It is all about the data

Maarten L. Buis
maarten.buis@uni.kn

Finding your way around your data
- Searching
- Getting an overview
- Describing variables
- Describing variables (special cases)

Working with your data

Workflow
- Workflow
- Settings for your project
- Between program and .do file

Working with ID variables
- getting at parts of a variable
- typing large numbers
- Even and odd

Adding more meta data
- notes
- char

Some commands can be slow
- collapse can be slow
- reshape can be slow

- References

Finding your way around your data

Searching

The **lookfor** command is helpful for quickly finding the relevant variables in your dataset

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

. lookfor foreign

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin
```

lookfor only looks in the variable names and variable labels

Sometimes I know a term that should appear in the value labels

That is what **lookfor2** (available from SSC) is for

```
. lookfor2 domestic

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin
```

Once I have found a variable, I often want to see what variables are close

This is what **closedes** is for (it is part of the **lookfor2** package)

```
. closedesc headroom

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
rep78           int     %8.0g                 Repair record 1978
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
```

```
length          int     %8.0g                 Length (in.)
```

sometimes I want to look for variables by other characteristics. For example, they have a certain value label, i.e. they are part of the same batery of questions.

```
. ds, has(type numeric)
price         headroom      length        gear_ratio
mpg           trunk         turn          foreign
rep78         weight        displacement

. ds, has(vallabel "*origin*")
foreign
```

index >>

Finding your way around your data

Getting an overview

For getting a quick list of variables the **desc** command is useful

A similar command is **codebook** together with the **compact** option (the default is too verbose for my taste)

```
. codebook, compact

Variable      Obs Unique      Mean    Min     Max  Label
-------------------------------------------------------------------------------
make           74     74         .      .       .  Make and model
price          74     74  6165.257   3291   15906  Price
mpg            74     21   21.2973     12      41  Mileage (mpg)
rep78          69      5  3.405797      1       5  Repair record 1978
headroom       74      8  2.993243    1.5       5  Headroom (in.)
trunk          74     18  13.75676      5      23  Trunk space (cu. ft.)
weight         74     64  3019.459   1760    4840  Weight (lbs.)
length         74     47  187.9324    142     233  Length (in.)
turn           74     18  39.64865     31      51  Turn circle (ft.)
displacement   74     31  197.2973     79     425  Displacement (cu. in.)
gear_ratio     74     36  3.014865   2.19    3.89  Gear ratio
foreign        74      2  .2972973      0       1  Car origin
-------------------------------------------------------------------------------

. desc

Contains data from C:\Program Files\Stata17\ado\base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            12                  13 Apr 2020 17:45
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
```

The **htmlcb** package (available from SSC) can be used getting a more detailed description of the data.

```
. htmlcb, saving(cb.html) replace
Output written to cb.html
```

<< index >>

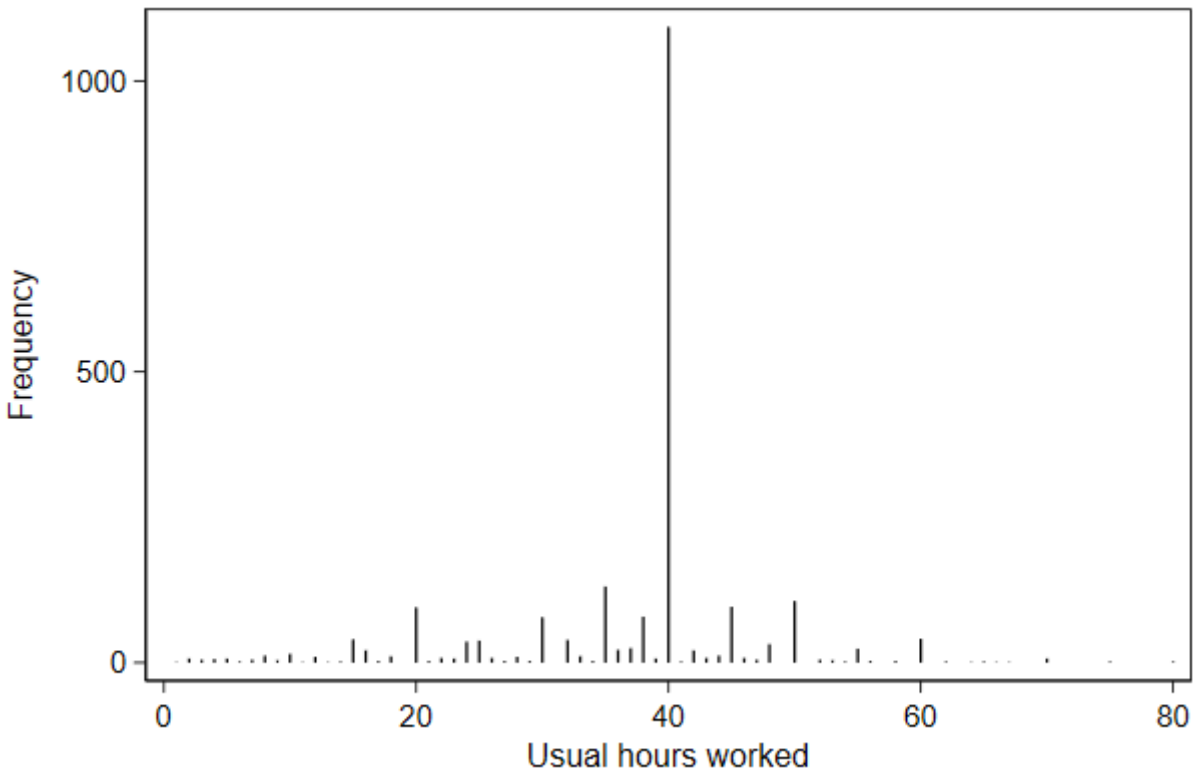
Finding your way around your data

Describing variables

An ugly, but extremely useful, graph for getting a good overview of a variable is **spikeplot**

```
. sysuse nlsw88
(NLSW, 1988 extract)

. spikeplot hours
```



For categorical variables the **fre** command (from SSC by Ben Jann) is just absolutely necessary

```
. fre union

union -- Union worker
------------------------------------------------------------------
                    |      Freq.    Percent      Valid       Cum.
--------------------+---------------------------------------------
Valid   0 Nonunion  |        1417      63.09      75.45      75.45
        1 Union     |         461      20.53      24.55     100.00
        Total       |        1878      83.62     100.00
Missing .           |         368      16.38
Total               |        2246     100.00
------------------------------------------------------------------
```

Finding your way around your data

Describing variables (special cases)

Sometimes I have a number of variables that together represent a concept

For those cases I find **groups** (from SSC by Nick Cox) extremely helpful

```
. groups married never_married

  +--------------------------------------------------+
  | married      never_married   Freq.   Percent |
  |--------------------------------------------------|
  |  Single   Has been married     570     25.38 |
  |  Single      Never married     234     10.42 |
  | Married   Has been married    1442     64.20 |
  +--------------------------------------------------+


. tab married never_married

           |     Never married
   Married | Has been   Never mar |     Total
-----------+----------------------+----------
    Single |       570        234 |       804
   Married |     1,442          0 |     1,442
-----------+----------------------+----------
     Total |     2,012        234 |     2,246
```

With two variables we can get away with **tab**, but what if I also had variables living_together, and divorsed?

For panel data I sometimes want to see the entire history of a person if a certain condition is met.

In this example we are looking at the retrospectively reported educational history of a respondent

And we find some persons who after primary school immediately entered a university of applied sciences.

```
. use education.dta
(cleaned up spell data from NEPS)


. lany ID_t sort start finish if start == 8 & sort == 2, by(ID_t) sepby(ID_t)
(2 real changes made)
(8 real changes made)


     +-----------------------------------------+
```

	ID_t	sort	start	finish
36.	8000814	1	enter Gym	finish Ab
37.	8000814	2	enter Fac	finish Fa
38.	8000985	1	enter Gru	finish Re
39.	8000985	2	enter Fac	finish Fa
40.	8001265	1	enter Gym	finish Ab
41.	8001265	2	enter Fac	finish Fa
42.	8008101	1	enter Gru	finish Re
43.	8008101	2	enter Fac	finish Fa
44.	8008488	1	enter Rea	finish Re
45.	8008488	2	enter Fac	finish Fa
46.	8010928	1	enter Gru	finish Ha
47.	8010928	2	enter Fac	finish Fa
48.	8011551	1	enter Gym	finish Ab

<< index >>

Workflow

An excellent resource for improving ones workflow in Stata is (Long 2009)

The mkproject package from SSC can help setting up a new projct:

> It creates the different folders and initial .do files with a commands in them that most .do files should have (sometimes called boilerplate code)
>
> It also creates a project, which you can fill with files so you can have easy access to them.
>
> Additional .do files with the standard commands already in them can be created witht the boilerplate command, which is part of **mkproject**

If you typed **mkproject proj**, it would create the following folders and files:

```
proj/
├─ proj.stpr
├─ admin/
├─ docu/
   ├─ research_log.txt
├─ posted/
|  ├─ data/
├─ work/
|  ├─ proj_main.do
|  ├─ proj_ana01.do
|  ├─ proj_dta01.do
```

After that it will open the do-file editor with the project open like so

The newest version of **mkproject** also has a **git** option that will setup a git repository.

It als has a **smclpres** option, for setting up the folder structure and a .do file with boilerplate code for a **smclpres** presentation like this.

Settings for your project

Larger projects need to be split up over different files

But I often need some settings throughout the project, that could be
some file names,
a list of variables, or something like that

I want to define them once, but I don't want to use globals

Instead, I typically have one .do file that defines my settings as locals, and use **include** at the top of all my other .do files

Between program and .do file

You can run a .do file called child.do by typing **do child.do**

You can also run that .do file by typing **do child.do something else**

This will make the following local macros available at the beginning of child.do:

`0' containing: something else

`1' containing: something

`2' containing: else

child.do could do some complicated/fiddly manipulations of a variable, maybe multiple levels of **if** conditions

You want to apply those to multiple variable, say **var1** and **var2**

If you were programming you would have put it in a sub-routine.

Now you can create another .do file (e.g. parent.do) that contains the lines:

```
foreach var of varlist var2 var2 {
    do child.do `var'
}
```

To me this is the ".do file equivalent" of a sub-routine.

Inside child.do I would start with **args** in order to give the local macros `1', `2', etc. more meaningful names

getting at parts of a variable

Say we have an ID variable where the eight digit is the continent, the seventh and sixth digit the country within that continent, the fifth digit the cite, fourth digit the village, and the first three the household within the village.

You want the country, i.e. the last three digits

If I divide by 100,000, then I am getting close; all I need to do is get rid of the bit after the decimal point, i.e. round down

That can be done with the **floor()** function

```
gen country = floor(ID/100000)
```

typing large numbers

If I have to type large numbers, like 100,000, I alway worry that I miscount the zeros

Instead one can also type 1e5

So, for the previous slide I would actually write:

```
gen country = floor(ID/1e5)
```

Even and odd

Sometimes I work with data where the odd observations in a unit are the origins and the following even observations are the destinations

so I want to find out if a number is even or odd

```
gen origin = mod(sort,2)
```

Working with your data -- Adding more meta data

---

notes

Notes can be attached to the data and to variables

You can add multiple notes to the same variable or dataset

notes can be longer than labels

```
. sysuse auto
(1978 automobile data)

. note

_dta:
  1.  From Consumer Reports with permission

. note foreign : this is american data, so foreign is non-US

. note: this is example data that comes with Stata

. note

_dta:
  1.  From Consumer Reports with permission
  2.  this is example data that comes with Stata

foreign:
  1.  this is american data, so foreign is non-US

. note foreign
```

<< index >>

---

char

A char is a more general version of notes

notes are numbered, but chars can have any name

```
. char rep78[Description] repair status

. char list rep78[]
  rep78[Description]:         repair status
```

For people making data available to others, and who want to add meta data to the data there are two options:

They could add the meta data as **notes**. This makes it probably a bit easier for potential users to find that information, but at the same time can interfere with the user who may also want to use notes.

They could add the meta data as **chars**. This is less likely to interfere with how the user may want to use the data, but makes it a bit harder for the user to find that information.

Some commands can be slow

---

collapse can be slow

for some datasets commands like **collapse** can be uncomfortably slow

```
. set rmsg on
r; t=0.00 9:19:08

. import delimited using "time_series_covid19_confirmed_global.csv", clear
(1,028 vars, 289 obs)
r; t=0.78 9:19:09

. drop lat v4
r; t=0.00 9:19:09

. collapse (sum) v*, by(countryregion)
r; t=10.38 9:19:19

. set rmsg off
```

The **gtools** package (from SSC, by Mauricio Caceres Bravo) contains various commands that are faster than the official Stata version.

```
. set rmsg on
r; t=0.00 9:19:19

. import delimited using "time_series_covid19_confirmed_global.csv", clear
(1,028 vars, 289 obs)
r; t=0.35 9:19:20

. drop lat v4
r; t=0.00 9:19:20

. gcollapse (sum) v*, by(countryregion)
r; t=5.22 9:19:25

. set rmsg off
```

<< index >>

Some commands can be slow

---

reshape can be slow

Similarly, **reshape** can be painfully slow

```
. set rmsg on
r; t=0.00 9:19:25

. import delimited using "time_series_covid19_confirmed_global.csv", clear
(1,028 vars, 289 obs)
r; t=0.21 9:19:25

. drop lat v4
r; t=0.00 9:19:25

. reshape long v, i(provincestate countryregion) j(date)
(j = 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 3
> 1 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
> 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
>  83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 1
> 06 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 12
> 5 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
>  145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
> 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 1
> 83 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 20
> 2 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
>  222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
> 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 2
> 60 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 27
> 9 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298
>  299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317
> 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 3
> 37 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 35
```

fastreshape (available from SSC, by Michael Droste) is faster

```
. set rmsg on
r; t=0.00 9:20:33

. import delimited using "time_series_covid19_confirmed_global.csv", clear
(1,028 vars, 289 obs)
r; t=0.28 9:20:33

. drop lat v4
r; t=0.00 9:20:33

. fastreshape long v, i(provincestate countryregion) j(date)
pos: 0, len: 1
(note: j = 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
> 9 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
> 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
>  81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104
>  105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123
```

```
> 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 1
> 43 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 16
> 2 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
>  182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
> 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 2
> 20 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 23
> 9 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258
>  259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277
> 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 2
> 97 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 31
> 6 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335
```

tolong (available from SSC, by Rafal Raciborski) is even faster, but less general

```
. set rmsg on
r; t=0.00 9:20:42

. import delimited using "time_series_covid19_confirmed_global.csv", clear
(1,028 vars, 289 obs)
r; t=0.37 9:20:42

. drop lat v4
r; t=0.00 9:20:42

. tolong v, i(provincestate countryregion) j(date)
r; t=0.42 9:20:42

. set rmsg off
```

[<< index]

References

J. Scott Long (2009), *The workflow of data analysis using Stata*. College Station, TX: Stata Press.

---

index
