R and Stata Side-by-Side

Basic Commands and Processes

Created: August 2013

Author:
Richard Blissett

Author:
Benjamin Skinner

Department of Leadership, Policy & Organizations

Contents

1) Beginning the Work File

1.1 Commenting

It is important to annotate your code. This is for the benefit of others who will be reading your code as well as for yourself, as it's often difficult to keep track of what your code is doing. Comments are preceded by special characters, which indicate to the program that the comments are not to be evaluated as instructions. Comments will be used throughout this document to explain language-specific details.

R

```
# This is a comment
```

Stata

```
* This is a comment
/* This is also a comment */
// This, too, is a comment
```

1.2 Set Working Directory

Your statistical package needs to know where your files are in order to access them. It also needs a location to store your output should you save any. While it is possible to link to different locations throughout the file, it is a practice not recommended, especially if you are collaborating. Put all of your projects files—data, program file, output—in one folder.

R

```
setwd("~/working/directory/location") # For OSX
setwd("C:/Users/You/working/directory/location") # For Windows
```

Stata

```
cd "~/working/directory/location"  // OS X
cd "C:/Users/You/working/directory/location" // PC
```

1.3 Open a New Log

A log file is record of the input/output from the file that you run. This can be a useful diagnostic tool when some bit of code doesn't seem to be working or the file crashes entirely. In addition, saving new log files after making substantial changes to code essentially saves a history of your work such that you have files to refer to should you feel the need to revert to a previous code version.

R

```
# Strictly speaking, R does not have a simple way to store log
# files that contain both the code and code output. However, the
# two functions below will give you the same information.
# Save command history (all of the code that you ran, goes at the
```

Stata

```
# end of the code)
savehistory(file="myhistory.Rhistory")
# Save output to a file (goes at the beginning of the code)
sink(file="myoutput.txt")
```

```
// Opens log file (goes at beginning of code)
capture log using "projectname.log", replace
// Closes the current log file (goes at end of code)
log close
```

1.4 Clear Memory/Set Preferences

Because EVERYTHING in R is an object, clearing your workspace only requires you to clear all of the objects in your workspace. Since Stata has more automated features/preferences, you have to clear/(re)set more options.

R

```
# The ls() function returns a list of all objects, and the rm()
# command removes them.
rm(list=ls())
```

Stata

```
// Clear everything in memory: data, graphs, matrices,
// labels, etc.
clear all
```

1.5 Installing/Loading Libraries & Packages

Packages consist of commands/functions (plus associated help files). The base installations of R and Stata both come with a standard set of packages, but there are many others written by users that can be downloaded from the internet to extend the functionality of your statistical software. You only need to install the package one time; once installed, it is on your computer. The key difference is that to use the package in R, you need to load its library in your file before the command will work. Once you have the package in Stata, the command is now available like any other command without any extra steps.

R

```
# Download the package from the internet
install.packages("xtable")
# Load the package for use
library(xtable)
```

Stata

```
// Get list of possible installations
net search estout
// Or, direct installation
ssc install sutex
```

2) Reading and Viewing Data

2.1 Reading Data

Data must be read into the statistical software in a way that the latter understands. R stores the data in an object; therefore, R can hold multiple datasets in memory at the same time. Stata, on the other hand, can only access/hold one dataset at a time.

R

```
# Reads in the dataset. Other commands exist for other file types.
d <- read.csv("plans.csv")
# Makes this dataset the "primary" dataset
attach(d)
```

Stata

```
// Must clear other dataset that may be in memory
use plans, clear
```

2.2 Viewing Data

Raw data and information about your dataset as a whole can be viewed in a number of ways depending on what information you want. Both R and Stata have commands to view your dataset in a spreadsheet format. Both have the ability to edit data in this way as well: DON'T!

R

```
names(d)  # see list of variable names
View(d)  # open spreadsheet of dataset
```

Stata

```
describe  // general info about data
browse  // open spreadsheet of dataset
```

3) Manipulating Data

3.1 Keeping & Dropping Variables

Sometimes predefined datasets have more variables than you need for your project. Rather than trying to keep and re-load overly large datasets, which can slow your computer, you should drop unnecessary variables. You can go about this in two ways, either keeping what you want or dropping what you don't.

R

```
# Copy a subset of the data, with these variables, into a new object
newd <- d[c("byses1", "byses2", "bynels2m", "bynels2r")]
# Overwrite copy with full original data
```

Stata

```
// Keep only what you want...
keep byses1 byses2 bynels2m bynels2r
```

```
newd <- d
# Refer to all variables using the $ symbol. If only using one
# dataset, this can be avoided by using attach() on the dataset
# object so that you don't have to reference it anymore.
# Remember to reattach the data if you modify it.
newd$bynels2m <- NULL  # drop by making null
```

```
// Or, drop whatever you don't want...
use plans, clear  // need to reload dataset
drop by*
```

3.2 Summarizing Variables

Before you do anything to a variable or begin analyzing your data, it's a good idea to know about your variables: mean, min, max, etc.

R

```
summary(bysex)
```

Stata

```
tab bysex
```

3.3 Recoding Variables

Sometimes you might want to recode your variable. For instance, your data may code `sex` so that male = 1 and female = 2, but you think that it makes more sense to have males = 0 and females = 1. Recoding allows you to change this. Note that missing values are denoted by `NA` in R and by . and the empty string in Stata.

R

```
# This is the raw way to do it - the revalue() function in the plyr
# package is much cleaner
d$bysex[d$bysex==-8|d$bysex==-4] <- NA
d$bysex[d$bysex==1] <- 0
d$bysex[d$bysex==2] <- 1
```

Stata

```
// Recode and label
recode bysex (1 = 0) (2 = 1) // order matters here
label define sex 0 "Male"
label define sex 1 "Female", add
label values bysex sex
```

3.4 Creating Variables

Your dataset may not contain a variable that you need for your analysis, but often you can create a new variable to meet your needs. For example, you may know the number of men and the number of women in the dataset, but need the total population. Simply creating a new variable that is the sum of the other two is what you need to do. Since data is sacrosanct (NEVER manually modify values), you can transform or modify existing variables in your code and create new variables as well.

Some common rules/guidelines for variable names: don't start them with a number, keep them to a reasonable length, keep a consistent capitalization style, don't use common programming words (like "sum"), and make names easily interpretable.

In the examples below, `bynsels2m` and `bynels2r` are math and reading scores, respectively.

R

```
test <- bynels2m + bynels2r
avtest <- test / 2
zavtest_magic <- (avtest - 36.72297) / 12.17733
```

Stata

```
generate test = bynels2m + bynels2r
generate avtest = test / 2
generate zavtest_magic = (avtest - 36.72297) / 12.17733
```

You may notice that both examples above make use of "magic numbers," i.e. numbers that were hard-coded into the file in order to create a new combined test z-score. Magic numbers, because they are entered by hand, encourage mistakes and are generally discouraged in programming. They will also need to be changed again should any of your values change or if you add additional observations. A better solution is below:

R

```
# The mean() and sd() functions calculate means and standard
# deviations, respectively
zavtest = (avtest-mean(na.omit(avtest)))/sd(na.omit(avtest))

# Note that R does not automatically exclude missing values from
# analysis. Use the na.omit() to specify the subset of data that
# does not include missing data.
```

Stata

```
sum avtest
return list // list of locals stored in memory after sum command
generate zavtest = (avtest - r(mean)) / r(sd)
```

3.5 Coded Missing Data

Your datasets will contain observations with coded missing data. This is different from data that are simply missing. Instead, the makers of the dataset have given non-responses a nonsensical value that would never normally arise. If you do not take these values into account, however, you may mistakenly create variables or use values that have been influenced by these numbers. Mean values are one such example.

R

```
hist(bynels2m)  # note 'fake' values
bynels2m[bynels2m==-8] <- NA
bynels2m[bynels2r==-8] <- NA
avtest2 = (bynels2m + bynels2r) / 2
zavtest2 = ((avtest2 - mean(na.omit(avtest2)))
               / sd(na.omit(avtest2)))

# Compare mean values of prior and new variable
summary(zavtest)
summary(na.omit(zavtest2)) # OR do...
```

Stata

```
hist bynels2m  // note 'fake' values
replace bynels2m = . if bynels2m == -8
replace bynels2r = . if bynels2r == -8
generate avtest2 = (bynels2m + bynels2r) / 2
sum avtest2  // sum to put locals in memory
generate zavtest2 = (avtest2 - r(mean)) / r(sd)

/* Compare mean values of prior and new variable */
sum zavtest
sum zavtest2
```

```
summary(zavtest2, na.rm=T)

# NB: R does not automatically exclude missing values from analysis.
# Use the na.omit() to specify the subset of data that does not
# include missing data.
```

4) Programming Logic

4.1 And/Or

In both R and Stata, & is used for AND statements and | (a pipe) is used for OR statements. They may be combine in various combinations, but be careful to use parentheses as necessary so that the logic follows exactly what you intend.

R

```
# AND
summary(avtest2[bysex==1 & byrace==3])
# OR
summary(avtest2[bysex==1 | byrace==3])
```

Stata

```
// AND
sum avtest2 if bysex == 1 & byrace == 3
// OR
sum avtest2 if bysex == 1 | byrace == 3
```

4.2 While

The `while` command is commonly used for creating loops that iterate a set number of times (though the condition can be anything that moves from one position to another). The condition of the while loop must be something that will eventually be untrue (or true) inside of the loop, or else you will end up with an infinite loop.

R

```
i <- 1
while(i<=10) {
  print(paste0("Now on iteration ", i))
  i <- i + 1
}
```

Stata

```
local i = 1
while `i' <= 10 {
    di "Now on iteration `i'"
    local i = `i' + 1
}
```

4.3 For/Forvalues

The `for` and `forvalues` (in Stata) commands are like the `while` command, but require fewer lines since the counter is, in a sense, built into the command structure. Unless you have a compelling reason to use a `while` command, these are just as useful for most tasks in R and Stata.

R

```
for(i in 1:10) {
  print(paste0("Now on iteration ", i))
}
```

Stata

```
forvalues i = 1/10 {
    di "Now on iteration ‘i’"
}
```

4.4 Lapply/Foreach (with Locals)

`foreach` loops are a subset of the FOR command used by Stata. They are useful for running through a list of variables. This can also be done using the `for()` function in R, combined with the `list()` function, but the `lapply()` function is preferred and is more efficient.

Stata is able to hold information in two ways other than generated variables. First, data may be stored temporarily in locals. Stata will hold the values in memory only as long as the work file/command/function runs; it will delete them after that. They are useful for storing information that may change (such as a list of variables). Properly set up, locals will allow you to make only one change to your work file that will propagate throughout the rest. Scalars in Stata hold numerical values only. They remain until memory is cleared (i.e. Stata is closed). R, due to its different logic, does not make use of locals.

R

```
varlist <- list(bysex, byrace, bymothed, byfathed, byincome)
lapply(varlist, summary)
```

Stata

```
foreach var in bysex byrace bymothed byfathed byincome {
    sum ‘var’ if ‘var’ > 0
}

local demo bysex byrace bymothed byfathed bystincome

foreach var of local demo {  // same as above
    sum ‘var’ if ‘var’ > 0
}
```

5) Visualizing Data

5.1 Graphs

There are many different graphics you can produce in R and Stata. These are just a few. Also, graphics in both languages can be extensively customized. Getting exactly what you want takes much practice and patience.

R

```
hist(avtest2)
plot(bynels2m, bynels2r)
```

Stata

```
hist avtest2
scatter bynels2m bynels2r)
```

6) Regression

6.1 OLS

The practical difference between R and Stata in terms of regression is that whereas Stata displays most information without intermediate steps, R does not. To see regression output in R, you must ask the program to explicitly display it.

R

```
# When operations are stored in objects in R, it by default does not
# display the results of the operation. To store the results AND
# show them, just wrap the entire line in parentheses.

(model1 <- lm(bynels2m ~ byses1))
summary(model1)
```

Stata

```
regress bynels2m byses1
```

6.2 Logistic

Other regression models in R and Stata may be run. A logistical regression example is given below.

R

```
glm(formula = bysex ~ byses1, family=binomial(logit), data=d)
```

Stata

```
logit bysex byses1
```

7) Output

The output in this document is produced in LaTeX format. However, similar output can be produced in rtf (Word-compatible) and csv (Excel-compatible) formats as well.

7.1 Descriptive Table

With a little effort, both R and Stata can produce useful descriptive tables. With more effort, they can be very professional looking. The two examples below are rough around the edges, but give two good examples of what is possible. Know that there are many other ways to create descriptive tables. Also note that each command below only produces the LaTeX code for the table; you will need to paste that code into a full document in order for the table to generate.

R

```
# Attach data
attach(d)
# Store summaries of score variables
mathsum <- summary(bynels2m)
readsum <- summary(bynels2r)
# Glue them together
sumtable <- cbind(mathsum, readsum)
# Change the column names
colnames(sumtable) <- c("Math Summary", "Read Summary")
# Load xtable library
library(xtable)
# Create xtable object
textble <- xtable(sumtable, caption="Math and Reading Summary")
# Output to file
print.xtable(textble, type="latex", file="sumtable_r.tex",
             append=FALSE, caption.placement="top")
```

Stata

```
// Need to install sutex
ssc install sutex
// Create table
sutex bynels2m bynels2r, minmax ///
                         title("Math and Reading Summary") ///
                         file(sumtable_stata.tex) ///
                         replace
```

7.2 Regression Table

Professional-looking regression tables can also be produced with a little practice. As with the descriptive table examples, the two below might need a bit more polish before public consumption.

R

Stata

```
library(texreg)
model1 <- lm(bynels2m ~ byses1)  # store model 1
model2 <- lm(bynels2m ~ byses1 + bynels2r)  # store model 2
# Output to file
regt1 <- texreg(list(model1, model2), file="mathreg.tex")
```

```
eststo: reg bynels2m byses1  // store model 1
eststo: reg bynels2m byses1 bynels2r  // store model 2
esttab est* using "mathreg.tex", ///
    b(2) ///
    se(2) ///
    width(\linewidth) ///
    booktabs ///
    page ///
    replace
```

7.3 Graphs

When printing a graph, make sure you know its ultimate use (print or screen) and size. Two basic file types are available for your figures: raster/bitmap (e.g. jpg, jpeg, gif) and vector (e.g. eps, pdf). Unless you have a compelling reason to use raster images, save and export your graphs as vector files. They are scalable, meaning that you won't loss image quality as the you increase the image size. The tradeoff is that vector files tend to be larger, as they have more information, and can sometimes be slower to load.

R

```
pdf("math.pdf")
hist(bynels2m)
dev.off()
```

Stata

```
hist bynels2m, name(math)
graph export math.pdf, name(math) replace
```

A) Full Side-by-side Code

Below is the full code taken from the sections above. Each side should run from top to bottom in its respective software package.

R

```
## START WORK FILE ###############################################

# NB: This is a comment




# set working directory
setwd("~/working/directory/location") # For OSX
setwd("C:/Users/You/working/directory/location") # For Windows

# save results
sink(file="myoutput.txt")

# clear memory
rm(list=ls())

# download the package from the internet (first time only)
install.packages("xtable") # comment out in subsequent runs

# load the package for use
library(xtable)

## READING/VIEWING DATA ##########################################

# read in the dataset
d <- read.csv("plans.csv")

# make this dataset the "primary" dataset in memory
attach(d)

# see list of variable names
names(d)
```

Stata

```
/* START WORK FILE **********************************************/

* NB: This is a comment
/* NB: This is also a comment */
// NB: This, too, is a comment

// set working directory
cd "~/working/directory/location"  // OS X
cd "C:/Users/You/working/directory/location" // PC

// opens log file
capture log using "projectname.log", replace

// clear memory
clear all

// get list of possible installations (first time only)
net search estout
// or, direct installation (first time only)
ssc install estout


/* READING/VIEWING DATA *****************************************/

// read in dataset (must clear other data in memory)
use plans, clear





// list of variable names and other info
describe
```

```r
# open spreadsheet of dataset
View(d)

## MANIPULATING DATA #############################################

# copy a subset of the data into a new object
newd <- d[c("byses1", "byses2", "bynels2m", "bynels2r")]

# overwrite copy with full original data
newd <- d

# drop variable by making it NULL
newd$bynels2m <- NULL

# summarize variable
summary(bysex)

# summarize variable as factor
summary(factor(bysex))

# recode variables
d$bysex[d$bysex==-8|d$bysex==-4] <- NA
d$bysex[d$bysex==1] <- 0
d$bysex[d$bysex==2] <- 1

# label values
d$bysex <- factor(d$bysex,
                  levels=c(0, 1),
                  labels=c("Male", "Female"))

# create new variables
d$test <- d$bynels2m + d$bynels2r
d$avtest <- d$test / 2
d$zavtest_magic <- (d$avtest - 36.72297) / 12.17733

# better way...
d$zavtest = (d$avtest-mean(na.omit(d$avtest)))/sd(na.omit(d$avtest))
```

```stata
// open spreadsheet of dataset
browse

/* MANIPULATING DATA ********************************************/

// keep only what you want...
keep byses1 byses2 bynels2m bynels2r

// or, drop whatever you don't want
use plans, clear  // need to reload dataset
drop by*

// summarize variable (table format)
use plans, clear  // need to reload dataset
tab bysex

// recode and label variables
recode bysex (1 = 0) (2 = 1) // order matters here
label define sex 0 "Male"
label define sex 1 "Female", add
label values bysex sex

// create new variables
generate test = bynels2m + bynels2r
generate avtest = test / 2
generate zavtest_magic = (avtest - 36.72297) / 12.17733

// a better way...
sum avtest
return list // list of locals stored in memory after sum command
```

```
# recode coded missing values
hist(d$bynels2m)  # note ‘fake’ values
d$bynels2m[d$bynels2m==-8] <- NA
d$bynels2m[d$bynels2r==-8] <- NA
d$avtest2 = (d$bynels2m + d$bynels2r) / 2
d$zavtest2 = ((d$avtest2 - mean(na.omit(d$avtest2)))
                        / sd(na.omit(d$avtest2)))

# reattach data
attach(d)

# compare mean values of prior and new variable
summary(zavtest)
summary(na.omit(zavtest2)) # OR do...
summary(zavtest2, na.rm=T)

## LOGIC #########################################################

# AND
summary(avtest2[bysex==1 & byrace==3])

# OR
summary(avtest2[bysex==1 | byrace==3])

# WHILE
i <- 1
while(i<=10) {
  print(paste0("Now on iteration ", i))
  i <- i + 1
}

# FOR
for(i in 1:10) {
  print(paste0("Now on iteration ", i))
}

# LAPPLY
```

```
generate zavtest = (avtest - r(mean)) / r(sd)

// recode coded missing values
hist bynels2m  // note ‘fake’ values
replace bynels2m = . if bynels2m == -8
replace bynels2r = . if bynels2r == -8
generate avtest2 = (bynels2m + bynels2r) / 2
sum avtest2  // sum to put locals in memory
generate zavtest2 = (avtest2 - r(mean)) / r(sd)

/* compare mean values of prior and new variable */
sum zavtest
sum zavtest2

/* LOGIC ******************************************************/

// AND
sum avtest2 if bysex == 1 & byrace == 3

// OR
sum avtest2 if bysex == 1 | byrace == 3

// WHILE
local i = 1
while ‘i’ <= 10 {
    di "Now on iteration ‘i’"
    local i = ‘i’ + 1
}

// FORVALUES
forvalues i = 1/10 {
    di "Now on iteration ‘i’"
}

// FOREACH
```

```r
varlist <- list(bysex, byrace, bymothed, byfathed, byincome)
lapply(varlist, summary)

## VISUALIZING DATA ##############################################

# histogram
hist(avtest2)

# two-way scatterplot
plot(bynels2m, bynels2r)

## REGRESSION ####################################################

# linear
(model1 <- lm(bynels2m ~ byses1))
summary(model1)

# logistic
glm(formula = bysex ~ byses1, family=binomial(logit), data=d)

## OUTPUT ########################################################

# Descriptive table

# store summaries of score variables
mathsum <- summary(na.omit(bynels2m))
readsum <- summary(na.omit(bynels2r))
# glue them together
sumtable <- cbind(mathsum, readsum)
# change the column names
colnames(sumtable) <- c("Math Summary", "Read Summary")
```

```stata
foreach var in bysex byrace bymothed byfathed byincome {
    sum ‘var’ if ‘var’ > 0
}

// ...with locals

local demo bysex byrace bymothed byfathed byincome
foreach var of local demo {  // same as above
    sum ‘var’ if ‘var’ > 0
}

/* VISUALIZING DATA ********************************************/

// histogram
hist avtest2

// two-way scatterplot
scatter bynels2m bynels2r)

/* REGRESSION **************************************************/

// linear
regress bynels2m byses1


// logistic
logit bysex byses1

/* OUTPUT ******************************************************/

// Descriptive table

// create table
sutex bynels2m bynels2r, minmax ///
                         title("Math and Reading Summary") ///
                         file(sumtable_stata.tex) ///
                         replace
```

```
# load xtable library
library(xtable)
# create xtable object
textble <- xtable(sumtable, caption="Math and Reading Summary")
# output to file
print.xtable(textble, type="latex", file="sumtable_r.tex",
             append=FALSE, caption.placement="top")

# Regression table

# load texreg library
library(texreg)

# store models
model1 <- lm(bynels2m ~ byses1)
model2 <- lm(bynels2m ~ byses1 + bynels2r)

# output to file
regt1 <- texreg(list(model1, model2), file="mathreg_r.tex")

# Graphs

pdf("math_r.pdf")
hist(bynels2m)
dev.off()

## CLOSE WORK FILE ###############################################

savehistory(file="myhistory.Rhistory") # save commands
sink() # end sinking
```

```
// Regression table

// store models
eststo: reg bynels2m byses1  // store model 1
eststo: reg bynels2m byses1 bynels2r  // store model 2

// output to file
esttab est* using "mathreg_stata.tex", ///
    b(2) ///
    se(2) ///
    width(\linewidth) ///
    booktabs ///
    page ///
    replace

// Graphs

hist bynels2m, name(math)
graph export math_stata.pdf, name(math) replace

/* CLOSE WORK FILE ********************************************/

log close
exit
```