Statistical Software Camp: Introduction to R

Day 1
Basic Concepts and Data Manipulation

January 26, 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 `;`

```
> pol572 <- "Quant II"; enrollment <- 26
> pol572

[1] "Quant II"

> enrollment

[1] 26
```

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

```
> pol.572      <-     "second      in quant  sequence"
> pol.572

[1] "second      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 Matrices

1.5.1 Creating, Indexing and Combining

- R can also handle **matrices**, two-dimensional array of numbers.
- The `matrix(x, N, K)` function reshapes a vector `x` into an `N` $\times$ `K` matrix.
- By default, a matrix is (counterintuitively) filled by columns. Setting the `byrow` argument to `TRUE` (or just `T`) change this behavior.

```
> a <- c(1,2,3,4)
> X <- matrix(a, 2, 2)
> X
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> Y <- matrix(a, 2, 2, byrow=TRUE)
> Y # Notice how X and Y differ
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

- A matrix can also be built up from other vectors/matrices.
- The `cbind(x,y)` and `rbind(x,y)` functions combine `x` and `y` by columns and rows, respectively.

```
> b <- c(5,6,7,8)
> V <- cbind(a,b)
> V
```

```
     a b
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

```
> W <- rbind(a,b)
> W
```

```
  [,1] [,2] [,3] [,4]
a    1    2    3    4
b    5    6    7    8
```

- The `nrow(X)` function returns the number of rows in `X`; `ncol(X)` returns the number of columns.
- The `dim(X)` function reports both at the same time.
- Use square brackets and a comma (`[,]`) to specify rows and columns.

```
> nrow(V)
```

```
[1] 4
```

```
> ncol(V)
```

```
[1] 2
```

```
> dim(V)
```

```
[1] 4 2
```

```
> V[1,] # returns the first row
```

```
a b
1 5

> V[3,2] # returns the (3,2) element

b
7
```

1.5.2 Manipulating

- The addition operator (`+`) works on matrices in the same way as usual

```
> X + Y

     [,1] [,2]
[1,]    2    5
[2,]    5    8
```

- However, the multiplication (`*`) works element by element
- Instead, the `%*%` operator is used for usual matrix multiplication

```
> X * Y

     [,1] [,2]
[1,]    1    6
[2,]    6   16

> X %*% Y

     [,1] [,2]
[1,]   10   14
[2,]   14   20
```

- Other matrix operations include:
  - `t(X)` (transpose, X')
  - `det(X)` (determinant, $|X|$)
  - `solve(X)` (inversion, X^{-1})
  - `crossprod(X,Y)` (cross product, $X'Y$)
  - `kronecker(X,Y)` or `X %x% Y` (Kronecker product, $X \otimes Y$)

1.6 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
```

1.7 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] "a"           "apple"      "b"           "endow"      "endowment"
 [6] "endow.old"   "endow.penn" "endow.ps"    "enrollment" "orange"
[11] "pol572"      "pol.572"    "ratio"       "regions"    "schools"
[16] "V"           "W"          "X"           "Y"          "yield.ps"

> rm(schools)
> ls()

 [1] "a"           "apple"      "b"           "endow"      "endowment"
 [6] "endow.old"   "endow.penn" "endow.ps"    "enrollment" "orange"
[11] "pol572"      "pol.572"    "ratio"       "regions"    "V"
[16] "W"           "X"          "Y"           "yield.ps"
```

1.8 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/imai/software/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

1.9 Using Additional Packages

- R can be extended via **packages**, which contains additional functions and example data.
- Many packages are already available, and the library is being expanded by statisticians and methodologists.
- Packages can be installed directly from the internet using `install.packages("xxx")` where `xxx` is the package name.

```
> install.package("mvtnorm") # a package for multivariate normal distribution
```

- To use an installed package, type `library(xxx)`.

```
> # Draws from the bivariate standard normal distribution
> rmvnorm(1, mean=c(0,0), sigma=matrix(c(1,0,0,1), nrow=2)) # generates error

> library(mvtnorm)
> rmvnorm(1, mean=c(0,0), sigma=matrix(c(1,0,0,1), nrow=2))

          [,1]     [,2]
[1,] 0.2674600 1.082827
```

Package	Description	Pre-Installed?
`boot`	Functions for bootstrap analysis	Yes
`foreign`	Read data files created in other softwares, e.g., Stata, SAS, SPSS.	Yes
`MASS`	A large collection of useful functions by Venables & Lipley (2002)	Yes
`MCMCpack`	Bayesian analysis via Markov chain Monte Carlo	No
`mvtnorm`	Multivariate normal and t distributions.	No
`pcse`	Panel-corrected standard errors by Beck & Katz (1995)	No
`Zelig`	Easy-to-use frontend for a range of statistical models by Imai, King & Lau	No

Table 1: List of Add-on Packages Often Used in Political Science.

1.10 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 Working with Data

2.1 Loading Workspace and 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 Reading Data Created in Other Softwares

- The `foreign` package must be loaded to read files created in other softwares.
- `read.dta()` reads a Stata data file (`xxx.dta`)
- `read.spss()` reads a SPSS data file (`xxx.sav`)

```
> library(foreign)
> Humph <- read.dta("Humph.dta") # Data on African Countries by Humphreys
> BPCD <- read.spss("BPCD.sav") # British Parliament Constituency Data by Norris
```

- To read an Excel spreadsheet file (`xxx.xls`), the simplest way is to open it in Excel, save it as a .csv file and read it with `read.csv()`.
  (Note: The `gdata` package contains the `read.xls()` function, which can directly read an Excel file. However, you need to have the Perl language installed to get it to work.)

2.3 Looking at Data

- `names()` gives variable names in the data frame
- `nrow()`, `ncol()` and `dim()` work for data frames in the same as they do for matrices
- Also like matrices, data frames can be indexed by `[,]`
- Use `$` to access an individual variable
- `summary()` works for data frames too

```
> names(Africa)

[1] "Country" "GDP"     "GDP.pc"  "HDI"     "Region"

> dim(Africa)

[1] 52  5

> Africa[1:4,]  # Display the first four rows, and all columns

   Country    GDP GDP.pc  HDI          Region
1  Algeria 298448   8649 0.73 Northern Africa
2   Angola  91825   5463 0.45  Central Africa
3    Benin  12217   1507 0.44  Western Africa
4 Botswana  28454  18402 0.65 Southern Africa

> Africa[3:8, c(2,4)]  # Display rows 3-8, columns 2 and 4
```

```
    GDP  HDI
3 12217 0.44
4 28454 0.65
5 22132 0.37
6  5913 0.41
7 45777 0.53
8  4271 0.74
```

```
> Africa$HDI  # Displays only the HDI values
```

```
 [1] 0.73 0.45 0.44 0.65 0.37 0.41 0.53 0.74 0.38 0.39 0.56 0.43 0.41 0.52 0.71
[16] 0.64 0.48 0.41 0.68 0.50 0.55 0.46 0.37 0.52 0.55 0.33 0.82 0.53 0.44 0.38
[31] 0.55 0.80 0.65 0.38 0.65 0.37 0.47 0.55 0.45 0.65 0.50 0.84 0.34 0.67 0.53
[46] 0.55 0.47 0.51 0.77 0.51 0.43 0.51
```

```
> summary(Africa)
```

```
         Country        GDP              GDP.pc           HDI        
 Algeria     : 1   Min.   :   616   Min.   :  500   Min.   :0.3300  
 Angola      : 1   1st Qu.:  6178   1st Qu.: 1366   1st Qu.:0.4300  
 Benin       : 1   Median : 19654   Median : 2162   Median :0.5100  
 Botswana    : 1   Mean   : 61118   Mean   : 4616   Mean   :0.5294  
 Burkina Faso: 1   3rd Qu.: 55461   3rd Qu.: 5569   3rd Qu.:0.6425  
 Burundi     : 1   Max.   :703709   Max.   :23294   Max.   :0.8400  
 (Other)     :46                                                    
            Region
 Central Africa : 9
 Eastern Africa :16
 Northern Africa: 6
 Southern Africa: 5
 Western Africa :16
```

2.4 Logical, Conditional Statements and Subsetting

- Logical operators: `<`, `<=`, `>`, `>=`, `==` and `!=`
- an object class of `logical` (i.e., `TRUE` or `FALSE`) is returned
- "==" versus "="
- Works with vectors too

```
> 4 > 3
```

```
[1] TRUE
```

```
> "Hello" == "hello"
```

```
[1] FALSE
```

```
> x <- 4 > 3
> x

[1] TRUE

> class(x)

[1] "logical"

> x <- c(3, 2, 1, -2, -1)
> x >= 2

[1]  TRUE  TRUE FALSE FALSE FALSE
```

- Combine logical statements with `&` (and) and `|` (or)

```
> x > 0 & x <= 2

[1] FALSE  TRUE  TRUE FALSE FALSE

> x > 0 | x <= 2

[1] TRUE TRUE TRUE TRUE TRUE
```

- Use logical statements to subset the data. Two important arguments of `subset()` are `subset` (for observations) and `select` (for variables)

```
> Africa[Africa$Region == "Northern Africa",]

   Country    GDP GDP.pc  HDI          Region
1  Algeria 298448   8649 0.73 Northern Africa
15   Egypt 423464   5643 0.71 Northern Africa
27   Libya  93402  15041 0.82 Northern Africa
33 Morocco 198785   6406 0.65 Northern Africa
45   Sudan 129447   3395 0.53 Northern Africa
49 Tunisia 107185  10269 0.77 Northern Africa

> subset(Africa, subset = (Region == "Northern Africa"))

   Country    GDP GDP.pc  HDI          Region
1  Algeria 298448   8649 0.73 Northern Africa
15   Egypt 423464   5643 0.71 Northern Africa
27   Libya  93402  15041 0.82 Northern Africa
33 Morocco 198785   6406 0.65 Northern Africa
45   Sudan 129447   3395 0.53 Northern Africa
49 Tunisia 107185  10269 0.77 Northern Africa

> Africa[(Africa$GDP.pc >= 8000) & (Africa$Region != "Northern Africa"), ]
```

```
             Country    GDP GDP.pc  HDI          Region
4           Botswana  28454  18402 0.65 Southern Africa
8         Cape Verde   4271   8481 0.74  Western Africa
16 Equatorial Guinea  26428  21316 0.64  Central Africa
32         Mauritius  19015  14954 0.80  Eastern Africa
35           Namibia  20100   9653 0.65 Southern Africa
42        Seychelles   1921  23294 0.84  Eastern Africa
44      South Africa 703709  14529 0.67 Southern Africa

> subset(Africa, subset = ((GDP.pc >= 8000) & (Region != "Northern Africa")))

             Country    GDP GDP.pc  HDI          Region
4           Botswana  28454  18402 0.65 Southern Africa
8         Cape Verde   4271   8481 0.74  Western Africa
16 Equatorial Guinea  26428  21316 0.64  Central Africa
32         Mauritius  19015  14954 0.80  Eastern Africa
35           Namibia  20100   9653 0.65 Southern Africa
42        Seychelles   1921  23294 0.84  Eastern Africa
44      South Africa 703709  14529 0.67 Southern Africa

> Africa.sub <- subset(Africa, select = c("Country", "GDP"))
> names(Africa.sub)

[1] "Country" "GDP"

> Africa.sub <- Africa[, c(1,2)]
> names(Africa.sub)

[1] "Country" "GDP"
```

- We can add up the number of `TRUE` statements using the `sum()` command

```
> x

[1]  3  2  1 -2 -1

> x >= 2

[1]  TRUE  TRUE FALSE FALSE FALSE

> sum(x>=2)  # Adds up the number of TRUE statements

[1] 2
```

- Conditional Statements evaluate a logical statement, then perform an action
- `ifelse(X, Y, Z)` performs `Y` if the statement `X` is true; performs `Z` if `X` is false

```
> regions
```

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

```
> ifelse(regions == "Africa", "Yes", "No")
```

```
[1] "Yes" "Yes" "No"  "No"  "Yes" "No"
```

2.5 Working with Missing Values

- Missing values in your dataset can cause you trouble, and sometimes mess up your analysis completely.
- R uses `NA` ("not available") to indicate missing values.
- When a calculation involves `NA`s, it often behaves in a rather counterintuitive manner. Examples:

```
> x <- c(0, 1, NA)
> x > 0 # Returns "NA" for NA, rather than "FALSE"
```

```
[1] FALSE  TRUE    NA
```

```
> x == NA # Returns "NA" for everything
```

```
[1] NA NA NA
```

- The `is.na()` function tells whether a value is `NA`.
- Many R functions have an argument `na.rm=`. Setting it to `TRUE` allows the function to automatically drop `NA`s.

```
> is.na(x) # Shows which element is missing
```

```
[1] FALSE FALSE  TRUE
```

```
> mean(x) # Returns NA
```

```
[1] NA
```

```
> mean(x, na.rm=T) # Omits missing values
```

```
[1] 0.5
```

- Other special values similar to `NA` include `NaN` ("not a number") and `Inf`.

```
> 1/0
```

```
[1] Inf
```

```
> 0/0
```

```
[1] NaN
```

```
> is.na(c(1/0, 0/0, NA)) # NaN is treated as NA; Inf is not
```

```
[1] FALSE  TRUE  TRUE
```

2.6 Exporting Data Created in R

- Once finished with manipulating a data frame, you might want to save it into a non-R file.
- The `write.table(x, file)` function saves the object `x` as an ASCII file, with its name specified by the `file` argument.
- Alternatively, you can save it into a csv file by `write.csv(x, file)`

```
> write.table(Africa.sub, file="africa_sub.txt")
> write.csv(Africa.sub, file="africa_sub.csv")
```