Statistical Software Camp: Introduction to R

Day 1

August 24, 2009

1 Introduction

1.1 Why Use R?

- Widely-used (ever-increasingly so in political science)
- Free
- Power and flexibility
- Graphical capabilities
- Pedagogical usefulness
- Learning curve is initially steep, but it gets easier later

1.2 Installing R

1. Go to `http://www.r-project.org`.
2. Follow the link under the `Download` section on the left hand side of the page.
3. Make sure to select an appropriate mirror (the closer the faster) and operating system.
4. Once the download is finished, you can install R in the same way you install other software.

1.3 Using R

1.3.1 Typing Directly

- R as a calculator
- `<-` or `=` to assign a name to an object (`<-` is recommended)
- `#` for comments

```
> apple <- 7
> apple

[1] 7
```

```
> ## Can use = but not recommended
> orange = 12
> orange

[1] 12
```

- Hello ≠ hello ≠ HELLO

```
> obama <- "president" # character string
> Obama # generates error
```

- Variable names cannot begin with numbers
- One command per line or use ;

```
> mathcamp09 <- "Fun"; enrollment <- 15
> mathcamp09

[1] "Fun"

> enrollment

[1] 15
```

- Space between syntaxes will be ignored, but not in ""

```
> pol.571      <-     "first      in quant  sequence"
> pol.571

[1] "first      in quant  sequence"
```

- Can use <up> and <down> to navigate through previously entered commands
- An example: calculate the amount of Princeton's endowment the university could spend for each student without harming the principle.

```
> 15787000000 / (4918+2416)  # Endowment divided by the total number of students

[1] 2152577

> endow.ps <- 15.787*(10^9) / (4918+2416)  # Stores the result as `endow.ps'
> yield.ps <- endow.ps*0.08  # Creates new object `yield.ps'
> yield.ps

[1] 172206.2
```

1.3.2 Using a Text Editor

- Almost always want to use a text editor
- Efficiency, Replicability
- Built-in R editor; what we'll use in the camp
- Notepad, TextPad, TextEdit, etc.
- More fancy editors: WinEdt (`http://www.winedt.com`) for Windows and Aquamacs (`http://aquamacs.org`) for Mac OS X
- Send code from R editor to **console** by:
  - Copy & Paste
  - Highlight and press <Ctrl + R> on Windows
  - Highlight and press <Apple + Return> on Mac
- An example: Repeat the endowment calculation above using the R editor.

1.4 Numeric Vectors

A **vector** is simply string of numeric values connected in a specific order.

1.4.1 Creating, Indexing and Combining

- Use `c()` to enter a data vector
- Use square brackets (`[ ]`) to access elements of a vector
- `c()` can also combine more than one vectors

```
> endow <- c(9928, 11207, 13045, 15000)  #Princeton endowment 2004-2007
> endow

[1]  9928 11207 13045 15000

> endow[2]  # Use square brackets to access elements of a vector

[1] 11207

> endow[c(2, 4)]

[1] 11207 15000

> endow[4] <- 15787
> endow

[1]  9928 11207 13045 15787

> endow[-1]
```

```
[1] 11207 13045 15787

> endow <- endow / 1000
> endow

[1]  9.928 11.207 13.045 15.787

> endow.old <- c(8.398, 8.359, 8.320, 8.730)  #Years 2000-2003
> endow <- c(endow.old, endow)  #combine vectors
> endow

[1]  8.398  8.359  8.320  8.730  9.928 11.207 13.045 15.787
```

1.4.2 Manipulating

- Named vectors can be manipulated like numbers

```
> endow.penn <- c(3.201,3.382,3.393,3.547,4.019,4.370,5.313,6.635)  #UPenn endowment
> endow.penn

[1] 3.201 3.382 3.393 3.547 4.019 4.370 5.313 6.635

> ratio <- endow / endow.penn
> ratio

[1] 2.623555 2.471614 2.452107 2.461235 2.470266 2.564531 2.455298 2.379352
```

1.4.3 Basic Functions

- General format of a function: `output <- functionname(input)`
- More than one input: `output <- functionname(input1, input2, ...)`
- Inputs are often called **arguments**
- The colon operator (`:`) creates a simple sequence
- Use `seq()` for more complex sequences

```
> 1:10

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

> seq(from = 2, to = 10, by = 2)

[1]  2  4  6  8 10
```

- `names()` gives named entries to a vector

```
> names(ratio) <- 2000:2007
> ratio
```

```
    2000      2001      2002      2003      2004      2005      2006      2007
2.623555 2.471614 2.452107 2.461235 2.470266 2.564531 2.455298 2.379352
```

- Simple calculations: `min()`, `max()`, `range()`, `length()`, `sum()` and `mean()`

```
> min(ratio)  # Shows the minimum value of our vector

[1] 2.379352

> max(ratio)  # Same, but for the maximum value

[1] 2.623555

> range(ratio)

[1] 2.379352 2.623555

> length(ratio)  # Tells us the number of values in our vector

[1] 8

> ratio[length(ratio)]

    2007
2.379352

> sum(ratio)  # Sum of all values in the vector

[1] 19.87796

> sum(ratio)/length(ratio)  # Long way to calculate the mean

[1] 2.484745

> mean(ratio)  # Easier

[1] 2.484745
```

1.5 Object Class

- An object belongs to a certain class (e.g. `numeric`, `character`, `function`)
- The function `class()` returns the class of an object
- The function `summary()` to get a summary of an object

```
> endowment <- c(9928, 11207, 13045, 15787)
> class(endowment)

[1] "numeric"

> summary(endowment)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   9928   10890   12130   12490   13730   15790
```

```
> schools <- c("Harvard", "Princeton", "Yale", "MIT", "Stanford", "CalTech")
> class(schools)

[1] "character"

> summary(schools)  #Not so useful when dealing with "character" vectors

   Length     Class      Mode
        6 character character
```

- `factor` is more useful for categorical variables than `character`
- The function `as.factor()` coerces an object into a factor (`as.numeric()`, `as.character()`)
- The function `levels()` returns the categories of a factor

```
> regions <- c("Africa", "Africa", "Asia", "Asia", "Africa", "Middle East")
> regions

[1] "Africa"      "Africa"      "Asia"        "Asia"        "Africa"
[6] "Middle East"

> class(regions)

[1] "character"

> regions <- as.factor(regions)
> regions

[1] Africa      Africa      Asia        Asia        Africa      Middle East
Levels: Africa Asia Middle East

> class(regions)

[1] "factor"

> levels(regions)  # Displays all possible categories for our factor

[1] "Africa"      "Asia"        "Middle East"

> summary(regions) # Useful output for factors

     Africa        Asia Middle East
          3           2           1
```

1.6 Workspace

- The workspace is the sandbox where R temporarily saves everything you've created or loaded into R
- The functions `objects()` and `ls()` list all objects in the workspace
- To remove an object from the workspace, use `remove()` or `rm()`
- You can remove multiple objects via `rm(object1, object2, ...)`

```
> objects()

 [1] "apple"      "endow"      "endow.old"  "endow.penn" "endow.ps"
 [6] "endowment"  "enrollment" "mathcamp09" "orange"     "pol.571"
[11] "ratio"      "regions"    "schools"    "yield.ps"

> rm(schools)
> ls()

 [1] "apple"      "endow"      "endow.old"  "endow.penn" "endow.ps"
 [6] "endowment"  "enrollment" "mathcamp09" "orange"     "pol.571"
[11] "ratio"      "regions"    "yield.ps"
```

1.7 Saving Objects and Workspace

- Use `save()` to save an object or multiple objects (or go to the pulldown menu under File: "Save Workspace...")
- The file name should be `xxx.RData`

```
> save(endowment, file = "H:/endowment.RData")
> save(endowment, regions, file = "H:/endowment.RData") # save multiple objects
```

- If you want to save all objects (i.e., the entire workspace), then use `save.image()`

```
> save.image("H:/Handout1.RData")
```

- The working directory is where R loads and saves your files.
- For example, `save(endowment, file = "endowment.RData")` will save `endowment.RData` in the working directory
- When you close down R, you will be asked if you'd like to save your current workspace. You normally want to choose NO.
- If you choose YES, R will save your files in the working directory as "`.RData`" (empty file name + extension).
- `getwd()` displays the working directory

```
> getwd()
```

```
[1] "/home/teppei/Documents/teaching/software09/handouts"
```

- `setwd()` changes the working directory (or go to the pulldown menu)

```
> setwd("H:/")  # Start saving to H drive
```

- Change the working directory to an appropriate directory at the beginning of your code

1.8 Getting R Help

- Use `help()` or `?` to look up help on a function
- `help.search("xxx")` to look for functions that mention the word `xxx`

```
> help("save")
> ?save       # Same thing
> ?read.table
> help.search("save")
```

- Many resources on the web (e.g., search, mailing list at `http://cran.r-project.org`)

2 Reading Data and Doing Simple Analysis in R

2.1 Loading Workspace and Data

- Data analysis almost always begins by reading some data.
- If you have the file named ".`RData`" in the working directory, it will get automatically loaded when you start R in that directory
- `load()` for a previously saved R object/workspace (`xxx.RData`)
- `read.table()` for a simple text (ASCII) file (`xxx.txt`, etc.)
- `read.csv()` for a comma-delimited file (`xxx.csv`)
- `read.delim()` for a tab-delimited file (`xxx.tab`, etc.)
- Use `header = T` (for "`TRUE`") for a file with variable name headers in the first row
- `read.table()` defaults to `header = T`; others to `F` (= "`FALSE`")
- The object class for datasets is `data.frame`

```
> load("Africa.RData")  # Looks for the file in our working directory
> Africa <- read.table("Africa.txt", header = TRUE)
> Africa <- read.csv("Africa.csv", header = TRUE)
> class(Africa)
```

```
[1] "data.frame"
```

- We can also pull data straight from the web

```
> primates <- "http://www.hopkinsmedicine.org/FAE/primate_structural_properties.txt"
> hop <- read.delim(primates, header=T)
```

2.2 Summary Statistics

- After reading new data, we usually compute summary statistics to get an overview.
- Various functions for summary statistics are available in R.
- `mean()` returns the (arithmetic) mean of an imput vector.
- `max()` and `min()` return the largest and smallest element of a numeric vector, respectively. `range()` gives you the distance between these two numbers.
- The function `summary()` will provide the mean, median, minimum, maximum, and quartiles of a `numeric` object and a table for a `factor` object (you can also use `table()` for this)

```
> mean(Africa$GDP.pc)  # Simple mean

[1] 4616.115

> Africa$pop <- Africa$GDP / Africa$GDP.pc  # Population of each country
> mean(Africa$GDP.pc)

[1] 4616.115

> median(Africa$GDP.pc)

[1] 2162.5

> range(Africa$GDP.pc)

[1]   500 23294

> summary(Africa$GDP.pc)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    500    1366    2162    4616    5569   23290
```

2.3 Graphs

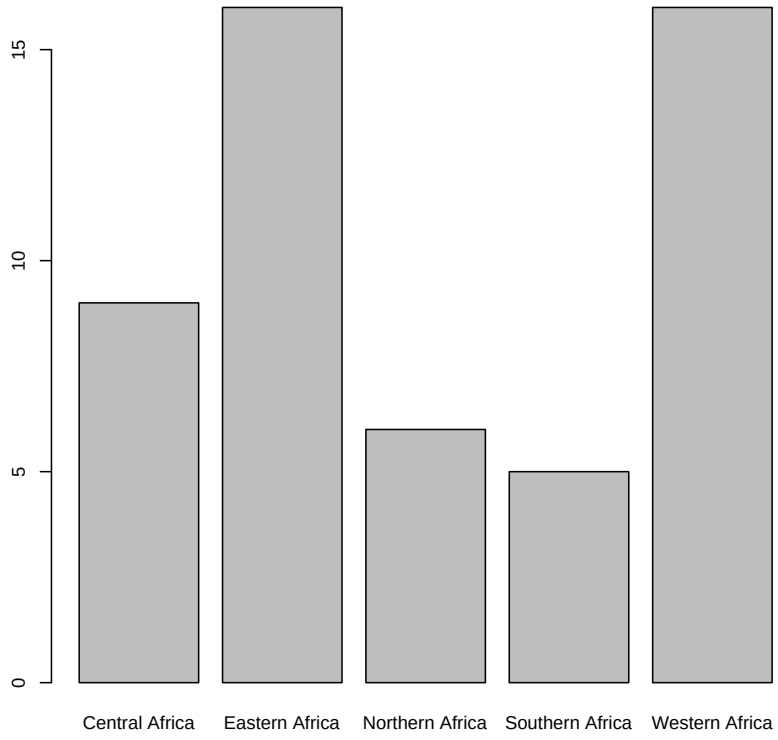
- Graphs are an excellent device to summarize and present quantiative information. They are often way superior to numerical summaries, such as tables.
- Great graphics: Easy to understand the "story" without much explanation or eye-balling
- Bad graphics:
  - Inefficient (leaving out easy-to-add information)
  - Potentially misleading (leaving the reader with the wrong impression)
  - Too complicated (taking too much time to understand)
- The function `barplot()` will produce a **barplot** figure
- The function `pie()` will produce a **pie chart** figure

`main =`	Title to put on the graphic.
`xlab =`	Label for the x-axis. Similarly for `ylab`.
`xlim =`	Specify the x-limits, as in `xlim = c(0,10)`, for the interval $[0, 10]$. Similar argument for the y-axis is `ylim`.
`type =`	Type of plot to make. Use `"p"` for points (the default), `"l"` for lines, and `"h"` for vertical lines.
`pch =`	The style of point that is plotted. This can be a number or a single character. Numbers between 0 and 25 give different symbols. The command `plot(0:25,pch = 0:25)` will show those possible.
`lty =`	When lines are plotted, specifies the type of line to be drawn. e.g., `"solid"`, `"dashed"`, `"dotted"` (See `?par` for full details.)
`lwd =`	The thickness of lines. Numbers bigger than 1 increase the default.
`col =`	Specifies the color to use for the points or lines, e.g., `"blue"`, `"red"`, `"yellow"`.

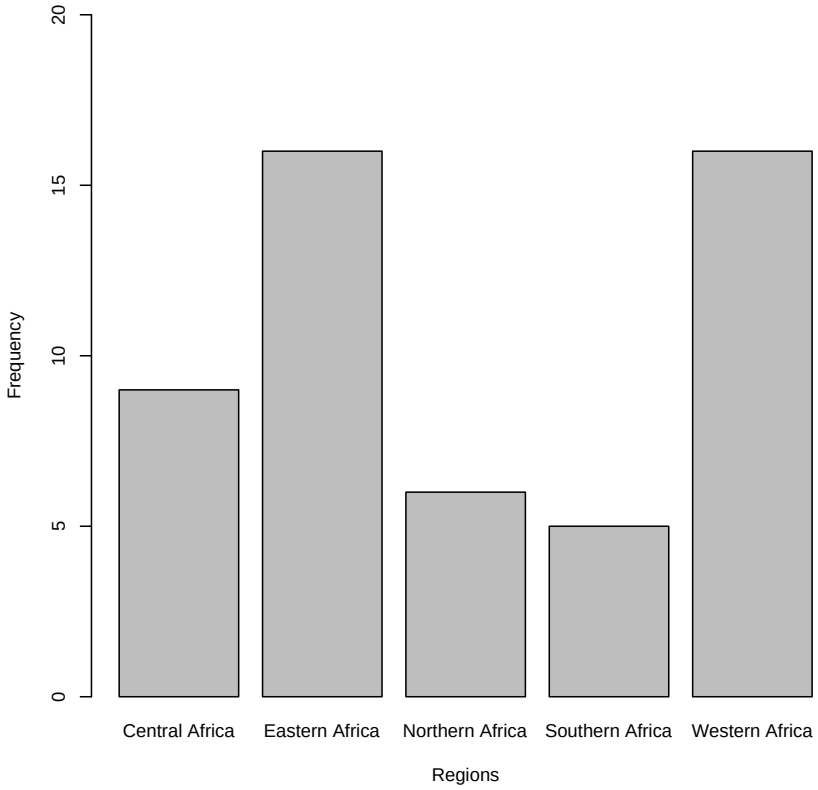
Table 1: Useful arguments for `plot()` and other graphic functions. From Verzani (2005), Table 3.7 (p. 86).

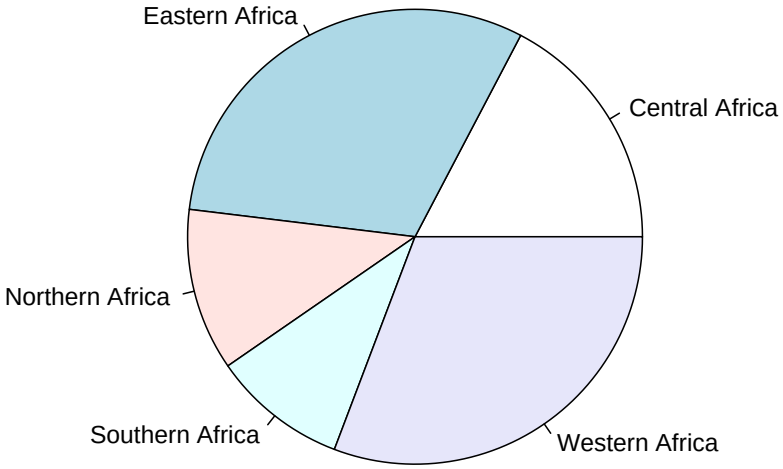
- Specify **arguments** within each function to tweak the graphs to our exact specifications
- To learn the arguments for a particular command, use the help function (e.g., `?barplot` or `help("pie")`)

```
> load("Africa.RData")
> x <- table(Africa$Region) # make a table for a factor variable
> x

 Central Africa  Eastern Africa Northern Africa Southern Africa  Western Africa
              9              16               6               5              16

> barplot(x) # Bar graph; takes in a numeric vector
> barplot(x, xlab="Regions", ylab="Frequency", ylim=c(0,20),
+     main="Geographical Distribution of African Countries")  # Adds labels
> pie(x) # Pie chart; takes in a numeric vector
```



Geographical Distribution of African Countries




Eastern Africa
Central Africa
Northern Africa
Southern Africa
Western Africa