, fruit\_farms)  
apple\_farms # Look at apple\_farms

num\_apples short\_names fruit\_farms  
1 1 Tantre Tantre Farm  
2 2 Kapnick Kapnick Orchards  
3 3 Wasem Wasem Fruit Farm  
4 4 Erie Erie Orchard and Cider Mill

# Get information about apple\_farms  
str(apple\_farms)

'data.frame': 4 obs. of 3 variables:  
 $ num\_apples : num 1 2 3 4  
 $ short\_names: chr "Tantre" "Kapnick" "Wasem" "Erie"  
 $ fruit\_farms: chr "Tantre Farm" "Kapnick Orchards" "Wasem Fruit Farm" "Erie Orchard and Cider Mill"

# Use $ to call up columns within data frames  
apple\_farms$num\_apples

[1] 1 2 3 4

apple\_farms$fruit\_farms

[1] "Tantre Farm" "Kapnick Orchards"   
[3] "Wasem Fruit Farm" "Erie Orchard and Cider Mill"

# Index (look around) inside the data frame based on the position

#Index based on position  
apple\_farms[3, 1] # structure: apple\_farms[rows,columns]

[1] 3

apple\_farms[4, ]

num\_apples short\_names fruit\_farms  
4 4 Erie Erie Orchard and Cider Mill

apple\_farms[ , 1]

[1] 1 2 3 4

apple\_farms[c(1, 2, 3), ] # Read your R code from the inside out. Start with the innermost set of parentheses.

num\_apples short\_names fruit\_farms  
1 1 Tantre Tantre Farm  
2 2 Kapnick Kapnick Orchards  
3 3 Wasem Wasem Fruit Farm

apple\_farms[c(FALSE, TRUE, TRUE, FALSE), ]

num\_apples short\_names fruit\_farms  
2 2 Kapnick Kapnick Orchards  
3 3 Wasem Wasem Fruit Farm

#Index based on value  
apple\_farms$num\_apples > 2 #Find a logical vector (True/False) of the rows that meet the value of interest (in this case num\_apples > 2)

[1] FALSE FALSE TRUE TRUE

apple\_farms[apple\_farms$num\_apples > 2,] #Now show the rows that are True

num\_apples short\_names fruit\_farms  
3 3 Wasem Wasem Fruit Farm  
4 4 Erie Erie Orchard and Cider Mill

## Recode a value in the data frame based on the position

apple\_farms

num\_apples short\_names fruit\_farms  
1 1 Tantre Tantre Farm  
2 2 Kapnick Kapnick Orchards  
3 3 Wasem Wasem Fruit Farm  
4 4 Erie Erie Orchard and Cider Mill

apple\_farms[3, 1] <- 5 # recode a value  
# What do you expect?  
  
# What do you get?  
apple\_farms

num\_apples short\_names fruit\_farms  
1 1 Tantre Tantre Farm  
2 2 Kapnick Kapnick Orchards  
3 5 Wasem Wasem Fruit Farm  
4 4 Erie Erie Orchard and Cider Mill

# Do they match?

## Specify options in a function

seq(from = 10, to = 23, by = 1) # count from 10 to 23 by 1

[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23

seq(from = 10, to = 23) # equivalent

[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23

seq(10, 23) # equivalent

[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23

seq(10, -3) # count from 10 to -3

[1] 10 9 8 7 6 5 4 3 2 1 0 -1 -2 -3

seq(from = -1, to = 11, by = 3) # count from -1 to 11 by 3

[1] -1 2 5 8 11

## Perform calculations on the data frame

sum(apple\_farms$num\_apples)

[1] 12

mean(apple\_farms$num\_apples)

[1] 3

## Pipes!

* Important part of coding with tidyverse using this set of symbols %>%

# Reminder of values in num\_apples  
num\_apples

[1] 1 2 3 4

# This is a pipe: %>%  
# This pipe is loaded from the tidyverse package  
library(tidyverse)

── Attaching packages ─────────────────────────────────────── tidyverse 1.3.2 ──  
✔ ggplot2 3.3.6 ✔ purrr 0.3.5   
✔ tibble 3.1.8 ✔ dplyr 1.0.10  
✔ tidyr 1.2.1 ✔ stringr 1.4.1   
✔ readr 2.1.3 ✔ forcats 0.5.2   
── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
✖ dplyr::filter() masks stats::filter()  
✖ dplyr::lag() masks stats::lag()

# Pipes are used to link objects to function(s)  
num\_apples %>%  
 sum()

[1] 10

num\_apples %>%  
 mean() %>%  
 length()

[1] 1

# Can store the results in a new object  
median\_apples <- num\_apples %>%  
 median()

## Commenting in Scripts

# is the comment symbol in R

# Comments in R follow the hashtag symbol  
# Anything from the # to the end of the line (on the right) will be ignored by R. Note the color of your code changes after you use a hashtag  
  
# This # is still a comment  
### You can use many #'s as you want  
1 + 2 # Can be the right of code

[1] 3

# Best practice is to take a ton of notes to help your future self and anyone who comes later to re-run your code

## White Space in Scripts

# Use lots of white space - it makes your code more readable for yourself and others!  
  
  
# Examples:  
  
#1)  
# Spaces after operators  
2 + 2

[1] 4

#2)  
# Spaces after commas  
paste("Hello,", "welcome", "to", "class")

[1] "Hello, welcome to class"

#compared to:  
paste("Hello,","welcome","to","class")

[1] "Hello, welcome to class"

#3)  
# New lines  
fruits <- c("apple",  
 "pear",  
 "orange",  
 "cherry",  
 "grape",  
 "strawberry",  
 "lime")  
# compared to:  
fruits <- c("apple","pear","orange","cherry","grape","strawberry","lime")

## Use the help viewer

Any time I use a new function, I navigate to the lower right panel and search for the function. This describes the purpose of the function, the default settings, and the options you can change.

# Another option is to use the 'help' function to search. Look in the lower right panel and the same view will appear.  
help(class)

## Test your comprehension:

You want to create a new numeric vector that repeats the number 7 in the first 4 positions 7 7 7 7 To do this you can use the rep() function. Make a new code chunk. Look up the rep function in the help viewer. Identify the option for the number of times something is repeated. Create this vector and store it as a new object.

## Common new R users frustrations

1. Different versions of software
   * RStudio Cloud solves this
2. Working directory problems: trying to read files that R “can’t find”
   * RStudio Cloud solves this and so does RStudio Projects
3. Data type problems (is that a character or a numeric object?)
   * discussed throughout
4. Typos (R is **case sensitive**, x and X are different)
   * RStudio helps with “tab completion”
   * discussed throughout
5. R often does not include any error/warning messages. Need to train yourself in the following sequence:
   1. What do I expect?
   2. What do I get?
   3. Do they match?

# Remember to save your R script!

If the name of the .Rmd at the top of the code editor is in red font with an asterisk at the end, you have unsaved changes.

# To exit R

q()

Click the **Knit** button at the top of this script to run all of the code together and generate a markdown report!

## Review

* Creating a new script
* Using R as a calculator
* Assigning values to objects (vectors and data frames)
* Performing algebra on numeric variables