Programming, Data Management and Visualization

Module C: **Data management**

Alexander Ahammer

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

β version, may still be updated
Last updated: Monday 9th December, 2019 (13:31)

Introduction

- In this module cover data validation, we hear briefly about database structures in general, and we learn how to reorganize and combine datasets is Stata.

- At the end of the module, you should know
  - how to validate data through scripted data checking
  - the mechanics of the `collapse` and `reshape` commands
  - how to combine datasets with `merge`, `append`, and `joinby`

- The book covers also issues concerning saving and reusing intermediate results, as well as presenting outputs in tables.

  $\Longrightarrow$ We postpone these topics to module D.

- **Big data** issues are discussed at different points in this module.

C.1

Data validation

Sanity checking

- Preparing data should always start with **sanity checking.**
  - Do all values of the raw data make sense?
  - Are there any coding errors that are apparent in the range of data values?
  - Are there numeric values that should be coded as missings?

- Always perform a series of checks on new data you input. This is the start of your data transformation.

- Write another do-file that corrects these errors (don't do this in Excel), and keep both the initial and the final data set separately.

- Your best friends here are `describe`, `summarize`, `tabulate`, and `histogram`, which provide useful information on imported data.

Sanity checking

```
. su p_age sl_dur e_wage
```

Variable	Obs	Mean	Std. Dev.	Min	Max
p_age	322,375	36.82896	11.19948	18	65
sl_dur	322,375	6.020889	5.386769	1	44
e_wage	322,375	25994.96	16373.24	.0033333	1144276

```
. ta p_female
```

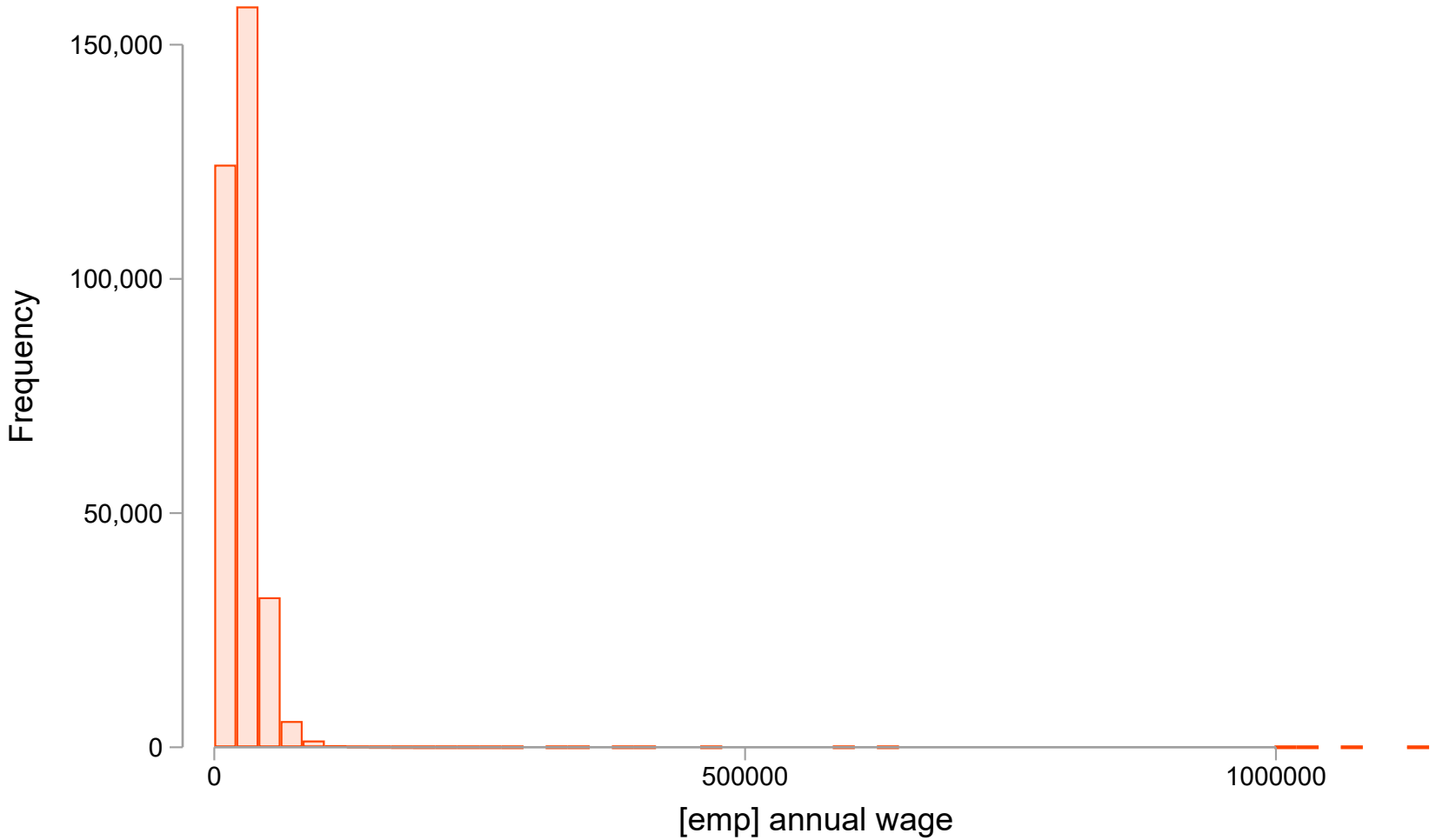
[worker] =1 if female	Freq.	Percent	Cum.
0	192,279	59.64	59.64
1	130,096	40.36	100.00
Total	322,375	100.00	

```
. g year = yofd(sl_start)
. ta year
```

year	Freq.	Percent	Cum.
2004	444	0.14	0.14
2005	35,650	11.06	11.20
2006	33,631	10.43	21.63
2007	38,242	11.86	33.49
2008	42,408	13.15	46.65
2009	40,596	12.59	59.24
2010	40,211	12.47	71.71
2011	45,379	14.08	85.79
2012	45,814	14.21	100.00
Total	322,375	100.00	

```
. hist e_wage, lcolor("255 69 0") fcolor("255 69 0%30") xsize(6.5) freq ///
>         ylab(, format(%9.0fc) nogrid) xlab(, nogrid)
(bin=55, start=.00333333, width=20805.022)
. gr export "slides/graphs/wagedist.pdf", as(pdf) replace
(file slides/graphs/wagedist.pdf written in PDF format)
```

Sanity checking



Sanity checking

```
. // comment these three commands out if you want to download the dataset below
. *net from "http://www.stata-press.com/data/itsp2"
. *net set other data/itsp2
. *net get itsp2-data
. use data/itsp2/census2b.dta, clear
(Version of census2a for data validation purposes)

. des

Contains data from data/itsp2/census2b.dta
  obs:            50                          Version of census2a for data validation purposes
 vars:             5                          9 Oct 2015 12:43
 size:         1,650
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
state           str14   %14s
region          str7    %9s
pop             float   %9.0g
medage          float   %9.0g
drate           float   %9.0g
-------------------------------------------------------------------------------
Sorted by:

. su pop-drate, sep(0)
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         pop |         49     4392737     4832522        -9   2.37e+07
      medage |         50       35.32    41.25901      24.2        321
       drate |         50       104.3    145.2496        40       1107
```

Using the `assert` command

- In the last example based on census data, several anomalies are revealed that have to be corrected.
  - Population data is missing for one state (unlikely in proper census data)
  - At least one state has negative population numbers $\longrightarrow$ coding error
  - Maximum of the median age var of 321 $\longrightarrow$ coding error
  - Mean death rate is 104, a value 10 times the mean is unlikely as well

- In your research, you have **subject-matter knowledge,** which helps you define **sensible ranges of values** for the vars in your data.

- You may also find the `codebook` command helpful in identifying data errors.

- Let's write a code now that uses data validation techniques that can be applied for datasets with millions of observations. We use `assert` to perfom sanity checks $\longrightarrow$ if the code runs without error, all checks are passed.

Using the `assert` command

```
use data/itsp2/census2b.dta
// check pop
   list if !inrange(pop,300000,3e7)
   assert inrange(pop,300000,3e7)
// check medage
   list if !inrange(medage,20,50)
   assert inrange(medage,20,50)
// check drate
   su drate
   list if !inrange(drate,10,r(mean)*r(sd))
   assert inrange(drate,10,r(mean)*r(sd))
```

Correction data mistakes

```
. use data/itsp2/census2b.dta, clear
(Version of census2a for data validation purposes)
. // check pop
. list if !inrange(pop,300000,3e7)
```

	state	region	pop	medage	drate
4.	Arkansas	South	-9	30.6	99
10.	Georgia	South	.	28.7	81
15.	Iowa	N Cntrl	0	30	90

```
. assert inrange(pop,300000,3e7)
3 contradictions in 50 observations
assertion is false
r(9);
end of do-file
r(9);
. do "C:\Users\ALEXAN~1\AppData\Local\Temp\STDf0a8_000000.tmp"
```

Sanity checking

```
. su pop
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         pop |         49     4392737     4832522          -9   2.37e+07

. replace pop = r(mean) if !inrange(pop,300000,3e7)
(3 real changes made)

. // check pop
. list if !inrange(pop,300000,3e7)

. assert inrange(pop,300000,3e7)
```

IMPORTANT There is **no right or wrong way** of correcting such mistakes, imputing the sample mean for non-valid values is only one possibility. You may also impute other values (for example the minimum or the maximum if you want to **censor** a variable, consider **winsorizing** or certain **imputation** techniques, or **dropping** the observations altogether. What option you choose always depends on the particular problem at hand.

Other useful data validation techniques

- Twoway tables with `tab` can also be helpful. Suppose you have a dataset with medical questionnaires, you may want to check something like

  ```
  tab pregnant gender
  ```

  which should display non-zero values only in the female column. You can also use `assert pregnant == 1 if gender == "Male"`, which should not return an error.

- With `tabstat` you can assess means of continuous variables by realizations of categorical variables.

- You can use `duplicates` to check for duplicate observations (pro tip for RA's working with the ASSD and GKK data $\longrightarrow$ always check this at the beginning of your do-files).

- Download the user-written code `distinct`, which returns the distinct number of observations for a specified varlist. This is especially informative with panel data.

Other useful data validation techniques

```
. use "data/pdmv_sl.dta", clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. tabstat sl_dur, by(gp_sex) s(mean sd)
Summary for variables: sl_dur
     by categories of: gp_sex ([GP] sex)
```

gp_sex	mean	sd
M	6.044573	5.400271
W	5.997163	5.350472
Total	6.038739	5.394182

```
. duplicates list
Duplicates in terms of all variables
(0 observations are duplicates)
. distinct id_worker id_GP id_firm
```

	Observations	
	total	distinct
id_worker	322375	52739
id_GP	322375	1033
id_firm	322375	17620

Unit tests

- *Real* programmers write **unit tests** for just about every piece of code. These scripts check whether the piece of code does everything it is expected to do.
- For a program to compile, it must pass all the unit tests. Many bugs are thus caught automatically. A large program will often have as much testing code as program code.
- Econometricians typically don't write unit tests, instead we interactively apply different validation checks to see whether our code produces the right data or regression output.
- This is inefficient $\Longrightarrow$ write unit tests.

C.2

Reorganizing data

The `collapse` command

- With `collapse` you can make a new dataset from summary statistics. Alternatively, you can also use `contract` to create a dataset of *frequencies*.

- The syntax of `collapse` is

  `collapse (stat1)` *`varlist`* `(stat2)` *`varlist`* `...`

  where `stat` can be `mean`, `median`, `min`, `max`, `sum`, etc. (see the help file). It also allows multiple vars in a `by` options, which computes the summary statistics of *`varlist`* for every unique combination of the vars in `by`.

  - *`varlist`* can also consist of *`newvar=oldvar`*, which is especially helpful if you want to generate different statistics of the same var, or you want to rename the statistic of a var immediately.

- `collapse` takes the dataset in memory and creates a new dataset containing summary statistics of the original data. This means that the original dataset will vanish unless you store it in memory using `preserve`.

- We will see how the command works in some of the examples in this module.

The `collapse` command

```
. collapse (firstnm) p_female (sum) sl_dur (max) p_age, by(id_worker)
. list in 1/5
```

	id_wor~r	p_female	sl_dur	p_age
1.	166	1	17	59.83333
2.	276	1	9	55.75
3.	548	1	34	52.66667
4.	579	1	58	59.75
5.	1063	1	47	57.91667

TIPP Here we use the function `firstnm`, which carries the first non-missing observation of `p_female` over to the collapsed dataset. Do this only for vars that are **constant** across obs in the by-group.

Alternatively, preserve the data, make a dataset containing only one obs per `id_worker` with all constant vars, save it, and restore the original data. Then collapse the statistics you need and merge back the dataset from before.

Collapse and big data

- Big data often consist of millions of rows of information (e.g., all sick leaves in Upper Austria, all medication prescriptions, all employment spells) and/or perhaps (tens of) thousands of columns (large N / large K)
- It is impossible to work with such data (i.e., running regressions) without rearranging them in some shape or form first.
- Typically these data can be aggregated on a meaningful level
  $\longrightarrow$ e.g., aggregate sick leaves to a worker-year panel.
- This can be done with `collapse`.
- Check `this slide set` on `ftools` for a variety of user-written procedures that work faster with big data

`collapse` vs. `fcollapse`

Performance

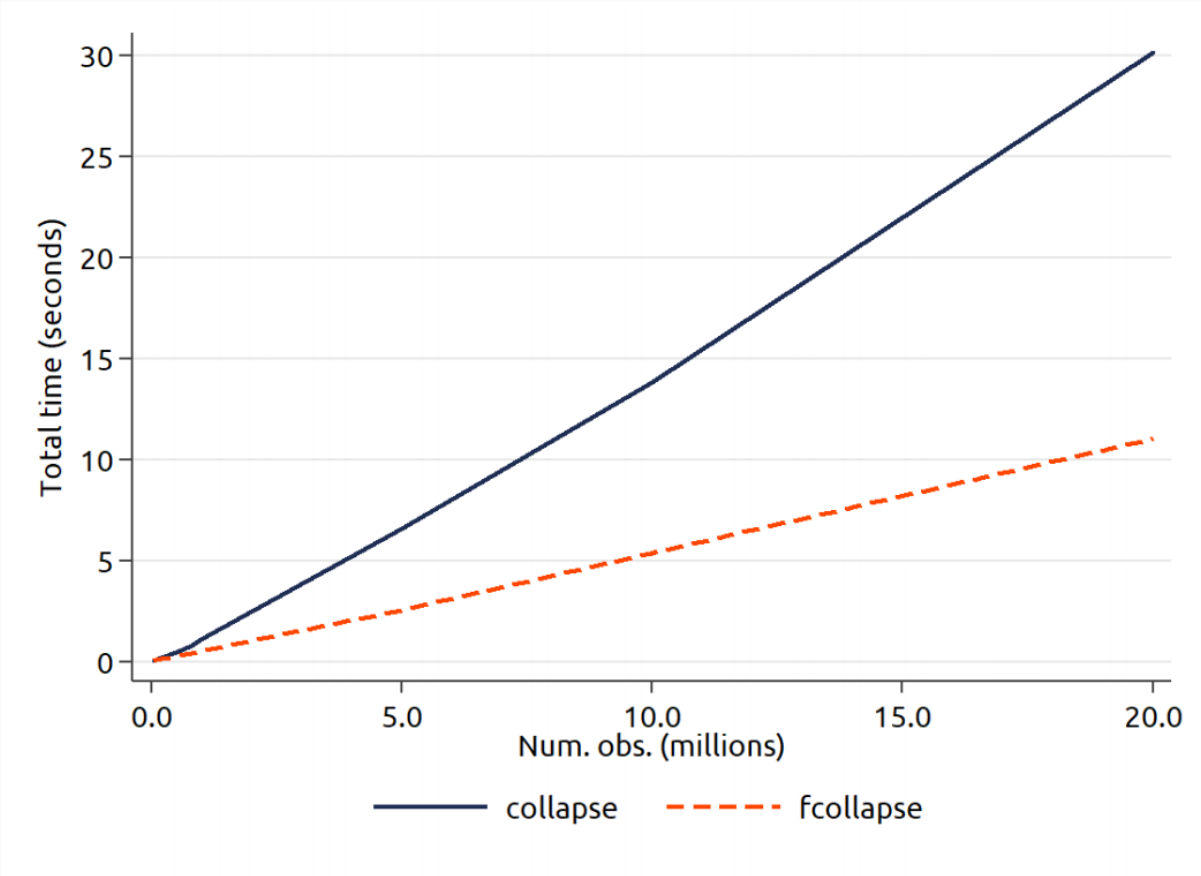


Figure 4: Elapsed time of `collapse` and `fcollapse` by num. obs.

Source: Correia (2017), '**ftools**: a faster Stata for large datasets'

```
. g year = yofd(sl_start)
. collapse (count) nosls=sl_start (mean) p_age sl_dur, by(id_worker year)
. format nosls %9.0f
. list if id_worker <= 579, sepby(id_worker)
```

	id_wor~r	year	nosls	p_age	sl_dur
1.	166	2011	2	58.70834	6.5
2.	166	2012	1	59.83333	4
3.	276	2009	1	55.75	9
4.	548	2005	3	49.91667	3.666667
5.	548	2006	2	51	9.5
6.	548	2008	1	52.66667	4
7.	579	2005	3	57.36111	9.666667
8.	579	2006	1	58.75	4
9.	579	2007	3	59.25	8.333333

Using `reshape` to reorganize data

Scott Imberman
@scottimberman

Folgen

Hello. My name is Scott. I have a PhD in Economics and I work for a major Big10 university and I just had to look up the syntax for the reshape command in Stata for the thirty thousandth time.

Joby Hollis @Jobium
Hello my name is Joby, I have a PhD in Physics and I work for NASA and I just had to look up the equation for the volume of a sphere
Diesen Thread anzeigen

Tweet übersetzen

04:21 - 4. Sep. 2018

146 Retweets **1.511** „Gefällt mir"-Angaben

39 146 1,5 Tsd.

Using `reshape` to reorganize data

- `reshape` is an essential command to use for data transformation. It's syntax is difficult and it often takes multiple runs of debugging to get it working.
- Before we see what it does, you have to understand how Stata distinguishes between **wide** and **long** data formats.
- This is the long format:

```
. list if id_worker <= 579, sepby(id_worker)
```

	id_wor~r	year	sl
1.	276	2009	9
2.	548	2005	3.666667
3.	548	2006	9.5
4.	548	2008	4
5.	579	2005	9.666667
6.	579	2006	4
7.	579	2007	8.333333

Using `reshape` to reorganize data

- This is the wide format:

```
. list if id_worker <= 579
```

	id_wor~r	sl2005	sl2006	sl2007	sl2008	sl2009
1.	276	.	.	.	.	9
2.	548	3.666667	9.5	.	4	.
3.	579	9.666667	4	8.333333	.	.

- In long format, one datum (i.e., one unique observation) is identified as a combination of `id_worker` and `year`, in wide format a datum is identified by a single variable.

- Why would you want to change between those formats?
  - Panel data are typically stored in long format.
  - The wide format is useful to perform rowwise calculations.
  - Graphical representations of results often require one or the other format.

Using `reshape` to reorganize data

Syntax generally

long

i	j	$stub$
1	1	x_{11}
1	2	x_{12}
2	1	x_{21}
2	2	x_{22}

`reshape` ⟷

wide

i	$stub\mathbf{1}$	$stub\mathbf{2}$
1	x_{11}	x_{12}
2	x_{21}	x_{22}

- From **long** to **wide** (with j being an existing var):

```
reshape wide stub, i(i) j(j)
```

- From **wide** to **long** (with j being a new var):

```
reshape long stub, i(i) j(j)
```

- **NOTE** Use the *string* option if `j()` *may* contain string values, and be aware that you can use `@` if your stubs contain suffixes after the j's.

Using `reshape` to reorganize data

Stata examples

```
. // as before, let´s create a dataset containing averages sl durs for every worker
. collapse (mean) sl=sl_dur if inrange(year,2005,2009), by(id_worker year)
. // now, reshape the data into wide format
. reshape wide sl, i(id_worker) j(year)
(note: j = 2005 2006 2007 2008 2009)
Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                   108169   ->   43184
Number of variables                   3   ->       6
j variable (5 values)              year   ->   (dropped)
xij variables:
                                     sl   ->   sl2005 sl2006 ... sl2009
-----------------------------------------------------------------------------
```

```
. list if id_worker <= 1803, sep(0)
```

	id_wor~r	sl2005	sl2006	sl2007	sl2008	sl2009
1.	276	.	.	.	.	9
2.	548	3.666667	9.5	.	4	.
3.	579	9.666667	4	8.333333	.	.
4.	1063	12	.	.	8	8
5.	1085	.	.	5.5	.	.
6.	1689	8.666667	.	.	9	.
7.	1738	4	3	3.5	1	.
8.	1788	28	7.5	.	.	.
9.	1803	.	19.5	13	.	.

Using `reshape` to reorganize data

Stata examples

```
. // suppose our using data is in wide format, as on the last slide, but the sl vars contain suffixes
. foreach v of varlist sl* {
  2.         rename `v´ `v´_total
  3. }
. // then the reshape command back to long format has to look like this:
. reshape long sl@_total, i(id_worker) j(y) /* with year being a newly generated var */
(note: j = 2005 2006 2007 2008 2009)
Data                                wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                     43184   ->  215920
Number of variables                    6   ->       3
j variable (5 values)                      ->   y
xij variables:
sl2005_total sl2006_total ... sl2009_total->   sl_total
-----------------------------------------------------------------------------
```

```
. list if id_worker <= 548, sepby(id_worker)
```

	id_wor~r	y	sl_total
1.	276	2005	.
2.	276	2006	.
3.	276	2007	.
4.	276	2008	.
5.	276	2009	9
6.	548	2005	3.666667
7.	548	2006	9.5
8.	548	2007	.
9.	548	2008	4
10.	548	2009	.

C.3

Combining data

Relational databases in general

- A **database** is simply a set of related datasets. So far we have encountered only single datasets, but these are parts of much larger databases.
  - `pdmv_sl.dta` is drawn from the *Austrian Social Security Database*
  - `UA_population.csv` is from the *Statistics Austria* database
- A **relational database** organizes one or more datasets (or 'relations') consisting of columns and rows, with a **unique key** identifying each row.
  - In CS, rows are also called records or tuples. Columns are also called attributes.
- Generally, each table/relation/dataset represents one **entity type** (such as workers, sick leaves, or districts). The rows represent **instances** of that type of entity (one sick leave), and columns represent **values** attributed to that instance (start of the sick leave, information on the worker, and so on).
- Different datasets can be linked through the **unique keys.**
  - What is the unique key in `pdmv_sl.dta`?

Store data in relational databases!

- Avoid storing data in tables like this one:

zip	district	zip_pop	district_pop
4020	401	107236	182304
4040	401	75068	182304
.	401	.	182304
4400	402	30509	38347
.	402	.	38347

- Instead, set up a relational database:

zip	district	pop
4020	401	107236
4040	401	75068
4400	402	30509

district	pop
401	182304
402	38347

- Now there is **no ambiguity** $\longrightarrow$ no missing zip codes and no conflicting definitions. If the pop of a district is required, the second data can be used directly. The database is self-documenting, and each table has a **unique key.**

- These datasets can easily be combined through their unique keys.

Types of relations in databases

Consider two datasets A and B. Depending on the information in these datasets, more specifically **how many obs are attached to every unique id**, these datasets can be related in the following ways:

- A **one-to-one** (1:1) relationship between A and B exists, when one row in table A *may be* linked with only one row in table B, and vice versa.[1]
- A **one-to-many** (1:m) relationship exists, when one row in table A may be linked with many rows in table B, but one row in table B is linked to only one row in table A (note the equivalence with a many-to-one relationship).
- A **many-to-many** (m:m or n:m) relationship is given when multiple entries in A relate to multiple entries in B. Typically we would transform one of the datasets before merging, in order to get a 1:m relationship. More on that later.

[1] Note the conditional *may be*: These relationships can exist, but they don't necessarily have to. For example, there may be rows in A that cannot be mapped to any row in B, yet the datasets may still be in a 1:1, 1:m, or m:m relationship.

New datasets

- We consider three new datasets in this module:
  - `pdmv_children.dta`
  - `pdmv_pnat.dta`
  - `pdmv_addinfo.dta`

`pdmv_sl`	
Sick leaves 2005-2012 for 10% sample of Upper Austrian workers	
Unique ID:	`id_worker sl_start`

`pdmv_addinfo`	
Diagnosis and living place for all sick leaves in Upper Austria (2005-2012).	
Unique ID:	`id_worker sl_start`

`pdmv_pnat`	
Nationalities of all UA workers that had a sick leave (2005-2012).	
Unique ID:	`id_worker`

`pdmv_children`	
All children along with their birth year for mothers in Austria, 1972-2009.	
Unique ID:	`id_mother`

Combine datasets

Dataset 1

id	var1	var2
111	...	...
112	...	...
113	...	...

Dataset 2

id	var1	var2
121	...	...
122	...	...
123	...	...

Dataset 3

id	var3	var4
111	...	...
112	...	...
113	...	...

Combine datasets

- Note that these three datasets contain different information $\longrightarrow$ Dataset 1 and Dataset 2 contain the same vars for different `id`'s, while Dataset 3 contains the same `id`'s but different vars as in Dataset 1.

- These are simple cases, because one `id` always identifies one **unique observation**, or equivalently, per unique observation (`id`) there is only one row with information.

- As long as `id` has the same name and data type (string vs. numeric) across the two datasets, 1 and 2 can be combined using the `append` command. To combine 1 and 3 we would use the `merge` command.

- In Stata, the dataset in memory is called the **master** data, while the dataset which is to be merged or appended is called **using** data.

Append data

- Sometimes you have to download datasets from different years or different entities **separately** (e.g., survey waves). Combining these datasets into one would be a task for `append`. [Manual link: Append]

- In terms of the example above, it would mean to simply **stack** Dataset 1 and Dataset 2:

id	var1	var2
111	...	...
112	...	...
113	...	...
121	...	...
122	...	...
123	...	...

- Importantly, it is no problem if one of the datasets contains different vars apart from `id`. In this case Stata attaches missings for `id`'s that don't have information on the var which is non-existing in the original data.

Append data

Datasets with different vars

Dataset 1

id	var1	var2
111	...	...
112	...	...
113	...	...

Dataset 2

id	var1	var2	var3
121	...	...	...
122	...	...	...
123	...	...	...

Dataset 1+2 (after append)

id	var1	var2	var3
111	...	...	
112	...	...	
113	...	...	
121	...	...	...
122	...	...	...
123	...	...	...

Append data

In Stata

```
. // store data by years for illustrative purposes
. count
  322,375
. forval i = 2004/2012 {
  2.         preserve
  3.         keep if year == `i´
  4.         qui save data/tmp_`i´.dta, replace
  5.         restore
  6. }
(321,931 observations deleted)
(286,725 observations deleted)
(288,744 observations deleted)
(284,133 observations deleted)
(279,967 observations deleted)
(281,779 observations deleted)
(282,164 observations deleted)
(276,996 observations deleted)
(276,561 observations deleted)
. // append data again
. use data/tmp_2004.dta, clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. forval i = 2005/2012 {
  2.         qui append using data/tmp_`i´.dta
  3.         erase data/tmp_`i´.dta
  4. }
. count
  322,375
. erase data/tmp_2004.dta
```

Merge data

- Now consider the case where you have two datasets, and both contain **different** information on a unique identifier that you want to combine into a single dataset. [Manual link: Merge]

- For example, you may want to combine Dataset 1 and Dataset 3 from above:

id	var1	var2	var3	var4
111	...	...	...	...
112	...	...	...	...
113	...	...	...	...

- This is a simple 1:1 merge. In Stata, such a merge can be done using the `merge` command. Its syntax is as follows:

  `merge 1:1` *`varlist`* `using` *`filename`*

  where *`varlist`* is the common unique identifier in both datasets.

- The newly introduced dataset `pdmv_addinfo.dta` also allows a 1:1 merge, because it contains exactly one observation for every `id_worker sl_start` combination (i.e., every individual sick leave) as well.

- Let's try the merge and examine what it does:

```
. merge 1:1 id_worker sl_start using "D:\Dropbox\pdmv\data\pdmv_addinfo.dta"

    Result                           # of obs.
    -----------------------------------------
    not matched                     7,139,443
        from master                         0  (_merge==1)
        from using                  7,139,443  (_merge==2)

    matched                           322,375  (_merge==3)
    -----------------------------------------
```

- First, we observe that the data contains now 7,461,818 obs. Browsing the data, we find that 7,139,443 obs have information on the newly added vars `sl_diag p_plz p_gkz`, but not on the original vars.

- Second, we observe that Stata generates a new variable called `_merge`. This variable takes on
  - 1 if an observation was found only in the master dataset (`pdmv_addinfo`),
  - 2 if it was found only in the using dataset, or
  - 3 if it was found in both datasets.

1:1 merges

- This means that a **successful merge** is flagged with `_merge == 3`. Obs with `_merge == 1` appear only in the master data, such obs do not exist as all worker–sick leave combinations also have entries in the master data.

- Why are there 7m obs in the using data that could not be found in `pdmv_sl.dta`? Remember the data description $\longrightarrow$ this is only a 10% sample of all workers!

- In most applications, it makes sense to automatically drop obs with `_merge == 2`, and then drop the `_merge` var altogether.

```
. merge 1:1 id_worker sl_start using "D:\Dropbox\pdmv\data\pdmv_addinfo.dta"

    Result                           # of obs.
    -----------------------------------------
    not matched                     7,139,443
        from master                         0  (_merge==1)
        from using                  7,139,443  (_merge==2)

    matched                           322,375  (_merge==3)
    -----------------------------------------

. drop if _merge == 2
(7,139,443 observations deleted)

. drop _merge
```

  - If you want to keep `_merge` for future reference, give it a different name!

1:1 merges

```
. list id_worker sl_start sl_diag p_plz p_gkz if id_worker <= 1063, sepby(id_worker)
```

	id_wor~r	sl_start	sl_diag	p_plz	p_gkz
1.	166	04jun2011	J069	4331	411
2.	166	09nov2011	J040	4331	411
3.	166	20oct2012	J40	4331	411
4.	276	15jan2009	J069	4470	410
5.	548	18feb2005	J10	4770	414
6.	548	04jul2005	M544	4770	414
7.	548	05dec2005	I95	4770	414
8.	548	09aug2006	J02	4770	414
9.	548	28aug2006	M53	4770	414
10.	548	10apr2008	M5384	4770	414
11.	579	24jan2005	M53	4600	403
12.	579	02mar2005	J068	4600	403
13.	579	02dec2005	M53	4600	403
14.	579	10oct2006	J068	4600	403
15.	579	11jan2007	J068	4600	403
16.	579	19jan2007	J068	4600	403
17.	579	12oct2007	J068	4600	403
18.	1063	01apr2005	M652	4040	401
19.	1063	22sep2008	J068	4040	401
20.	1063	01oct2008	M199	4040	401
21.	1063	08sep2009	M771	4040	401
22.	1063	07apr2010	M545	4040	401
23.	1063	27oct2010	M5494	4040	401

1:1 merges

- Note that in this example, it appears that `p_plz` and `p_gkz` do not change across sick leaves. If this is really the case, it would make more sense to store the information on living place in a **separate database which contains only one obs per unique id** `id_worker`.

- However, since the zip code is stored in numeric format, we can easily check whether it contains different values across `id_worker`'s by computing its sd:

```
. egen sd_plz = sd(p_plz), by(id_worker)
(10051 missing values generated)

. su sd_plz
```

Variable	Obs	Mean	Std. Dev.	Min	Max
sd_plz	312,324	23.79653	126.7822	0	3852.318

- Thus, if people move, their living place changes between sick leaves. This is the reason why I stored `p_plz` and `p_gkz` in this data table.

Other merges

- Instead of 1:1 merges, Stata can also perform m:1 (or 1:m) and m:m merges, these *joins* are also the only part of the syntax you have to change.

```
merge m:1 varlist using filename
merge 1:m varlist using filename
merge m:m varlist using filename
```

- **Avoid m:m merges at all cost!**
  - An m:m relationship means you have multiple observations in both datasets for some values of the merge key variable(s).
  - Usually the result of merging two datasets that both have more than one value of the merge key variable(s) is unpredictable, because it **depends on the sort order** of the datasets.
  - Repeated execution of the same do-file will likely result in a different number of cases in the result dataset without any error indication. **A m:m merge has no unique outcome.** When it is encountered, it usually results from a coding error in one of the files.
  - Use `joinby` instead, which generates all possible pairwise combinations within groups (see later).

m:1 merges

- Suppose you want to extend the sick leave data with workers' nationalities. These datasets are in a m:1 relationship, as one `id_worker` appears only once in `pdmv_pnat.dta`, but every worker in `pdmv_pnat.dta` can appear many times in `pdmv_sl.dta`.

```
. merge m:1 id_worker using "D:\Dropbox\pdmv\data\pdmv_pnat.dta"

    Result                           # of obs.
    -----------------------------------------
    not matched                       721,213
        from master                         0  (_merge==1)
        from using                    721,213  (_merge==2)

    matched                           322,375  (_merge==3)
    -----------------------------------------

. drop if _merge == 2
(721,213 observations deleted)

. drop _merge
```

- Again, we drop the 721,213 workers that have nationalities but are not in our sick leave data (`_merge == 2`). We have 322,375 perfect matches, and no obs. in the sick leave data that do not appear in the nationalities data (although they can have missings).

m:1 merges

```
. list id_worker sl_start p_nat if id_worker <= 1063, sepby(id_worker)
```

	id_wor~r	sl_start	p_nat
1.	166	09nov2011	.
2.	166	04jun2011	.
3.	166	20oct2012	.
4.	276	15jan2009	Österreich
5.	548	09aug2006	Österreich
6.	548	28aug2006	Österreich
7.	548	04jul2005	Österreich
8.	548	10apr2008	Österreich
9.	548	18feb2005	Österreich
10.	548	05dec2005	Österreich
11.	579	10oct2006	Österreich
12.	579	11jan2007	Österreich
13.	579	12oct2007	Österreich
14.	579	02mar2005	Österreich
15.	579	24jan2005	Österreich
16.	579	19jan2007	Österreich
17.	579	02dec2005	Österreich
18.	1063	07apr2010	Österreich
19.	1063	22sep2008	Österreich
20.	1063	08sep2009	Österreich
21.	1063	01oct2008	Österreich
22.	1063	27oct2010	Österreich
23.	1063	01apr2005	Österreich

Merging data

Merging data can be extremely **memory-intensive** with big datasets. If your data matrix is already huge, its dimension grows even further when you attempt a merge. However, there are certain tricks you can use:

- Open a second Stata instance and load the using data. Check whether you can collapse the information you need first, or reduce its size by some other means.
- Use the `keepusing` option to merge only variables you really need from the using data. This can often make the difference between a successful merge and Stata returning a memory error.
- Sometimes, there may also be vars in the **master data you don't necessarily need**. Drop these before the merge.
- Specify `keep(1 3)` to save some additional time.
- If possible, increase Stata's `matsize`.

m:m merges

- Sometimes you have to combine datasets where multiple entries in the using data refer to multiple entries in the master data.
- For example, you may want to match data on all children of women in the data from `pdmv_children`.
- Forget about `merge m:m`.
- Sometimes it's viable to reshape one of the data sets first (for example, put children in columns instead of rows in `pdmv_children` before merging).
  - This is only possible if the second data dimension is small (i.e., few children per mother).
- Instead, use `joinby`.

What `joinby` does

- `joinby` builds all pairwise combinations within defined groups in the merged dataset. It's syntax looks as follows:

  `joinby` *`varlist`* `using` *`filename`*

  where *`varlist`* defines groups in the data.

- Example:

Dataset 1

id	var1	var2
1	...	...
2	...	...

Dataset 2

id	child_id	var3
1	x1	...
1	x2	...
2	y1	...

Dataset 1+2 (after `joinby` on `id`)

id	child_id	var1	var2	var3
1	x1	...	...	...
1	x2	...	...	...
2	y1	...	...	...

```
. use "D:\Dropbox\pdmv\data\pdmv_children.dta", clear
([PDMV Ahammer 2018] all births between 1971-2007 for Austrian women)
. list if id_mother == 166
```

	id_child	id_mot~r	child_~b
396068.	4497187	166	1977
396069.	.	166	1977
396070.	10041105	166	1981

```
. use if p_female == 1 using "data/pdmv_sl.dta", clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. g double id_mother = id_worker
. list id_worker sl_start sl_end if id_mother == 166
```

	id_wor~r	sl_start	sl_end
1.	166	09nov2011	13nov2011
2.	166	04jun2011	11jun2011
3.	166	20oct2012	23oct2012

```
. joinby id_mother using "D:\Dropbox\pdmv\data\pdmv_children.dta", _merge(_kidsjoin) unm(m)
. sort id_mother id_child
. list id_worker sl_start sl_end *child* if id_mother == 166, sepby(id_child)
```

	id_wor~r	sl_start	sl_end	id_child	child_~b
1.	166	20oct2012	23oct2012	4497187	1977
2.	166	04jun2011	11jun2011	4497187	1977
3.	166	09nov2011	13nov2011	4497187	1977
4.	166	20oct2012	23oct2012	10041105	1981
5.	166	04jun2011	11jun2011	10041105	1981
6.	166	09nov2011	13nov2011	10041105	1981
7.	166	09nov2011	13nov2011	.	1977
8.	166	20oct2012	23oct2012	.	1977
9.	166	04jun2011	11jun2011	.	1977

What `joinby` does

- In the last example you saw how to use `joinby` to combine two datasets where each contains multiple entries for `id_mother`.
- In fact we may have multiple sick leaves per mother in `pdmv_sl` and possibly multiple children in `pdmv_children`.
- Mother 166 has had 3 sick leaves and 3 children, she will therefore have 3×3 obs in the resulting dataset.
- In `joinby` you should always specify the `unmatched()` option, which specifies how Stata treats observations that cannot be matched between the joined datasets:
  - `unm(master)` keeps only unmatched obs from the **master** data in memory.
  - `unm(using)` keeps only unmatched obs from the **using** data.
  - `unm(both)` keeps both unmatched obs from **both** the master and using data.
  - `unm(none)` **drops** all unmatched obs.

C.4

Further data transformation commands

expand

- You may remember the `expand` command, we have used it in the past.
- With `expand` you can copy each observation in the data by a certain factor.
  - `expand` *c* copies each observation by a constant *c*.
  - `expand` *var* copies each obs by the value of *var*.
- This can sometimes be helpful, especially if you work with spell data.
  - *Example:* If you make an event study based on spell data, you may want to expand the data by the number of pre- and post-event time periods you want to analyze (see module B).
  - *Example:* You have data on employment spells which you want to convert into a yearly panel. In this case you have to expand the data first by the number of years each spell spans before collapsing.
  - *Example:* See my solution to the week 6 problem set, where we use `expand` to calculate the number of days worked per quarter based on spell data.
- `help expand`

expand

```
. // expand each obs by the same factor
. use "data/pdmv_sl.dta", clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. sort id_worker sl_start
. list id_worker sl_start if id_worker <= 276
```

	id_wor~r	sl_start
1.	166	04jun2011
2.	166	09nov2011
3.	166	20oct2012
4.	276	15jan2009

```
. expand 2, gen(_e)
(322,375 observations created)
. sort id_worker sl_start _e
. list id_worker sl_start _e if id_worker <= 276, sepby(id_worker sl_start)
```

	id_wor~r	sl_start	_e
1.	166	04jun2011	0
2.	166	04jun2011	1
3.	166	09nov2011	0
4.	166	09nov2011	1
5.	166	20oct2012	0
6.	166	20oct2012	1
7.	276	15jan2009	0
8.	276	15jan2009	1

expand

```
. // expand according to the value of a variable
. use "data/pdmv_sl.dta", clear
(All sick leaves 2004-2012 for 10% sample of Austrian employees)
. sort id_worker sl_start
. g r = int(runiform()*3) + 1              /* generates a random integer */
. list id_worker sl_start r if id_worker <= 276
```

	id_wor~r	sl_start	r
1.	166	04jun2011	3
2.	166	09nov2011	1
3.	166	20oct2012	2
4.	276	15jan2009	1

```
. expand r, gen(_e)
(322,481 observations created)
. sort id_worker sl_start _e
. list id_worker sl_start r _e if id_worker <= 276, sepby(id_worker sl_start)
```

	id_wor~r	sl_start	r	_e
1.	166	04jun2011	3	0
2.	166	04jun2011	3	1
3.	166	04jun2011	3	1
4.	166	09nov2011	1	0
5.	166	20oct2012	2	0
6.	166	20oct2012	2	1
7.	276	15jan2009	1	0

`fillin`

- Suppose you have panel data (as in `pdmv_sl`). Remember the definition of a balanced panel: Each $i \in n$ observations has **exactly** T time-series observations.

- Many panels are unbalanced, meaning that each observation may have a different number of time-series observations $T_i \leq T, T_j < T$ for some j.

- Sometimes you may want to fill-up these obs to create a **balanced panel**.
  - *Example:* You have data on medical drug usage for 2005–2012, but some patients do not consume drugs in every year. A balanced panel would require to create obs for missing years and replacing drug usage with 0 (given that you know you would indeed observe drug usage if the patient consumed drugs).
  - *Example:* Again, see my solution to the week 6 problem set, where we make a balanced quarterly panel using the data in `pdmv_sl`.

- This works both with the `fillin` command or using `tsfill` after `tsset`.

- `help fillin`

cross, stack, separate, xpose

- `cross` forms every pairwise combination of the data in memory with the data in the using dataset.
  - Syntax: `cross using` *`usingdata.dta`*
  - `help cross`
- `stack` allows you to vertically stack the variables in a varlist.
  - Syntax: `stack` *`x1 x2 x3 x4`*`, into(`*`z1 z2`*`)`
  - `help stack`
- `separate` can convert one var into several new vars, either on the basis of a Boolean condition or in terms of another var given by *`by()`*.
  - Syntax: `separate` *`var1`*`, by(`*`var2`*`)`
  - `help separate`
- `xpose` turns observations into variables and vice versa.
  - `help cross`

User's corner: ftools and gtools

Do you need to perform data management tasks or compute summary statistics with very large datasets? If so, and if you find yourself wishing you could do these things more quickly, you won't want to miss the solutions provided in the community-contributed **ftools** and **gtools** suites.

ftools, written by Sergio Correia, includes commands such as **fcollapse** for creating datasets of summary statistics, **fmerge** for merging datasets with one-to-one or many-to-one matching, and **flevelsof** for identifying distinct levels of a variable. The **ftools** suite also includes a Mata class for working with factors. The Stata commands in **ftools** are based on this Mata class. You can use it to implement your own commands too. **ftools** is available from the **Statistical Software Components (SSC) archive**, and you can read more about these commands in **Sergio's slides** from his presentation at the 2017 Stata Conference and his **GitHub page**.

gtools, by Mauricio Caceres, includes commands for quickly creating datasets of summary statistics (**gcollapse**), identifying distinct levels of a variable (**glevelsof**), and much more. **gtools** uses a C plugin and is even faster than **ftools** for tasks that can be performed using both suites of commands. **gtools** is available from the **Statistical Software Components (SSC) archive**, and you can read all about these commands **here**.

$\Longrightarrow$ `ftools` https://github.com/sergiocorreia/ftools

$\Longrightarrow$ `gtools` https://gtools.readthedocs.io/en/latest/

Both are available from the SSC archive, so `ssc install` them!