Guide to Automating Your Work in Stata 2015-05-15

Scott Long and Bianca Manago

Part 1. Introduction to automation in Stata

This guide supplements materials from Long's *The Workflow of Data Analysis Using Stata* (WFDAUS) and class lectures to help you write code using automation. The guide shows how to use automation to create a table of descriptive statistics that would be appropriate for a research paper. We illustrate one format for the table and provide do-files with the code. As you read the guide, we encourage you to experiment with each command and customize the table of descriptive statistics you are producing. All exercises should be completed using do-files. When you finish this guide, you will be able to create a table of descriptive statistics that you can export to Word or Excel. And, most importantly, you will know how to customize the table as your needs change.

Stata commands and output are in **`this font`**. Commands might be preceded by "**`.`**" or "**`>`**" as they would be in the Stata Results window. Do not type these in your do-file or from the command line.

Why automation?

Why would you write your own program rather than use commands like **`codebook, compact`** or **`summarize`**?

```
. use wf-lfp, clear
(Workflow data on labor force participation \ 2008-04-02)

. codebook, compact

Variable   Obs Unique      Mean        Min       Max  Label
--------------------------------------------------------------------------------
lfp        753      2  .5683931          0         1  In paid labor force? 1=...
k5         753      4  .2377158          0         3  # kids < 6
<snip>
lwg        753    676  1.097115  -2.054124  3.218876  Log of wife's estimated...
inc        753    621  20.12897  -.0290001        96  Family income excluding...
--------------------------------------------------------------------------------

. sum

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         lfp |       753    .5683931    .4956295          0          1
          k5 |       753    .2377158     .523959          0          3
<snip>
         lwg |       753    1.097115    .5875564  -2.054124   3.218876
         inc |       753    20.12897     11.6348  -.0290001         96
```

These commands provide the numbers you want in your table, but not the format you would use in a research paper. To create a table, you could copy, paste, and reformat the results. Then, if a variable changes, you have to re-do the entire process. With automation, you create a template for a descriptive table that you can use with any dataset and that can be easily modified to change the format. When changes are made to your dataset, simply re-run the code, and you have an updated table! Here is the table we will create using automation:

```
Name        Mean       SD      Min      Max  Description
lfp        0.568    0.496    0.000    1.000  In paid labor force? 1=yes 0=no
k5         0.238    0.524    0.000    3.000  # kids < 6
k618       1.353    1.320    0.000    8.000  # kids 6-18
age       42.538    8.073   30.000   60.000  Wife's age in years
wc         0.282    0.450    0.000    1.000  Wife attended college? 1=yes 0=n
hc         0.392    0.488    0.000    1.000  Husband attended college? 1=yes
lwg        1.097    0.588   -2.054    3.219  Log of wife's estimated wages
inc       20.129   11.635   -0.029   96.000  Family income excluding wife's
```

Part 2. Display command (wfauto-02display.do)

The **display** command displays characters or the results of mathematical expressions. We use **display** to write the names and number in our table.

2.1. Using display to show text

If you type **display "***text***"** or its abbreviation **di "***text***"**, the content of *text* is sent to the results window. Importantly, *text* can include macros, which are discussed in detail later. For example:

```
. di "Hello! Welcome to automation."
Hello! Welcome to automation.

. di "We can display whatever we place between quotations"
We can display whatever we place between quotations
```

This is the simplest use of **display**.

2.2. Using display as a calculator

The **display** command can also be used as a calculator.

```
. di 5+1
6

. di exp(-.2134)
.80783294

. di 70*8
560
```

Note that we do not put quote marks around these expressions. You should modify these examples to see what happens if you put quote marks around the expressions. These are simple examples, but **display** is a very powerful calculator.

2.3. Using display to create columns in tables

The **_col(#)** option places text in a specific column. In the example below, each **display** command moves **"hello"** 5 columns to the right of the previous command:

```
. display _col(5) "hello"
    hello

. display _col(10) "hello"
         hello

. display _col(15) "hello"
              hello

. display _col(20) "hello"
                   hello
```

You can use multiple **_col(#)** options in the same command:

```
. di "Variable" _col(15) "Mean" _col(25) "SD" _col(36) "Label"
Variable      Mean      SD         Label

. di _col(5) "age" _col(15) "Mean = " 1.23328924 _col(35) "SD = " 23.4256
    age       Mean = 1.2332892    SD = 23.4256
```

In the next step we show why we didn't put the numbers within the quotes.

2.4. Formatting

In your table you will need to round the numbers. You could round them yourself, but it is easier and more consistent to let Stata do this. And, if you change your mind on how many digits, Stata can make the change quickly. (Remember, *never* round a number that has already been rounded!) Suppose I want the mean and SD to be rounded to two decimal digits. To do this, I use the format **`%-10.2f`** . Here is what each part6 of the format means (use **`help format`** for full details):

%	→	A format is being specified.
-	→	Results are left-aligned.
10	→	Place the number in a column that is 10 digits wide.
.2	→	Round to two decimal points.
f	→	Use a fixed format (**`help format`** for more information).

For example,

```
. di _col(5) "age" _col(15) ///
>   "Mean = " %-10.2f 1.23328924 _col(35) "SD = " %-10.2f 23.4256

    age       Mean = 1.23         SD = 23.43
```

If we had included the numbers within quotes, they would not have been rounded.

Part 3. Stata macros (wfauto-03macros.do)

Macros are abbreviations for text and numbers. Instead of typing the content of the macro, which can be 1000s of characters long, you simply type the name of the macro and Stata inserts the content of the macro. Macros are incredibly easy and useful. There are two types of macros: *local* and *global*. Global macros persist until you delete them. This can be very useful, but it can also result in do-files that are not robust. Accordingly, we focus on local macros. A local macro (hereafter, simply referred to as a local) created in a do-file disappears after the do-file is run. A local created in the Command window, is not available for use in a do-file.

3.1. Defining a local macro

To define a local, you type: **`local`** *name-of-local* **"***content***"** For example:

```
local varlist "lfp k5 k618 age wc hc lwg inc"
```

We can recall the contents of a local by entering the local's name between a grave accent **`** (under the escape key on American keyboards) and a single quotation mark **'** (near Enter). For example,

```
. di "The local varlist contains: `varlist'"

The local varlist contains: lfp k5 k618 age wc hc lwg inc
```

We can use the local **`varlist`** to specify the variables we want to summarize:

```
. summarize `varlist'

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         lfp |       753    .5683931    .4956295          0          1
          k5 |       753    .2377158     .523959          0          3
<snip>
         lwg |       753    1.097115    .5875564  -2.054124   3.218876
         inc |       753    20.12897     11.6348  -.0290001         96
```

3.2. Locals with long strings

Often you will need a long string of characters. A common example is a list of control variables to be used in various specifications of statistical models. Rather than typing the same names in each regression command, you can use a local. If the list of variables is longer than 80 characters, the end of the string will wrap, which reduces the legibility of your do-file. There are several ways to avoid this. Here is one approach:

```
. local hardshipvars "hsphone hsmortg hsevict hsutil hsmedical hsdoctor"
. local hardshipvars "`hardshipvars' hsdentist hsrent hsfood"
```

In the second local above, **`` `hardshipvars' ``** inserts the content from the first local. The result is:

```
. di "`hardshipvars'"
hsphone hsmortg hsevict hsutil hsmedical hsdoctor hsdentist hsrent hsfood
```

This can be confusing at first. Experiment with it using a do-file until you are sure you understand what is happening.

3.3. Setting options with locals

Locals are useful for specifying options for a command, particularly when you are running the same command many times. By using a local, all commands will have exactly the same options. If you want to change them, you only need to change them once. Locals are also invaluable for creating graphs that look similar. Here's a simple example:

```
. local opt_tab "cell miss nolabel chi2"

. tabulate wc hc, `opt_tab'

+-----------------+
| Key             |
|-----------------|
|    frequency    |
| cell percentage |
+-----------------+

      Wife |
  attended |   Husband attended
  college? |  college? 1=yes 0=no
1=yes 0=no |         0          1 |     Total
-----------+----------------------+----------
         0 |       417        124 |       541
           |     55.38      16.47 |     71.85
-----------+----------------------+----------
         1 |        41        171 |       212
           |      5.44      22.71 |     28.15
-----------+----------------------+----------
     Total |       458        295 |       753
           |     60.82      39.18 |    100.00

          Pearson chi2(1) = 213.1042   Pr = 0.000
```

```
. tabulate wc lfp, `opt_tab'

+-----------------+
| Key             |
|-----------------|
|    frequency    |
| cell percentage |
+-----------------+

      Wife |
  attended | In paid labor force?
  college? |      1=yes 0=no
1=yes 0=no |         0          1 |     Total
-----------+----------------------+----------
         0 |       257        284 |       541
           |     34.13      37.72 |     71.85
-----------+----------------------+----------
         1 |        68        144 |       212
           |      9.03      19.12 |     28.15
-----------+----------------------+----------
     Total |       325        428 |       753
           |     43.16      56.84 |    100.00

          Pearson chi2(1) =  14.7804   Pr = 0.000
```

3.4. Locals for formatting

Let's say that you are creating multiple tables in a do-file and want each table to have the same style. To do this, we save the desired width, number of decimal places, and so on in locals. For example,

```
* C == column
local C1 = 5    // set 1 of output begins at 5
local C2 = 20
local C3 = 30

* W == width
local W2 = 10   // width of both columns to be 10
local W3 = 10

* D == decimal
local D2 = 2    // set 2 has 2 decimal places
local D3 = 3    // set 3 has 3 decimal places
```

We use these locals to print the mean and standard deviation with formatting. We use **///** to continue our command across lines:

```
. di _col(`C1') "age" ///
>    _col(`C2') "Mean = " %-`W2'.`D2'f   1.23328924 ///
>    _col(`C3') "SD = "   %-`W3'.`D3'f  23.4256

    age            Mean = 1.23      SD = 23.426
```

While this may look complicated, it is very useful since we can easily make changes that are applied uniformly to all of the commands that follow. For example, if we decide we want 2 decimal points for the third set of results, we simply change the local to **local D3 = 2**.

3.5. Extended macros

Extended macro functions contain strings or numbers that are retrieved by Stata. Here is a simple example. For our table, we want to include each variable's variable label. The extended macro:

: variable label

retrieves this information:

```
. local varnm "age"
. local varlbl : variable label `varnm'

. display "The label for `varnm' is: `varlbl'"
The label for age is: Wife's age in years
```

See **`help extended macro`** for other macro functions. If you are trying to do something with locals and get stuck, review these functions and you might find an easy way to do something that would otherwise be very difficult.

3.6 Global macros

To be added in a future version of this document.

Part 4. Stata returns (wfauto-04returns.do)

After a command runs, Stata saves in memory information about what the command did, including both numerical results that have been displayed and other information related to the command. The information that is saved is referred to as "returns"—information that is returned to memory. We consider two types of returns: e-returns from estimation commands like **`regress`** and r-returns from non-estimation commands such as **`summarize`**. In general, you should extract information from returns immediately after the command is run so that other commands do not overwrite the information.

4.1 r-returns

`summarize` has the statistics we need for our table and saves the information in r-returns:

```
. local varnm age
. summarize `varnm'

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         age |       753    42.53785    8.072574         30         60
```

You can examine r-returns by entering the command **`return list`**:

```
. return list
scalars:
                  r(N) =  753
              r(sum_w) =  753
               r(mean) =  42.53784860557769
                r(Var) =  65.16645121641095
                 r(sd) =  8.072574014303674
                r(min) =  30
                r(max) =  60
                r(sum) =  32031
```

We can place the information from the returns into locals which are listed with **`display`**:

```
. local mn = r(mean)
. di `mn'
42.537849

. local sd = r(sd)
. di `sd'
8.072574
```

Adding this to our prior work, we can display means and SD's without having to type in the values of these statistics:

```
. local C1 = 5
. local C2 = 20
. local C3 = 30
. local W2 = 10
. local W3 = 10
. local D2 = 2
. local D3 = 3

. di  _col(`C1') "`varnm'" ///
>     _col(`C2') "Mean = " %-`W2'.`D2'f  `mn' ///
>     _col(`C3') "SD = "   %-`W3'.`D3'f  `sd'

    age            Mean = 42.54     SD = 8.073
```

Later we will loop over variables, retrieve the returns, and create our table.

4.2. e-returns

e-returns follow estimation commands, such as **regress**. These returns include scalars, matrices, and macros. We don't need e-returns for our table, but it is useful to see what they are. As a challenge, try to create a table the combines the returns from multiple estimation commands.

```
. regress age inc

      Source |       SS       df       MS              Number of obs =     753
-------------+------------------------------           F(  1,   751) =    2.59
       Model |  168.541498     1  168.541498           Prob > F      =  0.1078
    Residual |  48836.6298   751  65.0288014           R-squared     =  0.0034
-------------+------------------------------           Adj R-squared =  0.0021
       Total |  49005.1713   752  65.1664512           Root MSE      =   8.064

------------------------------------------------------------------------------
         age |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         inc |   .0406898   .0252746     1.61   0.108    -.0089276    .0903072
       _cons |    41.7188   .5875276    71.01   0.000     40.56541     42.8722
------------------------------------------------------------------------------

. ereturn list

scalars:
                  e(N) =  753
               e(df_m) =  1
               e(df_r) =  751
                  e(F) =  2.59179770676773
                 e(r2) =  .0034392594434477
               e(rmse) =  8.064043734600771
                e(mss) =  168.5414982219954
                e(rss) =  48836.62981651902
               e(r2_a) =  .0021122810938384
                 e(ll) =  -2639.282981593837
               e(ll_0) =  -2640.580094609168
               e(rank) =  2

macros:
            e(cmdline) : "regress age inc"
              e(title) : "Linear regression"
          e(marginsok) : "XB default"
                e(vce) : "ols"
             e(depvar) : "age"
                e(cmd) : "regress"
         e(properties) : "b V"
            e(predict) : "regres_p"
              e(model) : "ols"
          e(estat_cmd) : "regress_estat"
```

```
matrices:
                  e(b) :  1 x 2
                  e(V) :  2 x 2
```

Part 5. Loops (wfauto-05loops.do)

Loops execute one or more commands multiple times. With a loop, Stata sequences through a list of names or numbers. Each name or number is placed in a local. Then, a set of commands is run that can use the information in the local whose value changes each time through the loop. This loop is repeated for each element in the list. Here we use **`foreach`** loops, but Stata has other types of loops such as **`forvalues`** or **`while`**.

5.1. foreach loop using display and summarize

An easy way to see how loops operate is to execute a single command for multiple variables. The variable list is contained in a local and the command inside the loop is performed on each of the variables in the local. First, we define a list of variables:

```
. local varlist "lfp k5 k618 wc hc"
```

The **`foreach`** loop looks like this:

```
foreach varnm in `varlist' {
    display "`varnm'"
}
```

Note that the content of the loop is contained within curly brackets. The first time through the loop, local **`varnm`** is set to **`lfp`**, the next time through the loop to **`k5`**, and so on. Here are the results:

```
. foreach varnm in `varlist' {
  2.     display "`varnm'"
  3. }
lfp
k5
k618
wc
hc
```

Next, we pass the variable name to the **`summarize`** command:

```
. foreach varnm in `varlist' {
  2.     summarize `varnm'
  3. }

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         lfp |       753    .5683931    .4956295          0          1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
          k5 |       753    .2377158     .523959          0          3

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        k618 |       753    1.353254    1.319874          0          8

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
          wc |       753    .2815405    .4500494          0          1

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
          hc |       753    .3917663    .4884694          0          1
```

5.2. foreach loop creating rows of summary statistics

Now we incorporate what we learned about returns to create a list of means and standard deviations for each variable in the local **`varlist`**:

```
. foreach varnm in `varlist' {
  2.     quietly summarize `varnm' // compute statistics
  3.     local mn = r(mean) // grab returns
  4.     local sd = r(sd)
  5.     di  _col(`C1') "`varnm'" ///
  >         _col(`C2') "Mean = " %-`W2'.`D2'f  `mn' ///
  >         _col(`C3') "SD = "   %-`W3'.`D3'f  `sd'
  6. }

    lfp            Mean = 0.57      SD = 0.496
    k5             Mean = 0.24      SD = 0.524
    k618           Mean = 1.35      SD = 1.320
    wc             Mean = 0.28      SD = 0.450
    hc             Mean = 0.39      SD = 0.488
```

This is starting to look like a table. Next, we use **`display`** to add a header and arrange our columns of summary statistics more elegantly. We also include the minimum, maximum, and variable label.

```
. di  _col(`C1') "Variable" _col(`C2') "Mean"  _col(`C3') "SD " ///
>     _col(`C4') "Min"      _col(`C5') "Max"   _col(`C6') "Label"

Variable Mean    SD      Min     Max     Label

. foreach varnm in `varlist' {
  2.     quietly summarize `varnm'
  3.     local mn = r(mean)
  4.     local sd = r(sd)
  5.     local min = r(min)
  6.     local max = r(max)
  7.     local varlbl : variable label `varnm'
  8.     di  _col(`C1') "`varnm'" ///
  >         _col(`C2') %-`W'.`D3'f  `mn'  _col(`C3') %-`W'.`D3'f  `sd'  ///
  >         _col(`C4') %-`W'.`D3'f  `min' _col(`C5') %-`W'.`D3'f  `max' ///
  >         _col(`C6') "`varlbl'"
  9. }

lfp      0.568   0.496   0.000   1.000   In paid labor force? 1=yes 0=no
k5       0.238   0.524   0.000   3.000   # kids < 6
k618     1.353   1.320   0.000   8.000   # kids 6-18
wc       0.282   0.450   0.000   1.000   Wife attended college? 1=yes 0=no
hc       0.392   0.488   0.000   1.000   Husband attended college? 1=yes 0=no
```

In our text editor, we strip out the commands and get close to what we want for a table:

```
Variable Mean    SD      Min     Max     Label
lfp      0.568   0.496   0.000   1.000   In paid labor force? 1=yes 0=no
k5       0.238   0.524   0.000   3.000   # kids < 6
k618     1.353   1.320   0.000   8.000   # kids 6-18
wc       0.282   0.450   0.000   1.000   Wife attended college? 1=yes 0=no
hc       0.392   0.488   0.000   1.000   Husband attended college? 1=yes 0=no
```

Part 6. Improving the table (wfauto-06table.do)

This section is a more sophisticated version of the program created above.

6.1. List of variables to be included

We start by creating a local with the list of variables to appear in our table.

```
. local varset "lfp k5 k618 age wc hc lwg inc"
```

6.2. Locals for formatting preferences

We define locals with columns and formats as before. We organize the locals and include comments to improves the do-file’s legibility and to facilitate later changes:

```
* C == column where each statistics will start
local Cvar  = 1
local Cmn   = 8
local Csd   = 17
local Cmin  = 26
local Cmax  = 35
local Clbl  = 45

* W == width
local Wmn   = 8
local Wsd   = 8
local Wmin  = 8
local Wmax  = 8

* D == decimal digits
local Dmn   = 3
local Dsd   = 3
local Dmin  = 3
local Dmax  = 3

* L == labels for statistics (spaces help center names)
local Lvar "Name"
local Lmn  "    Mean"
local Lsd  "      SD"
local Lmin "     Min"
local Lmax "     Max"
local Llbl "Description"
```

Next, we set a local with the length at which we will truncate the variable label and create a local indicating that we want to list a header for the table. More about this later.

```
local VLlen = 32
local listheader yes
```

6.3. Using a loop, calculate means and return

We are now prepared to loop through the local **`varset`** and create a table of descriptive statistics. The loop is similar to before:

```
foreach varnm in `varset' {
    if "`listheader'"=="yes" {
        di _col(`Cvar') "`Lvar'" _col(`Cmn')  "`Lmn'"  _col(`Csd')  "`Lsd'" ///
           _col(`Cmin') "`Lmin'" _col(`Cmax') "`Lmax'" _col(`Clbl') "`Llbl'"
       local listheader "no" // we no longer need to list the header
    }
    qui sum `varnm'
    * place returns into locals
    local mean = r(mean)
    local stdv = r(sd)
    local min= r(min)
    local max= r(max)
    * retrieve variable label
    local varlbl : variable label `varnm'
    local varlblshort = abbrev("`varlbl'",`VLlen') // abbreviate label
    * display statistics with formatting
    di _col(`Cvar') "`varnm'"  ///
          _col(`Cmn')  %`Wmn'.`Dmn'f `mean'  _col(`Csd')  %`Wsd'.`Dsd'f `stdv' ///
          _col(`Cmin') %`Wmin'.`Dmin'f `min' _col(`Cmax') %`Wmax'.`Dmax'f `max' ///
          _col(`Clbl') "`varlblshort'"
}
```

The biggest addition is that we check if local **`listheader`** is **yes**. If it is, we add a header to our table and change the content of local **`listheader`** to **no** so that the next time through the loop we don't list the header. The resulting table is:

```
Name        Mean       SD      Min      Max  Description
lfp        0.568    0.496    0.000    1.000  In paid labor force? 1=yes 0=no
k5         0.238    0.524    0.000    3.000  # kids < 6
k618       1.353    1.320    0.000    8.000  # kids 6-18
age       42.538    8.073   30.000   60.000  Wife's age in years
wc         0.282    0.450    0.000    1.000  Wife attended college? 1=yes 0=n
hc         0.392    0.488    0.000    1.000  Husband attended college? 1=yes
lwg        1.097    0.588   -2.054    3.219  Log of wife's estimated wages
inc       20.129   11.635   -0.029   96.000  Family income excluding wife's
```

Part 7. Matrices and Excel (wfauto-07matrixexcel.do)

Sometimes it is useful to save your table in Excel where you can add formatting and move the table to a word processor. To do this, we save our Stata results to a matrix and move the matrix to Excel using the **putexcel** command. We start with using **putexcel** to create a spreadsheet you can use for planning names and labels.

7.1. Planning names and labels

We begin by writing the provenance of the results to the spreadsheet:

```
. putexcel B1 = ("Dataset: wf-lfp.dta") using `pgm'-varplan.xlsx, replace
file wfauto-07matrixexcel-varplan.xlsx saved
```

The **putexcel** writes **Dataset: wf-lfp.dta** in cell **B1** in the file **`pgm'-varplan.xlsx**, where the local **pgm** has the name of the do-file we are using. The **replace** option over-writes the file if it exists. Next we add more information:

```
. putexcel B2 = ("Author: `who'") using `pgm'-varplan.xlsx, modify
file wfauto-07matrixexcel-varplan.xlsx saved

. putexcel B3 = ("Date: `dte'") using `pgm'-varplan.xlsx, modify
file wfauto-07matrixexcel-varplan.xlsx saved

. putexcel B4 = ("Pgm: `pgm'.do") using `pgm'-varplan.xlsx, modify
file wfauto-07matrixexcel-varplan.xlsx saved
```

Each line of **putexcel** specifies a different cell where the information is written, where the letter indicates the column placement and the number indicates the row placement. The **modify** indicates that we want to modify an existing spreadsheet.

Next we create a local **varnms** of a list of variables in our dataset by using the **unab** command, and we use **display `varnms'** to see what it holds:

```
. unab varnms : _all

. display "`varnms'"
lfp k5 k618 age wc hc lwg inc
```

. Our loop starts by modifying the local **row**, indicating which row in the .xlsx file we want to write our information. The syntax **local ++<local name>** increases a numerical local by 1 unit for each loop iteration, meaning we can place our variable information one row below the last. We loop through the variables, retrieve the variable label, and write it to the Excel file:

```
. local irow = 6 // start names in this row

. foreach varnm in `varnms' {
  2.     local varlbl : variable label `varnm'
  3.     putexcel B`irow' = ("`varnm'") /// place in col B, row irow
   >              C`irow' = (`"`varlbl'"') /// place in col C, row irow
   >              using `pgm'-varplan.xlsx, modify
  4.     local ++irow // increment the irow by 1
  5. }
```

That's it! Open wfauto-07matrixexcel-varplan.xlsx and you have a spreadsheet you can use for planning names and labels.

7.2. A spreadsheet with descriptive statistics

We will now create a new .xlsx file of descriptive statistics. This section uses some advanced matrix operations. Refer to WFDAUS for discussion of matrices. We first add provenance information as in our last example:

```
putexcel B1 = ("Dataset: wf-lfp.dta") using `pgm'-descstats.xlsx, replace
putexcel B2 = ("Author: `who'") using `pgm'-descstats.xlsx, modify
putexcel B3 = ("Date: `dte'") using `pgm'-descstats.xlsx, modify
putexcel B4 = ("Pgm: `pgm'.do") using `pgm'-descstats.xlsx, modify
```

Next we obtain a list of variables and count them using the string function **`wordcount`** so we know how large to make the matrix. String functions are similar to extended macros. See **`help string functions`** for more information.

```
unab varnms : _all
local varnmsN = wordcount("`varnms'") // # of variables
```

We create a local **`statnms`** with the names of our statistics which are used to label our matrix. We count them and add locals to indicate which matrix column to place each type of statistic.

```
local statnms "Mean SD Min Max"
local statnmsN = wordcount("`statnms'")
local Cmn  = 1
local Csd  = 2
local Cmin = 3
local Cmax = 4
```

We create a matrix and add row names and column names:

```
. matrix stattable = J(`varnmsN',`statnmsN',-99)
. matrix rownames stattable = `varnms'
. matrix colnames stattable = `statnms'

. matlist stattable

             |      Mean         SD        Min        Max
-------------+--------------------------------------------
         lfp |       -99        -99        -99        -99
          k5 |       -99        -99        -99        -99
        k618 |       -99        -99        -99        -99
         age |       -99        -99        -99        -99
          wc |       -99        -99        -99        -99
          hc |       -99        -99        -99        -99
         lwg |       -99        -99        -99        -99
         inc |       -99        -99        -99        -99
```

Next we loop through the variables, run **`summarize`** for each variable, and populate the matrix with summary statistics:

```
. local ivar = 0

. foreach varnm in `varnms' {
  2.     local ++ivar
  3.     qui sum `varnm'
  4.     matrix stattable[`ivar',`Cmn']  = r(mean)
  5.     matrix stattable[`ivar',`Csd']  = r(sd)
  6.     matrix stattable[`ivar',`Cmin'] = r(min)
  7.     matrix stattable[`ivar',`Cmax'] = r(max)
  8. }
```

The matrix looks like this:

```
. matlist stattable, format(%8.4f) title(The stattable matrix)

The stattable matrix

             |     Mean        SD       Min       Max 
-------------+----------------------------------------
         lfp |   0.5684    0.4956    0.0000    1.0000 
          k5 |   0.2377    0.5240    0.0000    3.0000 
        k618 |   1.3533    1.3199    0.0000    8.0000 
         age |  42.5378    8.0726   30.0000   60.0000 
          wc |   0.2815    0.4500    0.0000    1.0000 
          hc |   0.3918    0.4885    0.0000    1.0000 
         lwg |   1.0971    0.5876   -2.0541    3.2189 
         inc |  20.1290   11.6348   -0.0290   96.0000 
```

We save the matrix to the spreadsheet, starting in cell B6:

```
local startrow = 6
putexcel B`startrow' = matrix(stattable, names) ///
    using `pgm'-descstats.xlsx, modify
```

Using the same approach as our first example, we add variable labels:

```
foreach varnm in `varnms' {
    local ++startrow
    local varlbl : variable label `varnm'
    putexcel G`startrow' = (`"`varlbl'"') /// export label
        using "`pgm'-descstats.xlsx", modify
}
```

You can open the matrix, format it, and paste it into Word:

	Mean	SD	Min	Max	
lfp	0.57	0.50	0.00	1.00	In paid labor force? 1=yes 0=no
k5	0.24	0.52	0.00	3.00	# kids < 6
k618	1.35	1.32	0.00	8.00	# kids 6-18
age	42.54	8.07	30.00	60.00	Wife's age in years
wc	0.28	0.45	0.00	1.00	Wife attended college? 1=yes 0=no
hc	0.39	0.49	0.00	1.00	Husband attended college? 1=yes 0=no
lwg	1.10	0.59	-2.05	3.22	Log of wife's estimated wages
inc	20.13	11.63	-0.03	96.00	Family income excluding wife's

With a little effort, you can create complex tables using these tools. The putexcel command also allows you to add formatting to the cells in the table. You can define column tabs in Word to adjust how columns are aligned. You can even have dynamic links so your Word document updates automatically each time you change the spreadsheet.

Part 8. Regression tables with estout (wfauto-08estout.do)

The Stata package **estout** by Ben Jann can make professional regression tables. We will create a simple table with three regression models and discuss a few of the many options you can use. The commands in the **estout** package have *many* options. To see what kinds of tables you can create, see http://repec.org/bocode/e/estout/. (Professor Jann's web site was created with a Stata do-file!)

8.1. Setup

We install the **estout** package (or **search estout** in Stata)

```
ssc install estout , replace
```

We set up locals for specifying regression models:

```
local demog     k5 k618 age
local college   i.wc i.hc
local earn      lwg inc

local model1 `demog'
local model2 `model1' `college'
local model3 `model2' `earn'
```

Using a loop, we estimate each model and store the results using **eststo**:

```
local mytable ""
forvalues i = 1(1)3 {
  quietly logit lfp `model`i''
  eststo results`i'
  local mytable "`mytable' results`i'"
}
```

After estimating the model with the variables contained in local **model1**, the estimates are stored with the name **results1**. We build the local **mytable** to contain the names of the stored results. You could add **display `i': `mytable'** at the end of the loop if you want to see what is happening. Next we display a table with **esttab**.

8.2. The esttab default table

The most basic syntax to create a regression table is to include the **esttab** command followed by the models of interest:

```
. esttab `mytable'

------------------------------------------------------------
                      (1)             (2)             (3)
                      lfp             lfp             lfp
------------------------------------------------------------
lfp
k5                 -1.369***       -1.450***       -1.463***
                  (-7.21)         (-7.54)         (-7.43)

k618               -0.126          -0.108          -0.0646
                  (-1.94)         (-1.63)         (-0.95)

age               -0.0683***      -0.0684***      -0.0629***
                  (-5.61)         (-5.51)         (-4.92)

0.wc                                   0               0
                                     (.)             (.)

1.wc                               0.883***        0.807***
                                  (4.09)          (3.51)

0.hc                                   0               0
                                     (.)             (.)

1.hc                              -0.120           0.112
                                 (-0.62)          (0.54)

lwg                                                0.605***
                                                  (4.01)

inc                                              -0.0344***
                                                 (-4.20)

_cons               3.688***        3.499***        3.182***
                   (6.20)          (5.73)          (4.94)
------------------------------------------------------------
N                     753             753             753
------------------------------------------------------------
t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001
```

Often, you will want to customize the table.

8.3. Customizing the table with labels

Below are our options to label variables, models, and tables.

```
esttab `mytable' , ///
  title("Table 1: Models of female labor force participation") ///
  mtitle("Demographics" "(1)+College" "(2)+Earnings") /// label models
  coef(                                  /// label variables in model
    k5       "Number of kids under 6"      ///
    k618     "Number of kids 6-18"         ///
    age      "Age"                         ///
    1.wc     "Wife attended college"       ///
    1.hc     "Husband attended college"    ///
    lwg      "ln(Wife's estimated wages)"  ///
    inc      "Family income"               ///
    _cons    "Constant"                    ///
    )                                      ///
  varwidth(30)                             ///
  nobase                                   ///
```

`title` adds a title for the table.

`mtitle` creates titles for each model in the table. The model titles are contained within quotation marks.

`coef` adds your own labels for each coefficient. Alternatively, the option **`label`** uses the variable labels.

`varwidth(30)` increases the width of the column holding variable labels, avoiding truncated labels.

`nobase` removes base categories of factor variables from your regression table. By default the omitted category (e.g. **`0.wc`**, **`0.hc`**) are included in your **`esttab`** output.

With these options, we now have a table that is almost complete.

Table 1: Models of female labor force participation

	(1) Demographics	(2) (1)+College	(3) (2)+Earnings
lfp			
Number of kids under 6	-1.369***	-1.450***	-1.463***
	(-7.21)	(-7.54)	(-7.43)
Number of kids 6-18	-0.126	-0.108	-0.0646
	(-1.94)	(-1.63)	(-0.95)
Age	-0.0683***	-0.0684***	-0.0629***
	(-5.61)	(-5.51)	(-4.92)
Wife attended college		0.883***	0.807***
		(4.09)	(3.51)
Husband attended college		-0.120	0.112
		(-0.62)	(0.54)
ln(Wife's estimated wages)			0.605***
			(4.01)
Family income			-0.0344***
			(-4.20)
Constant	3.688***	3.499***	3.182***
	(6.20)	(5.73)	(4.94)
N	753	753	753

t statistics in parentheses
* p<0.05, ** p<0.01, *** p<0.001

8.4. Adding sub-table text

We can add notes, measures of fit, and other scalars to the bottom of the table:

```
esttab `mytable' , ///
<snip>
  nolegend /// suppress significance legend, I'm adding it myself
  addnote("* p<0.05 ** p<0.01 *** p<0.001" ///
          "" /// blank line
          "`tag'") /// add notes below table
  scalar("ll Log lik.") bic pr2 /// show model fit statistics
```

`nolegend` removes the default note on significance levels. Since commas can cause problems when exporting to Excel, we create our own note without commas.

addnote creates notes below the regression table. A note is contained in quotation marks. A second note enclosed in quotation marks places a new line of notes. We add provenance to our table with this option using the local **tag**.

scalar() adds scalars from e-returns to the table. We grab the scalar **e(ll)** and name it **Log lik**. We also add the

bic adds the BIC; **pr2** adds the pseudo-R2.

You can see the table that is created by running the do-file.

8.5. Enhance table

Next, we add several additional options that we frequently use. See Jann's documentation for other ways to customize your table.

```
esttab `mytable' , ///
<snip>
  eform /// show odds ratios
  constant /// do not suppress constant
  stardrop(_cons , relax) ///
  b(3) /// decimal points for coefficients
  t(2) /// decimal points for test statistic
  indicate("College controls = 1.wc 1.hc" "Earnings controls = lwg inc") ///
  nobase
```

eform transforms the raw coefficients to odds ratios.

constant forces **esttab** to include the constant term which by default is dropped when **eform** is specified.

b(3) and **t(2)** indicate the number of decimal points for the coefficients and z-statistics, respectively.

stardrop removes significance indicators from the variables included inside the parentheses.

indicate is useful when you only want to show the coefficients for some variables. The bundle of variables **1.wc** and **1.hc** are clumped under the label **College controls**. The bundle of variables **lwg** and **inc** are clumped under the label **Earnings controls**. If **1.wc** and **1.hc** are included in a regression model, then **College controls** will have a **yes** in that regression model. If they aren't, then **College** will have a **no**. We could easily add footnotes and notes to indicate which variables are included in these labels with the **addnote** option.

nobase drops the reference category for factor variables. For example, with **i.wc**, both **0.wc** and **1.wc** are added to the model. We only want to see **1.wc** which is what happens with **nobase**.

With these final options, our regression table now looks like this:

Table 1: Models of female labor force participation

	(1) Demographics	(2) (1)+College	(3) (2)+Earnings
lfp			
Number of kids under 6	0.254***	0.235***	0.232***
	(-7.21)	(-7.54)	(-7.43)
Number of kids 6-18	0.881	0.898	0.937
	(-1.94)	(-1.63)	(-0.95)
Age	0.934***	0.934***	0.939***
	(-5.61)	(-5.51)	(-4.92)
Constant	39.950	33.092	24.098
	(6.20)	(5.73)	(4.94)
College controls	No	Yes	Yes
Earnings controls	No	No	Yes
N	753	753	753
pseudo R-sq	0.067	0.087	0.121
BIC	987.204	979.506	958.258
Log lik.	-480.354	-469.881	-452.633

Exponentiated coefficients; t statistics in parentheses
* p<0.05 ** p<0.01 *** p<0.001

wfauto-08estout scott long 2015-05-13

8.5 Export to Excel, Word, and LaTeX

We can export our tables in rtf format for Word, csv format for Excel, and tex format for LaTeX. To export to Word:

```
esttab `mytable' using `pgm'.rtf , replace ///  export to word document
  <snip rest of code from above>
```

An rtf file of our table is saved to our working directory. To insert our table into this Word document, we select the Insert tab; select Insert Object; select Create from File tab; and Browse for the RTF document in your working directory. Our new table is:

Table 1: Models of female labor force participation

	(1) Demographics	(2) (1)+College	(3) (2)+Earnings
lfp			
Number of kids under 6	0.254***	0.235***	0.232***
	(-7.21)	(-7.54)	(-7.43)
Number of kids 6-18	0.881	0.898	0.937
	(-1.94)	(-1.63)	(-0.95)
Age	0.934***	0.934***	0.939***
	(-5.61)	(-5.51)	(-4.92)
Constant	39.950	33.092	24.098
	(6.20)	(5.73)	(4.94)
College controls	No	Yes	Yes
Earnings controls	No	No	Yes
N	753	753	753
pseudo R^2	0.067	0.087	0.121
BIC	987.204	979.506	958.258
Log lik.	-480.354	-469.881	-452.633

Exponentiated coefficients; *t* statistics in parentheses
* p<0.05 ** p<0.01 *** p<0.001

wfauto-08estout scott long 2015-05-13

Conclusions

To learn more, see *The Workflow of Data Analysis Using Stata* by Long, *Introduction to Stata Programming* by Kit Baum, and Stata's NetCourse 151.

stata-automation-guide 2015-05-15.docs