ISTA 116 Lab: Week 1

Last Revised August 21, 2011

1 The R environment

1.1 What is R?

- R is a *free* and *open-source* implementation of the S programming language for statistics.
- Originally Developed in 1990s by Ross Ihaka and Robert Gentleman
- Fully functional (Turing complete) *scripting language*
  - Can write functions and scripts to do anything you can do in any other language
- R is also *interactive* (like Python, MATLAB; unlike C/C++, FORTRAN)
  - Can enter commands directly into interpreter and get a result
  - No need to *compile* code
  - Slower to execute than compiled languages, but more flexible and easy to use

1.2 Starting and quitting R

Starting R

Linux:

1. Open the **Terminal** application
2. Type `R` (capitalized) at the prompt

Windows: Find the `R` application icon and double-click

Mac: Can use either of the above approaches

Quitting `R`

Type `q()`, or close the GUI.

You will be prompted to save the workspace. If you do, all the variables, etc. you have created will be there next time. Usually a bad idea: results in clutter and confusion. Instead, save to a particular file that can be opened later (we'll see how to do this later).

1.3 Paths and Navigation

To input and output from `R`, you need to be able to get around your computer.

`R` looks for files in the *working directory* ("directory" = "folder" in a graphical interface)

- Find the present working directory with `getwd()`. The parens are there because this is a *function*, with no arguments. More on that later.
- See what's in the pwd with `dir()` (just like `ls` in Unix. `ls()` has an analogous function in the `R`environment).
- Change the wd with `setwd("/the/path/to/the/directory")`
  - Example: The Desktop is at `"/home/Desktop"`

(*Note for Windows users*: The backslash is a special "escape character" in `R`. Therefore, you need to specify your paths either with forward slashes, or *double* backslashes: `"C:/Documents/RStuff"`, or ''`C:\\Documents\\RStuff`'')

- You can always use *relative paths* as well.
  - Example: if the pwd is `"/home"`, get to Desktop with just `"Desktop"`
  - Go up one level (back to home) with `".."`

  - Even get from Desktop to, say, `/home/Documents` with `"../Documents"`

Exercise: Create a directory called `RData` on the Desktop, and set it as the working directory in `R`.

1.4 Getting Help

It's impossible to memorize how every function works in `R`. Fortunately, there is documentation for each one.

- To see documentation, type `help("theFunctionName")`, or just `?theFunctionName`.
  - Example: `?getwd()`
- If you want general help related to a topic (or you only know part of the function name, use `apropos("topicOrPartialName")` to see a list of possible functions. With some versions of `R`, can also use `??topicOrPartialName`.
  - Example: `apropos("wd")`

2 `R` as a Calculator

We can use `R` as an interactive calculator by entering in expressions at the command line (much as we'd do for, say, an Excel formula):

2.1 Arithmetic

```
> 2+2
> (14 + 13) / 3^3 + 4*2 / 3
```

- The usual order of operations (PEMDAS) applies

Exercise: The distance from Dullsville to Mediocrity is 33792 feet. The distance from Mediocrity to Thrillsberg is 16360 feet. How many miles is it to go from Dullsville to Thrillsberg via Mediocrity? (There are 5280 feet in one mile.)

2.2 Strings

Sometimes we want to work with text, rather than numbers. Anything placed in quotes (single or double, as long as they match) is treated as a *string*.

```
> "x"
> "This string is an entire sentence"
> "2"
> "2" + "2"
> "The word 'pants' has five letters"
> 'The word "pants" has five letters'
```

Some basic string functions:

`nchar(SomeString)`	Returns the number of characters
`paste(a, b, c, ... , sep=" ")`	Joins strings into one, separated by `sep`
`substr(SomeString, first, last)`	from the `first`th to the `last`th character in the string

3 Using Functions

3.1 What is a function?

All the interesting work in `R` takes place using *functions*. We've seen several already, but what are they?

- a function takes some inputs and produces one or more outputs
- for our purposes, the inputs and ouputs can be (collections of) numbers, text, or a variety of other objects (which in `R` can even include other functions!)

3.2 Arguments

- Some functions (like `getwd()`) just operate by themselves (if you want, you can think of the input as the "state of the world").
- But more commonly, functions take *arguments*, which are values specified when the function is called.
  - When we type `sqrt(2)`, the `2` is the argument
  - `log(42, base=10)` uses 2 arguments
  - In `R`, can give arguments in any order by using their names (e.g. `base=`)
  - Many functions have default values for some args (e.g., `base` is set to e by default
  - See the help page on a function for its argument structure

4 Variables and Assignment

So far, everything has happened in one line, with no "memory". To do anything interesting, we need *variables*.

4.1 The Assignment Operator

Assign a value to a name using `<-`

```
> a <- 3
> b <- 4
> c <- sqrt(a^2 + b^2) #Solving for the length of the hy-
potenuse!
> c
```

Equivalently, can use `=`. Verzani uses `=`; I prefer `<-` because it helps keep assignments distinct from argument specification. Use whichever you prefer.

4.2 Variable Names

- Variable names are case-sensitive (so `X` is a different variable from `x`)
- Names can include letters, numbers, periods and underscores
- Must start with a letter

```
> The_Answer <- 42
> the_answer <- 12
> six.multiplied.by.9 <- The_Answer
```

4.3 Evaluation of Expressions

Question: What is the value of c at the end of the following?

```
> a <- 3
> b <- 4
> c <- sqrt(a^2 + b^2)
> a <- 1
> b <- 0
> c
```

5 Data Objects: Vectors and Data Frames

R is geared toward doing things with *data*. This means working with multiple values at once.

5.1 Vectors

- The basic data object is a *vector*
- Here, this just means a collection of values of the same type
- Can create with the `c()` function

```
> x <- c(2,5,3,7)
> a <- c("this","is","a","character","vector")
```

- The `scan()` function can be used instead

```
> x <- scan()
```

- Create regular or repeating numeric sequences with `seq()` and `rep()`

```
> seq(from=0, to=100, by=10)
> seq(0, 10, length=3)
> rep(c(1,2), times=5)
> rep(c(1,2), each=5)
```

- For basic incremental sequences, can use `start:end`

```
> 1:10
> 14:8
```

Exercise: Create the following sequences

a. "a", "a", "a", "a", "a"

b. $1, 3, 5, ..., 99$

c. $1, 1, 1, 2, 2, 2, 3, 3, 3$

5.2 Vector Arithmetic

- Most functions operate *element-wise* over vectors

```
> x <- c(2,3,9,4)
> log(x)
> y <- c(1,6,2,0)
> x * y
> y^x
```

- If lengths are different, `R` *recycles* the shorter vector

```
> x <- c(2,3,9,4)
> x + 1
> x + c(1,2)
> x ^ c(1,2,3,4,5)
```

Some functions give a single-number summary:

`length()`	How many elements?
`sum()`, `prod()`	The sum or product of all the elements
`mean()`	The average
`min()` and `max()`	The lowest and highest values

Some tell you how each element relates to the others:

```
> x <- c(2,3,9,4)
> sort(x)
> order(x)
> s <- c("the","quick","brown","fox")
> sort(s)
```

Question: What is the value of `s` now?

5.3 Indexing

- Sometimes we want to pick out or alter parts of a vector. We use `[ ]` to do this:

```
> x <- c(2,5,3,9)
> x[2]
```

- We can even put a vector of indices inside:

```
> x[2:3]
> x[c(1,4)]
> x[order(x)]
```

- Here, negative numbers indicate "everything but"

```
> x[-2]
> x[-c(1,3)]
```

- A logical vector of the same length as the outside vector gives you the elements at the `TRUE` positions

```
> x <- c(2,6,3,9)
> x[x <= mean(x)]
> y <- c("a","b","a","b")
> x[y == "a"]
```

Exercise: Create a vector that contains all the years from 1933 to 1980, *except* 1939-1945.

- Can also "edit" vectors by assignment to a subset

```
> x <- 1:10
> x[x %% 2 == 0] <- 0 # %% is the "modulo" or "remainder" op-
erator
```

Exercise: Create a vector containing $1, 2, 3, 14, 42, 6, 7, 8$ using a sequence and replacement operation

5.4 Data Frames

What if you have more than one type of information (e.g., sex, height, major)? Can store each one in a separate, named vector, but this gets messy.

- The *data frame* in R provides a convenient, spreadsheet-like format for organizing data
- Rows for different observations (e.g. people)
- Columns for different variables

```
> sex <- c("M","M","F","F","M","F")
> height <- c(74,68,65,68,70,61)
> major <- c("ISTA","Art","ISTA","CS","English","ISTA")
> roster <- data.frame(sex,height,major)
```

- `names()`: What variables are in the data frame?

5.4.1 Subsets of Data Frames

Data frames have two main ways to identify subsets

`frameName[row(s), column(s)]`	Pick out some entries by their "coordinates"
`frameName$variable`	Pick out a particular variable

```
> roster[c(1,3) , c("sex","height")]
> roster[roster$sex == "M" & roster$major != "ISTA" ,
"height"]
```

Exercise: List the majors for women over 5'4"

- Edit data frames by "reassignment" just like vectors
- Create a new variable by `frame$NewVariable <- Stuff$`

6 Data Input/Output

Most of the time we won't be entering data directly. We usually want to read it in from a file.

- `read.table()` and `read.csv()` produce data frames from tab-delimited and comma-separated text files
- `write.table()` and `write.csv()` do the opposite

- If the file has column names in the first row, specify `header=TRUE`

7 Workspace Management

7.1 Manipulating the Workspace

The *workspace* consists of all the stuff lying around on your virtual desk.

- See what variables and objects you have defined with `ls()`
- Remove select variables from the workspace with `rm()`
- Remove everything with `rm(list=ls())`

7.2 Saving and Restoring the Workspace

When quitting a session, you are asked if you want to save the workspace. This results in everything returning when you next start `R`.

- Usually better, less cluttery, to save to a file that you can load later
- `save()` and `load()` accomplish this

```
> ls()
> save(roster,file="Roster.RData")
> #Save specific variables
> rm(roster)
> load("Roster.RData")
> save.image(file="IstaLab.RData")
> #Save everything
> rm(list = ls())
> load("IstaLab.RData")
> q()
```