Programming, Data Management and Visualization

Module A: **Elementary concepts and data organization**

Alexander Ahammer

Department of Economics, Johannes Kepler University, Linz, Austria
Christian Doppler Laboratory Ageing, Health, and the Labor Market, Linz, Austria

γ version, final
Last updated: Monday 12th October, 2020 (13:27)

A.1

Introduction and opening remarks

Introduction

- Programming is nothing more than writing **codes** $\longrightarrow$ a succession of commands that can be executed using software.

- Before we cover how to program loops, merge data, make fancy tables, and write an estimation command; we discuss some preliminaries:
  - How to set up and organize a project
  - How to make your work replicable for others
  - Data types and memory
  - How to import and export data

- For now, the only thing you need is a net-aware version of Stata running on your computer. I use version 16, but any v between 12 and 16 is fine.

- Make sure you keep Stata updated. With `ssc install` *`command`* you can download user-written commands from the SSC library.

Introduction

Do-files

- Codes for Stata are called **do-files.** They are nothing else than text files, although we use Stata's built-in do-file editor to write and edit them.
- In contrast to most external editors, the do-file editor allows you to execute only a section of the do-file.
- I advise coding with Stata on one half of your monitor and the do-file editor on the other half. To execute, press
  - CTRL + D in Windows
  - ⌘ + Shift + D on Mac

  If you code a lot, get a second monitor.
- ado-files are similar, they allow you to write programs for tasks you often perform (we may have a small section on ado-file programming later on).

Exemplary data

- In module A we work mainly with `pdmv_sl.dta`. This is a data extract from the *Austrian Social Security Database* taken from a 2017 paper of mine.[1]
- These data contain all sick leaves between 2004–2012 for a 10% sample of Upper Austrian employees. Everything is anonymized.
- The unit of observation is a single sick leave spell, thus it is a worker–sick leave panel. Covariates are measured at the beginning of the sick leave.
- The dataset is password protected, you have to sign a form first which requires you, amongst other things, not to share the data with others and to delete the dataset after the semester. `[Link to DB folder]`
- Check the data at home and familiarize yourself with their structure and particularities. Requires at least Stata version 12.

[1] "Physicians, sick leave certificates, and patients' subsequent employment outcomes," *Health Economics*, https://onlinelibrary.wiley.com/doi/10.1002/hec.3646.

Exemplary data

```
. des
Contains data from data/pdmv_sl.dta
  obs:       322,375                          All sick leaves 2004-2012 for 10% sample of Austrian employees
 vars:            18                          19 Sep 2018 19:11
 size:    27,079,500
-------------------------------------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------------------------------------
id_worker       long    %16.0f                worker ID
id_GP           str32   %32s                * GP ID
id_firm         double  %13.0f                firm ID
p_age           float   %9.0g                 [worker] age in years
p_female        byte    %8.0g                 [worker] =1 if female
p_educ          byte    %27.0g     educ       [worker] education
gp_sex          str1    %9s                   [GP] sex
sl_start        int     %td                   [sick leave] start date
sl_end          int     %td                   [sick leave] end date
sl_dur          byte    %9.0g                 [sick leave] duration
e_start         int     %d                    [emp] start date
e_end           int     %d                    [emp] end date
e_class         byte    %19.0g     classlab   [emp] occupation
e_tenure        int     %9.0g                 [emp] job tenure
e_exper         float   %8.0f                 [emp] experience
e_wage          double  %10.0g                [emp] annual wage
f_firmsize      double  %16.0f                [firm] firm size
f_industry      byte    %8.0g      industry   [firm] NACE95 industry
                                            * indicated variables have notes
-------------------------------------------------------------------------------------------------------------
Sorted by: id_worker  sl_start
```

Glossary

I use different abbreviations that are common in Stata lingo, here is an extract of ones we use in this module:

Abbreviation	Explanation	Stata help file
var	Variable	
varname	Variable name (new or already existing)	[11.4 varlists]
varlist	List of variable names	[11.4 varlists]
numlist	List of numbers	[11.1.8 numlist]
Macro	Variables of Stata programs	[18.3 Macros]

A.2

Project organization and replicability

My Five Coding Commandments

- Thou shalt not use the command line or the user interface.
- Thou shalt not overwrite datasets.
- Thou shalt comment your do-files.
- Honor thy Google and Stata's built in help function.
- Thou shalt write your do-files as efficiently as possible.

Replicability

- Replicability means that your analysis (e.g., a homework, a scientific study, etc.) should produce the same results **if repeated exactly.**

- What does that imply for programming? You should organize your projects in a way that allows other researchers (or co-workers or other collaborators) to **retrace** and **replicate** your data preparation and analysis.

- Always keep do-files. They not only ensure replicability, but also help you in many other ways (e.g., in troubleshooting).

- More specifically, it means that anybody who has the same folder structure and data as you should be able to **understand** and **run your code without error** and obtain the exact **same results** as you.

  $\Longrightarrow$ Ideally, the other person should only change the **current directory** to run your code without error.

Excursus Current directory

- Similar to other languages Stata uses the concept of *current working directory* (CWD).
- Set with `cd "path"`
  - Always enclose file paths in `" "`
  - Avoid capital letters, spaces, and symbols in your folder structure
  - Also in Windows environments, use `/` as a directory separator
- Can be located on your hard disk or external drives, such as your *Dropbox* or a network drive.
- If you open or save a file, Stata will automatically refer to your CWD, unless you specify a file path in the command:
  - `save filename, replace` vs.
  - `save "C:/project/filename", replace`

Replicability

Do-file basics

- Always make sure to keep codes as **tidy** and **perspicuous** as possible.
  - Use comments, not only for others, but also for your future self.
  - Use tab stops to indicate different hierarchies in your code.
  - Make sections in your code, and distinguish them cleanly.
  - Use different comments as dividers (`*`, `//`, `/* */`)

- Do files should be self-contained, meaning they should not rely on something left in memory and not use a dataset unless it loads the dataset before.

- If you simulate data or you draw randomly from the data, always set a random number seed in your do-file with `set seed` *`number`*. This guarantees that you always get the same results.

- Be consistent, always name do-files according to their function, and, again, **NEVER** save over another data file!

Replicability

Do-file basics

- Type expressions so that they are readable:
  - Put spaces around each binary operator except `^`, e.g., `g z = x + y^2`
  - Avoid spaces around `*` and `/`
  - Use parentheses for readability
  - Put a space after each comma in a function, e.g., `inlist(a, b, c)`
- To deal with long lines, use `///`. Use `#delimit ;` only for commands that spread many lines (e.g., graphs, `estouts`)
- Logical negations can be expressed using `!` or `~`, you can sometimes save a lot of coding if you put them in front of variables or functions.
  - `g male = !female` instead of `g male = female == 0` (if female is binary)
  - `g out = !inrange(x, 0, 5)`
  - `g educ_nonmi = !missing(educ)`
- Use macros or scalars instead of "magic numbers" — e.g., save the mean of a variable as a scalar if you need it in your code, refer to `_b[`var`]` if you need the coefficient on var, etc.

Project setup

Do-file basics

```
* time after
bys penr_p benzo (date): g time_after = date[_n+1] - date if !missing(benzo)
la var time_after "time until next benzo prescription in days"

// hypothetical date second prescription starts if p_n > 1
g packpr_p_day = old_total/time_bw if time_bw <= 180
replace packpr_p_day = old_total/time_after if missing(packpr_p_day)

egen tmp = mean(packpr_p_day), by(penr_p benzo year)    /* packages needed per year */
g tagdrop = missing(tmp) & p_n > 1                      /* no information on past+future prescriptions */

* generate date
bys penr_p benzo (date): g date_shyp = date if p_n == 1
format date_shyp %td
la var date_shyp "hypothetical starting date of second package"

g tmp2 = old_total / tmp                        /* days package used on avg */
replace date_shyp = date + tmp2 if p_n > 1      /* fill in days */
order date_shyp, after(date)

* drop obs and generated vars
drop if tagdrop == 1
drop packpr_p_day tmp tmp2 tagdrop

*** | DEFINE GROUPS (continue, stop, switch)
loc x = 90  /* range of days around reform where prescriptions are considered */
foreach p in FZ OT {

    if "`p'" == "FZ" {
      loc i = 1
    }
    else {
      loc i = 0
    }

    g tmp_`p'_pre  = benzo_flunitrazepam == `i' & inrange(date_shyp,mdy(12,15,2012)-`x',mdy(12,15,2012)-1)
    g tmp_`p'_post = benzo_flunitrazepam == `i' & inrange(date_shyp,mdy(12,15,2012),mdy(12,15,2012)+`x')

    egen `p'_pre = max(tmp_`p'_pre), by(penr_p)
    egen `p'_post = max(tmp_`p'_post), by(penr_p)

}
```

Project setup

- Keep multiple do-files for different preparation and analysis steps, use a master do-file to trigger the others.

- Specify a current directory — this is the **only line of the code other people should have to change** if they want to replicate your work.
  - Easier if you work on Dropbox or a shared network folder

- Generate a folder structure from within Stata using the `mkdir` command.
  - `cap mkdir` *`foldername`*
  - Placing `capture` in front makes sure that the do-file continues executing even if *`foldername`* already exists

- Make logs for every do-file and put today's date in the title of the log file, this allows you to track changes.
  - Save the date in a global macro (see later) and open logs with `log using` *`filename.smcl`*`, replace` at the beginning of every do-file
  - Close with `log close` and convert to pdf with `translate` *`filename.smcl filename.pdf`* (works only on Windows)

Project setup

Master do-file

```
/* ////////////////////////////////////////////////////////////////////////////

    MATERNITY LEAVE REFORM 1974

    Task            ::  Master file
    Author          ::  Alexander Ahammer
    Date started    ::  October 12, 2016

//////////////////////////////////////////////////////////////////////////// */

*** | PRELIMINARIES
set more off
set trace off
set seed 1010101

// where do I work?
global mach     = 0
global drive    E

// set current directory
if ${mach} == 1 {
  cd "/nfn/k2xx/vwlmembers/Ahammer/reform74"
}
else {
  cd "${drive}:\Dropbox\reform74\micro_data_analysis"
}

// set directory for ASSD raw data
global HVdir    "/nfn/k2xx/rohdaten/Hauptverband/2014"
global PANELdir "/nfn/k2xx/vwlmembers/Ahammer/nacho_trade/"
global fetddir  "${drive}:\Dropbox\health at birth - mit Alex\alex\"
global paperdir "${drive}:\Dropbox\reform74\paper"

// folder structure
cap mkdir logs
cap mkdir data
cap mkdir graphs
cap mkdir graphs/data
cap mkdir graphs/data/base
cap mkdir graphs/data/het
cap mkdir graphs/data/lw
cap mkdir tables
cap mkdir logs/stlog
cap mkdir logs/desc
cap mkdir logs/analyses
cap mkdir est_data

// define threshold for days spells are allowed to have inbetween to count as 1 spell
global t  = 180     /* 360 days */
global ts = 7       /* 1 week */

// define sharp cutoff date for graphs
global c = mdy(5,1,1974)
```

```
// obtain date for log files
loc date: display %td_CCYY_NN_DD date(c(current_date), "DMY")
global date_string = subinstr(trim("`date'"), " " , "-", .)

// perform saveold?
global so = 0   /* switch to 1 in order to perform */

// define set of covariates
global covars wedlock i.m_age_cat i.m_relig_new m_citizen i.m_province

// define set of outcomes
* health at birth
global outcomes    ln_birth_weight bwght_low low_PI_bwgth ln_length preb
global didoutcomes ln_birth_weight bwght_low low_PI_bwgth ln_length
global suout       birth_weight bwght_low low_PI_bwgth length

* mum's subsequent fertility outcomes
global moutcomes   nb another_child ln_ttnb surv20 surv40
global msuout      nb another_child ttnb_yrs surv20 surv40

////////////////////////////////////////////////////////////////////////////////

*** | EXECUTE DO-FILES
*do do-files/01.1_idprepost.do
*do do-files/01.3_describe-idprepost.do
*do do-files/02.1_prepare-data.do
*do do-files/02.2_prepare-match.do
*do do-files/02.3_nace-match.do
*do do-files/03.1_cleandata.do
*do do-files/04.1_raw-comp.do
*do do-files/04.2_summarize.do
*do do-files/04.4_msurv.do
*do do-files/04.5_average_durations.do
*do do-files/05.1_regs_ols.do
*do do-files/05.1_regs_did.do
*do do-files/05.1_regs_rdd.do
*do do-files/05.1_regs_rdd-did.do
*do do-files/05.1.1_regs_did_lgthMLdur.do
*do do-files/05.1.1_regs_rdddid_lgthMLdur.do
*do do-files/05.1.1_regs_otherCG.do
*do do-files/06.1_regs_splits.do
*do do-files/06.2_regs_splits_working.do
*do do-files/06.2_regs_splits_income.do
do do-files/06.3_regs_rdd_firststages.do
*do do-files/07.1_balancing.do
*do do-files/07.2_graphs.do
*do do-files/07.3_graphs_73q74.do
*do do-files/08.1_charts_base.do
*do do-files/08.2_charts_het.do
*do do-files/08.3_charts_het_mum.do
*do do-files/09.1_longterm.do
*do do-files/09.3_longwindow-robust.do
```

Project setup

Example code

00_master.do

```
cd "D:/pdmv/project1"
cap mkdir data
cap mkdir logs

* obtain date for log files
loc date: display %td_CCYY_NN_DD date(c(current_date), "DMY")
global date_string = subinstr(trim("‘date’"), " " , "-", .)

* trigger do-files
do 01_ps1
```

IMPORTANT Be careful when copying the above code in your do-file, the `date` local may not be addressed properly (you have do use the adequate apostrophes, see [18.3 Macros] or `help local`).

Project setup

Example code

01_ps1.do

```
cap log close
log using logs/ps1_log_${date_string}.smcl, replace

/* here comes your analysis */

log close
translate logs/ps1_log_${date_string}.smcl logs/ps1_log_${date_string}.pdf
erase logs/ps1_log_${date_string}.smcl
```

Project setup

Example code — remarks

- With `global` *name* you generate a global macro (more on macros later).[2] It is addressed by `$`*name* or `${`*name*`}`. With `local` *name* you define a local macro which can only be accessed within the do-file.

- Note that we placed `cap log close` in the beginning of `01_ps1.do`. The problem here is that, if your do-file has an error and stops executing, your log is still open. If, after troubleshooting, you want to re-execute your do-file, you would get an error saying that the log file is still open. Again, `capture` makes sure to execute the command afterwards only if a do-file is open.

- Note also that we erased the smcl log which we converted to pdf anyways. To save capacity, erasing also makes sense for temporary datasets, e.g., ones you have to save in order to merge them to others.

[2] Global macros are global, that is, there is only one global macro with a specific name in Stata, and its contents can be accessed by a Stata command executed at any Stata level.

Other tips for an efficient and secure workflow

- If possible, use Dropbox or any other cloud service to store your codes and data.
- If you work on multiple computers, make a harddrive partition on each which you use only for your work. Put the Dropbox folder on this partition.
- If you don't want to use Dropbox, synchronize your files with FTP or torrent, e.g., with Resilio Sync.
- Backup data as often as possible, ideally on external harddrives.
- If you collaborate a lot, it may make sense to dive into **git,** which allows versioning.
  - `https://github.com/fpinter/git-for-economists/blob/master/git-for-economists-presentation.pdf`

A.3

Data types

String variables

- Load `pdmv_sl.dta` and `des` the data. In the column 'storage type' you will see the data type of each variable.

- Variables come in different shapes or forms, the major distinction is between **numeric** and **string** data variables.

- **String variables**
  - Can hold up to 244 characters in length, 1 byte per character.
  - Typically ordinal information is stored as strings (e.g., names, job or industry classifications, addresses, etc.)
  - It may make sense to convert string to numeric variables, the commands `destring` (converting numbers stored as strings to numeric variables), and `encode` (creating a new variable which attaches numbers to every realization of a string variable) may prove useful.
  - For a finite and small number of realizations, also `tab, gen(`*varname*`)` is possible to convert from string to numeric.

String variables

- For the next exercises, you need the following two user-written packages:
  - `ssc install egenmore`
  - `ssc install fre`

String variables

Working with strings — `destring`

```
. // extract all real numbers from id_GP
. egen id_GP_real = sieve(id_GP), char(0123456789)
(965 missing values generated)

.
. // destring to numeric format
. des id_GP_real
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
id_GP_real      str12   %12s
. list id_GP_real in 1/5

     +-----------+
     | id_GP_~l |
     |-----------|
  1. |    06374 |
  2. |    06374 |
  3. |    06374 |
  4. |       09 |
  5. |     7812 |
     +-----------+

. destring id_GP_real, replace
id_GP_real: all characters numeric; replaced as double
(965 missing values generated)
. list id_GP_real in 1/5

     +-----------+
     | id_GP_~l |
     |-----------|
  1. |     6374 |
  2. |     6374 |
  3. |     6374 |
  4. |        9 |
  5. |     7812 |
     +-----------+
```

String variables

Working with strings — encode

```
. list gp_sex in 1/5
```

	gp_sex
1.	M
2.	M
3.	M
4.	W
5.	M

```
. encode gp_sex, gen(gp_sex_num)
. list gp_sex_num in 1/5
```

	gp_sex~m
1.	M
2.	M
3.	M
4.	W
5.	M

```
. fre gp_sex_num
gp_sex_num ── [GP] sex
```

		Freq.	Percent	Valid	Cum.
Valid	1 M	276263	85.70	87.69	87.69
	2 W	38767	12.03	12.31	100.00
	Total	315030	97.72	100.00	
Missing	.	7345	2.28		
Total		322375	100.00		

String variables

Working with strings — `strpos()`, `substr()`

```
. decode e_class, gen(class_str)
. ta class_str
```

[emp] occupation	Freq.	Percent	Cum.
blue collar worker	195,760	60.72	60.72
white collar worker	126,615	39.28	100.00
Total	322,375	100.00	

```
. count if strpos(class_str,"blue")
  195,760
. g class = substr(class_str, 1, strpos(class_str,"collar")-2)
. ta class
```

class	Freq.	Percent	Cum.
blue	195,760	60.72	60.72
white	126,615	39.28	100.00
Total	322,375	100.00	

String variables

Working with strings

- Sometimes you have to extract information from string variables, which can be rather tricky. These functions may prove helpful:
  - `split` *`s`*`, p(" ")` (splits string `s` by whatever is inside the `" "`)
  - `strpos(`*`s1`*`,`*`s2`*`)` (the position in `s1` at which `s2` is last found; otherwise 0)
  - `substr()`, `subinstr()`, and others (change certain portions of strings)

- The built-in Stata help contains very detailed information on all aspects of working with strings, consult especially:
  - `help strings` [Link to v14 pdf manual]
  - `help string functions` [Link to v14 pdf manual]

String variables

Working with strings

```
. list
```

	fullname
1.	John Adams
2.	Adam Smiths
3.	Mary Smiths
4.	Charlie Wade

```
. // generate separate vars for first and last name
. split fullname, p(" ")
variables created as string:
fullname1  fullname2
. rename (fullname1 fullname2) (firstname lastname)
. list
```

	fullname	firstn~e	lastname
1.	John Adams	John	Adams
2.	Adam Smiths	Adam	Smiths
3.	Mary Smiths	Mary	Smiths
4.	Charlie Wade	Charlie	Wade

Numeric variables

- Numeric variables can be stored as `byte`, `int`, `long`, `float`, and `double`. The first three can only store integer values.
- The differences in storage requirements between these variable types is minimal, let Stata decide how to generate a variable and use `compress` before saving a data file.
- **IMPORTANT:** Sometimes identifier variables are extremely long, and Stata may choose a format that **truncates** the variable. Make sure to `list` such variables, and tell Stata that you need new variables in `double` format if you want to copy or recode them.
  - `gen double` *`newvarname = oldvarname`*
- Experts in *fine digit programming* suggest storing variables with many digits as strings instead of numeric variables.

Numeric variables

Variable lengths and storage requirements

Storage type	Minimum	Maximum	Bytes
`byte`	−127	100	1
`int`	−32,767	32,740	2
`long`	−2,147,483,647	2,147,483,647	4
`float`	-1.701×10^{38}	1.701×10^{38}	4
`double`	-8.988×10^{307}	8.988×10^{307}	8

Numeric variables

The variable length fallacy

```
. gsort -id_firm          /* sorts id_firm in descending order */
. list id_firm in 1/5
```

	id_firm
1.	1919003067000
2.	1918508076000
3.	1917300028000
4.	1917300028000
5.	1917300028000

```
. g tmp1 = id_firm        /* STATA saves in float format, which is too short */
. g double tmp2 = id_firm
. format tmp1 tmp2 %13.0f
. list tmp1 tmp2 in 1/5
```

	tmp1	tmp2
1.	1919003131904	1919003067000
2.	1918508072960	1918508076000
3.	1917299982336	1917300028000
4.	1917299982336	1917300028000
5.	1917299982336	1917300028000

```
. compress
  variable gp_sex_num was long now byte
  (967,125 bytes saved)
```

Date and time handling

- It's very common to work with dates and times, for example if you work with spells (e.g., employment spells) or survival data (how long until a certain event occurs). In `pmdv_sl.dta` you have several date variables.

- Dates are represented by numbers known as `%t` values measuring the time interval from a reference date or *epoch*. The epoch for Stata is **midnight** on **January 1, 1960.** Days following that date have positive integer values, days prior negative ones.

- Days represented in days are known as `%td` values, you also have other frequencies: weeks (`%tw`), months (`%tm`), quarters (`%tq`), or half-years (`%th`). Intradaily times are supported since Stata 10, a date-time variable is known as `%tc` and can be as granular as seconds or miliseconds.

- Helpful Stata documents (that even I refer to all the time):
  - `help datetime` [Link to v14 pdf manual]
  - `help datetime_translation` [Link to v14 pdf manual]

Date and time handling

Examples for dates, relative to Stata epoch

Format	Date	Value	how to address
Daily dates			
	January 1, 1960	0	`mdy(1,1,1960)`
`%td`	January 30, 1960	29	`mdy(1,30,1960)`
	October 15, 2018	21,472	`mdy(10,15,2018)`
Monthly dates			
	January, 1960	0	`ym(1960,1)`
`%tm`	March, 1960	2	`ym(1960,3)`
	October, 2018	705	`ym(2018,10)`
Quarterly dates			
	Q1, 1960	0	`yq(1960,1)`
`%tq`	Q3, 1960	2	`yq(1960,3)`
	Q4, 2018	235	`yq(2018,4)`

Date and time handling

Some useful tips:

- Always store date variables in `long` format and times in `double` format to avoid overflow conditions.
- Sometimes, you only have limited information on dates (e.g., birth year and birth month) — pick the appropriate format (`%tm`)! Make sure not to mix up different formats (e.g., when calculating durations).
- Date conversions can be tricky, but the Stata help provides a solution (in the form of specific functions that have to be used) for every possible conversion you can imagine. Some examples you will find on the next two slides.
- Use the `inrange()` function if you want to check whether a variable (could be a date) lies within two boundaries.
- Dates are numbers, with `format` *`varname`* you can specify the format in which it is displayed.
  - `format sl_start sl_end %td`

Date and time handling

Basics of date time conversion

- Converting dates between different formats can be tricky, but — again — you will find the appropriate function in the Stata help file or manual.
- If your date is saved as a string, use the functions in `help datetime translation` to translate to a date.
- If you have a date saved in **several variables** (e.g., *birthday*, *birthmonth*, *birthyear*), use the function on slide 33 to generate a date.
- You want to convert from a daily date $\longrightarrow$ *SIF-SIF conversion* in the [manual]
  - To a monthly var: `tm = mofd(td)`
  - To a quarterly var: `tq = qofd(td)`
  - To a yearly var: `ty = yofd(td)`
- Sometimes you may also want to convert between missing entries, e.g.,
  - From monthly to quarterly var: `tq = qofd(dofm(tm))`
  - From weekly to monthly var: `tw = mofd(dofw(tw))`

Date and time handling

Basics of date time conversion

- You can extract also portions of dates:

Portion to be extracted	Function	Example
Calender year	`year(td)`	2013
Calender month	`month(td)`	7
Calender day	`day(td)`	5
Day of week	`dow(td)`	2 (*Tuesday*)
Week within year	`week(td)`	27
Quarter within year	`quarter(td)`	3

  - Note that these functions return particular values in the domain of the portion that is extracted (e.g., month 12 instead of a monthly var as on slide 33).
  - You always need a `%td` date for these extractions. If you want to extract from a less granular date, you have to convert first (slide 35): `quarter(dofq(tq))`.

Date and time handling

Date saved as string

```
. list    /* manually imputed dates as strings */

     +-------------+
     | date_str~g  |
     |-------------|
  1. |  20jan2007  |
  2. |   16June06  |
  3. | 06sept1985 |
  4. |   21june04  |
     +-------------+

. g date = date(date_string, "DM20Y")
. list

     +-------------------------+
     | date_str~g    date      |
     |-------------------------|
  1. |  20jan2007   17186      |
  2. |   16June06   16968      |
  3. | 06sept1985    9380      |
  4. |   21june04   16243      |
     +-------------------------+

. format date %td
. list

     +-------------------------+
     | date_str~g        date  |
     |-------------------------|
  1. |  20jan2007   20jan2007  |
  2. |   16June06   16jun2006  |
  3. | 06sept1985   06sep1985  |
  4. |   21june04   21jun2004  |
     +-------------------------+
```

Date and time handling

Date saved in multiple vars

```
. list    /* again, manually imputed dates */
```

	bday	bmonth	byear
1.	20	1	1960
2.	1	5	1980
3.	5	12	1975
4.	8	10	1930
5.	30	4	1989

```
. g bdate = mdy(bmonth,bday,byear)
. list
```

	bday	bmonth	byear	bdate
1.	20	1	1960	19
2.	1	5	1980	7426
3.	5	12	1975	5817
4.	8	10	1930	-10677
5.	30	4	1989	10712

```
. format bdate %td
. list
```

	bday	bmonth	byear	bdate
1.	20	1	1960	20jan1960
2.	1	5	1980	01may1980
3.	5	12	1975	05dec1975
4.	8	10	1930	08oct1930
5.	30	4	1989	30apr1989

Date and time handling

Calculate duration with different date formats

```
. list    /* manually impute also random death dates */
```

	bday	bmonth	byear	bdate	dmonth	dyear
1.	20	1	1960	20jan1960	2	2018
2.	1	5	1980	01may1980	12	2018
3.	5	12	1975	05dec1975	5	2016
4.	8	10	1930	08oct1930	1	2017
5.	30	4	1989	30apr1989	12	2018

```
. g ddate = ym(dyear,dmonth)
. list
```

	bday	bmonth	byear	bdate	dmonth	dyear	ddate
1.	20	1	1960	20jan1960	2	2018	697
2.	1	5	1980	01may1980	12	2018	707
3.	5	12	1975	05dec1975	5	2016	676
4.	8	10	1930	08oct1930	1	2017	684
5.	30	4	1989	30apr1989	12	2018	707

```
. format ddate %tm
. g age = ddate - ym(byear,bmonth)
. g age_alt = ddate - mofd(bdate)
. su age*
```

Variable	Obs	Mean	Std. Dev.	Min	Max
age	5	607.2	269.214	356	1035
age_alt	5	607.2	269.214	356	1035

Date and time handling

Exercise I

Exercise: Day of the week

Use the data is `pdmv_sl.dta` for this exercise. Find out which day of the week (Monday–Sunday) sick leaves end most often. Make a histogram with sick leave ending day densities per day of the week.

Date and time handling

Exercise I

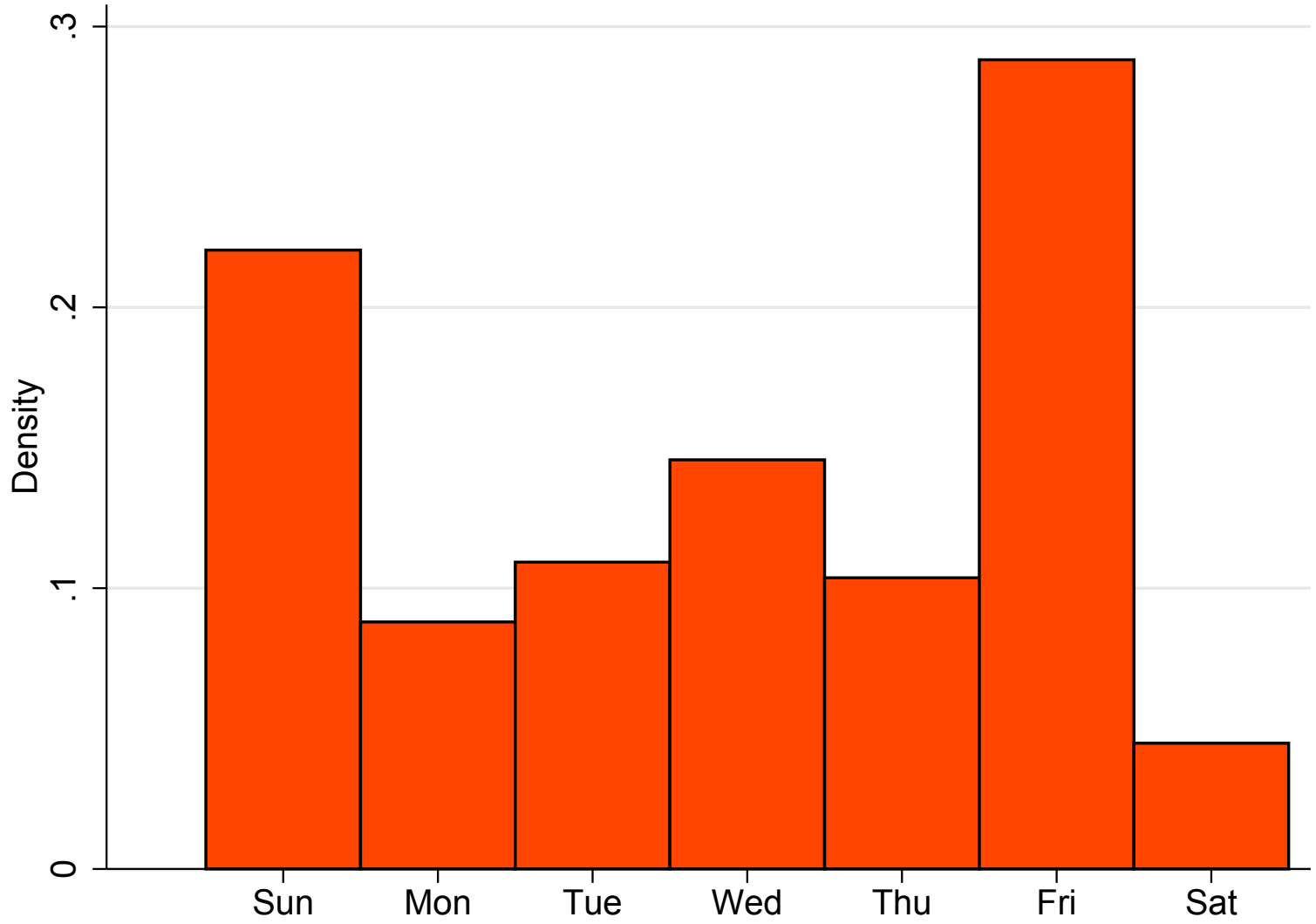
```
. use "data/pdmv_sl.dta", clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. g dow_end = dow(sl_end)
. la def dow 0 "Sun" 1 "Mon" 2 "Tue" 3 "Wed" 4 "Thu" 5 "Fri" 6 "Sat", replace
. la val dow_end dow
. fre dow_end, order
dow_end
```

		Freq.	Percent	Valid	Cum.
Valid	0 Sun	71050	22.04	22.04	22.04
	1 Mon	28358	8.80	8.80	30.84
	2 Tue	35226	10.93	10.93	41.76
	3 Wed	46971	14.57	14.57	56.33
	4 Thu	33430	10.37	10.37	66.70
	5 Fri	92896	28.82	28.82	95.52
	6 Sat	14444	4.48	4.48	100.00
	Total	322375	100.00	100.00	

```
. hist dow_end, d xlab(0(1)6, val) xtitle("") fcolor("255 69 0") scheme(s2mono) graphregion(color(white))
(start=0, width=1)
. gr export "slides/graphs/dow.pdf", as(pdf) replace
(file slides/graphs/dow.pdf written in PDF format)
```

Date and time handling

Exercise I

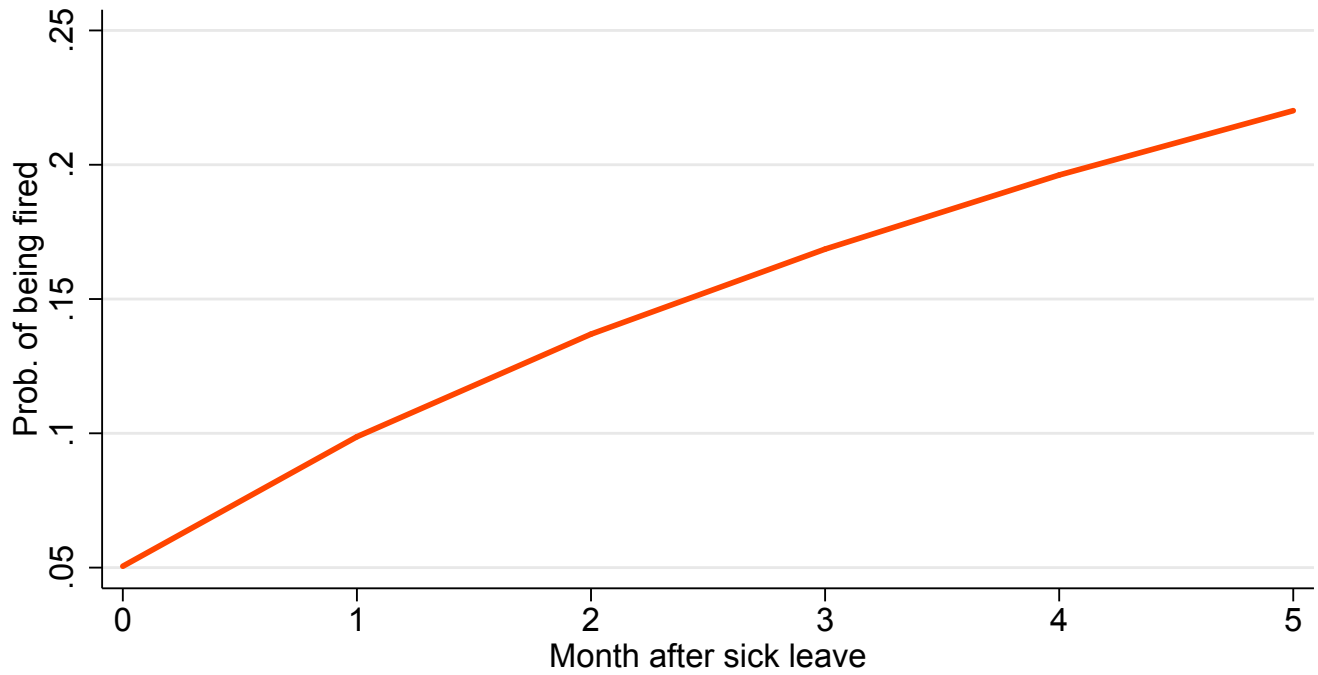


Date and time handling

Exercise II

Advanced Exercise: Event study

Continue to use the data in `pdmv_sl.dta`. Replicate the event study below, which shows the average unconditional probability that employment ends (`e_end`) for up to 5 months after the sick leave ends (`sl_end`).



Date and time handling

Exercise II

```
use "data/pdmv_sl.dta", clear

egen id_sl = group(id_worker sl_start)
g sl_end_month = mofd(sl_end)
format sl_end_month %tm

// expand to five obs per sick leave
expand 6, gen(_ex)
sort id_sl _ex
bys id_sl: g t = _n - 1
g sl_month = sl_end_month + t
format sl_month %tm

// check whether e_end falls within t=0 and t=5
g fired = mofd(e_end) <= sl_month
list sl_start sl_end e_end t sl_month fired if id_sl == 3, sep(0)

// make event study
collapse (mean) fired, by(t)
```

Date and time handling

Exercise II *(remarks)*

- This code contains some functions and tricks we will learn this semester, for example the `by` prefix and observation counting using `_n` (which will be a huge topic in module B), but it illustrates how you work with date conversion.
- `sl_end` is a `%td` variable, so we use `mofd()` *("month of daily date")* to convert it into a `%tm` var.
- `format` *varname* `%tm` is used here only for us to check whether the conversion went right, it's not necessary for Stata to handle the var as `%tm`.
- This is the graph command I used in case you are interested:

```
tw line fired t, ytitle("Prob. of being fired") xtitle("Month after sick
leave") lcolor("255 69 0") lwidth(*2) xsize(7.5) scale(1.3) scheme(s2mono)
graphregion(color(white))
```

Date and time handling

Exercise II *(remarks)*

```
. list sl_start sl_end e_end t sl_month fired if id_sl == 3, sep(0)
```

	sl_start	sl_end	e_end	t	sl_month	fired
13.	20oct2012	23oct2012	31dec2012	0	2012m10	0
14.	20oct2012	23oct2012	31dec2012	1	2012m11	0
15.	20oct2012	23oct2012	31dec2012	2	2012m12	1
16.	20oct2012	23oct2012	31dec2012	3	2013m1	1
17.	20oct2012	23oct2012	31dec2012	4	2013m2	1
18.	20oct2012	23oct2012	31dec2012	5	2013m3	1

Date and time handling

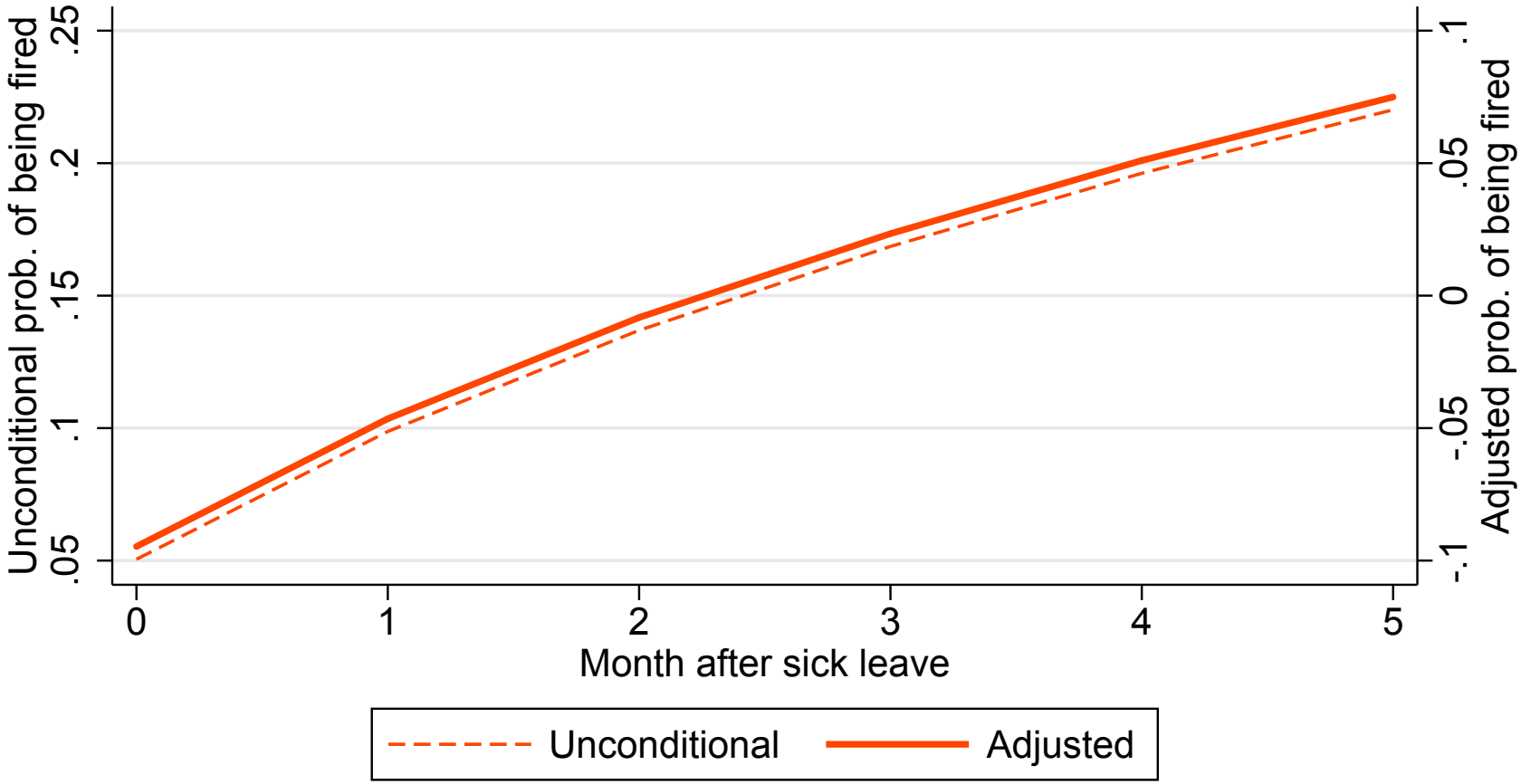
Exercise II (additional exercise)

Advanced Exercise: Event study with adjusted probabilities

Consider again the event study from Exercise II. Suppose you want to adjust the firing probabilities for age, gender, tenure, and occupation. Produce a graph that contains the adjusted probabilities and include also the unconditional firing probabilities from before.

Date and time handling

Exercise II (additional exercise)



Date and time handling

Exercise II (additional exercise)

```
* adjust for several vars
egen p_age_cat = cut(p_age), at(15(5)70)
reg fired i.p_age_cat p_female e_tenure i.e_class
predict fired_adj, r
su fired*

collapse (mean) fired*, by(t)
#delimit ;
tw  (line fired t, lcolor("255 69 0") lpattern(dash))
    (line fired_adj t, lcolor("255 69 0") lpattern(solid) lwidth(*2)
      yaxis(2)),
    ytitle("Unconditional prob. of being fired", axis(1))
    ytitle("Adjusted prob. of being fired", axis(2))
    xtitle("Month after sick leave")
    xsize(7.5) scale(1.3) scheme(s2mono) graphregion(color(white))
    legend(order(1 "Unconditional" 2 "Adjusted"))
  ;
#delimit cr
```

Time series operators

- Stata provides time series operators `L.` (lags), `F.` (leads or forward values), `D.` (differences), and `S.` (seasonal differences).
- It's not necessary to create new variables for these constructs if the data are `tsset` or `xtset` (Stata knows that you handle time series or panel data).
  - Operators can be used almost everywhere a varlist is required, e.g., in regressions or summary statistics, simply put in front of varname.
- Combined with a numlist, you can even include multiple of these constructs, e.g., `L(1/4).`*varname* includes 4 lags of *varname*.
- Operators are easy to use and less erratic than manually generating lags/leads/etc. For example, `gen lag1 = x[_n-1]` can be erratic when time series/panel is not balanced.
- [11.4.4 Time series varlists]

A.4

Memory and data import/export

Memory

- Newer Stata versions do a pretty good job of automatically allocating the appropriate amount of memory your dataset requires.
- Stata MP can do multi-core processing with up to 20 bn observations and 10,998 variables, your local PC is most likely restricted far below this limit.
- With `memory` you can check how much memory Stata has allocated, certain processes can slow Stata down, e.g., creating large matrices.
- Big data are typically handled on large capacity mainframes/clusters, such as the new MACH2 at the JKU (allows jobs up to 512 GB RAM, 260 TB mass storage).
- You don't have to load full datasets, use the `use if` option to select only certain vars/obs you need (can save some time).
- Single best tip I can give you to handle massive datasets:

  $\Longrightarrow$ Draw a **random sample** of the data and write your codes on that. As soon as your code runs without error, execute it on the full data. Lean back.

`preserve` and `restore`

- Several Stata commands replace the data in memory with a new dataset. For example, `collapse`, which makes a dataset of summary statistics.
- In your code you may want to invoke one of these commands, but you may also want to retain the existing contents in memory for further use.
- You need the `preserve` and `restore` commands:
  - `preserve` sets aside the current contents in memory
  - `restore` brings them back when needed

`preserve` and `restore`

```
use "data/pdmv_sl.dta", clear
preserve
    collapse (mean) p_age, by(id_GP)
    rename p_age gp_meanage
    la var gp_meanage "[GP] avg age of patient pop"
    save data/meanage_GP.dta, replace
restore
merge m:1 id_GP using data/meanage_GP.dta
erase data/meanage_GP.dta
```

This code

- loads the sick leave data and saves it in memory using `preserve`,
- calculates the average age of a GP's patient stock (for every GP in the data),
- and then merges the newly generated variable `gp_meanage` back to the original dataset, which is brought back from memory with `restore`.

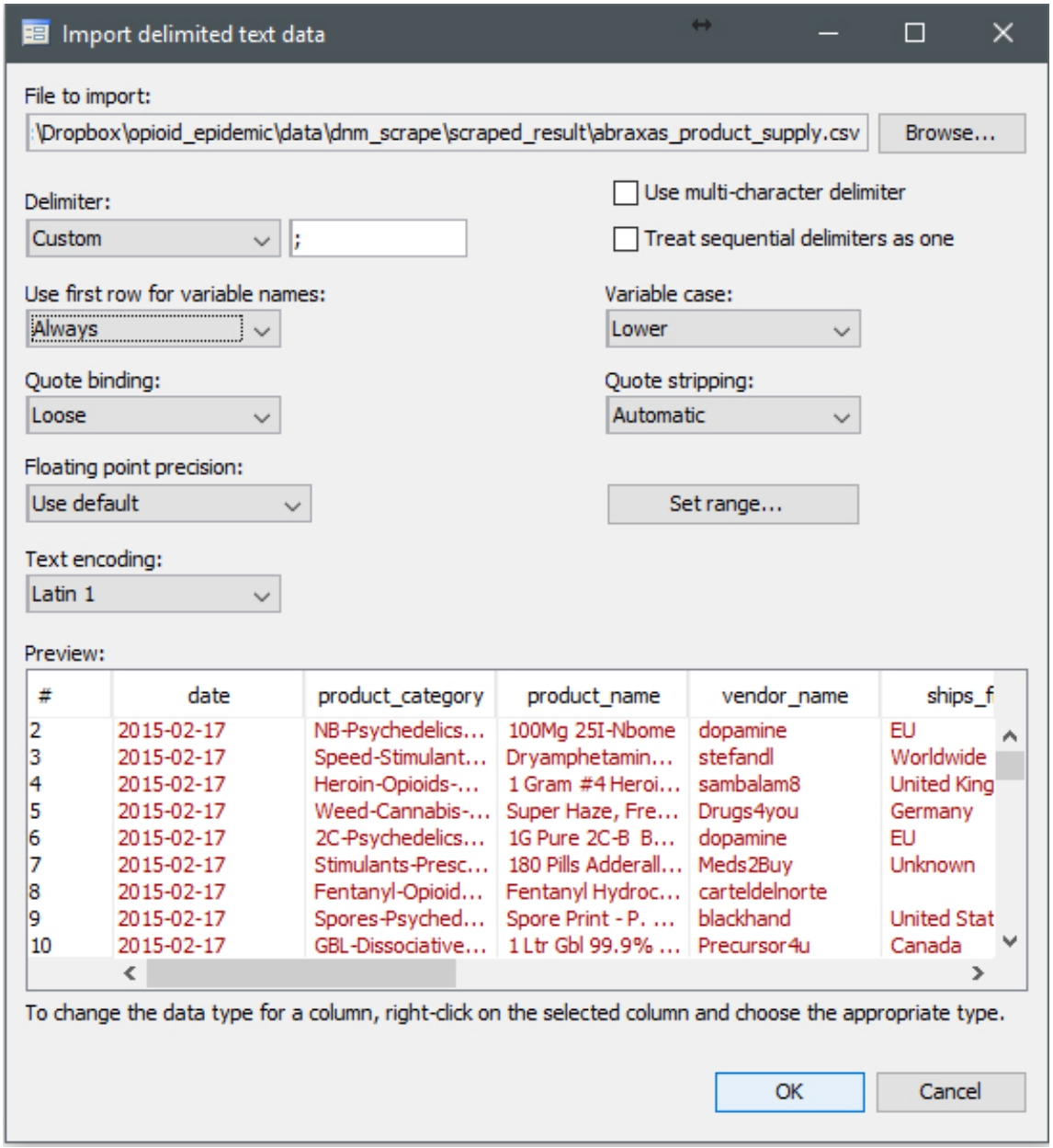
Getting your data into Stata

- Source data do not necessarily have to be in Stata format, they can be downloaded from a website, acquired in spreadsheet format, or made available in a format of a different statistical package.

- When importing non-native Stata datasets, I usually break the first of my five commandments (*"Thou shalt not use the user interface"*), because it allows you to see a preview of what the data will look like after importing.
  - Go to *File* $\longrightarrow$ *Import* and select the file type from the dropdown menu
  - Copy the command in your do-file

- Exporting data is similar. In module C we will discuss the `parmest` and `putexcel` commands, both are very powerful and allow you to export not only data, but also estimation results or stored matrices (which is useful if you want to plot, for example, a set of regression coefficients).

Getting your data into Stata



Getting your data into Stata

General tips and remarks on importing data

- **IMPORTANT** Get your data as early as possible into Stata, and perform all manipulations via well-documented codes.
- Typically microeconomic data comes in text (or ASCII) files (such as .raw or .csv), these can be in free vs. fixed format (columns separated by tabstops or certain delimiters). Make sure to choose the right delimiter.
- If the data contain both numbers and strings, Stata may not automatically import the numeric variable as such. This requires you to use the `destring` command from before.
- For some datasets that do not come in Stata format (e.g., the US NHANES) there may be a directory (.dct) file available, which contains value labels for certain variables (this can be extremely helpful).
- **IMPORTANT** Beware that there are many ways of coding **missing values** (., 9, 99, m). Make sure to read the data documentation and find out how missings are coded in the data you are importing.