Stat 406, Spring 2020

Introduction to R and exercises

R and RStudio: R is the computing / data analysis program. RStudio is an Integrated Development Environment (IDE) for R. RStudio organizes windows, files, and output for you. Most folks learning R now use RStudio. I learned R years ago (long before RStudio) so I generally don't. These notes are set up for both R and RStudio. Differences between the two interfaces are indicated.

R and RStudio are installed on all Stat computing lab machines. Look in the Stat Applications folder on the desktop.

If you want to install R and RStudio on your own, or a lab group, computer:
1) install R: go to the ISU R mirror site, `http://mirror.las.iastate.edu/CRAN/`, click the appropriate link (Linux, OS X, Windows) at the top of the page. You will need adminstrator privledges when you run the install file. If you want to use R and not RStudio, you're now ready to go.
2) install RStudio: install R (previous step), then go to the RStudio download site, `https://www.rstudio.c` and click the appropriate link. The RStudio installer will automatically detect your copy of R and link to it.
Note: If you install RStudio, you want vanilla RStudio, not RStudioServer.

R is open source and completely free. RStudio is produced by a commercial company. The individual version of RStudio is free; at least some of the other versions are not.

1. To run R or RStudio on one of machines in a Stat computing room, open the Stat Applications folder on the desktop and click on the program you want. If you use R, you have a choice of a 32 bit or a 64 bit version. For what we're using, it shouldn't matter which you use. (If you don't know the difference, ignore the previous two sentences).

2. When your program starts, you will see a workspace. The organization of this depends on your choice of program.

   - R: you see a workspace with room for multiple subwindows. The only subwindow initially is labeled R Console.
     RStudio: your workspace is filled with three or four subwindows: the bottom left is labelled Console. If you have four windows, the other three (clockwise from the bottom left) are:
     the code / data set window, where you can edit files of R code or look at data frames,
     the Environment/History window, where you see the contents of your R workspace (more below), or the History of commands you've typed, and
     a multipurpose window that is most commonly used to show plots or R help files.

3. R is a command driven language. You type something into the R Console (R) or Console (RStudio) window, hit the enter key, and R does it. E.g., 1*2/3 <enter> returns 0.6666667

4. The ">" symbol is the R prompt. If you haven't finished a command when you hit the enter key, R will start the new line with +. If you type 1*2/ <enter>, you get a +. type 3 <enter> and you get the result.

5. If you get a + and don't understand why, the most common reason is character strings without an ending quote. Type the matching quote symbol and the rest of your command.

6. Capitalization MATTERS. A is not the same as a. The function sum() is not the same as the functions Sum() or SUM().

7. The bulk of the work is done by functions. These accept arguments and return results. For example, sum(x) accepts a vector or matrix named x and computes the sum of all the elements.

8. To see the help file for a function, type ?NAME or help(NAME), where NAME is the name of the function, probably lower case. If R this will open a browser window with the help file contents. In RStudio, the help file is displayed in the help window.

9. Help files are succinct. They remind you of the format of the function, its options, and something about the output. Often the help file tells you about related commands, which can be very helpful if you don't remember the name of a function, but you know something it's related to.

10. The result of a function can be stored in another object using either <- (two characters making a back pointing arrow), or = (the equals sign). I learned <-, so I continue to use that. E.g., tot <- sum(c(1,2,4,5,7,8)) stores the sum in the variable tot.

11. You can print an object either by typing its name <enter>, e.g. tot <enter>, or using the print() function, e.g., print(tot).

12. Historical function names often look like t.test(). More recent function names often look like SpatialPoints().

13. Functions are organized into libraries. To use a library, you have to download and install it, then activate it.

    In R:

    - To download: Select Packages/Install Package from the R main menu. You will be asked for a mirror site. Choose US(IA). You then get an alphabetic list of all available packages. Scroll down, select on the desired package, then click on ok. It will be downloaded, unzipped, and installed in the appropriate directory. This only has to be done once (per package and computer).
    - To activate: type library(package) at the R prompt, or select Packages/Load Package from the main menu. You will be given a list of all installed packages. Select the desired package, then click ok. You need to do this every session in which you want to use that package.

    In RStudio:

    - To download: When R is running, select Packages/Install Package from the main menu. You will be asked for a mirror site. Choose US(IA). You then get an alphabetic list of all available packages. Scroll down, select on the desired package, then click on ok. It will be downloaded, unzipped, and installed in the appropriate directory. This only has to be done once (per package and computer).

- To activate: type library(package) at the R prompt, or select Packages/Load Package from the main menu. You will be given a list of all installed packages. Select the desired package, then click ok. You need to do this every session in which you want to use that package.

14. Packages provide extensions to commonly used functions. For example, there are many different plot functions. They do different things to different types of data. You don't have to worry which is the correct plot function, you just type plot(object). However, if the appropriate package has not been activated, those specific versions of plot() aren't available. If a command does something unexpected, check that you have activated the appropriate package, e.g. by library(gstat).

15. To see a list of all the functions in a package, type library(help=PACKAGE), where PACKAGE is the name of the package. library(help=base) gives you basic R functions. library(help=stats) gives basic statistical functions.

16. To leave R, type q(). q() is the quit function. If you just type q, you see the definition of that function.

Types (Classes) of R objects:
What R does to something depends on its class. If you know about object-oriented programming, that's what R is based on. Some of what we do will require manipulating the class, so R knows how to deal with us. If R does something you didn't expect or gives you an unusual error message, check to make sure the class is set correctly. Some of the commonly used classes:

- Scalars and Vectors: e.g. 1, 1.4, pi or c(1,2,5,7). These are have the class integer or numeric (real) but you don't need to worry about the difference. You can create a vector with specified numbers using the collect, c(), function. So c(1,2,5,7) is a vector of four integers. You can find out the length of a vector using length()

- Character: A character string, e.g. 'hello' or "hello". Either single or double quotes work, but they need to match. You can not do arithmetic on character strings. You can have vectors of character strings, e.g. c('a','b','c')

- Logical: T (true) or F (false) values. If you do arithmetic on these, T is converted to 1, and F is converted to 0.

- Matrix: rows and columns. You can create matrices directly various ways. If all entries are the same number, matrix(0, nrow=5, ncol=6) will create a matrix with 5 rows and 6 columns filled with 0's. If you want to specify the values, matrix(c(1,2,4,5,7,8), nrow=3, byrow=T) will create a matrix with 3 rows and 2 columns filled with the values that were provided. R knows there are 2 columns because you gave 6 values and specified 3 rows. You can specify either the number of rows or the number of columns. A matrix has to be entirely numbers or entirely character strings. You can subset matrices. x[1,1] is the value in the first row and first column. x[,1] is the all rows of the first column. x[1:3,] is all columns of the first three rows. 1:3 is a shortcut for the vector of integers starting with 1 and ending with 3. 1:3 is the same as c(1,2,3). c(1:3, 5:7) is the same as c(1,2,3,5,6,7).

- Data frame: rows and columns of data. This can include both numbers and character strings, so a column can be site names, the second and third can be x and y coordinates, the fourth and fifth are two measured values, and the sixth is the soil type, stored as a character string. Raw data from a file is usually read into a data frame. Columns of a data frame have names. Specific columns can be extracted either by subsetting columns (as done for a matrix), or by using $. So soildata$X refers to the column labelled X in the soildata data frame.

  A data set (e.g. an excel spreadsheet) will usually be stored in R as a data frame.

- Spatial objects: The sp package defines a large number of spatial classes (points, lines, pixels, dataframes). I'll introduce these shortly. In order to use many of the spatial functions, your data needs to be converted to a spatial class. I'll show you how to do this.

- Classic R uses what are called System 3 (S3) objects. These have components accessed by $. Some packages, e.g. sp (which we will use) and lme4 (which you may have used), use System 4 (S4) objects. S4 objects are defined more stringently, which simplifies programming. In my mind, S4 objects are a lot harder to use (but that may just be because of my being more very familiar with S3 and less familiar with S4). The main difference is that S4 objects have slots accessed by @. I'll illustrate the difference in the sp package part of the exercises.

Useful tips:

1. Both R and RStudio have a text (script) editor. This allows you to save commands. You can run the entire file of commands or specific lines. Some details differ between R and RStudio:

   - R: File/new script opens a new window (R editor window). File/open script opens a previously existing file. R expects script files to have a .R or .r extension. If you open multiple files, each file is in a separate window.
     To run the entire file: save it, return to the console and type `source('name')`, where `'name'` is the name of the file, including its extension.
     To run a single line from the editor window: put the cursor on the line to be executed (anywhere in the line), then type ctrl-r (control key and r). The line will be copied to the console and executed.
     To run multiple lines: highlight the desired commands (have to start this at the beginning of the line), then type ctrl-r
   - RStudio: File / New file / R script opens the file editor window in the top left of the screen. File / Open file opens a previously existing file. R expects script files to have a .R or .r extension. If you open multiple files, they become new tabs in the file editor window.
     To run the entire file: click the source button.
     To run a single line from the editor window: put the cursor on the line to be executed (anywhere in the line), then type ctrl-enter (control key and enter key) or click the run button in the file editor window. The line will be copied to the console and executed.
     To run multiple lines: highlight the desired commands (have to start this at the beginning of the line), then type ctrl-enter or click run.

2. The # character defines the start of a comment. Anything from # to the next <enter> are ignored. This allows you describe / document what you are doing in a file of commands.

3. The console window has a history facility (both R and RStudio). If you make an error, you can type the up arrow to bring up the previous command. Repeated up arrows go back up through previous commands. You can edit the command (using right or left arrow keys or the mouse to move around), then hit <enter> to execute it.

4. Graphs are displayed in a new window.

    - R: A new window pops up, with the requested graph. If you select that graph window, the R main menu changes to provide options for graphs. One of those is History/Recording. When recording is on, R remembers each plot, so you can page back and forth to compare different plots. Another is File / Save as which allows you to export a graph in one of various formats. Another is File / Copy to the clipboard. For copying into a Word document, saving as meta file (.emf) or copying to the clipboard as a meta file make the nicest graphs. To include into a LaTex file, save the graph as a .pdf file.
    - RStudio: The plot appears in the lower right window (plots component). The Export button allows you to save the file (Export / Save as image) then choose your file type or copy to the clipboard (Export / copy to clipboard) and select whether to copy as bitmap or metafile. Again, metafiles (or .emf files) are the best to copy into Word or you want a .pdf file to include in a LaTex document.

5. RStudio makes it easy to use Sweave, knitr, and Markdown to create reproducible documents. These are documents with embedded R code. When you compile the document, the code is run and the output (or figure) is automatically inserted into the document. You are welcome to use these tools for class projects but I won't teach their use.

Some useful functions, especially valuable if something isn't working as expected:

- head(object): print the head (first ca 5 lines of the object).
- args(function): print the arguments accepted by a function.
- str(object): print the structure of the object. Includes class and a short summary of the object. If object is a data frame, prints a short summary of each column.

Lab exercise - part 1 - using R

These repeat the material in Bivand section 2.2, except with they emphasize things I find most useful. If you're familiar with R, you probably don't need to do any of these exercises. For each, I give you an R command. You should type it in and see what happens. If you don't understand what the command is doing or why the result is what it is, look in the first section of material or ask. You don't need to type in my comments (lines starting with #). Those are there to explain the previous command.

1. Basic operations:
   (a) c(1.1, 2.2, 4.1)
   (b) x <- c(1.1, 2.2, 4.1)
   (c) x
   (d) length(x)
   (e) sum(x)
   (f) mean(x)
   (g) mean(X)
   (h) str(x)
   (i) x2 <- rnorm(100, 5, 0.2)
   # simulate 100 random values from a normal distribution with mean 5 and sd 0.2
   (j) summary(x2)
   (k) c(mean(x2), sd(x2))
   (l) 1:9
   (m) ?sum
2. Matrix operations: (code below uses x defined above)
   (a) xm <- matrix(1:9, nrow=3)
   (b) xm
   (c) diag(xm)
   # extract the diagonal of the matrix
   (d) xm[1:2,1:2]
   (e) xm[1,]
   (f) xm2 <- matrix(0, nrow=3, ncol=3)
   (g) xm2
   (h) diag(xm2) <- 1
   # some functions can be used "backwards" to assign values
   (i) xm2
   (j) xm3 <- as.matrix(x)

(k) xm3
    # a vector is not the same as a matrix with one column

(l) t(xm3)
    # t() is the transpose function. flips rows and columns.

(m) t(xm)

3. Reading data:
R provides various ways to read data. The two I will illustrate are read.table(), which reads a text file, and read.csv(), which reads a .csv (comma separated values) version of an excel spreadsheet. The presumption is that the first line contains variable names. The default behaviour of both functions is to read numbers as numbers and to convert character strings to factors (a class we won't use until much later). I prefer to leave character strings "as is". If something needs to become a factor, I prefer to explicitly convert it. Both functions create data frames.

The data section of the class web site has a data set containing rainfall amounts on 8 May 1986 at various places in Switzerland. The rainfall units are 1/10 mm of rain, so 24 is 2.4 mm. The x and y locations are in meters, relative to a point in the middle of the country. The four columns are the site number, the x and y coordinates, and the rainfall. This is .csv format (Excel, comma separated values) file so values are separated by commas and two commas with nothing between is a missing value.

Download and save the swiss.csv data file. Remember to save it in someplace useful (e.g. the desktop or a stat 406 folder). Or save it in the downloads folder and then move the file to someplace useful.

(a) swiss <- read.csv(file.choose(), as.is=T)
    # file.choose() opens a menu to choose a file name interactively
    # as.is=T suppresses converting character strings to factors

(b) str(swiss)

(c) setwd(choose.dir())
    # I find it very helpful to specify a working directory.
    # This keeps various pieces of my work separate. setwd() does that.
    # choose.dir() opens a window to select interactively the directory.
    # you probably want a folder in the c:/Users/yourNetId folder.

(d) swiss <- read.table('swiss.csv', header=T, as.is=T, sep=',')
    # read swiss.csv from the working directory using a more general function
    # read.table that reads any delimited text file
    # header=T says the first line is variable names
    # as.is=T suppresses factor conversion
    # sep=',' specifies the separator between values. The default for read.table() is a space.

4. Reading Excel worksheets
Functions to read excel worksheets can be found in various packages. My favorite is read_excel() in the readxl package. It allows you to choose which sheet to read and even which subset of rows and columns to read. See the help file for read_excel() for more information. read_excel() creates a tibble, a special form of data frame. Tibble (tidy tables) are the major data object

in the tidyverse, a collection of packages for structured manipulation of data. Look at the Data Wrangling Cheat Sheet for more information on the tidyverse.

For the most part, tibbles behave like data frames, but there are sufficient exceptions that I would not trust spatial functions to work with tibbles. So, if you use read_excel(), I recommend converting the tibble to a data frame. Here's how:

- swiss.tbl <- read_excel('swiss.xlsx')
  Will look for the file in your working directory. Creates a tibble
- swiss.tbl
  If you print out a tibble, you see that you get a small fraction of rows (and would get a small fraction of columns if there were more) and things like negative values are highlighted.
- swiss <- as.data.frame(swiss.tbl)
  Converts swiss.tbl to a base R data frame