MODULE 4

GETTING DATA INTO R

Contents

4.1 Setting Up R and Helpers

R is a software environment for performing statistical analyses. RStudio is a helper program that makes it easier to use R. `NCStats` is a set of R functions that make the statistical methods used in this class easier. Herein, I will refer to R but you will interact with R through RStudio.

Detailed methods for downloading, installing, and configuring R, RStudio, and `NCStats` on your personal computer are given on the Resources page of the course website.

4.2 Working With R Basics

4.2.1 Saving Results

Results are not saved in R. Rather, "scripts" of successful R commands are saved and, then, if the analysis needs to be re-done, the entire set of commands is opened and run again. When writing a report, all tabular and graphical output should be copied from R and pasted into your report document. This document will serve as your analysis report and can be modified to include answers to questions, references to the tables and graphs, etc.[1] All data that is not a simple vector (see Section 4.3.4) should be entered into R through "comma-separated values" text files (see Section 4.3.2).

R does allow one to save a "workspace", though I urge you not to do that. Rather, save your "good" commands in a script and save your "good" results in a report document; do not save the workspace.

[1] Specifics for how to format homework assignments is on the course syllabus

4.2.2 Expressions and Assignments

Expressions in R are mathematical "equations" that are evaluated by R with a result seen immediately. An example of an expression in R is

```
> 5+log(7)-pi
[1] 3.804317
```

where `log()` and `pi` are built-in functions used to compute the natural log and find the value of π, respectively. Expressions in R are like using a calculator where the result is shown, but not saved for subsequent analyses. In addition, expressions in R follow the same order of operations and use of parentheses as expressions entered into your calculator.

Results from an expression are typically saved for further computations by assigning the results to an object with the assignment operator (i.e., `<-`). The general form for saving the result of an expression into an object is `object <- expression`. The result of the expression will not be seen unless the object name is subsequently typed into R (but see below). For example, the result of the previous expression is saved into an object called `x` and then viewed with

```
> x <- 5+log(7)-pi
> x
[1] 3.804317
```

The result of an expression can be both assigned and printed by surrounding the command in parentheses. For example, the following assigns the result of the expression to `y` and prints the result.[2]

```
> ( y <- 15*exp(2) )
[1] 110.8358
```

An object can be named whatever you want, with the exception that it cannot start with a number, contain a space, or be the name of a reserved word or function in R (e.g., `pi` or `log`). Furthermore, object names should be short and simple enough that you can remember what is contained in the object. It is also good practice to view the object immediately after making the assignment to make sure that it contains results that seem appropriate.

4.2.3 Functions and Arguments

R contains many "programs," or functions, to perform particular tasks. A function is "called" by typing the function name followed by open and closed parentheses. Arguments, which the function will use to perform its task, are contained within the parentheses. The `log()` function, used in the previous section, is an example of a function. The name of the function is `log` and the argument, the number for which to compute the natural log, is contained within the parentheses following the function name. Many other functions will be described below and in subsequent modules.

◇ Regular curved parentheses have two primary uses in R: (1) to control order of operations in expressions (as with a calculator) and (2) to contain the arguments sent to a function.

[2] The spaces between the expression and the parentheses are only needed to increase legibility.

4.3 Working With Data

4.3.1 Data Types

Data in R will be designated as an integer (whole numbers), numeric (non-integer numeric values), character (strings), factor (group membership), or logical (TRUE/FALSE). The type of data largely dictates the type of analysis that can be performed. Data types will be discussed in more detail as needed. Note, however, that the **factor** data type is a special case of the character data type, where the specific items describe the group to which an individual belongs. This description allows for certain analyses in later modules.

4.3.2 Entering Data

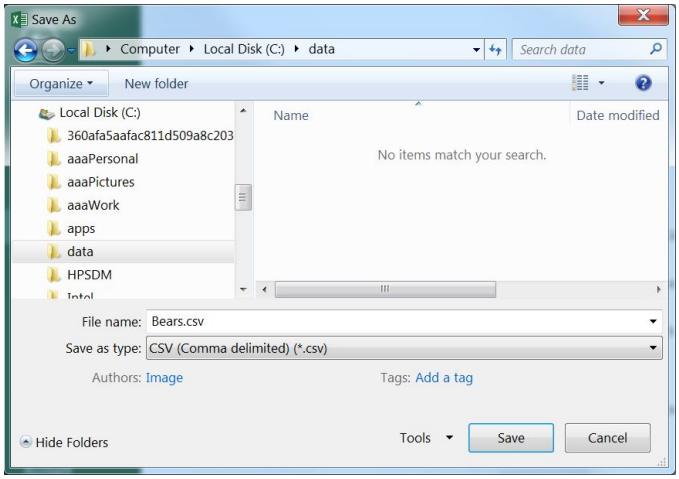
For real data (i.e., several variables from many individuals) it is most efficient to enter data into a comma-separated values (CSV) file and then import that file into R. Creating a CSV file with Microsoft Excel is described below, though there are other ways to create CSV files (see FAQs on class webpage). This explanation assumes that you have a basic understanding of Excel (or other spreadsheet softwares).

The spreadsheet should be organized with variables in columns and individuals in rows, with the exception that the first row should contain variable names. The example spreadsheet below shows the length (cm), weight (kg), and capture location data for a small sample of Black Bears.

	A	B	C
1	length.cm	weight.kg	loc
2	139	110	Bayfield
3	138	60	Bayfield
4	139	90	Bayfield
5	120.5	60	Bayfield
6	149	85	Bayfield
7	141	100	Ashland
8	141	95	Ashland
9	150	85	Douglas
10	166	155	Douglas
11	151.5	140	Douglas
12	129.5	105	Douglas
13	150	110	Douglas

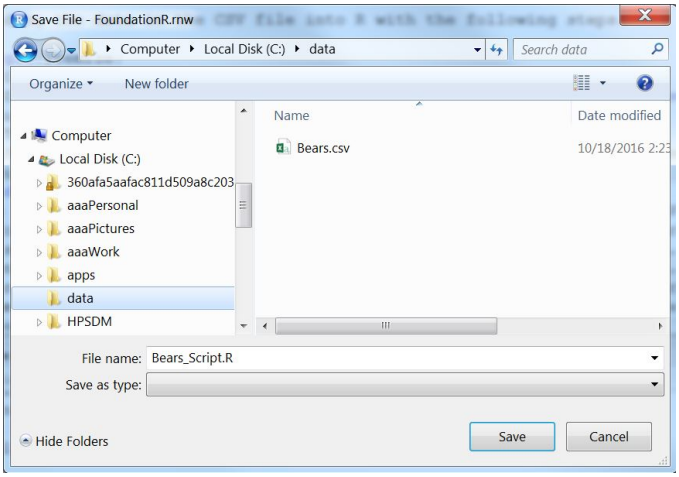
Variable names must NOT contain spaces. For example, don't use *total length* or *length (cm)*. If you feel the need to have longer variable names, then separate the parts with a period (e.g., *length.cm*) or an underscore (e.g., *length_cm*). Furthermore, numerical measurements should NOT include units (e.g., don't use `7 cm`). Finally, for categorical data, make sure that all categories are consistent (e.g., do not have a column that contains both `bayfield` and `Bayfield`).

The spreadsheet is saved as a CSV file by selecting the `File..Save As` menu item, which will produce the dialog box below. In this dialog box, change `Save as type` to `CSV (Comma delimited) (*.csv)` (you may have to scroll down), provide a file name (don't have any periods in the name besides for ".csv", which you should not have to type), select a location to save the file (don't forget this location!!), and press `Save`. Two "warning" dialog boxes may then appear – select `OK` for the first and `YES` for the second. You can now close the spreadsheet file (you may be asked to save changes – you should say `No`).



The following steps are used to load the data in the CSV file into RStudio.

- Open RStudio.
- Open a new script by selecting the `File`, `New File`, `R Script` menu items.
- Type `library(NCStats)` in the new script (i.e., in the upper-left pane).
- Save this script by selecting the `File`, `Save` menu items. In the ensuing dialog box, navigate to the **exact same directory** where you saved the data, type a name for the file in the `File name:` box (**do not use a period in this name!!**), and press `Save`.



- Set the working directory (tell R where the file is) with the `Session`, `Set Working Directory ...`, `To Source File Location` menu items in RStudio. RStudio will print an appropriate `setwd()` command to the console (lower-left pane). Copy this command from the console to the second line in your script.[3] For example, I stored the file created above in the `C:/data` directory, so that RStudio will create this `setwd("C:/data")`.
- The CSV file is read into R by including the name of the file (in quotes) in `read.csv()`. For example, `"Bears.csv"` is read into R and stored into an object called `bears` with `bears <- read.csv("Bears.csv")`.
- One should check the data in this object as descried in Section 4.3.3 below.

[3]Doing this will eliminate the need to manually select the menu options every time you want to run this script.

It is important that each row of the data.frame correspond to one individual. This is critically important when data are recorded for two different groups (e.g., for a two-sample t-test; see Module 18). For example, the following data are methyl mercury levels recorded in mussels from two locations labeled as "impacted" and "reference."

```
impacted   0.011  0.054  0.056  0.095  0.051  0.077
reference  0.031  0.040  0.029  0.066  0.018  0.042  0.044
```

To follow the "one individual per row" rule, these data are entered in stacked format where the "reference" data are stacked underneath the "impacted" data and a column is used to indicate to which group the individuals belong. For example, the Excel file for data entry would look like the following.

	A	B
1	loc	merc
2	impacted	0.011
3	impacted	0.054
4	impacted	0.056
5	impacted	0.095
6	impacted	0.051
7	impacted	0.077
8	reference	0.031
9	reference	0.04
10	reference	0.029
11	reference	0.066
12	reference	0.018
13	reference	0.042
14	reference	0.044

Alternative Forms of Getting Data

Some of the data files that you will use are provided on the Data for MTH107 resource page of the class webpage. In these cases, the data should be downloaded from the webpage and saved in the same directory or folder as your analysis script. The downloaded file is then read into R in the same manner as described previously (i.e., set the working directory with `setwd()` and use `read.csv()`).

A few data files used in these notes are supplied with R or the `NCStats` package. These files are loaded with `data()`. For example, the **iris** data file is loaded into R with

```
> data(iris)
```

4.3.3 Working With Data Frames

Viewing a Data Frame

Many users are disoriented in R because they cannot "see" their data in the same way that they see it in a spreadsheet program. There are, however, several options for viewing your data. First, you can type the name of the data.frame object to see its entire contents.

```
> bears
   length.cm weight.kg      loc
1      139.0       110 Bayfield
2      138.0        60 Bayfield
3      139.0        90 Bayfield
4      120.5        60 Bayfield
5      149.0        85 Bayfield
6      141.0       100  Ashland
7      141.0        95  Ashland
8      150.0        85  Douglas
9      166.0       155  Douglas
10     151.5       140  Douglas
11     129.5       105  Douglas
12     150.0       110  Douglas
```

Typing the name is adequate for small data.frames, but not useful for large data.frames. The entire data.frame is opened in a separate window by double-clicking on the name of the data.frame in the `Environment` tab of RStudio (in upper-right pane). Alternatively, the first and last three rows of a data.frame are viewed by including the data.frame object in `headtail()`.

```
> headtail(bears)
   length.cm weight.kg      loc
1      139.0       110 Bayfield
2      138.0        60 Bayfield
3      139.0        90 Bayfield
10     151.5       140  Douglas
11     129.5       105  Douglas
12     150.0       110  Douglas
```

In addition to viewing the contents, it is useful to examine the structure of the data.frame as returned from `str()`. In this example, it is seen that three variables were recorded on 12 individuals. The first variables – *length.cm* and *weight.kg* – are numerical measurements made on the bears. The last variable – *loc* – is a factor variable that records the capture location for each bear.

```
> str(bears)
'data.frame': 12 obs. of  3 variables:
 $ length.cm: num  139 138 139 120 149 ...
 $ weight.kg: int  110 60 90 60 85 100 95 85 155 140 ...
 $ loc      : Factor w/ 3 levels "Ashland","Bayfield",..: 2 2 2 2 2 1 1 3 3 3 ...
```

The levels of the *loc* variable may be seen by including this variable (with the data.frame name) as the argument to `levels()`.

```
> levels(bears$loc)
[1] "Ashland"  "Bayfield" "Douglas"
```

In the previous example, the `$` notation was used to identify a particular variable (i.e., `loc`) within a data.frame (`bears`). Think of variables as being nested inside data.frames and, thus, to access the variable you must first identify the data.frame in which it exists and then the name of the variable. The `$` simply separates the data.frame from the variable.

```
> bears$length.cm
 [1] 139.0 138.0 139.0 120.5 149.0 141.0 141.0 150.0 166.0 151.5 129.5 150.0
> bears$loc
 [1] Bayfield Bayfield Bayfield Bayfield Bayfield Ashland  Ashland  Douglas  Douglas
[10] Douglas  Douglas  Douglas
Levels: Ashland Bayfield Douglas
```

Filtering a data.frame

It is common to create a new data.frame that contains only some of the individuals from an existing data.frame. For example, a researcher may want only bears captured in Bayfield County or bears that weighed more than 100 kg. The process of creating the newer, smaller data.frame is called filtering (or subsetting) and is accomplished with `filterD()`. The `filterD()` function requires the original data.frame as the first argument and a condition statement as the second argument. The condition statement is used to either include or exclude individuals from the original data.frame. Condition statements consist of the name of a variable in the original data.frame, a comparison operator, and a comparison value (Table 4.1). The result from `filterD()` should be assigned to an object, which is then the name of the new data.frame.

Table 4.1. Condition operators used in `filterD()` and their results. Note that *variable* generically represents a variable in the original data.frame and *value* is a generic value or level. Both *variable* and *value* would be replaced with specific items (see examples in main text).

Condition Operator	Individuals Returned from Original Data Frame
variable == *value*	all individual that are **equal** to the given value
variable != *value*	all individuals that are **NOT equal** to the given value
variable > *value*	all individuals that are **greater than** the given value
variable >= *value*	all individuals that are **greater than or equal** to the given value
variable < *value*	all individuals that are **less than** the given value
variable <= *value*	all individuals that are **less than or equal** to the given value
condition , *condition*	all individuals that **meet both conditions**
condition | *condition*	all individuals that **meet one or both conditions**[4]

The following are examples of new data.frames created from *bears*. The name of the new data.frame (i.e., object left of the assignment operator) can be any valid object name. As demonstrated below, the new data.frame (or its structure) should be examined after each filtering to ensure that the data.frame actually contains the items that you desire.

- Only individuals from *Bayfield* county.

```
> bf <- filterD(bears,loc=="Bayfield")
> bf
  length.cm weight.kg      loc
1     139.0       110 Bayfield
2     138.0        60 Bayfield
3     139.0        90 Bayfield
4     120.5        60 Bayfield
5     149.0        85 Bayfield
```

- Individuals from both *Bayfield* and *Ashland* counties.

```
> bfash <- filterD(bears,loc %in% c("Bayfield","Ashland"))
> bfash
  length.cm weight.kg      loc
1     139.0       110 Bayfield
2     138.0        60 Bayfield
3     139.0        90 Bayfield
4     120.5        60 Bayfield
5     149.0        85 Bayfield
6     141.0       100  Ashland
7     141.0        95  Ashland
```

- Individuals with a weight greater than 100 kg.

```
> gt100 <- filterD(bears,weight.kg>100)
> gt100
  length.cm weight.kg      loc
1     139.0       110 Bayfield
2     166.0       155  Douglas
3     151.5       140  Douglas
4     129.5       105  Douglas
5     150.0       110  Douglas
```

- Individuals from *Douglas* County that weighed at least 150 kg.

```
> do150 <- filterD(bears,loc=="Douglas",weight.kg>=150)
> do150
  length.cm weight.kg     loc
1       166       155 Douglas
```

4.3.4 Vectors

Data.frames are the primary structure in which to store real data. However, much simpler situations that don't require a data.frame may arise. In R, items of the same data type (Section 4.3.1) are stored in a one-dimensional *vector*. Vectors are usually displayed in one row (with many columns), but they may also be thought of as a single column (with many rows). Items are entered into a vector with `c()`, where the individual arguments are specific numbers, characters, or logical values.[5] Items for a vector of characters must be contained within paired quotes.

```
> ( v <- c(1,2,5) )
[1] 1 2 5
> ( y <- c("Iowa","Minnesota","Wisconsin") )
[1] "Iowa"      "Minnesota" "Wisconsin"
```

Single variables from a data.frame are vectors.

```
> bears$length.cm
 [1] 139.0 138.0 139.0 120.5 149.0 141.0 141.0 150.0 166.0 151.5 129.5 150.0
```

Vectors that are not extracted from a data.frame will only be used in this course for very simple lists of items, usually as arguments in a function.

[5] Note that `c` comes from the word "concatenate."

Selecting Individuals

In some instances, you may need to select or exclude an individual from a data.frame or vector. Vectors are one-dimensional objects that are indexed simply by the position within the object. Positions within an object are identified with square brackets. For example, the second item from the `y` vector created above is extracted with

```
> y[2]
[1] "Minnesota"
```

Alternatively, the second item can be excluded by preceding the position number with a negative sign.

```
> y[-2]
[1] "Iowa"      "Wisconsin"
```

Data.frames are two-dimensional objects that are indexed by a row and a column, in that order. Again, positions within an object identified within square brackets. However, for data.frames, both a row and a column specification must be given. For example, the item in the third row and second column of `bears` is selected below.

```
> bears[3,2]
[1] 90
```

An entire row or column may be selected by omitting the other dimension. For example, one could select the entire second column with `bears[,2]`, but this is also the `weight.kg` variable and is better selected, as shown above, with `bears$weight.kg`. As a better example, the entire third row is selected below (note that the column designation was omitted).

```
> bears[3,]
  length.cm weight.kg      loc
3       139        90 Bayfield
```

Multiple rows are selected by combining row indices together with `c()`. For example, the third, fifth, and eighth rows are selected below (again, the column index is omitted).

```
> bears[c(3,5,8),]
  length.cm weight.kg      loc
3       139        90 Bayfield
5       149        85 Bayfield
8       150        85  Douglas
```

Finally, rows can be excluded by preceding the row indices with a negative sign.

```
> bears[-c(3,5,8,10,12),]
   length.cm weight.kg      loc
1      139.0       110 Bayfield
2      138.0        60 Bayfield
4      120.5        60 Bayfield
6      141.0       100  Ashland
7      141.0        95  Ashland
9      166.0       155  Douglas
11     129.5       105  Douglas
```