

Stata Programming Webinar

From typing commands to reusable Stata programs

June 2026 Public Webinar

What we will discuss

- Think about a practical project layout for Stata work
- A backup plan that combines backup and version history
- A mental model for `do`, `ado`, Mata, and plugins
- Several programming tips: handling file paths, passing inputs, returning results, and Mata matrix subscripting

The talk materials are at <https://github.com/huapeng01016/webinar2026>

Audience

This webinar is for Stata users who can run commands and do-files, and now want their analysis projects to be easier to rerun, review, extend, and share.

1. Plan the project

1. Protect the project: data privacy, code privacy, backup, and file sharing
2. Choose the right programming surface: `do`, `ado`, Mata, and plugins
3. Deal with the project's folder structure, input and output files, and absolute and relative paths
4. Move values through code deliberately: local macros, global macros, scalars, passed arguments, and returned results
5. Design output

Sample project layout

```
data/    # small public sample datasets
src/     # Stata do-files, ado files, and helper scripts
docs/    # slides, notes, and supporting documentation
```

The layout should make the first rerun boring in the best possible way. It should also support moving the project to a different machine or sharing it with a collaborator.

2. Protect the work

Every project needs two safety nets:

- Backup recovers a lost or deleted file
- Version control explains what changed, when, and why
- GitHub adds review, collaboration, issues, and release history

Use both. They solve different problems. Data privacy requirements can complicate the situation.

Backup is not revision control

| Question | Backup | Git/GitHub |
|---------------------------------------|-----------|------------|
| Can I meet data privacy requirements? | Maybe | Maybe |
| Can I recover the file? | Yes | Usually |
| Can I compare two versions? | Sometimes | Yes |
| Can I branch an experiment? | No | Yes |
| Can collaborators review changes? | No | Yes |
| Can I explain the project history? | No | Yes |

3. Programming tools

A good Stata setup makes structure visible:

- Use an editor with Stata syntax highlighting
- Keep code, data, and documents in predictable folders
- Within the project's data privacy requirements, use AI coding agents for scaffolding, review, and refactoring
- Treat generated code like a student's draft: inspect and test it

4. Choose the right Stata surface

| Surface | Best Use | Effort |
|-------------------|---|--------|
| <code>.do</code> | Reproducible analysis and reporting scripts | Low |
| <code>.ado</code> | Reusable commands that behave like Stata commands | Medium |
| Mata | Matrix-heavy or performance-sensitive work | Medium |
| Plugin | Compiled extensions for performance and specialized needs | High |

5. First example: a do-file to produce a Table 1

`src/ex1_table1.do` introduces a common reporting task:

- Load public NHANES II data with `webuse`
- Build descriptive statistics with `dtable`
- Export a first table to HTML
- Refine the output with `collect`
- Export a formatted table to various formats

5.1. Code snippet 1

```
*/! 1.0.0 18/may/2026
```

```
version 18
```

```
/*  
 * Include a table of descriptive statistics for data from the  
 * Second National Health and Nutrition Examination Survey  
 * (NHANES II) (McDowell et al. 1981).  
 *  
 * Use -dtable- to create the table and export it to many formats.  
 */
```

```
use ../data/nhanes2l.dta, clear
```

```
/*  
 * Format the means and standard deviations to two decimal places, add  
 * a title, and export the final table to an HTML file:  
 */  
dtable age weight bpsystol i.sex i.race,      ///  
      by(diabetes, nototals tests)           ///  
      continuous(age, test(none))            ///  
      factor(race, test(none))                ///  
      sample(, statistics(freq) place(seplabels)) ///  
      column(by(hide))                       ///  
      sformat("(N=%s)" frequency)            ///  
      nformat(%7.2f mean sd)                 ///  
      note(Total sample: N = 10,349)          ///  
      title(Table 1. Demographics)            ///  
      export(../docs/table1.html, replace)
```

5.2. Dissection of the code

- Date and file version - `*/! 1.0.0 18/may/2026`
- Version control - `version 18`
- File and folder navigation - `../nhanes2l.dta`, `../docs/table1.html`
- Line continuation - `///`
- Comments

5.3. Version control

```
stata-output
. do ex1_table1

. *! 1.0.0 18/may/2026
.
. version 18
this is version 17.0 of Stata; it cannot run version 18.0 programs
    You can purchase the latest version of Stata by visiting
https://www.stata.com.
r(9);
```

```
stata_code
/*
 * Stata's table command went through major changes in Stata 17,
the following
 * syntax no longer works in 17 without specifying version 16
 */
sysuse auto
version 16
table rep78, contents(n mpg)
```

```
version 18
capture table rep78, contents(n mpg)
version 16 : table rep78, contents(n mpg)
```

```
stata_output
. sysuse auto
(1978 automobile data)

. version 16

. table rep78, contents(n mpg)
```

```
-----
Repair      |
record      |
1978        |      N(mpg)
-----+-----
          1 |          2
          2 |          8
          3 |         30
          4 |         18
          5 |         11
-----
```

```
.
. version 18
```

```
. capture noi table rep78, contents(n mpg)
option contents() not allowed since Stata 17; see help table for
updated syntax
```

```
. version 16 : table rep78, contents(n mpg)
```

| ----- | | |
|-------------|---|--------|
| Repair | | |
| record | | |
| 1978 | | N(mpg) |
| -----+----- | | |
| | 1 | 2 |
| | 2 | 8 |
| | 3 | 30 |
| | 4 | 18 |
| | 5 | 11 |
| ----- | | |

5.4. Organize folders and files

For file and path names:

- Use forward slashes as separators; backslashes fail on macOS and Linux
- Use only lowercase English letters and underscores in file names; capital letters can cause problems on macOS and Linux
- Double-quote paths with spaces

5.5. Double-quote or compound double-quote paths with spaces

```
stata-output
. cd "hua peng"
c:\talks\webinar2026\hua peng

. do test_path

. display "Test file path current working directory = |`c(pwd)'"
Test file path current working directory =
|c:\talks\webinar2026\hua peng|

. do `c(pwd)'/test_path.do
file c:\talks\webinar2026\hua.do not found
r(601);

. do "`c(pwd)'/test_path.do"

. display "Test file path current working directory = |`c(pwd)'"
Test file path current working directory =
|c:\talks\webinar2026\hua peng|
```

```
. do ``c(pwd)'/test_path.do''

. display "Test file path current working directory = |`c(pwd)'"
Test file path current working directory =
|c:\talks\webinar2026\hua peng|
```

5.6. Another common pitfall when a tempfile path contains a space

```
stata-code
/* Temp directory is at c:/Users/Hua Peng/AppData/Local/Temp */
tempfile f
save `f'
/* Fails with a cryptic error message
* invalid 'Peng'
* r(198);
* It attempted to issue something like:
* save c:/Users/Hua Peng/AppData/Local/Temp/ST_93bc_000001.tmp
*/

/* Correct way */
save ``f'``'
```

5.7. File and path navigation

Paths in do-files can be relative or absolute. If the project's folder is `c:\talks\webinar2026`, the layout looks like:

```
c:/talks/webinar2026/
  data/
    nhanes21.dta
  src/
    ex1_table1.do
  docs/
    table1.html
    table1.pdf
    table1.md
```

Using a relative path in `ex1_table1.do` requires Stata's current working directory to be `c:/talks/webinar2026/src`, so the following works:

```
stata_code
cd "c:/talks/webinar2026/src"
do ex1_table1.do
```

The following fails:

```
stata_code
cd "c:/talks/webinar2026/"
do src/ex1_table1.do
```

Using an absolute path avoids this problem, but it makes the project difficult to run on other machines without modification. Therefore, a relative path is still the preferred method.

5.8. One way (of many ways) to simplify navigating folders on a machine

Use an ado program, which Stata can automatically find and load, and a global macro.

```
stata_code
*! 1.0.0 18/may/2026

program define waypoint
    version 16

    args fasttravel

    local talks "C:/talks/"
    local webinar "C:/talks/webinar2026/"

    if "`fasttravel'" == "" {
        // set all waypoints
        global waypoint_talks "`talks'"
        global waypoint_webinar "`webinar'"
    }
    else if "`fasttravel'" == "list" {
        display "waypoint:"
        display "waypoint_talks: |$waypoint_talks|"
        display "waypoint_webinar: |$waypoint_webinar|"
    }
    else if inlist("`fasttravel'", "talks", "webinar") {
        global waypoint_`fasttravel' "`fasttravel'"
        cd "${waypoint_`fasttravel'}"
    }
    else {
        di in red "Unknown argument `fasttravel'"
        error 198`
    }
    exit
end
```

To change directory to `webinar2026/src`

```
stata_output
. waypoint webinar
C:\talks\webinar2026

. cd src
C:\talks\webinar2026\src
```

```
. cd "$waypoint_webinar/src"
C:\talks\webinar2026\src

. waypoint list
waypoint:
waypoint_talks: |C:/talks/|
waypoint_webinar: |C:/talks/webinar2026/|
```

The disadvantage is that this is a machine-specific solution: change the code if you change machines or projects. The advantage is that the change is likely to be infrequent.

5.9. Use a file for the absolute path and use `-include-` in a do-file

Use `global_path.do` to store the project's absolute path in a macro.

```
stata_code
*! 1.0.0 18/may/2026

/*
 * Absolute path to the project
 */

local webinar2026 "c:/talks/webinar2026"
```

In the do-file, use `include global_path.do`.

```
stata_code
*! 1.0.0 18/may/2026

version 18
/*
 * Include project path
 */

include global_path.do

use "`webinar2026'/data/nhanes21.dta", clear
```

See `help include` for details. The `adopath` option is useful if multiple projects need to share common include files.

5.10. Keep comments updated with code

It is important to comment the code. It is more important to keep comments accurate when the code changes.

5.11. Be aware of system values and global settings

Stata has settable system parameters; see `help set` for more information. Depending on the setting, a command's behavior may change.

```
stata_code
. clear

. set obs 5
Number of observations (_N) was 0, now 5.

. gen id_row = _n

. summ id

      Variable |      Obs      Mean   Std. dev.      Min
-----+-----
      id_row |         5         3   1.581139         1
5

. set varabbrev off

. summ id
variable id not found
r(111);
```

6. Make a do-file more flexible

The second example:

```
stata_code
*! 1.0.0 18/may/2026

version 18
/*
Include a table of descriptive statistics for data from the
Second National Health and Nutrition Examination Survey
(NHANES II) (McDowell et al. 1981).

Use -dtable- to create the table and export it to many formats.
*/

/*
syntax:

do ex4_table1.do dir(string) name(string) replace

options:
    dir      : directory to store exported files
```

```

name      : filename without extension for exported
files
replace: replace if exported files already exist
*/

local 0 "", `0'"
syntax [, dir(string asis)      ///
        name(string)            ///
        replace]

if ``"dir'" == "" {
    local dir "../docs"
}

local export_file "`name'"
if ``"export_file'" == "" {
    local export_file "table1"
}

di in red "|`dir'/'`name'| |`replace'| "

use ../data/nhanes21.dta, clear

/*
 * Check variables
 */

confirm numeric variable age weight bpsystol sex race diabetes

/*
 * Check if output files exist
 */
if "`replace'" == "" {
    confirm new file "`dir'/'export_file'.html"
    confirm new file "`dir'/'export_file'.docx"
    confirm new file "`dir'/'export_file'.md"
    confirm new file "`dir'/'export_file'.pdf"
}

/*
 * Get sample size
 */
count if !missing(diabetes)
local total_sample = strofreal(`r(N)', "%9.0gc")

/*
 * Format the means and standard deviations to
 * two decimal places, add a title, and export
 * the final table to an HTML file:
 */
dtable age weight bpsystol i.sex i.race,      ///
        by(diabetes, nototals tests)          ///

```

```

        continuous(age, test(none))          ///
        factor(race, test(none))            ///
sample(, statistics(freq) place(seplabels)) ///
        sformat("(N=%s)" frequency)        ///
nformat(%7.2f mean sd)                     ///
        note(Total sample: N = `total_sample')    ///
        column(by(hide))                      ///
        title(Table 1. Demographics)           ///
        export("`dir'/'`export_file'.html", `replace')

/*
* Use -collect- to change the color of the borders
* and the background color for alternating rows
* in our table. Then export it to different formats.
*/

* Change the border color above the corner and column headers to
cyan
collect style cell border_block[column-header corner], border(top,
color(cyan))

* Change the border color above and below the results and row
headers to cyan
collect style cell border_block[row-header item], border(bottom,
///
        color(cyan)) border(top, color(cyan))

* Make the column headers bold
collect style cell cell_type[column-header], font(, bold)

/*
* Change the background color to cyan for the rows corresponding
to N,
* BMI, and cholesterol
*/
collect style cell var[_N bmi tcresult], shading(background(cyan))

/*
* Finally, we specify that the width of the columns be resized to
* fit the table contents and export the table:
*/

collect style putdocx, layout(autofitcontents)

collect export `dir'/'`export_file'.docx, `replace'
collect export `dir'/'`export_file'.md, `replace'
collect export `dir'/'`export_file'.pdf, `replace'

* end

```

6.1. Pass arguments to a do-file

```

stata_code
*! 1.0.0 18/may/2026

version 18

display "arguments:"
display `"'arg0 = |`0'|"'
display `"'arg1 = |`1'|"'
display `"'arg2 = |`2'|"'
display `"'arg3 = |`3'|"'
display `"'arg4 = |`4'|"'

stata_output
. do ex3_args arg1 " arg 2 " arg3(100) 3.14

. *! 1.0.0 18/may/2026
.
. version 18
.
. display "arguments:"
arguments:

. display `"'arg0 = |`0'|"'
arg0 = |arg1 " arg 2 " arg3(100) 3.14|

. display `"'arg1 = |`1'|"'
arg1 = |arg1|

. display `"'arg2 = |`2'|"'
arg2 = | arg 2 |

. display `"'arg3 = |`3'|"'
arg3 = |arg3(100)|

. display `"'arg4 = |`4'|"'
arg4 = |3.14|

```

See <https://www.stata.com/support/faqs/programming/passing-arguments-to-do-files/> for more information.

6.2. Use local macros and arguments to avoid code modification

```

stata_code
do "ex4_table1.do" name(doc3)

```

6.3. Parse command syntax with -syntax-

```

stata_code
local 0 ", `0'"
syntax [, dir(string asis)      ///
        name(string)          ///
        replace]

if "`dir'" == "" {
    local dir "../docs"
}

local export_file "`name'"
if "`export_file'" == "" {
    local export_file "table1"
}

```

6.4. Validate inputs before expensive or destructive work

```

stata_code
confirm numeric variable age weight bpsystol sex race diabetes

/*
 * Check if output files exist
 */
if "`replace'" == "" {
    confirm new file "`dir'/'export_file'.html"
    confirm new file "`dir'/'export_file'.docx"
    confirm new file "`dir'/'export_file'.md"
    confirm new file "`dir'/'export_file'.pdf"
}

```

Now, omitting the `replace` option will produce an error if the files already exist.

```

stata_output
. do "ex4_table1.do" name(table1)

. (output omitted...)

. if "`replace'" == "" {
.     confirm new file "`dir'/'export_file'.html"
file ../docs/table1.html already exists
r(602);
.     confirm new file "`dir'/'export_file'.docx"
.     confirm new file "`dir'/'export_file'.md"
.     confirm new file "`dir'/'export_file'.pdf"
. }
r(602);
```stata

```

See <https://blog.stata.com/2026/05/13/essential-tools-for-data-quality-checks/> for more examples.

## 7. Write ado program

- An ado program is a reusable command that extends Stata's functionality by adding new commands or modifying existing ones
- It must be located on your ado path, which contains directories defined in `sysdir`. Directories can be added to or removed from the ado path using `adopath`
- The file must have the same name as the command; for example, the `regress` program must be defined in `regress.ado`

See <https://blog.stata.com/tag/stataprogramming/> for a series of blog posts about Stata programming.

### 7.1. An example ado program to estimate the mean of a single variable by the sample average

```
stata_code
*! version 1.0.0 20may2026

/*
 * Compute the sample average and its estimated sampling variance,
 * assuming an iid process
 */
program define mymean, eclass
 version 18

 syntax varname [if] [in]

 marksample touse

 tempvar e2
 tempname b V
 quietly summarize `varlist' if `touse'
 local sum = r(sum)
 local N = r(N)
 matrix `b' = (1/`N')*`sum'
 quietly {
 generate double `e2' = 0
 replace `e2' = (`varlist' - `b'[1,1])^2 if `touse'
 summarize `e2' if `touse'
 }
 local sum = r(sum)
 local N = r(N)
 matrix `V' = (1/((`N')*(`N'-1)))*r(sum)
 matrix colnames `b' = `varlist'
 matrix colnames `V' = `varlist'
 matrix rownames `V' = `varlist'
 ereturn post `b' `V'
 di as smcl "{txt}{col 1}Mymean estimation{col 45}{lalign
```

```

13: Number of obs {col 58} = {res}{ralign 2: `N'}`"
 ereturn display, nopvalues
end

stata_output
. mymean mpg
Mymean estimation Number of obs = 74

 | Coefficient Std. err. [95% conf. interval]
-----+-----
 mpg | 21.2973 .6725511 19.97912 22.61547

. mymean mpg if foreign
Mymean estimation Number of obs = 22

 | Coefficient Std. err. [95% conf. interval]
-----+-----
 mpg | 24.77273 1.40951 22.01014 27.53532

```

## 7.2. Return results

Return values let the next command use the previous command's work:

- Use `return` for r-class results
- Use `ereturn` for estimation commands
- Store scalars, macros, and matrices with clear names
- Check outputs with `return list` or `ereturn list`
- Document what callers can rely on

## 7.3. Numeric precision and floating-point operations

Floating-point operations (FPO) perform arithmetic calculations on real numbers that include decimal points. Since computers can represent only finitely many real numbers, even basic arithmetic in computing differs from mathematics. For example, `a + b + c` may not equal `b + c + a`.

```

stata_output
. do ex6_precision.do

. *! version 1.0.0 24may2026
.
. clear

. set obs 3
Number of observations (_N) was 0, now 3.

. gen double x = 0.1

```

```
. replace x = 0.2 in 2
(1 real change made)
```

```
. replace x = 0.3 in 3
(1 real change made)
```

```
. gen double y = 0.2
```

```
. replace y = 0.3 in 2
(1 real change made)
```

```
. replace y = 0.1 in 3
(1 real change made)
```

```
. list
```

```

+-----+
| x y |
+-----+
1. | .1 .2 |
2. | .2 .3 |
3. | .3 .1 |
+-----+
```

```
. gen double sum_x = sum(x)
```

```
. gen double sum_y = sum(y)
```

```
. list
```

```

+-----+
| x y sum_x sum_y |
+-----+
1. | .1 .2 .1 .2 |
2. | .2 .3 .3 .5 |
3. | .3 .1 .6 .6 |
+-----+
```

```
. capture noi assert sum_x == sum_y in 3
assertion is false
```

```
.
. gen double diff = sum_x - sum_y in 3
(2 missing values generated)
```

```
. list
```

```

+-----+
| x y sum_x sum_y diff |
+-----+
1. | .1 .2 .1 .2 . |
```

|         |  |    |    |    |    |           |  |
|---------|--|----|----|----|----|-----------|--|
| 2.      |  | .2 | .3 | .3 | .5 | .         |  |
| 3.      |  | .3 | .1 | .6 | .6 | 1.110e-16 |  |
| +-----+ |  |    |    |    |    |           |  |

And in Python:

```
python_code
a1 = 0.1
a2 = 0.2
a3 = 0.3

s1 = a1+a2+a3
s2 = a2+a3+a1

if s1 == s2:
 print("s1 equals s2")
else:
 d = s1-s2
 print(f"diff = {d}")

runfile('C:/Users/speaker/zone1.py', wdir='C:/Users/speaker')
diff = 1.1102230246251565e-16
```

See <https://blog.stata.com/2012/04/02/the-penultimate-guide-to-precision/> for a great discussion about precision. Two more tips with precision:

- Use the `-double-` type when generating a datetime variable
- Use the system variable `-c(obs_t)-` when generating an ID variable. `-c(obs_t)-` returns a string equal to the optimal data type for storing `_n`; it ensures that the ID variable goes from 1 to `_N` without roundoff errors and without wasting space

```
stata_code
gen double dt = 0
generate `c(obs_t)' ind = _n
```

## 8. Use Mata with Stata

Mata is Stata's matrix programming language. It can be used as a matrix calculator or as the computational engine for new commands.

### 8.1. Use Mata as a matrix calculator

```
stata_output
. sysuse auto, clear
(1978 automobile data)

. mata:
```

```

----- mata (type end to
exit) -----

: /* Create mata matrix from Stata data */
: y = st_data(., "mpg")

: X = st_data(., "price weight trunk")

:
: /* Compute X'X */
: XTX = cross(X, X)

:
: /* Compute inverse of X'X */
: XTXi = invsym(XTX)

:
: /* Compute b */
: b = XTXi*cross(X, y)

: b
 1
+-----+
1 | -.0000461482 |
2 | .0040395724 |
3 | .5121996632 |
+-----+

: end

.
. quietly regress mpg price weight trunk, nocons

. mat list e(b)

e(b)[1,3]
 price weight trunk
y1 -.00004615 .00403957 .51219966

.
. mata:
----- mata (type end to
exit) -----

: st_b = st_matrix("e(b)")

: mrelidif(b, st_b')
5.65317e-15

```

```
: end
```

```



```

## 8.2. Consider using -st\_view()- if memory is a concern

```
stata_code
. sysuse auto, clear
(1978 automobile data)

. mata:
----- mata (type end to
exit) -----

: /* Create mata view into Stata data */
: st_view(y=., ., "mpg")

: st_view(X=., ., "price weight trunk")

:
: /* Compute X'X */
: XTX = cross(X, X)

:
: /* Compute inverse of X'X */
: XTXi = invsym(XTX)

:
: /* Compute b */
: b = XTXi*cross(X, y)

: b
1
+-----+
1 | -.0000461482 |
2 | .0040395724 |
3 | .5121996632 |
+-----+

: end

.
.
quietly regress mpg price weight trunk, nocons

. mat list e(b)
```

```

e(b)[1,3]
 price weight trunk
y1 -.00004615 .00403957 .51219966

.
. mata:
----- mata (type end to
exit) -----

: st_b = st_matrix("e(b)")

: mreldif(b, st_b')
 5.65317e-15

: end

```

### 8.3. Know views' limitations and pitfalls

See `help mf_st_view` for a detailed discussion. Also note that a view is based on variable position; if variable positions change, the underlying view may silently change.

```

stata_output
. mata:
----- mata (type end to
exit) -----

: /* Create mata view into Stata data */
: st_view(y=., ., "mpg")

: st_view(X=., ., "price weight trunk")

:

: X1 = st_data(., "price weight trunk")

:

: Xcopy = X

: /* X changes, but Xcopy and X1 stay the same */
: stata("drop headroom")

:

: assert(Xcopy == X1)

: assert(Xcopy == X)
 assert(): 3498 assertion is false
 <istmt>: - function returned error
(1 line skipped)

```

## 8.4. Use Mata in an ado file

```
stata_code
*! 1.0.0 25may2026

program mvarsum
 version 18
 syntax varname [if] [in]
 marksample touse
 mata: calcsun("`varlist'", "`touse'")
 display as txt " sum = " as res r(sum)
end

version 18.5
mata:
void calcsun(string scalar varname, string scalar touse)
{
 real colvector x

 st_view(x, ., varname, touse)
 st_numscalar("r(sum)", colsum(x))
}
end

stata_output
. sysuse auto, clear

. mvarsum price
 sum = 456229

. ret list

scalars:
 r(sum) = 456229

macros:
 r(fn) : "C:\Program
Files\Stata18\ado\base/a/auto.dta"

. mvarsum make
 sum = 0
```

See `help m1_ado` for more details.

## 8.5. Add error handling

We change the Mata function to exit with an error if all elements in the matrix are missing.

```

stata_code
version 18.5
mata:
void calcsun(string scalar varname, string scalar touse)
{
 real colvector x
 real scalar nms

 st_view(x, ., varname, touse)

 nms = missing(x)
 if(nms == rows(x)*cols(x)) {
 exit(error(2000))
 }
 else {
 st_numscalar("r(sum)", colsum(x))
 }
}
end

stata_output
. mvarsum2 make
no observations
r(2000);

. mvarsum2 price
sum = 456229

```

See `help mf_error` for more information.

## 9. Other language support

- Python
- Java
- C/C++

Some considerations when choosing which language to use in a project:

- Both Python and Java can be used directly within Stata, interactively like `-mata:-`, through `-python:-` and `-java:-`
- Python can be used bidirectionally; you can call Stata from within Jupyter Notebook through PyStata
- A Stata installation includes a JVM, so you do not need to install one separately
- C/C++ plugins offer strong performance and can behave seamlessly as part of Stata

## 10. Recap

Key takeaways:

- Use a predictable project layout so code, data, and output are easy to find and rerun
- Combine backups with version control; they protect against different kinds of failure
- Choose the right Stata surface for the task: do-files for workflows, ado files for reusable commands, Mata for matrix-heavy work, and plugins when compiled performance is needed
- Pass inputs, validate assumptions, and return results deliberately so programs are easier to reuse and test
- Be careful with file paths, global settings, numeric precision, and Mata views because small details can change results

Thanks!