Worksheet 1 R basics CBES 677

Natalie Graham

2025-08-01

Getting started in R

In this class you are not required to have R and R studio on your computer. Instead, you can access R through Posit Workbench in the cloud here.

Whether you are using the Posit Workbench or your own copy of R and R studio *please read* how to make an R markdown and knit it into a PDF document in Google Drive here.

```r
library(tinytex)
```

How to download both R and R Studio to your computer.

First download R for your system at https://cran.rstudio.com/ To download R, you now need to choose a "CRAN Mirror". This is one of many locations across the world that host R- I suggest choosing a "CRAN mirror" from an American University. Once there, choose the R download option that is best for your operating system, and follow the instructions!

Second download RStudio

Download for R studio is here. Again follow the simple download instructions for your operating system.

If there are prompts during the download process asking if you want to allow additional packages, click "yes". This whole process shouldn't take more than 10-15 minutes. R-Studio should then show up in your apps or as an icon somewhere on your computer. From here on out, you can just click on R Studio to do your work, but it only works if you also have R on your computer.

There are four windows (sometimes three) in RStudio. The bottom left window is the command (console) window- this is where you can type commands and run code. The top left window is the editor (or script) window. That is the one that shows your imported datasets and is also a place where you can edit and save scripts. Sometimes the editor window isn't showing, in which case you can go up to the RStudio menu and click on "File-> New File-> R script". The top right window is the workspace window where you can see what R has in its memory and how it is viewing the data. It is also an easy way to see and save your past commands. The bottom right window is mainly for viewing plots and viewing and installing packages.

Some R Definitions:

Object: just about everything on the screen is an object in R

Vector: generally a one dimensional group of numbers, usually in a column

Function: A set of instructions to be carried out on an object (which is usually in parentheses), eg. mean()

Argument: The info that the function works on. Basically the data or expression inside the () following the function.

Operator: symbols with pre-defined meaning, such as +, -, /, and *.

Expression: a command that R evaluates and produces output for (example 2+2)

The command line

The prompt (>) is where you enter commands in the command window. The continuation prompt (+) at the beginning of a line means that R is waiting for more input before it can carry out a command. A good habit to get into is to annotate your commands with #. R will ignore anything following a # that is on a line.

Try out some expressions (use R as a calculator): *Anything in the gray 'code chunk' can be run by clicking the green arrow to the right in the code chunk box. Also, check out the options in the Run dropdown menu top right.*

```r
# Any text that is a comment for your code must have a # to remain inside the code chunk.

3*2
```

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

```r
2^5
```

```
## [1] 32
```

```r
48/3
```

```
## [1] 16
```

```r
log(32)
```

```
## [1] 3.465736
```

- Now repeat the same operations by copy / paste into the Console Window after the > prompt

Assignment

This is VERY commonly used and will be important for model building later in the semester. Use the assignment operator <- to assign a name to an object on the left side of the <- that is the result or output of the expression on the right side.

Variable1 <- log(32)

```r
Variable1 <- log(32) # no spaces in variable1
```

Note that you can not have spaces in your variable names and variable names cannot begin with numbers. Use "snakeCase" or "under_score" to make clearer variable names. Variable naming is an art. Do not just call your variables "one" "two" etc because you won't recall them!

Look for your object in the Environment Console. It is saved as a value. Different objects have different properties. The Environment is a good place to find what you have stored in the local memory for your R project. Also try typing list ls() and list structure ls.str() into the Console. What do you see?

Now, lets play with some data. We will first create a vector of numbers, then a matrix, followed by the dataframe. A matrix is basically just more than one vector of the same length bound together. A dataframe is a matrix in which the values for a given row all correspond to the same "individual" whereas that isn't necessarily the case with a matrix. Also, dataframes can consist of both factors and continuous variables.

Factor: A factor in R is a categorical variable that can take on a limited number of unique values, known as levels. Factors are often used to represent categorical data, such as gender, species, or treatment groups.

The concatenate function c() binds together a group of numbers into a vector

```r
ohia.diam<-c(5,8,12,15,20)
# now lets check the vector ohia.diam by "calling" that variable
ohia.diam
```

```
## [1]  5  8 12 15 20
```

```r
#lets create another vector of temperature
temp<-c(30,25,20,15,10)
temp
```

```
## [1] 30 25 20 15 10
```

```r
# the cbind() function binds two or more vectors into a matrix
cbind(ohia.diam, temp) # note that we haven't saved it yet bc it wasnt assigned as a object
```

```
##      ohia.diam temp
## [1,]         5   30
## [2,]         8   25
## [3,]        12   20
## [4,]        15   15
## [5,]        20   10
```

```r
# we can now create an object called ohia.diam.temp (or whatever we like), which is the matrix, for use
ohia.diam.temp<-cbind(ohia.diam,temp)
ohia.diam.temp
```

```
##      ohia.diam temp
## [1,]         5   30
## [2,]         8   25
## [3,]        12   20
## [4,]        15   15
## [5,]        20   10
```

Make the matrix ohia.diam.temp a dataframe as follows- first assign a name to the dataframe, then use the dataframe function

```
df1<-data.frame(ohia.diam.temp)
df1
```

```
##   ohia.diam temp
## 1         5   30
## 2         8   25
## 3        12   20
## 4        15   15
## 5        20   10
```

```
#now, when a vector is part of a dataframe, any operation that is done to the vector occurs separately 
mean(df1$ohia.diam)
```

```
## [1] 12
```

```
# you can now remove the original vector (outside of the dataframe) to avoid confusion and a buildup of

rm(ohia.diam.temp)

# Another way to see a list of objects on the workspace:
objects()
```

```
## [1] "df1"       "ohia.diam" "temp"      "Variable1"
```

```
#Again, the above can also be found by clicking on the "Environment" tab in the workspace window. Note 

#There is also a family of functions with the is. prefix (please see Table 1.3 in Logan) that evaluate 

is.data.frame(df1)
```

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

Saving and accessing your workspace and data

Everything that you save in R goes to the working directory. It is important to know the filepath to this directory (can vary from computer to computer). You can also specify any other filepath you like if you dont want it saved to the working directory. Other useful functions include setwd(), where you can change your working directory by specifying a particular filepath (inside the parentheses), and getwd() where you can review what the current working directory is. It is also good to note that you can separately select and copy code and output in the console and "plots" windows and paste in Word or other text editing program.

Additionally, the use of projects for R studio is highly recommended if you are working on your own computer. Projects are a way of organizing files such as data you read in and read out of R, as well as graphs or other things you might like to keep organized in subfolders associated with your coding. If you go to the dropdown menu top right by the R studio symbol you can choose a new project and click existing directory and navigate to the working directory above where you stored your R script (in this case an Rmd script.)

Reading in data

To import data from an excel spreadsheet into R Studio, use the "import dataset" tab in the upper right-hand portion of the Rstudio screen. You can then browse to the location of your spreadsheet on your computer, import it, and it will show up in the "editor window". HOWEVER, you must save the file you've imported using this shortcut by assigning the excel file (or csv) to an Object in R. If you used a particular library (R "librarys" are packages of coding functions executed together in R) to read in the data you must include the call for that library as well. Click on the Import Dataset shortcut now and see what the library and suggested object name are for the "null" Excel file. *Please ask me if you aren't sure about this as it causes a lot of trouble when you try to turn in your homework in knitted format*

There are various ways to save your work in R studio, depending on which window you are interested in.

There are four basic types of vectors in R: Integer (whole numbers), numeric (includes decimals), character (letters and names), and logical (True or False). In addition, there is what is called a factor variable- this is like a character variable except the characters are stored internally as numbers, where the characters are alphabetically converted to numbers, starting with 1. Factors are created using the factor() function, and are printed by R without the quotation marks.

If you want to create a character vector and add it to your dataframe:

```
plot<-c("lo","lo","lo","hi","hi")
plot
```

```
## [1] "lo" "lo" "lo" "hi" "hi"
```

Now you can add the character "plot" to the df1 dataframe:

```
df1<-data.frame(df1,plot)
df1
```

```
##   ohia.diam temp plot
## 1         5   30   lo
## 2         8   25   lo
## 3        12   20   lo
## 4        15   15   hi
## 5        20   10   hi
```

```
# try clicking on df1 in your environment and see how it opens using the View() command
```

And verify that you have made a dataframe again:

is.data.frame(df1)

If you had used cbind(df1,plot) what type of object would you have created? (note this is a trick question)

Now, back to that character plot that we created- it still exists:

```
is.factor(plot)
```

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

```
is.character(plot)
```

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

```
#to turn plot into a factor:
plot<-factor(plot)
plot
```

```
## [1] lo lo lo hi hi
## Levels: hi lo
```

```
#another way to verify what kind of vector you are dealing with, use the class() function
class(plot)
```

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

```
#or:
is.factor(plot)
```

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

```
# Similarly, you can convert objects using the as. prefix (see Table 1.4 in Logan)

as.character(plot)
```

```
## [1] "lo" "lo" "lo" "hi" "hi"
```

Some more vector functions:

See if you can determine what each of these is doing > xP<-c(1:10) > xP [1] 1 2 3 4 5 6 7 8 9 10 > rep(xP,2) [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 > rep(hi,4) Error: object 'hi' not found > rep("hi",4) [1] "hi" "hi" "hi" "hi" > xP[3:6] [1] 3 4 5 6 > letters[1:5] [1] "a" "b" "c" "d" "e" > LETTERS[1:5] [1] "A" "B" "C" "D" "E" > max(xP) [1] 10

and some more:

max(xP), min(xP), sum(xP),mean(xP), sd(xp), median(xP),range(xP),var(xP),sort(xP) rank(xP),quantile(xP),cumsum(xP),
etc.

Summary() is a generic function that is versatile and does different things depending on the type of object x might be; usually sensible and useful.

To practice some more vector and matrix functions, lets create a matrix: - Note: depending on time, you can do this at home

```r
eggs<-c(2,4,6,8,10,12,14,16)
temp<-c(20,21,22,23,24,25,26,28)
eggsvstemp<-cbind(eggs,temp)
```

- Note: you can shorten the name to something easier to type! (though I haven't done that here)

Row and column names

```r
# forget the column names?
colnames(eggsvstemp)
```

```
## [1] "eggs" "temp"
```

```r
rownames(eggsvstemp)
```

```
## NULL
```

As you can see, the resulting matrix does not have row names. If you want to add row names, you can do so manually using the rownames() function.

```r
# Assign row names
rownames(eggsvstemp) <- c("Sample1", "Sample2", "Sample3", "Sample4", "Sample5", "Sample6", "Sample7",
eggsvstemp
```

```
##         eggs temp
## Sample1    2   20
## Sample2    4   21
## Sample3    6   22
## Sample4    8   23
## Sample5   10   24
## Sample6   12   25
## Sample7   14   26
## Sample8   16   28
```

Subsetting

Often times, it may be necessary to specify a subset of rows within a vector or matrix. This can be done by appending an "index vector" to the vector name using square brackets. temp[2]

```r
# subsetting
temp[2:5]
```

```
## [1] 21 22 23 24
```

```r
temp[temp>24]
```

```
## [1] 25 26 28
```

```r
#now i'll add a character vector to the matrix:

light<-c("no", "no", "no", "no", "yes", "yes", "yes", "yes")

eggsvstemp<-cbind(eggs,temp,light)
eggsvstemp
```

```
##      eggs temp light
## [1,] "2"  "20" "no"
## [2,] "4"  "21" "no"
## [3,] "6"  "22" "no"
## [4,] "8"  "23" "no"
## [5,] "10" "24" "yes"
## [6,] "12" "25" "yes"
## [7,] "14" "26" "yes"
## [8,] "16" "28" "yes"
```

```r
is.matrix(eggsvstemp)
```

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

Conditional operators

Now we can use some more conditional operators inside the hard brackets for subsetting the data

```r
temp[light=="no"]
```

```
## [1] 20 21 22 23
```

```r
temp[eggs<14 & light == "no"]
```

```
## [1] 20 21 22 23
```

Index by row and col

You can also use R to index values based on [row,col]- see below

```r
eggsvstemp[6,2]
```

```
## temp
## "25"
```

```r
eggsvstemp[6,]  # this selects the entire 6th row
```

```
##   eggs  temp light 
##   "12"  "25" "yes" 
```

```r
eggsvstemp[,2]   #this selects all of column 2
```

```
## [1] "20" "21" "22" "23" "24" "25" "26" "28"
```

```r
eggsvstemp[,-1]  # this selects all columns except the first
```

```
##      temp light
## [1,] "20" "no" 
## [2,] "21" "no" 
## [3,] "22" "no" 
## [4,] "23" "no" 
## [5,] "24" "yes"
## [6,] "25" "yes"
## [7,] "26" "yes"
## [8,] "28" "yes"
```

```r
eggsvstemp[2,1:2]    #this selects row2, columns 1-2
```

```
## eggs temp 
##  "4" "21" 
```

"Apply" function shortcuts.

The apply family of functions in R is designed to perform repetitive tasks on data structures like lists, matrices, arrays, and data frames. These functions are often more efficient and concise than using loops.

The tapply function applies a function to a vector separately for each level of an associated factor. This can come in handy. For example, if you want to calculate the mean temp for each of the 2 levels of light:

```r
tapply(temp, light, mean)
```

```
##    no   yes 
## 21.50 25.75 
```

How do you think you can calculate the standard deviation for eggs at each light level?

If you don't know the function for standard deviation try looking it up online, or use the Help search. Note there are lots of stardard deviation calculators in various R packages. We want the one that is with the built in "stats". Sometimes there are conflicting function names among package authors but you can always say the specific package you want to call by typing package::function for example stats::sd

Note, that often you may want to group data based on multiple categorical variables. In such cases, you can use a list of factors to define the groups. In general, the data vector is grouped by the group factor, and the mean or sd (or even some function you define) of each group is calculated.

That's all! Hope you enjoyed this introduction to R. Next week we will do another worksheet and you will have your first Homework. Please try to knit this document so you know that it work.