

Part B. EpiData Analysis

Part B: EpiData Analysis

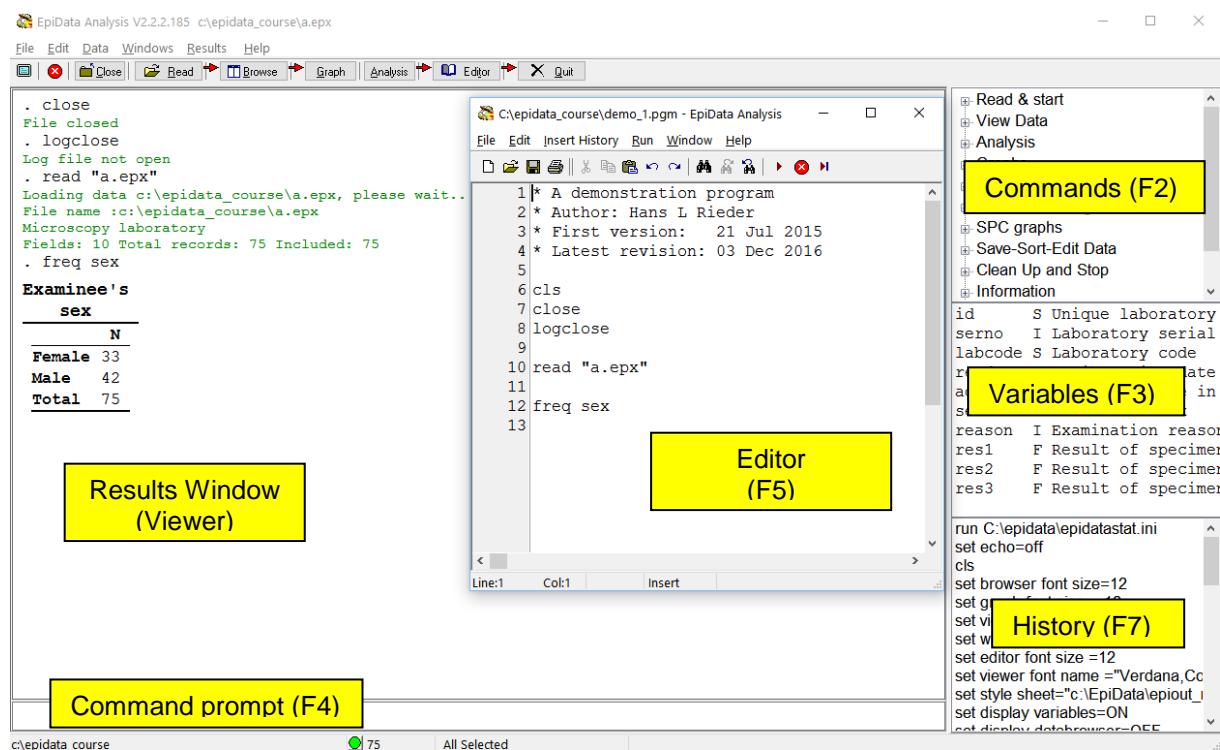
- Exercise 1 An introduction to EpiData Analysis
- Exercise 2 More on EpiData Analysis
- Exercise 3 Aggregating data and saving the summary data in a file
- Exercise 4 From a spreadsheet to an EpiData file

Exercise 1: An introduction to EpiData Analysis

At the end of this exercise you should be able to:

- Do some analyses using the commands on the menu bar.
- Use the 'Editor' to write commands into a program that can be saved to make a permanent record.
- Do some calculations.
- Create new variables.

The image below shows the EpiData Analysis interface with all components open:



The screen shot shows the various windows:

On the right:

- With **F2** (toggle key) you access various Commands
- With **F3** (toggle key) you see information on field names, types, and labels
- With **F7** (toggle key) you display the History of commands in the current session (which you can paste into the Editor which you access with **F5** (center)).

In the center:

- With **F5** you access the Editor in which we will be writing our program (script) which can be run as a batch and will produce output in the Results window (the Viewer)

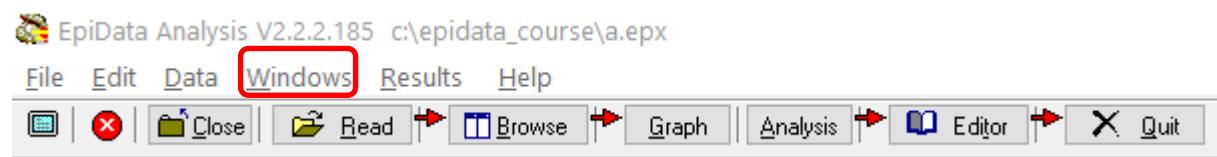
On the left:

- The Viewer or the Results window, here after running the program shown in the Editor. On the top you see in black the commands interspersed with green information on what is being done / has been done and finally the requested output table of a frequency for the variable sex.

At the bottom:

- Accessible with mouse-click or **F4** is the Command line into which you can type any command (we could have written FREQ sex here instead of into the Editor, provided that the file was open). Below the Command line, from left to right, you see the current folder including the path to it, a green button indicating readiness, the number of records (75) and that all of them from the dataset a.epx have been selected. That the open data file is the a.epx can be seen at the very top of the screen.

If we look at more details, we find on top of the screen:

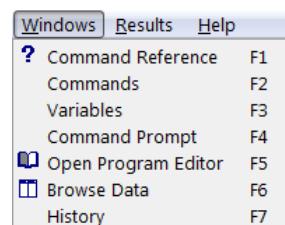


Top row: EpiData version (important to specify when reporting bugs) and path and currently open file name.

Center row: Menu bar

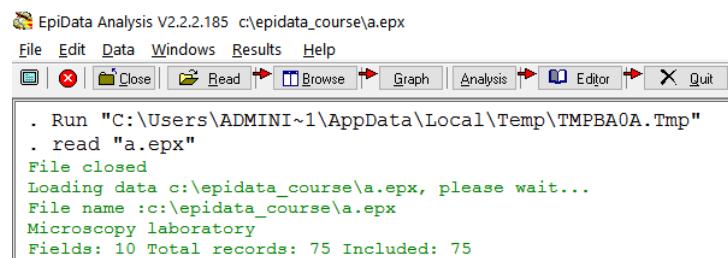
Bottom row: Work tool bar

If you select in the Menu bar Windows you get a drop-down menu:



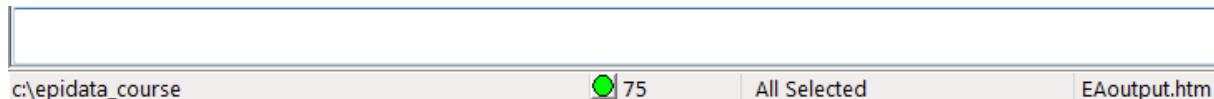
This gives you spelled out the names of the various command and windows and the shortcuts **F1** ,..., **F7** that can be used to access them.

If we open a file, the Viewer (Results window) gives information (written in green) on File name, File label, number of Fields and number of Records:



At the bottom of the screen (below the command line) we find the location (path) on the very left, in the center the green bullet indicating readiness, and right to it the number of records

(75 here) available, further to the right that the entire dataset is available (All selected), and at the very right that EpiData Analysis has opened an output file EAoutput.htm:



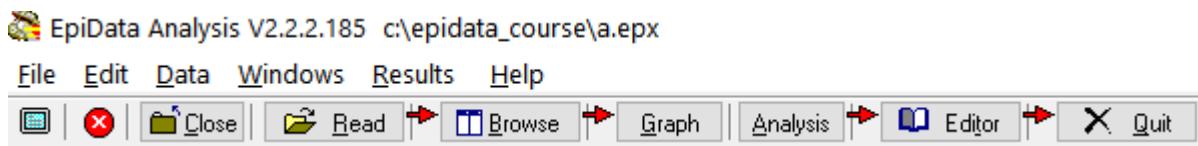
We can double-click on the path (c :\epidata_course here) if we wish to change it and get a pop-up box to Browse for the desired data Folder (see more on that below).

We can only summarize some basic essentials in the approach to the use of EpiData Analysis and we will split it into the following components:

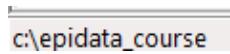
- Using the menu interactively
- Writing a script (program) to execute a series of commands
- Analysis of continuous variables
- Analysis of categorical variables

Using the menu interactively

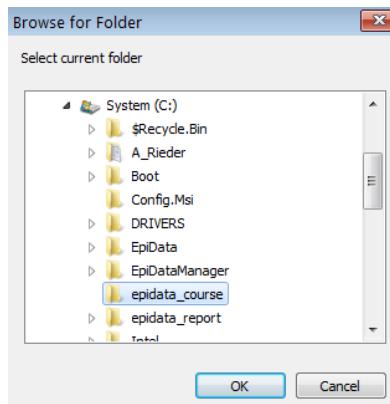
When you open EpiData Analysis, you see information on Version and currently open dataset in the top row, the Menu bar in the second and the Work tool bar in the third row:



For beginners, it is best to use the Menu and Work tool bars. Before anything else, we need to tell EpiData where our data files are located. This can be done in different ways. The simplest way is probably to double-click at the bottom left on the path that is there:



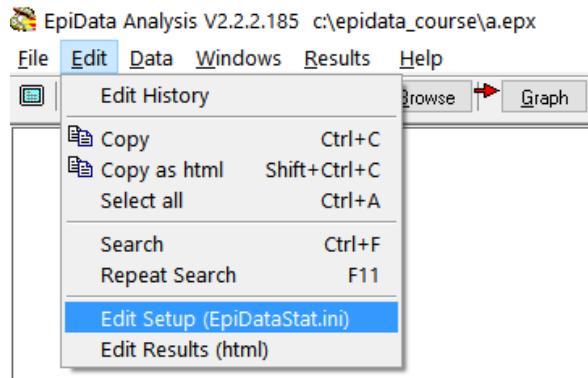
This will open an Explorer-like window:



allowing searching for the desired folder. This folder selection will persist during the current session (unless you change it during the session), but once you exit and re-enter EpiData Analysis, the path will again be the default path. Therefore, it is desirable to get a bit more control over the default path and folder that is opened when you start EpiData Analysis. Commonly, one works in project folders like we do here where the:

..\\epidata_course

is our project folder. On the top you have Edit with a pop-up menu:



If you open Edit Setup (EpiDataStat.ini), you get the file opened in the Editor allowing us to customizing it to our needs:

A screenshot of the EpiData Analysis software interface. The window title is "C:\EpiData\epidatadatastat.ini - EpiData Analysis". The menu bar includes File, Edit, Insert History, Run, Window, and Help. A context menu is open under the Edit menu, listing options like Copy, Copy as html, Select all, Search, Repeat Search, Edit Setup (EpiDataStat.ini) (which is highlighted in blue), and Edit Results (html).

```
1 set echo=off
2 cls
3
4 set browser font size=12
5 set graph font size =12
6 set viewer font size =12
7 set window font size =12
8 set editor font size =12
9 set viewer font name ="Verdana,Courier"
10 set style sheet="c:\epidata\epiout_monospace.css"
11 set output folder="C:\EpiData\TestData"
12 set language=english
13
14 set display variables=ON
15 set display databrowser=OFF
16 set output open=ON
17
18 * Working folders
19 cd c:\\temp
20 cd c:\\epidata_course
21
22 set echo=ON
```

The code block shows the contents of the EpiDataStat.ini file. Lines 4 to 16 define various system settings like font sizes and styles. Lines 18 to 20 are specifically for setting the working directory to "c:\\epidata_course". Line 22 sets the echo level back to ON.

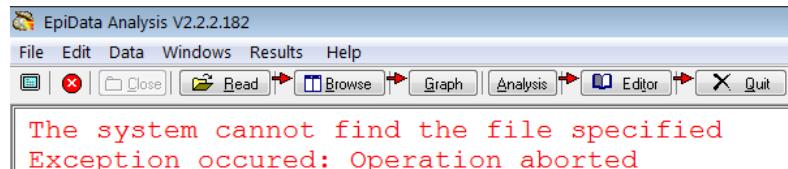
In lines 4 to 16 above we set font sizes, font types and other things (that can be changed according to your needs). It is lines 18 to 20 that are of interest here for the data folder. We start in line 19 with:

[drive name]:\\folder in the PC root\\subfolder\\sub-sub-folder

You note specifically in line 20 cd c:\\epidata_course. It is this line which directs EpiData Analysis to open at start-up in that folder. If we had not customized the starting folder in this EpiDataStat.ini file, EpiData Analysis would start in its program file folder.

In both lines 19 and 20 above you find the command “cd”. “CD” denotes “Change Directory” (a term formerly used to denote for what Microsoft denotes now as “folder”). In line 19, EpiData is instructed to go to a folder temp (and will do so if it exists, but stops if it doesn’t). In line 20, the previous instruction is overridden and it will end up (the last instruction sets the

final destination) in the `epidata_course` folder. We can thus list our different working folders and just rearrange them so that the currently desired is the last one. Note though that each of the listed folders must exist, else you get an error message at the start of EpiData Analysis and it will stop at the last “legal” folder found in the list of folders you may have:



Opening an EpiData data set

The command to open an EpiData EPX or REC file is `Read` which can be accessed by clicking on the respective icon in the menu (or with the short-cut **ALT+R**). Make sure (the first time) that you are in the right folder “`c:\epidata_course`” to choose the file:

`a.epx`

from the `[C:]\\epidata_course` folder. In the Viewer window you should then get:

```
. read
File closed
Loading data C:\epidata_course\ a.epx, please wait...
File name :C:\epidata_course\ a.epx
Microscopy laboratory
Fields: 10 Total records: 75 Included: 75
```

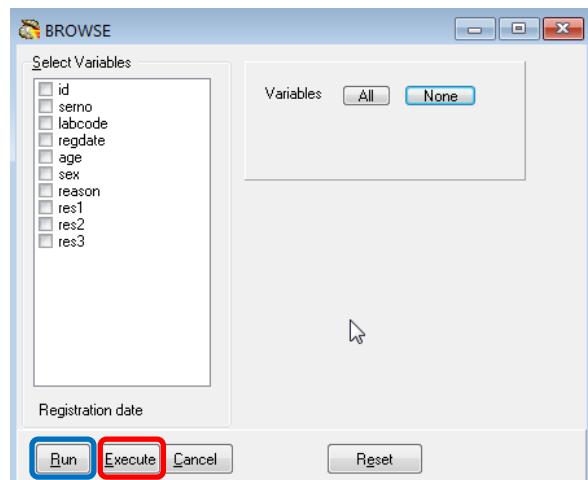
Below the command line you note:

C:\epidata_course 75 All Selected

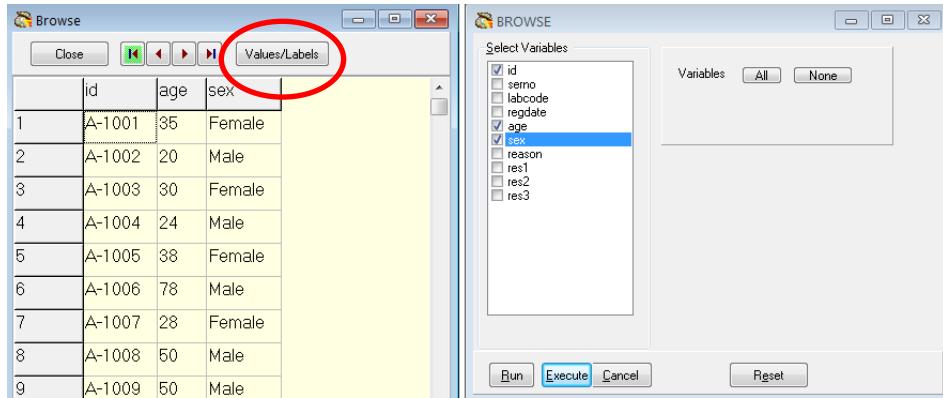
that all of the 75 records are selected. You will see later that any selections you make are listed here and the remaining number of records is shown.

Browsing the EpiData data set

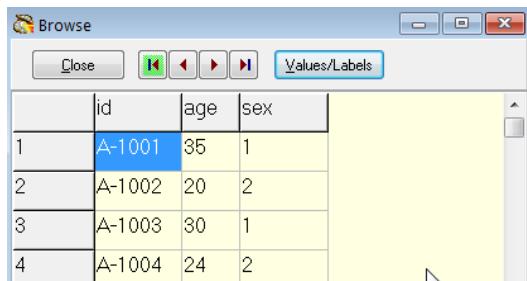
Clicking on `Browse` (or short-cut **ALT+B**) gives a menu box:



where you can choose All or ticking which ones you need specifically. Note at the bottom the possibilities Run and Execute (circled above in blue and red respectively). The difference is that Run shows the results and simultaneously closes this selection box. If you change your mind, you have to reopen it and start again from scratch. In contrast, with Execute this window with all selections stays open and you can modify easily if you dislike what you got:

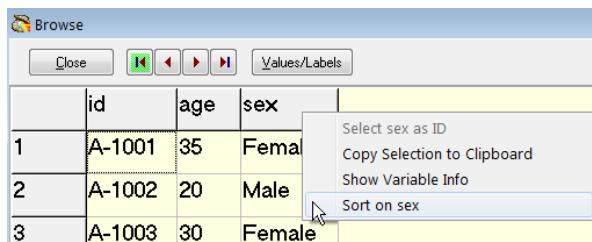


Note the “switch” (toggle key) box Values/Labels. If you click on it you get:



If the toggle key is on the Values, the information in the older EpiData file format comes from the *.REC file, if on the Labels, it comes from the *.CHK file, in the *.EPX file format, there is only one file. Thus, if the *.CHK was not in the folder, the toggle key would not work and only the Values would be seen. You can thus BROWSE either the Field values or the Field labels.

If you right-click on the field name (e.g., sex):



You can **Copy** the column (you can also copy one or more cells or rows) to **Clipboard**, or you can **Sort** the data set on the chosen field.

If you go to the command line (**F4**), and use the up-arrow key, you get the last command:

```
BROWSE id age sex
```

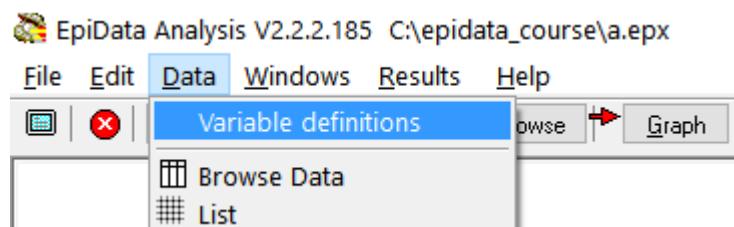
Now you know that the actual syntax to browse is the command BROWSE followed by the variable(s). Note that as in EpiData Entry, the capitalizing is just the way that EpiData shows the commands, it is not something you are required to use.

The use of the interactive approach with clicking can thus be used to see the actual commands that are behind the clicking which will come in handy when we start writing a program. All what we do is also written into the history which can be accessed from Windows or with the short-cut key **F7**, where the last lines read:

```
Read
Read "C:\epidata_course\a.epx"
BROWSE id serno labcode regdate age sex reason res1 res2 res3
BROWSE id age sex
```

Again, to make life easier for the beginner, the history can be invoked and pasted into the Editor, and then edited for the program.

Browsing allows seeing variables and their values for each record, but one would also like to know all possible values and labels for a given variable. In the Menu bar, choose Variable definitions from the drop-down menu of Data:

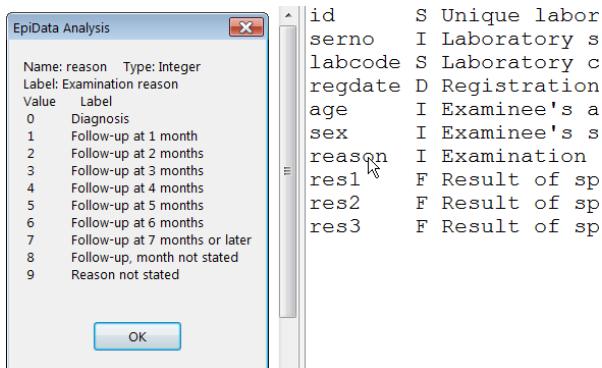


and in the Viewer you get (beginning only shown):

```
- var
File: C:\epidata_course\a.epx
      Microscopy Laboratory
Name   Type    Length Decimals Label           Value label          Missing
id     String   6       0    unique laboratory identifier
serno  Integer  4       0    laboratory serial number
labcode String   1       0    laboratory code
                                A = Laboratory A
                                B = Laboratory B
                                C = Laboratory C
                                D = Laboratory D
regdate Date DMY 10     0    registration date
age    Integer  2       0    examinee's age in years
sex    Integer  1       0    examinee's sex
                                1 = Female
                                2 = Male
                                9 = Sex not recorded
```

Note at the very beginning var which is the command equivalent to the selection from the Menu and which you can type into the command line to clicking on the menu.

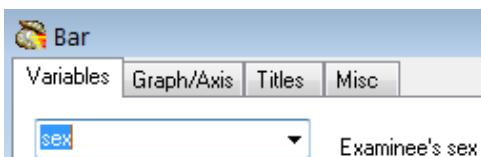
If you are only interested in one specific variable, for instant the variable reason, you can right-click on it in the variables window:



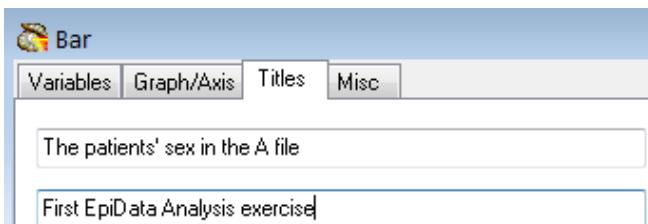
and you get to see what the label block looked like during data entry.

Making a graph

To make for instance a bar graph (or any other), open the graph dialogue by clicking on **Graph (ALT+G)** and pick for instance the field **sex**:

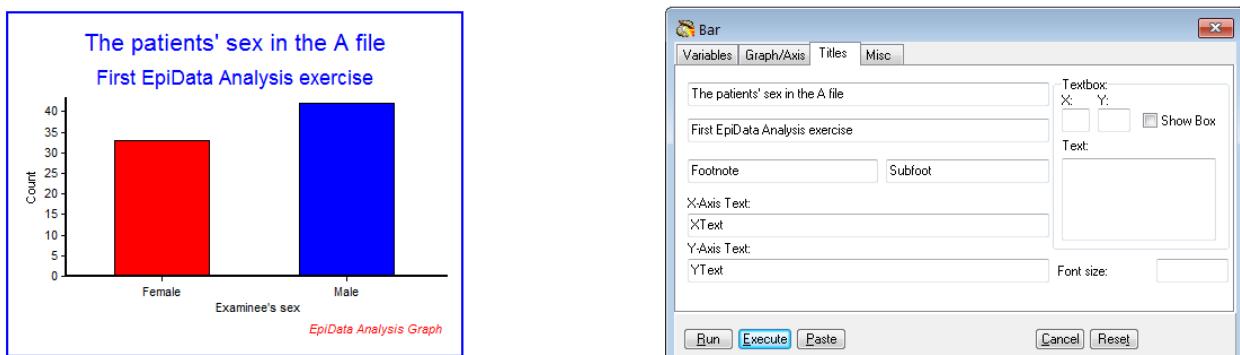


and modify the lines for Title and Sub-Title if you wish:



then click **Execute** get:

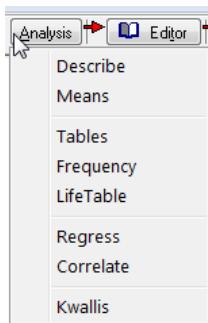
```
. BAR sex /Ti="The patients' sex in the A file" /Sub="First EpiData Analysis exercise" /SizeX=400 /SizeY=300
```



Note that you not only get the graph you asked for but above it in this instance also the syntax required to get it which you can copy and paste into your program.

Interactive analysis

Clicking on **Analysis (ALT+A)** give a drop-down menu:



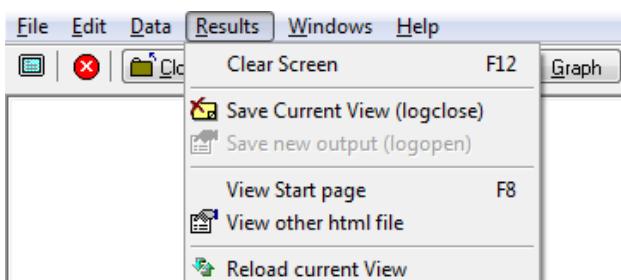
Let's use the first one, **Describe** to describe the content of a numeric field such as **age** and get:

. DESCRIBE age

Variable	N=75	Sum	Mean	(95% cfi)	Min	p5	p10	p25	Median	p75	p90	p95	Max
age	75	2997.00	39.96	35.35 44.57	14.00	16.60	19.00	24.00	35.00	50.00	72.20	81.00	99.00

We will come back later to the error of including here examinees with an age coded as missing (99) and how to fix that.

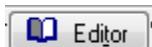
In the **Results** menu, you can clear the Viewer output (and memory) with **Clear screen** (short cut function key **F12**):



Writing a script (program) to execute a series of commands

This is all good and fine. It is what might be called “interactive analysis”. The advantage is that you can very quickly “fish” in your dataset and familiarize yourself with it. The disadvantage of interactive analysis is that everything is lost once you leave EpiData Analysis. It defies thus the underwritten purpose that everything you do in research must be thoroughly documented.

The solution of EpiData Analysis to documentation is to use:

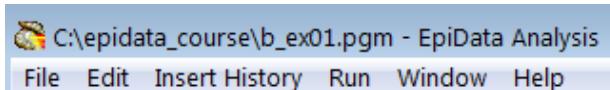


(Function key **F5** also opens the Editor)

It opens a new window which is simply a text editor for EpiData Analysis programs that take the extension ***.PGM**. There are many different ways to do it, but things become more quickly automated if done always in the same way. We propose that the first few command lines have the following content:

```
1 * Part B, Exercise 1
2 * Written by: Hans L Rieder
3 * First version: 23 Apr 2013
4 * Last revision: 21 Jul 2015
5
6 cls
7 close
8 logclose
```

Save the program immediately as “`b_ex01.pgm`” (the extension is supplied automatically) in our working folder and verify at the top that this is so:



It is always a good idea to make comments on what you do. Comments in EpiData Analysis are written by putting an asterisk * as the first thing on a line like in:

```
* The title of my program
```

If you want a command to be executed but would like to write a comment, you may do this with a double forward slash (“`//`”) after the command:

```
cls // This will clear the screen and free memory
```

It is also a good idea to write the date of the first version, lesser so of subsequent versions, because the original date will disappear from the file information once you update and save while one is often interested when it was started. It is also helpful to state who wrote the first draft, and add names if it is a collaborative work. We cannot overemphasize the importance of thorough annotations.

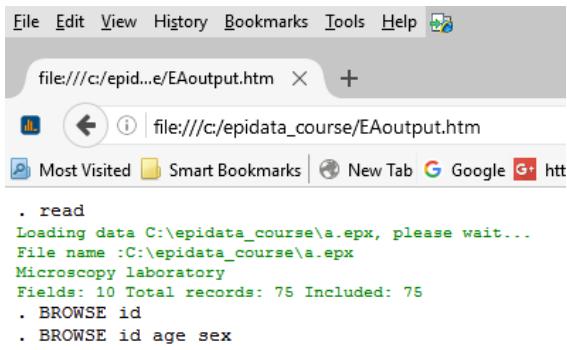
In lines 6 to 8 we start with a clean slate:

```
cls      // CLS clears the screen (identical to the interactive F12)
close    // CLOSE closes any open data (*.REC) file
logclose // LOGCLOSE closes any open log file
```

While `cls` and `close` are usually easily understood, the concept of “`logclose`” is perhaps intuitively not equally clear. When we opened EpiData Analysis and read the “`a.epx`”, EpiData Analysis gave the following information below the command line:



On the very right, the `EAoutput.htm` is the log file that EpiData Analysis opens and in which it records our doings and at which we can look (after quitting EpiData Analysis). We recognize some of the things we just did if we open it in our browser (the extension `*.htm` identifies it as a file to be examined in a browser):



A screenshot of the EpiData Analysis software interface. The menu bar includes File, Edit, View, History, Bookmarks, Tools, and Help. The toolbar includes icons for New, Open, Save, Print, and Exit. The main window shows a command line with the following text:

```
.read  
Loading data C:\epidata_course\A.epx, please wait...  
File name :C:\epidata_course\A.epx  
Microscopy laboratory  
Fields: 10 Total records: 75 Included: 75  
.BROWSE id  
.BROWSE id age sex
```

With our command “logclose”, this file is closed (and its name disappears from the line below the command line) and we are ready to write another log file (into which we perhaps wish to write specifically a table output or record anything else). We close it because like with data files only one log file can be open at a time. We could of course close it just before opening another one, but we propose to start with a clean slate where everything is closed, the screen is empty and the memory free.

We propose the above to be a standard approach to start any program: identifying, labeling and dating, followed by making a clean slate.

As we can work (and have open) only one data file at a time and we have closed any other that might be open with the above command, we can now open the file to analyze, with the command:

```
read "a.epx"
```

This does the same thing as we did above interactively when pressing the button Read: it opens an EpiData data file. The next three lines:

```
append /file="b.epx"  
append /file="c.epx"  
append /file="d.epx"
```

do something that is often required: we have several files with the same data structure and we wish to combine them vertically (“append”) to have a single data set for analysis. In this case, the four *.EPX files a, b, c, and d contain each 75 records from four different tuberculosis sputum smear microscopy laboratories (these are real data from Yangon, data courtesy Dr Ti Ti). Finally, we save the data in a new file, and thus have now:

```
read "a.epx"  
append /file="b.epx"  
append /file="c.epx"  
append /file="d.epx"  
savedata "abcd.rec" /replace
```

Important Note: The current version of EpiData Analysis (Build 2.2.2.183) can read *.EPX files, but it cannot yet save files in this format: any file you save will have the EpiData Entry 3.1 format with a *.REC file extension. Simultaneously, a paired *.CHK file containing the metadata will be saved.

Without the /replace, this would run once, but give an error the second time:

DataFile and/or Chk file exists.
Add /REPLACE or erase

```
c:\epidata_course\abcd.rec
c:\epidata_course\abcd.chk
Exception occurred: Operation aborted
Failed to save data to c:\epidata_course\abcd.rec
```

Creating a new *.REC file (as here) with “savedata” also creates an accompanying *.CHK file as we see from the error message.

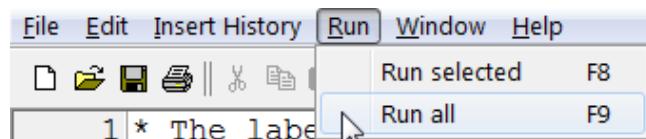
This requirement is a precaution to prevent the inadvertent overwriting of an existing file. The option /replace gives the explicit permission to overwrite the file in case it exists.

Finally, we close the file (a.epx) which is still open (the file that was read last is always the file that is open) and we need to close it first before we can open the newly created abcd.rec:

```
close
read "abcd.rec"
```

Now we are ready to analyze the data.

First run what we have. At the top of the Editor you see Run with two options:



F9 runs the entire program and **F8** whatever command line(s) you selected. To run a single command line, there is no need to mark it, just place the cursor anywhere onto it and press **F8**.

Analysis of continuous variables

We have one continuous variable in this data set, age in years (at last birthday). What you cannot know without a data documentation sheet is that examinees with unknown age were given a value 99 for age.

It is of course possible to make a frequency of age but this could give us up to 100 different values for a 2-digit field like age. Frequencies are better used for categorical variables while for continuous variables we use other methods to describe them. As is a common standard in any analysis package, we give a command followed by the name of the variable(s) on which the command should be executed. Type (on two lines):

```
describe age
means age
```

We get:

```
. describe age
Variable N=300      Sum  Mean  (95%    cfi)  Min   p5   p10   p25   Median   p75   p90   p95   Max
age        300 11656.0  38.85 36.82 40.88  5.00 17.00 19.00 26.25    35.00 49.00 65.00 74.95 99.00
```

```
. means age
```

Examinee's age in years

Obs.	Sum	Mean	Variance	Std Dev	(95% CI	mean)	Std Err
300	11656.0	38.85	319.26	17.87	36.82	40.88	1.03
Minimum	p5	p10	p25	Median	p75	p90	p95 Max
5.00	17.00	19.00	26.25	35.00	49.00	65.00	74.95 99.00

The first output is from “describe”, the second from “means”. Comparison shows that means is more informative than describe. We also note that the maximum is 99.00, meaning that at least one person had an unknown age coded as 99. Thus, our calculations are incorrect: we need to exclude the unknowns before calculating this kind of statistics:

```
select age<>99
```

The command to select a subset is select followed by the variable var01 in question, followed by an operator and the value. You may need these operators:

```
= Equal to
< Less than
> Greater than
>= Greater or equal to
<= Less or equal to
<> Unequal
```

There are more operators which you may look up in the Help file (**F1**).

Repeat means age after the appropriate selection and get:

Obs.	Sum	Mean	Variance	Std Dev	(95% CI	mean)	Std Err
297	11359.0	38.25	285.46	16.90	36.32	40.18	0.98
Minimum	p5	p10	p25	Median	p75	p90	p95 Max
5.00	17.00	19.00	26.00	35.00	48.00	65.00	74.00 86.00

Note at the bottom of the EpiData Analysis screen:

```
c:\epidata_course          297      (age<>99)
```

Apparently, our selection led to the exclusion of 3 examinees from the set of 300. Often we wish to compare a continuous variable in subgroups, for instance sex. The command is:

```
means age /by=sex
```

which gives us:

Examinee's age in years								
sex	Obs.	Sum	Mean	Variance	Std Dev	(95% CI	mean)	Std Err
Female	107	4065.00	37.99	301.98	17.38	34.66	41.32	1.68
Male	190	7294.0	38.39	277.68	16.66	36.00	40.77	1.21
sex	Minimum	p5	p10	p25	Median	p75	p90	p95 Max
Female	13.00	16.00	18.00	26.00	34.00	50.00	61.40	74.60 86.00
Male	5.00	17.00	19.10	27.00	35.00	45.25	65.00	73.45 86.00

Please note that the stratifying variable (sex, in this case) has to be a numeric variable for this command to work, another reason why we prefer numeric variables with value labels to text variables. The output shows that there is a small difference in the mean age between females and males. To know if this difference was just by chance or is statistically significant,

we may wish to do an unpaired t test. We then simply add the option /T (capitalization not required):

```
means age /by=sex /T
```

which gives us (only the lower part shown here):

Source	SS	df	MS	F	p Value
Between	10.89	1	10.89	0.04	0.846
Within	84486.17	295	286.39		
Total	84497.06	296	285.46		

Bartlett's test for homogeneity of variance
 $\text{Chi}^2 = 0.239$ df(1) p= 0.625

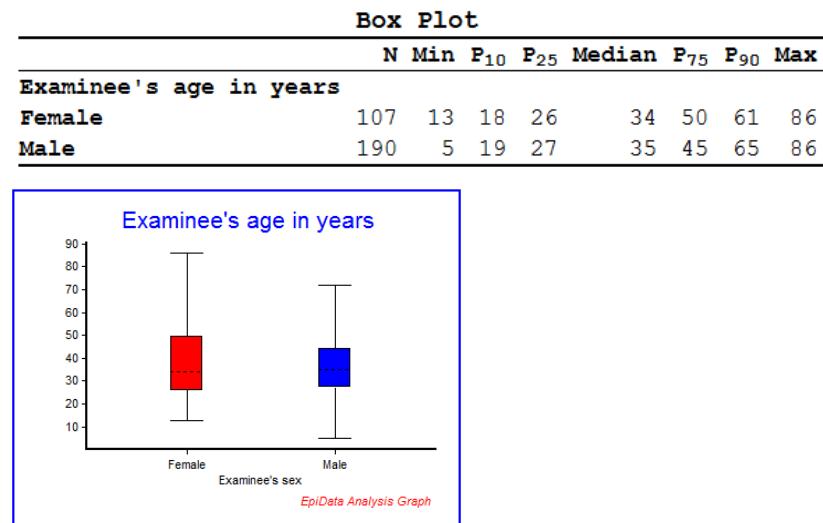
The p value of 0.846 (being more than 0.05, the level used conventionally to assess statistical significance) here thus tells that the difference between mean ages of males and females is not statistically significant. Please note that this also gives the results of Bartlett's test of homogeneity of variance at the bottom of the output, one of the assumptions to be satisfied before using the t test. If the p value of Bartlett's test is <0.05, then the assumption of homogeneity of variance is violated and it suggests to use one of the non-parametric tests for assessing statistical significance. (Read about a non-parametric test called Kruskal-Wallis test and the command `kwallis` in the help file)

Note:

- If there are paired observations for the same individuals, then the paired t test is recommended. To calculate this in EpiData, calculate the difference in paired values in a new variable (let us say “diff”) and run the command line `means diff /t`.
- If there are more than two means to be compared, then we use ANOVA (Analysis of Variance), though the EpiData command to achieve it will be the same `/t`.

In addition to the numeric output, one might like to see it graphically represented with a boxplot:

```
boxplot age /by=sex
```



The command to de-select is simply `SELECT`. You thus select (one or more selections), execute a program and then deselect to return to the full data set:

```
select age<>99
boxplot age /by=sex
```

```
select
```

Sometimes, we wish to categorize a continuous variable. For age, a common categorization is “WHO age groups”. This is how we would write the commands:

```
cls
select
define agegrp #
if age>=00 and age<15 then agegrp=1
if age>=15 and age<25 then agegrp=2
if age>=25 and age<35 then agegrp=3
if age>=35 and age<45 then agegrp=4
if age>=45 and age<55 then agegrp=5
if age>=55 and age<65 then agegrp=6
if age>=65 and age<99 then agegrp=7
if age =99 then agegrp=9
label agegrp "WHO standard age groups"
labelvalue agegrp /1="0- to 14-yr-old"
labelvalue agegrp /2="15- to 24-yr-old"
labelvalue agegrp /3="25- to 34-yr-old"
labelvalue agegrp /4="35- to 44-yr-old"
labelvalue agegrp /5="45- to 54-yr-old"
labelvalue agegrp /6="55- to 64-yr-old"
labelvalue agegrp /7="65-yr-old and older"
labelvalue agegrp /9="Unknown age"
```

After ensuring that we have the full data set (“SELECT”), we define a new integer variable `agegrp` and categorize the patients’ age into the seven WHO age groups plus one for those with no age recorded. After categorization, we give a Field label and then Value labels. Looking at how to use this new variable leads us over to the analysis of categorical variables.

Because “CLS” does more than just clearing the screen – it also frees memory – we tend to write frequently a `cls` into the program. You can also manually, while a program runs, press **F12**, and it will do the same thing, speeding up the running of the program.

We could have defined a text variable:

```
define agegrptxt1 _____
define agegrptxt2 <AAAAAAA>
```

Analysis of categorical variables

The most frequent command we use for a categorical variable is, well, frequencies:

```
cls
freq agegrp
```

WHO standard age groups	
	N
0- to 14-yr-old	6
15- to 24-yr-old	57
25- to 34-yr-old	84
35- to 44-yr-old	67
45- to 54-yr-old	28
55- to 64-yr-old	25
65-yr-old and older	30
Unknown age	3
Total	300

The output displays thus the Value labels we wrote, not the values, but we can have both with the option /vl:

```
freq agegrp /vl
```

WHO standard age groups		N
1	0- to 14-yr-old	6
2	15- to 24-yr-old	57
3	25- to 34-yr-old	84
4	35- to 44-yr-old	67
5	45- to 54-yr-old	28
6	55- to 64-yr-old	25
7	65-yr-old and older	30
9	Unknown age	3
	Total	300

This is important to know if we want to make a selection like for instance including those in the category “Unknown age” by writing “select agegrp<>9”. With frequencies we often want to know the percentage distribution:

```
freq agegrp /c
```

WHO standard age groups		
	N	%
0- to 14-yr-old	6	2.0
15- to 24-yr-old	57	19.0
25- to 34-yr-old	84	28.0
35- to 44-yr-old	67	22.3
45- to 54-yr-old	28	9.3
55- to 64-yr-old	25	8.3
65-yr-old and older	30	10.0
Unknown age	3	1.0
Total	300	100.0

The option “/c” is thus for column percent, while an option “/r” would be for row percent (not applicable here but in tables). With frequencies, one can also get 95% confidence intervals (not very sensible here, just shown for the principle):

```
freq agegrp /c /ci
```

WHO standard age groups			
	N	%	(95% CI)
0- to 14-yr-old	6	2.0	(0.9-4.3)
15- to 24-yr-old	57	19.0	(15.0-23.8)
25- to 34-yr-old	84	28.0	(23.2-33.3)
35- to 44-yr-old	67	22.3	(18.0-27.4)
45- to 54-yr-old	28	9.3	(6.5-13.2)
55- to 64-yr-old	25	8.3	(5.7-12.0)
65-yr-old and older	30	10.0	(7.1-13.9)
Unknown age	3	1.0	(0.3-2.9)
Total	300	100.0	

If you use more than one variable:

```
freq sex reason
```

you get frequencies for each of the (as many as you like) variables:

Examinee's

sex	N
Female	109
Male	191
Total	300

Examination reason

	N
Diagnosis	134
Follow-up at 2 months	40
Follow-up at 3 months	12
Follow-up at 4 months	3
Follow-up at 5 months	35
Follow-up at 6 months	31
Follow-up at 7 months or later	16
Follow-up, month not stated	27
Reason not stated	2
Total	300

Let's make one more new variable for a case definition. The dataset is, as mentioned above, from laboratory registers. Each examinee can have up to three sequential examinations. Let's define that an examinee is a "case" if any of the three possible examinations has at least 1 acid-fast bacillus (a proxy for tubercle bacilli) in it. If we make a frequency of the first result and also ask for the display of the values (not just the value labels):

```
freq res1 /vl
```

Result of specimen 1

	N
0.4 Scanty, 4 AFB per 100 fields	1
0.0 Negative	272
1.0 1+ positive	18
2.0 2+ positive	5
3.0 3+ positive	4
Total	300

We get something, but not quite what we need. What we need is to know what values were possible, not just which values were actually present. We can obtain that information by typing var (as explained above) into the command line and we get the coding for the entire dataset. Alternatively, for just one categorical variable, we can right-click the variable in the variable (**F3**, if not open) window (see above).

We can repeat this for each of the three result variables and we see that all use the same label block. Based on this information, we can now safely define a case:

```
define case #
      let case=0
      if res1>0 and res1<9 then case=1
      if res2>0 and res2<9 then case=1
      if res3>0 and res3<9 then case=1
      label case "Smear microscopy defined case"
      labelvalue case /0="Non-case"
      labelvalue case /1="Case"
```

and try:

```
tables case sex /r /c /tp /pct
```

Smear microscopy defined case								
Examinee's sex	Non-case	%	%	% Case	%	%	% Total	%
Female	94 (86.2)	{35.9}	[31.3]	15 (13.8)	{39.5}	[5.0]	109 (100.0)	{36.3} [36.3]
Male	168 (88.0)	{64.1}	[56.0]	23 (12.0)	{60.5}	[7.7]	191 (100.0)	{63.7} [63.7]
Total	262 (87.3)	{100.0}	[87.3]	38 (12.7)	{100.0}	[12.7]	300	

Percents: (Row) (Col) [Total]

This gives us row (/r), column (/c), and total (/tp) percentages. The /pct aligns them properly in separate columns.

Note that in the command, whichever variable you write first after the command tables is the column variable (outcome variable in epidemiologic terms) and the second variable is the row variable (exposure variable).

Meanwhile you have noted that EpiData Analysis gives just the raw numbers as default output in a frequency or table, but you can add multiple options, and combine them as needed. Options are preceded by the forward slash (/).

```
tables case sex /r /t
```

Smear microscopy defined case			
Examinee's sex	Non-case	% Case	% Total
Female	94 (86.2)	15 (13.8)	109 (100.0)
Male	168 (88.0)	23 (12.0)	191 (100.0)
Total	262 (87.3)	38 (12.7)	300

Percents: (Row)

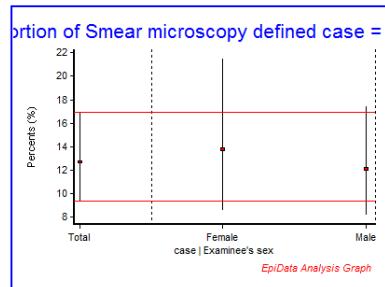
Chi²= 0.185 df(1) p= 0.6667

shows us with the option “/t” (for performing the Chi-square test) that there is no significant difference between males and females in being a case. In some instances, when the expected cell value of a 2-by-2 table is less than 5, it is recommended to use Fischer’s exact test instead of the Chi-square test. For Fischer’s exact test, use the option “/ex”. This is indicated by EpiData automatically. We could make 95% confidence intervals with frequencies, but we cannot use the “/ci” option for tables. We need a little trick to get that, and that is with the confidence interval plot:

```
ciplot case sex
```

Crude: Proportion of Smear microscopy defined case = Case among all.					
variable	stratum	Total N	n _{Smear microscopy defined case}	%	(95% CI)
Smear microscopy defined case	Total	300	38	12.7	(9.4-16.9)
Examinee's sex	Female	109	15	13.8	(8.5-21.5)
	Male	191	23	12.0	(8.2-17.4)

Crude: Proportion of Smear microscopy defined case = Case among all.



And if we don’t want to see the graph, only the table we use the option “/ng” (“no graph”)

```
ciplot case sex /ng
```

By default, the higher value of the variable is considered as outcome. In case you want to change that, use the option “/O=value” (O for outcome).

Tasks:

- o Determine the year of birth (new variable created from age and date of registration), then make groups of examinees (another variable) born respectively before 1930, from 1930 to including 1949, 1950 and later, and those without known year of birth.*
- o Use two approaches, one with text field coding and the other with numeric coding and value labels.*

Solution to Exercise 1: An introduction to EpiData Analysis

Key Point(s):

- It is important to make a comment first in the program. This preferably is what you want to do in that program. Comments are preceded by an asterisk and are bypassed by the analysis program.
- The F9 key runs the whole program whilst the F8 key runs only the selected part of the program.

Tasks:

- o Determine the year of birth (new variable created from age and date of registration), then make groups of examinees (another variable) born respectively before 1930, from 1930 to including 1949, 1950 and later, and those without known year of birth.
- o Use two approaches, one with text field coding and the other with numeric coding and value labels.

Solution:

The output solution is as follows:

```
. tables sex birthgrp1
Examinee's sex
Exercise birth years Female Male Total
text coding
1929 and before      6    9   15
1930-1949            14   25   39
1950 and later        83  152  235
Unknown               6    5   11
Total                 109 191  300
. tables sex birthgrp2
Examinee's sex
Exercise birth years Female Male Total
numeric coding
Born before 1930      6    9   15
Born 1930 to 1949     14   25   39
Born after 1950       83  152  235
Unknown birth year     6    5   11
Total                 109 191  300
```

The “Task” part of program B_EX01.PGM might look as follows:

```
*****
* Part B, Exercise 1
* Task
*****
cls
close
logclose
```

```

read "a.epx"
append /file="b.epx"
append /file="c.epx"
append /file="d.epx"
savedata "abcd.rec" /replace

close
read "abcd.rec"

* Define the year of birth
define birthyr #####
birthyr=year(regdate)-age
if age=99 or year(regdate)=1900 then birthyr=9999

* Using text variables
* define groupings for birth years
define birthgrp1 _____
                                let birthgrp1="other"
if birthyr <1930 then birthgrp1="1929 and before"
if birthyr>1929 and birthyr<1950 then birthgrp1="1930-1949"
if birthyr>1949           then birthgrp1="1950 and later"
if birthyr=9999            then birthgrp1="Unknown"
label birthgrp1 "Exercise birth years text coding"

* Using numeric variables
* define groupings for birth years
define birthgrp2 #
                                let birthgrp2=8
if birthyr <1930 then birthgrp2=1
if birthyr>1929 and birthyr<1950 then birthgrp2=2
if birthyr>1949           then birthgrp2=3
if birthyr=9999            then birthgrp2=9
label birthgrp2 "Exercise birth years numeric coding"
labelvalue birthgrp2 /1="Born before 1930"
labelvalue birthgrp2 /2="Born 1930 to 1949"
labelvalue birthgrp2 /3="Born after 1950"
labelvalue birthgrp2 /8="Unaccounted for"
labelvalue birthgrp2 /9="Unknown birth year"

cls
tables sex birthgrp1
tables sex birthgrp2

```

Exercise 2: More on EpiData Analysis

At the end of this exercise you should be able to:

- Know more about data set and variable manipulation
- Know more about tables
- Know more about graphs

More about data set manipulation

Merge and append

In the first exercise of Part B you learned to append files. It was said then that you append files if the files have the same structure. That is not exactly necessary. You can append files with different structures as long as you ensure 1) that the file that is read contains all the same variables as the file(s) you plan to append to it, and 2) that the variables have the same definitions. Let's show an example of two data sets, Set A and Set B, each with three records, where **id** is the unique identifier in each set:

Set A

	id	sex	age
1	a	m	23
2	b	f	30
3	c	m	50

Set B

	id	bs	ht
1	d	5.6	186
2	e	6.2	170
3	f	4.8	168

Note, that the identifiers in Set A are all different from the identifiers in Set B, the information in the two sets belongs thus to a total of six different individuals. We can append the two files if we first create the same variables in the file we read (let's say Set A) before we append Set B:

```
cls
close
read "b_ex02_01.rec"
define bs #.#
define ht ###
append /file="b_ex02_02.rec"
```

We get correctly a file with six records:

	id	sex	age	bs	ht
1	a	m	23	.	.
2	b	f	30	.	.
3	c	m	50	.	.
4	d	.	.	5.6	186
5	e	.	.	6.2	170
6	f	.	.	4.8	168

We could have gone the way with `merge`. `Merge` requires the identifier to be unique within a set which is the case for `id` within Set A and within Set B. If we use `merge`, there is no need to first ensure that all variables exist in the set that is read:

```
cls
close
read "b_ex02_01.rec"
merge id /file="b_ex02_02.rec"
```

We get:

	<code>id</code>	<code>sex</code>	<code>age</code>	<code>bs</code>	<code>ht</code>	<code>MergeVar</code>
1	a	m	23	.	.	Only in memory (Original)
2	b	f	30	.	.	Only in memory (Original)
3	c	m	50	.	.	Only in memory (Original)
4	d	.	.	5.6	186	Only in external file
5	e	.	.	6.2	170	Only in external file
6	f	.	.	4.8	168	Only in external file

In other words, we get exactly the same thing, except that EpiData Analysis added a variable `MergeVar` which can take 3 values:

Only in memory (Original)
Only in external file
In both [not existing here]

If the two data sets thus contain different individuals, we can choose either `append` or `merge` as long as we ensure that (if we choose `append`) the read file has all the variables that the appended file has. If one or more individuals are identical in Set A and set B, then using `append` becomes wrong because we would get two records from the same individual. In such a case, we must use `merge`. To exemplify this, we saved the above data Set B as `b_ex02_03.rec`, after changing the value for `id=d` to `id=c`. In other words, person “c” is in both data sets. If we juxtapose the results from appending and merging:

Appending						Merging							
	<code>id</code>	<code>sex</code>	<code>age</code>	<code>bs</code>	<code>ht</code>		<code>id</code>	<code>sex</code>	<code>age</code>	<code>bs</code>	<code>ht</code>	<code>MergeVar</code>	
1	a	m	23	.	.		1	a	m	23	.	Only in memory (Original)	
2	b	f	30	.	.		2	b	f	30	.	Only in memory (Original)	
3	c	m	50	.	.		3	c	m	50	5.6	186	In both
4	c	.	.	5.6	186		4	e	.	.	6.2	170	Only in external file
5	e	.	.	6.2	170		5	f	.	.	4.8	168	Only in external file
6	f	.	.	4.8	168								

The file resulting from appending has 6 records, two of which belong to individual “c”, the result is thus wrong. Conversely, the file resulting from merging has only 5 records corresponding to the total of 5 individuals and is thus correct. EpiData Analysis gives us with the automatically created field `MergeVar` also the relevant information about the source of the data.

In a later exercise you will learn about important options (look-up table) that you may need when merging data files.

Select

We used `select` as follows:

```
read "abcd.rec"
select age<>99
means age /by=sex
select
freq sex
```

We selected a subset, then carried out the command, and finally used `select` again to obtain back the full data set. Thus, before we wrote `select` again, we kept the reduced data set. Sometimes it is desirable to make a selection only for a specific command as in the above example. A powerful possibility in EpiData Analysis is to do precisely that by writing the condition for which a command is to be executed onto the same line as the command. Instead of the above, we thus write:

```
read "abcd.rec"
means age /by=sex if age<>99
freq sex
```

Drop and Keep

`Select` reduced the number of records in a data set. Sometimes it is useful to reduce the number of variables, not the number of records in a data set. This is where `drop` and `keep` come in. The two are complementary. We have identifier field `id` and `serno` in the data set `abcd.rec`. If we don't want to keep them, we would write:

```
cls
close
read "abcd.rec"
drop id serno
savedata "abcd_temp_01.rec" /replace
```

This is more efficient than the complementary approach with `keep`:

```
cls
close
read "abcd.rec"
keep labcode regdate age sex reason \
      res1 res2 res3
savedata "abcd_temp_02.rec" /replace
```

Note here also in passing that you can use the backslash \ to continue a command on the following line (this was not possible in the CHK file).

Sort

Sorting a dataset on one or more variables is a key component in programming. For instance, if you would like to sort our data set `abcd.rec` on the laboratory, you would write:

```
cls
close
read "abcd.rec"
sort labcode
```

and within each laboratory by registration date:

```
sort labcode regdate
```

8	A	24/10/2003
9	A	27/10/2003
...		
87	B	08/09/2003
88	B	09/09/2003

The last sorting command overrides any previous sorting command. Note that saving a data set after sorting will save it in the last sort order.

Define and gen

We used so far `define` to create a variable in the memory exactly as we did in the `CHK` file for temporary variables. For instance, if we create a new field `case` to define a sputum-smear positive case:

```
define case #
case=0
if res1>0 and res1<9 then case=1
if res2>0 and res2<9 then case=1
if res3>0 and res3<9 then case=1
label case "Microscopy case definition"
labelvalue case /0="Non-case"
labelvalue case /1="Case"
```

There is an alternative in EpiData Analysis, `gen` for generating a new variable. Be careful though to use “`gen`” and not “`generate`” because the latter is also a legitimate command, but with an entirely different meaning (it creates empty records). We would thus write instead of the above:

```
cls
gen i case2=0
if res1>0 and res1<9 then case2=1
if res2>0 and res2<9 then case2=1
if res3>0 and res3<9 then case2=1
label case2 "Microscopy case definition"
labelvalue case2 /0="Non-case"
labelvalue case2 /1="Case"
```

We don't have to assign it a value (as we do here with “`=0`”). If we don't, the value is set to missing with a period. With `define` we defined at the same time with the field name also both its type and length. An integer variable created with `gen` will automatically get a field length of 9.

There are other field types we can create this way:

```
gen d copydate=regdate
gen f agedays=age*365.25
gen s(2) labsex=labcode+sex
```

If you create a **float field**, it will have the length 12, including 4 decimal points. This could be inconvenient, if we create a label block for instance (like we had for the results) as we must pay attention that there is full compatibility. There is no option to change this default.

Wonderfully, we can change the default length (of 20) for **string fields**, in the non-sensical example here defined as having a length of 2.

Note that “gen” is a bit faster to code, “define” runs faster. This becomes important in large datasets. For date and string fields “gen” is virtually always preferable, for integer fields “define” is perhaps preferable if you know the expected length of the field, and for float fields, “define” is probably almost always preferable.

Recode

We had used DEFINE and / or GEN go make new variables. If we have a continuous variable from which we wish to make a categorical one, like converting the variable AGE to age groups, EpiData offers RECODE, as shown in this example:

```
define agegrp2 #
recode age to agegrp2 00-14=1 15-24=2 25-34=3 35-44=4 45-54=5 \
      55-64=6 65-98=7 99=9
freq agegrp2
```

gives:

```
agegrp2
  N
00-14  6
15-24  57
25-34  84
35-44  67
45-54  28
55-64  25
65-98  30
.99    3
Total  300
```

Thus, the groupings become the labels. Of course, one can override these default value labels.

String fields and substrings

Let’s assume we wish to make a string field for each microscopy result, which can take on the three values “N” (for negative), “P” (for any positive), and “9” (for not available). We thus write:

```
cls
close
read "abcd.rec"

      gen s(1) result1="P"
if res1=0 then result1="N"
if res1=9 then result1="9"
cls
      gen s(1) result2="P"
if res2=0 then result2="N"
if res2=9 then result2="9"
cls
      gen s(1) result3="P"
if res3=0 then result3="N"
if res3=9 then result3="9"
```

We then wish to combine the three results into one single string to obtain the pattern:

```
cls  
gen s(3) pattern=result1+result2+result3  
label pattern "Pattern of 3 serial smears"  
freq pattern
```

and get:

pattern of 3 serial smears	
	N
NN9	157
NNN	105
NP9	1
NPP	9
PP9	11
PPN	2
PPP	15
Total	300

Finally, we can extract a subset of a string field:

```
cls  
gen s(1) firstres=substr(pattern,1,1)  
label firstres "Result of 1st smear"  
freq firstres /c /ci
```

and get:

Result of 1st smear		
	N	% (95% CI)
N	272	90.7 (86.8-93.5)
P	28	9.3 (6.5-13.2)
Total	300	100.0

Admittedly, not the most efficient approach to something that could have been obtained directly from the original field `res1`, but the point here is to show the way how to extract a substring from a text field:

SUBSTR (fieldname, start position, number of characters including start position)

Thus:

```
cls  
gen s(2) res12=substr(pattern,1,2)  
gen s(2) res23=substr(pattern,2,2)  
freq res12 res23
```

gives:

res12	res23
	N
NN	262
NP	10
PP	28
Total	300
NN9	157
NNN	105
P9	12
PN	2
PP	24
Total	300

Date fields

We know that date fields are a hassle because they don't fit our decimal system and people use different ways to write dates. To further complicate matters, EpiData Analysis deals

somewhat differently with date definitions than we learned in the CHK file. There are different ways of doing it, but it might be best to learn one and learn to master it. Let's assume, we wish to calculate the number of years elapsed since the registration of the examinee in our data set and 1 January 2013. The syntax to accomplish this is:

```
cls
close
read "abcd.rec"
gen f intyrs=(dmy(01,01,2013)-regdate)/365.25
```

We thus tell EpiData Analysis 1) that it is a date and 2) the format of our date with dmy followed by the values for the three date components in parenthesis, separated by commas.

Results variables

If we execute certain commands, EpiData Analysis will produce variables in memory that keep temporarily certain values which we can visualize with the command `result`. For example:

```
cls
close
read "abcd.rec"

means age if (age<>99) and (sex=1)
result
```

produces:

System Variables Value
SYSTEMDATE 27/04/2013
SYSTEMTIME 15:55:43
SYSDIR C:\EpiData\
CURRENTDIR C:\epidata_course
LANG EN

Temporary Variables Value
\$VARIABLE1 age
\$LEVEL1
\$OBS1 107
\$SUM1 4065
\$MEAN1 37.9906542056075
\$VAR1 301.933874096279
\$SD1 17.3762445337386
\$CFILL1 34.6602378330551
\$CFIH1 41.3210705781599
\$STDERR1 1.67982496333453
\$MIN1 13
\$P051 16
\$P101 18
\$P251 26
\$P501 34
\$P751 50
\$P901 61.4
\$P951 74.6
\$MAX1 86

Our interest here is in the **Temporary Variables Value**. We can use them and save the values of any of them into another variable. For instance:

```
means age if (age<>99) and (sex=1)
result
gen meanfem=$mean1

means age if (age<>99) and (sex=2)
result
gen meanmal=$mean1
```

We don't even need to write the line "result" if we know how EpiData Analysis defines the variable we require.

Visualizing with BROWSE:

meanfem	meanmal
37.9907	38.3895
37.9907	38.3895
37.9907	38.3895

It is not of particular value here (every record has the same values), but this is a potentially very powerful tool to further process data.

Writing output into a file

You learned perhaps a bit mechanically to start every program with a command "logclose" to close any log file that might be open, but you haven't quite tested the opposite, the opening of a log file. You can write any output into three types of files, text, HTML, or Excel. We strongly discourage Excel, because for proprietary reasons EpiData is forced to use an outdated format. If you need your output in a spreadsheet program, it is way preferable to save the output into a text file and then open the text file in your spreadsheet application software.

The commands are straight forward:

```
logopen "my_output.txt" /replace
* Some EpiData Analysis commands
logclose
```

Specifically, if we write:

```
cls
close
read "abcd.rec"

logopen "output_01.txt" /replace
tables sex reason
logclose
```

If we look at the output_01.txt file in our text editor, we see:

```
Logopen output_01.txt
  EpiData Analysis V2.2.2.180 27-Apr-13 17:45
. tables sex reason
Examination reason      Female      Male   Total
Diagnosis      52       82      134
Follow-up at 2 months    18       22      40
Follow-up at 3 months    4        8      12
Follow-up at 4 months    0        3      3
Follow-up at 5 months    13       22      35
Follow-up at 6 months    11       20      31
Follow-up at 7 months or later  3        13      16
Follow-up, month not stated  8        19      27
Reason not stated 0       2        2
Total 109     191     300
```

```
. logclose
```

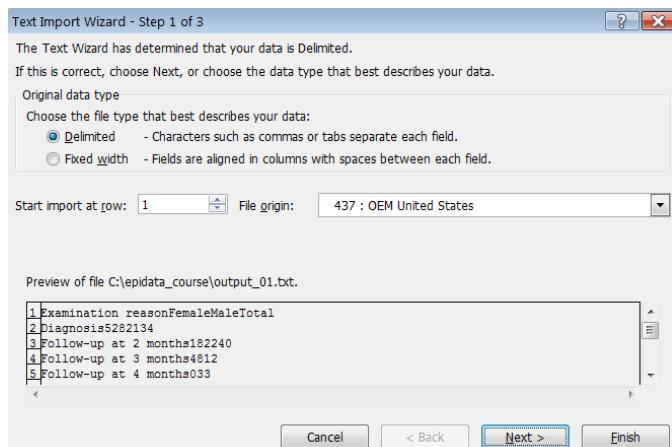
Apart from the “ragged” appearance, there is also too much superfluous output there, if we only want the table. We can greatly improve it by setting the echo first off before opening the log file and then setting it on again after it has been closed:

```
set echo=off
logopen "output_01.txt" /replace
tables sex reason
logclose
set echo=on
```

This way, we get it trimmed down:

Examination reason	Female	Male	Total	
Diagnosis	52	82	134	
Follow-up at 2 months	18	22	40	
Follow-up at 3 months	4	8	12	
Follow-up at 4 months	0	3	3	
Follow-up at 5 months	13	22	35	
Follow-up at 6 months	11	20	31	
Follow-up at 7 months or later		3	13	16
Follow-up, month not stated	8	19	27	
Reason not stated	0	2	2	
Total	109	191	300	

If we go now to our spreadsheet program and open it as a delimited text file:



it will come out nicely as intended:

	A	B	C	D
1	Examination reason	Female	Male	Total
2	Diagnosis	52	82	134
3	Follow-up at 2 months	18	22	40
4	Follow-up at 3 months	4	8	12
5	Follow-up at 4 months	0	3	3
6	Follow-up at 5 months	13	22	35
7	Follow-up at 6 months	11	20	31
8	Follow-up at 7 months or later	3	13	16
9	Follow-up, month not stated	8	19	27
10	Reason not stated	0	2	2
11	Total	109	191	300

More about tables

Let's first copy our case definition from above:

```

cls
close
read "abcd.rec"

define case #
case=0
if res1>0 and res1<9 then case=1
if res2>0 and res2<9 then case=1
if res3>0 and res3<9 then case=1
label case "Microscopy case definition"
labelvalue case /0="Non-case"
labelvalue case /1="Case"

```

and then make another categorical variable from the categorical variable `reason` to get only three levels and verify that we got what we intended to get, making a cross-table between the old variable (`reason`) and the new one (`reason2`) derived from it:

```

cls
gen i reason2=2
if reason=0 then reason2=1
if reason=9 then reason2=9
label reason2 "Reason for examination"
labelvalue reason2 /1="Diagnosis"
labelvalue reason2 /2="Follow-up"
labelvalue reason2 /9="Reason unknown"
tables reason2 reason

```

Reason for examination						
	Examination	reason	Diagnosis	Follow-up	Reason unknown	Total
Diagnosis		134	0	0	134	
Follow-up at 2 months		0	40	0	40	
Follow-up at 3 months		0	12	0	12	
Follow-up at 4 months		0	3	0	3	
Follow-up at 5 months		0	35	0	35	
Follow-up at 6 months		0	31	0	31	
Follow-up at 7 months or later		0	16	0	16	
Follow-up, month not stated		0	27	0	27	
Reason not stated		0	0	2	2	
Total		134	164	2	300	

Previously when we made a table also showing the values with:

```
tables case sex /vl
```

we got:

Microscopy case definition					
Examinee's sex	0	Non-case	1	Case	Total
1 Female	94	15	109		
2 Male	168	23	191		
Total	262	38	300		

If we look at the sorting sequence, we see that the sequence is ascending by value (not ascending by label alphabet, see labels for case definition). We can invert the sequence with:

```
tables case sex /vl /sd
```

Microscopy case definition					
Examinee's sex	1	Case	0	Non-case	Total
2 Male	23	168	191		
1 Female	15	94	109		
Total	38	262	300		

There are multiple sorting options but one should take note that some more complex sorting options do not deliver what we expect, thus always check carefully. For simple sorting like ascending on values (/sa) or descending on values (/sd) as we just did, we usually get what we want.

Important to note is what happens if we like to get an odds ratio, compare the two outputs:

```
cls
tables case sex
tables case sex /o
```

“Plain”

Microscopy case definition			
Examinee's sex	Non-case	Case	Total
Female	94	15	109
Male	168	23	191
Total	262	38	300

“Epidemiologic”

Outcome:Microscopy case definition			
Examinee's sex	Case	Non-case	Total
Male	23	168	191
Female	15	94	109
Total	38	262	300

Exposure: Examinee's sex = Male
 Outcome: Microscopy case definition = Case
 Odds Ratio = 0.86 (95% CI: 0.43-1.72) (Robins, Greenland, Breslow CI)

This is a reflection of the fact that EpiData Analysis is truly an epidemiologist's tool: most commonly in outbreak investigations or case-control studies we show the cases to the left and the non-cases to the right of the column, and show the exposure on top and the non-exposure at the bottom of the row.

If we now want to switch these, then we have to check carefully whether we get what we want, but one or the other options will give us what we need. Let's try then with our logic from above telling us that we should probably sort in ascending order:

```
tables case sex /o /sa
```

Outcome:Microscopy case definition			
Examinee's sex	Non-case	Case	Total
Female	94	15	109
Male	168	23	191
Total	262	38	300

Exposure: Examinee's sex = Female
 Outcome: Microscopy case definition = Non-case
 Odds Ratio = 0.86 (95% CI: 0.43-1.72) (Robins, Greenland, Breslow CI)

Indeed, it did invert it. If we wish to invert only one of the two, say keep the “epidemiologic” presentation for cases and non-cases, but change it for sex, then we might need to do some recoding:

```
cls
gen i sexinvert=1
if sex=1 then sexinvert=2
label sexinvert "Sex of person"
labelvalue sexinvert /1="Male"
labelvalue sexinvert /2="Female"
tables case sexinvert /o
```

and we get:

Outcome:Microscopy case definition			
Sex of person	Case	Non-case	Total
Female	15	94	109
Male	23	168	191
Total	38	262	300

Exposure: Sex of person = Female
 Outcome: Microscopy case definition = Case
 Odds Ratio = 1.17 (95% CI: 0.58-2.34) (Robins, Greenland, Breslow CI)

Again, be sure to always check that what you get is what you need and learn creatively to approach things as you need them to be.

Stratification

So far, we have dealt with two-by-two tables, but one of the common requirements is a stratified analysis. EpiData Analysis makes it easy for us in that we just list the variables in a `tables` command, though we must pay attention to the sequence of the variables. If we need to look at cases versus non-cases by sex, stratified by the (summary) reason, we write:

```
tables case sex reason2 if reason2<>9
```

and get:

Microscopy case definition			
Examinee's sex	Non-case	Case	Total
Female	94	15	109
Male	167	22	189
Total	261	37	298

Reason for examination: Diagnosis			
Microscopy case definition			
Examinee's sex	Non-case	Case	Total
Female	40	12	52
Male	64	18	82
Total	104	30	134

Reason for examination: Follow-up			
Microscopy case definition			
Examinee's sex	Non-case	Case	Total
Female	54	3	57
Male	103	4	107
Total	157	7	164

The first table is the crude, unstratified table, the following table(s) are the results of case by sex in the different strata. Of course, the interest here is commonly a summary odds ratio calculated by the Mantel-Haenszel procedure:

```
tables case sex reason2 /o if reason2<>9
```

We get more output here: first the “epidemiologic” table with crude odds ratios and 95% confidence intervals followed by these measures of association in the strata:

Outcome: Microscopy case definition			
Examinee's sex	Case	Non-case	Total
Male	22	167	189
Female	15	94	109
Total	37	261	298

Unstratified table
Exposure: Examinee's sex = Male
Outcome: Microscopy case definition = Case

Odds Ratio = 0.83 (95% CI: 0.41-1.67) (Robins,Greenland,Breslow CI)

Outcome: Reason for examination: Diagnosis			
Microscopy case definition			
Examinee's sex	Case	Non-case	Total
Male	18	64	82
Female	12	40	52
Total	30	104	134

Exposure: Examinee's sex = Male
Outcome: Microscopy case definition = Case

Odds Ratio = 0.94 (95% CI: 0.41-2.15) (Robins,Greenland,Breslow CI)

Outcome: Reason for examination: Follow-up			
Microscopy case definition			
Examinee's sex	Case	Non-case	Total
Male	4	103	107
Female	3	54	57
Total	7	157	164

Exposure: Examinee's sex = Male
Outcome: Microscopy case definition = Case

Odds Ratio = 0.70 (95% CI: 0.15-3.24) (Robins,Greenland,Breslow CI)

After these tables, we get the summary of the adjusted analysis:

Microscopy case definition by Examinee's sex adjusted for Reason for examination				
	N = 298	N	OR	(95% CI)
Crude		298	0.83	(0.41-1.67)
Adjusted		298	0.88	(0.42-1.82)
Reason for examination: Diagnosis	134	0.94	(0.41-2.15)	
Reason for examination: Follow-up	164	0.70	(0.15-3.24)	
Summary Estimates				
Total 2 strata. 2 informative & 0 non-informative.				
Exposure: Examinee's sex = Male				
Outcome: Microscopy case definition = Case				

Note also the all-important information at the bottom to help us ensuring that we got what we wanted:

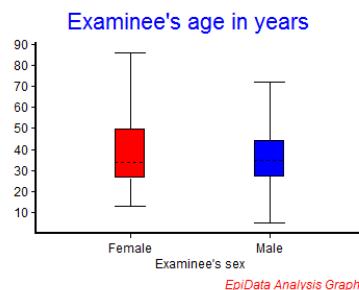
Exposure: Examinee's sex = Male
Outcome: Microscopy case definition = Case

More about graphs

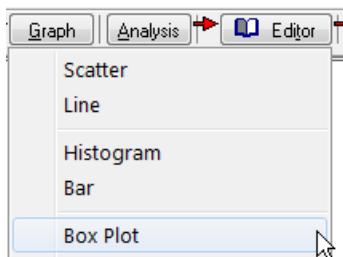
In the previous exercise, we did:

```
select age<>99  
boxplot age /by=sex
```

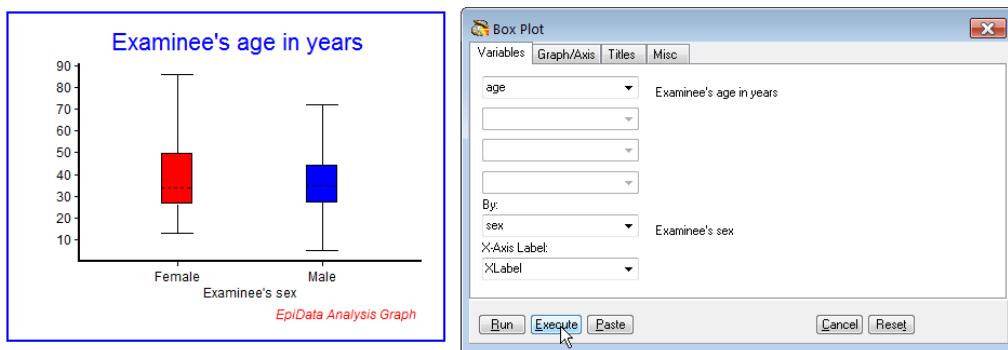
and got this graph:



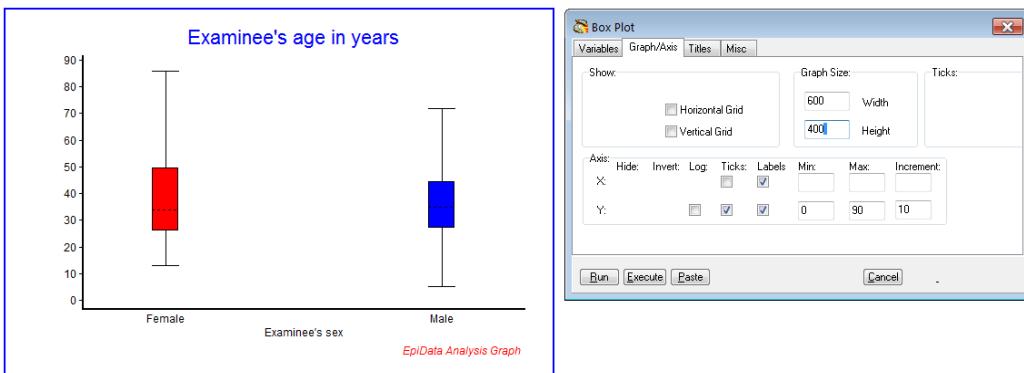
To explore the options a bit more without first taking recourse to the Help file, we will approach it by using the menu interactively:



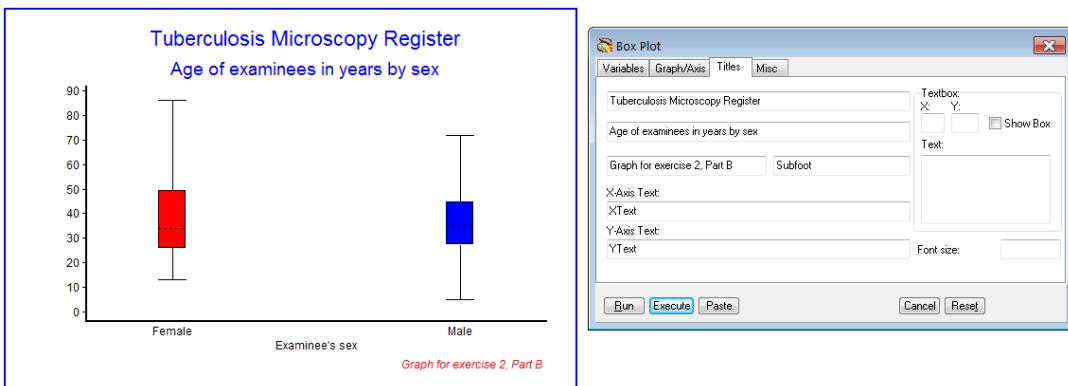
We will use Execute (not Run) to progressively edit the graph to our liking, completing the first tab “Variables”:



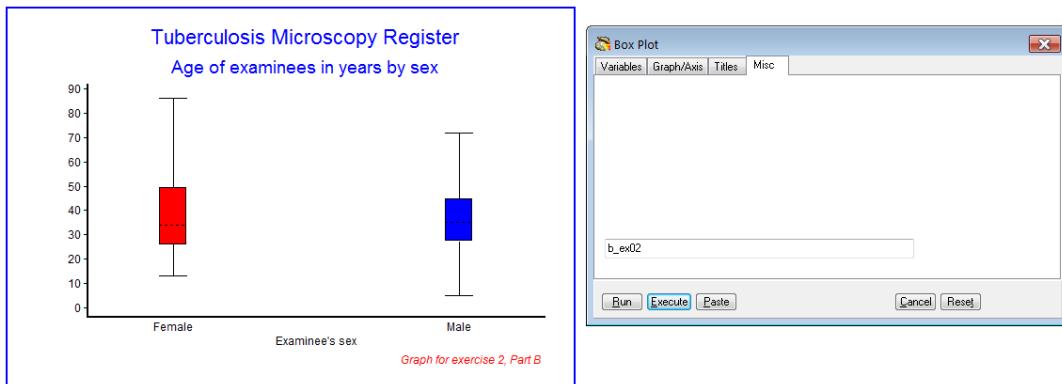
Then we refine the tab “Graph/Axis”:



then “Titles”:



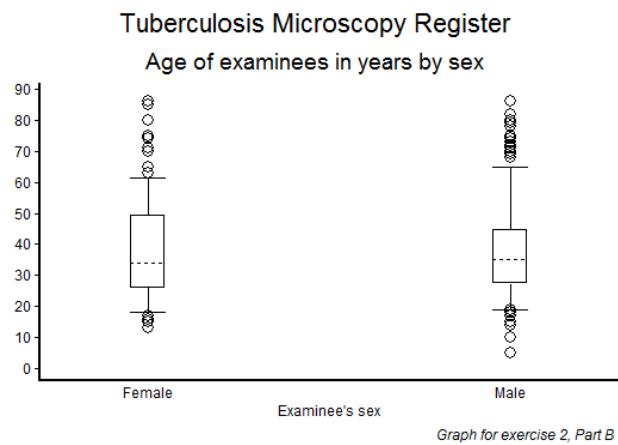
and finally “Misc”:



The most important part comes now in that we go to the command line (**F4**) and use the up cursor to get the entire list of commands, mark it, and paste it into our PGM file, making back slashes where appropriate and then have:

```
cls
BOXPLOT age /By=sex \
    /SizeX=600 /SizeY=400 \
    /Noxtick \
    /Ymin=0 /Ymax=90 /Yinc=10 \
    /Ti="Tuberculosis Microscopy Register" \
    /Sub="Age of examinees in years by sex" \
    /Fn="Graph for exercise 2, Part B" \
    /Save="b_ex02"
```

We do some additional editing, like allowing replacement of the graph (which is necessary if we name the output), and look some additional things up in the Help file, and finally have it refined to:



Note the difference in quality if we replace:

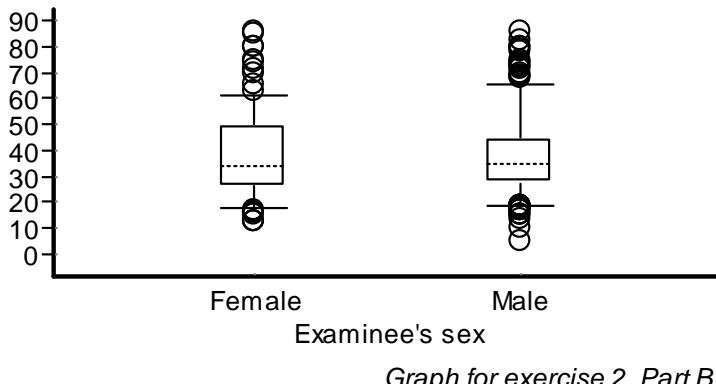
```
/Save="b_ex02" /replace \
```

with:

```
/Save="b_ex02.wmf" /replace \
```

Tuberculosis Microscopy Register

Age of examinees in years by sex

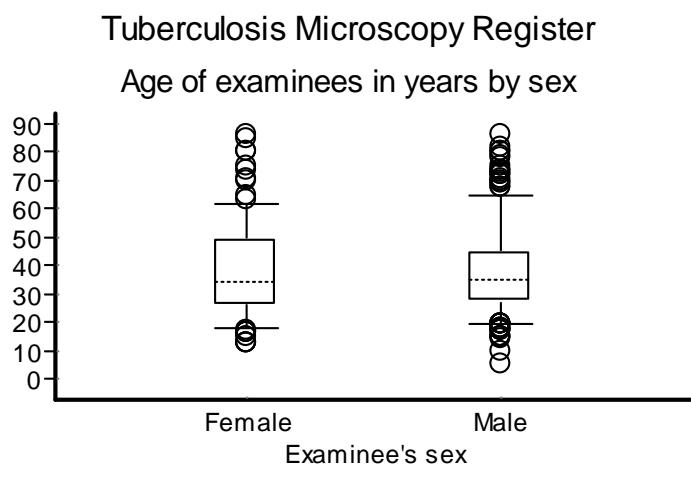


We note:

- If we save the graph as *.wmf (Microsoft metafile, a vector graph), it does not show up in the Results window
- Vector graphs are crisp and sharp and retain this irrespective of changing the graph size
- The result is not a faithful reproduction of the default *.png graph.

Sometimes the title gets too large in the metafile. This can be remediated by adding the SET to change the default font size from 10 to 9:

```
set graph font size=9
```



Graph for exercise 2, Part B

Task:

- Write a program **B_EX02.PGM** using the data set “**abcd.rec**”. Limit your analysis to patients with a diagnostic sputum smear examination. Create an output that shows the incremental yield of cases from the first, second and third of three serial smears. With incremental yield we mean determining the proportion of examinees who are positive already on the first, patients who are negative on the first, but positive on the second, and examinees who are negative on the first two but positive on the third serial smear examination. The denominator should be those who have had the required number of smears to determine the yield.

Solution to Exercise 2: More on EpiData Analysis

Key point(s):

- a) EpiData Analysis is a powerful tool to manipulate and restructure data sets
- b) Tabular outputs can be customized to specific needs
- c) To evaluate graph options, one may best use first the visual menu-driven interface and then copy and paste the resulting options into a program for permanent reproduction.

Task:

- o Write a program B_EX02.PGM using the data set “abcd.rec”. Limit your analysis to patients with a diagnostic sputum smear examination. Create an output that shows the incremental yield of cases from the first, second and third of three serial smears. With incremental yield we mean determining the proportion of examinees who are positive already on the first, patients who are negative on the first, but positive on the second, and examinees who are negative on the first two but positive on the third serial smear examination. The denominator should be those who have had the required number of smears to determine the yield.*

Solution:

The desired output is:

Yield of first smear: Px [exclude none]

Six essential patterns			
	N	%	(95% CI)
NNN	104	77.6	(69.8-83.8)
NPx	10	7.5	(4.1-13.2)
Px	20	14.9	(9.9-21.9)
Total	134	100.0	

Yield of second smear: NPx [exclude: N99]

Six essential patterns			
	N	%	(95% CI)
NNN	104	77.6	(69.8-83.8)
NPx	10	7.5	(4.1-13.2)
Px	20	14.9	(9.9-21.9)
Total	134	100.0	

Yield of third smear: NNP [exclude: N99, NN9]

Six essential patterns			
	N	%	(95% CI)
NNN	104	77.6	(69.8-83.8)
NPx	10	7.5	(4.1-13.2)
Px	20	14.9	(9.9-21.9)
Total	134	100.0	

The part of the program B_EX02.PGM which gave this output:

```
* Task Exercise 2, Part B

cls
close
read "abcd.rec"

define case #
case=0
if res1>0 and res1<9 then case=1
if res2>0 and res2<9 then case=1
if res3>0 and res3<9 then case=1
label case "Microscopy case definition"
labelvalue case /0="Non-case"
labelvalue case /1="Case"

cls
gen s(3) esspattern="NNN"
if res3=9 then esspattern="NN9"
if res2=9 then esspattern="N99"
if res3>0 and res3<9 then esspattern="NNP"
if res2>0 and res2<9 then esspattern="NPx"
if res1>0 and res1<9 then esspattern="Px"
label esspattern "Six essential patterns"

set echo=off
select reason=0
cls
type "Yield of first smear: Px [exclude none]" /h2
freq esspattern /c /ci
type "Yield of second smear: NPx [exclude: N99]" /h2
freq esspattern /c /ci if esspattern<>"N99"
type "Yield of third smear: NNP [exclude: N99, NN9]" /h2
freq esspattern /c /ci if esspattern<>"NN9" and esspattern<>"N99"
select
set echo=on
```

Exercise 3: Aggregating data and saving the summary data in a file

At the end of this exercise you should be able to:

- Understand “Aggregate” data and apply it to a specific task

We are quite familiar with:

```
cls  
close  
logclose  
  
read "abcd.rec"  
  
freq sex
```

getting as result:

Examinee's	
sex	N
Female	109
Male	191
Total	300

If we replace “freq” with “agg” (short for aggregate):

```
agg sex
```

we get essentially the same information (without the marginal, i.e. here the total):

sex	N
Female	109
Male	191

Making a frequency of a variable is nothing other than aggregating the values of that variable to the smallest common denominator and the same thing is done with aggregate. We can do it analogously for a table:

```
tables sex labcode
```

This gives (discounting the marginal) 8 inner cells for the 2 by 4 possibilities:

Examinee's sex			
Laboratory code	Female	Male	Total
A	33	42	75
B	31	44	75
C	24	51	75
D	21	54	75
Total	109	191	300

If we replace tables by aggregate:

```
agg sex labcode
```

we get the 8 inner cells listed in a column sorted by sex then by labcode:

	sex	labcode	N
Female	A	33	
Female	B	31	
Female	C	24	
Female	D	21	
Male	A	42	
Male	B	44	
Male	C	51	
Male	D	54	

As an option we can close the file that was open (the abcd.rec):

```
agg sex labcode /close
```

and we can see what remains in the Variables window:

sex	I	Examinee's sex
labcode	S	Laboratory code
n	I (N)	Total observations used in aggregate

We can browse it as it has been written to an EpiData REC file, aggregated into 8 records:

	sex	labcode	N
1	Female	A	33
2	Female	B	31
3	Female	C	24
4	Female	D	21
5	Male	A	42
6	Male	B	44
7	Male	C	51
8	Male	D	54

As an additional option we can save this aggregation to a REC file (note that the option is “/save”, not “/savedata”):

```
agg sex labcode /close /save="labsex_set.rec" /replace
```

then close everything and re-open either file.

The first property of the aggregate command is thus that it can replace the summary of a tables command and save the cells of the table (made from 1, 2, or more variables) in a REC file.

The second property of the aggregate command is related to the power it provides with options. For example:

```
cls
close
logclose
read "abcd.rec"
agg sex /mean=age if age<>99
```

gives:

sex	N	Nage	MEAage
Female	109	109	39.11
Male	191	191	38.71

We have now the mean age by sex, but we could also get the mean age by sex and laboratory:

```
agg sex labcode /mean=age if age<>99
```

sex	labcode	N	Nage	MEAage
Female	A	33	33	42.97
Female	B	31	31	34.87
Female	C	24	24	37.75
Female	D	21	21	40.86
Male	A	42	42	37.60
Male	B	44	44	35.11
Male	C	51	51	39.24
Male	D	54	54	42.00

and as above, all can be written into a REC file:

```
agg sex labcode /mean=age /close /save="labsex_set_2.rec" /replace \
if age<>99
cls
close
read "labsex_set_2.rec"
```

↪	sex	labcode	N	Nage	MEAage
1	Female	A	33	33	42.9696969697
2	Female	B	31	31	34.8709677419
3	Female	C	24	24	37.7500000000
4	Female	D	21	21	40.8571428571
5	Male	A	42	42	37.5952380952
6	Male	B	44	44	35.1136363636
7	Male	C	51	51	39.2352941176
8	Male	D	54	54	42.0000000000

The automatically created variable N denotes the number of records in the original file, Nage the number of records in the original file with non-missing information on age, and MEAage, the mean age in the aggregated stratum.

There are more options of this kind, like:

```
/min
/max
/stat
/sum
```

You can also cumulate different options, and you can use the same option for different variables, e.g. “/sum=var1 /sum=var2 /min=var3”.

It is the option /sum we are particularly interested here to apply. We can sum up all the values of a given variable across all records aggregated within a stratum. For instance, in the following task you will need to count the number of smears each examinee had, and then you can sum up all the smears within each stratum resulting from your aggregation.

Workload in the laboratory

One measure of the workload in laboratories is the number of smears they have to examine per day. Even the busiest laboratories do not work every day, they may close on weekends and public holidays. One may get some approximate estimate of the number of working days, but a much cleaner way is to count the actual working days. The tuberculosis laboratory register gives a possible good approximation with the date of registration. While smears are also examined on other than the registration date (the first specimen defines that date, and a patient gives the first specimen on the spot but brings in an early morning specimen one day later), this is a reasonably good approximation to the number of days work was actually carried out, certainly better than some other approximation without a good data basis.

We provide a data set “`b_ex03_workload.rec`” from three laboratories in Zimbabwe (data courtesy Dr Biggie Mabaera), each complete but also limited to during one calendar year.

Task:

- o The `B_EX03_WORKLOAD.REC` has been edited to contain only three laboratories (out of the original 30) and only the year 2002. Nonsensical results (e.g., first examination not recorded, followed by a valid result) have been excluded. Create a program `B_EX03.PGM` to provide the mean number of smears examined per registration day in each of the three laboratories.*

Solution to Exercise 3: Aggregating data and saving the summary data

Key point(s):

- Aggregate is a powerful tool to summarize the analogue to a tabular output in a REC file and to make calculations on the vertical for each stratum thus obtained.

Task:

- The B_EX03_WORKLOAD.REC has been edited to contain only three laboratories (out of the original 30) and only the year 2002. Nonsensical results (e.g., first examination not recorded, followed by a valid result) have been excluded. Create a program B_EX03.PGM to provide the mean number of smears examined per registration day in each of the three laboratories.*

The result:

laboratory	Obs.	Sum	(SUM) smears						
			Mean	Variance	Std Dev	(95% CI	mean)	Std	Err
BY_A	242	23044.0	95.22	1069.10	32.70	91.08	99.36	2.10	
ME_L	135	1211.00	8.97	53.34	7.30	7.73	10.21	0.63	
ML_L	241	6328.0	26.26	244.65	15.64	24.27	28.24	1.01	

The program B_EX03.PGM is very simple in the end (but admittedly it took us a while to get to this level of efficiency):

```
cls
close
logclose

read "b_ex03_workload.rec"

* Determine the number of smears for each examinee
gen i smears=1
if result2<>9 then smears=2
if result3<>9 then smears=3

* Sum up the number of smears done on each
* working day in each laboratory
aggregate regdate laboratory /sum=smears /close

* Calculate the average number of smears done
* each working day in each laboratory
cls
means sumsmears /by=laboratory
```

Exercise 4: From a spreadsheet to an EpiData file

At the end of this exercise you should be able to:

- a. Master the import of spreadsheet content into an EpiData file
- b. Recode the imported content to an analyzable file

Quite often data are captured in a spreadsheet. When trying to then analyze these data, the substantial limitations to the use of spreadsheet software in this respect become recognized. We are then compelled to import the spreadsheet data into a data file format more amenable to analysis. An EpiData file is an appealing solution as the software also disposes of powerful restructuring tools permitting reshaping the data into a format that is convenient for analysis using EpiData Analysis.

There are two approaches which are the same in principle: 1) copying the spreadsheet content to the clipboard, or 2) exporting the spreadsheet content to a delimited text file. EpiData Analysis is given a respective command to either read the clipboard content or the text file. We will demonstrate both approaches with two simple examples and then provide for the task a real dataset of somewhat more complexity.

Formatting the spreadsheet correctly

In the required files, you find the workbook “b_ex04_spreadsheet_data_1_original.xlsx” with 8 records (data rows) and 6 variables (column headings):

	A	B	C	D	E	F
1	Identification	sex	date of birth	marital status	Living arrangements	housing surface in sqm
2	A	m		married	apartment lease	
3	B	F	20-02-01	divorced	Living alone	80
4	C	f	15-12-95	widowed	house owner	150
5	D	m	26-07-85	cohabitating	appartment	105
6	E	M	24-03-03	married	own house	180
7	F		04-04-79	married	leasing an apartment	95
8	G	f	17-02-98	not married	appartment	110
9	H	m	12-01-05	unmarried	living with parents	75

The workbook has this single sheet labeled “original”. Under no circumstance do we wish to make any alteration whatsoever to the original. We therefore make as the first thing a copy of this content into a new sheet in the same workbook. We create a new sheet and label it “epidata”. We don’t want to copy and paste but we rather resolve the copying by using a formula.

In the new sheet “epidata” in cell A1 we type the “=” sign (no quotes) and then place the cursor into cell A1 in the “original” sheet and press the Enter key. This gets us back to cell A1 in the “epidata” sheet in which we now find written:

A1		X	✓	fx	=original!A1
	A	B	C	D	E
1	Identification				

Copy cell A1 in “epidata” to the first two rows, columns A to F and get:

	A	B	C	D	E	F	G
1	Identification	sex	date of birth	marital status	Living arrangement	housing surface in sqm	
2	A	m		0 married	apartment	0	0
3							

There are 3 rules that must be followed else the transfer of data from spreadsheet to EpiData will either not function or it will contain errors:

- 1) The variable names must be valid EpiData names i.e. a) single word, b) length not exceeding 10 characters, c) not beginning with a number.
- 2) The first row of data cannot contain any empty cell.
- 3) The data for a given variable must all be of the same type.

We thus overwrite the first row containing the variable names with valid EpiData names. Doing this, we remember that EpiData is not case sensitive, but many other analysis software is (like Stata or R). Despite EpiData not being case sensitive for variable names, we will make it always strictly unambiguous with a simple rule: we recommend and use only lower-case letters for variable names. Thus, we might write:

A	B	C	D	E	F
1	id	sex	dob	marital	living

While a length of up to 10 characters is allowed for EpiData variable names, we prefer to have the names even shorter: our longest names here have 7 characters. We will later make explicit variable labels so that it is unambiguously clear what the variable names refer to.

Next we modify the formula for the cells A1 to F1 in a way to take care of two things at the same time, i.e. providing a value for a given cell in case it is empty and defining the type of variable the column contains. We assume that column A with the identifier always contains a value (else we will discard records without identifier later in EpiData Analysis). Thus, cell A1 is left untouched. Column B is a string (text) field and it has length 1. Accordingly, we modify the current formula from:

=original!B2

to:

=IF(original!B2<>"",original!B2,"x")

If you are not familiar with the “IF” statements in spreadsheet syntax, you may consult the help file. Basically it is:

IF(logical test, if true write this, else write that)

We thus evaluate with the logical test whether the content of the cell in the sheet “original” is not empty (denoted as opening and closing double-quote “ ”), and if it is not empty, then we take that value, else an x is written (denoted as a string by surrounding it with double-quotes “x”).

We copy the thus revised formula of cell B2 to cell C2. “x” is obviously erroneous, because this is a date field, not a string field. It probably depends on the language of your spreadsheet software how a date is written, thus check first in the help file. In our English version it becomes:

```
=IF(original!C2<>"",original!C2,DATE(1900,1,1))
```

We need to choose a legal date, but one that obviously cannot be a genuine date of birth in our data set. We chose the earliest possible date in the spreadsheet after the anchor date (which is 31 Dec 1899, in EpiData it is 31 Dec 1799), thus the choice of 1 Jan 1900: it’s legal, it is written as an actual date, and it is obviously not a genuine date of birth in the context of the data set.

Cell D2 is a string analogous to cell B2 which we thus copy to here, but make perhaps 3 x’s as it is a longer field than just length 1:

```
=IF(original!D2<>"",original!D2,"xxx")
```

Cell E2 is actually the same as D2, so we copy cell D2 to cell E2. We introduce something more here. While it is actually not relevant for this specific case here, it can become relevant with text fields like comments: in EpiData a text field cannot exceed a length of 80 characters, else an error occurs when attempting to read a field with a larger length. To prevent that the value in such a string field exceeds the permissible length, we tell the spreadsheet that it should cut off anything beyond a certain length. We choose here (arbitrarily) that critical length to be 50 characters after which the rest is cut off and write:

```
=IF(original!E2<>"",MID(original!E2,1,50),"xxx")
```

The function in the spreadsheet software is “MID” and it requires three arguments, 1) the cell name, 2) the starting character position (1), and 3) the ending character position (50).

The final variable sqmeter (first data cell F2) is an integer field of a length 3, thus the formula becomes a copy of D2, yet the value for missing doesn’t have quotes and must be a number (because the variable is an integer field):

```
=IF(original!F2<>"",original!F2,999)
```

As the first two rows are now complete, row 2 can be copied up to and including row 9 and we get:

	A	B	C	D	E	F
1	id	sex	dob	marital	living	sqmeter
2	A	m	01-01-1900	married	apartment lease	999
3	B	F	20-02-2001	divorced	Living alone	80
4	C	f	15-12-1995	widowed	house owner	150
5	D	m	26-07-1985	cohabitating	appartment	105
6	E	M	24-03-2003	married	own house	180
7	F	x	04-04-1979	married	leasing an apartment	95
8	G	f	17-02-1998	not married	appartment	110
9	H	m	12-01-2005	unmarried	living with parents	75

If we mark the data rectangle A1:F9, copy it to clipboard and then paste it into our text editor, we get:

```

1 id sex dob marital living sqmeter
2 A m 01-01-1900 married apartment lease 999
3 B F 20-02-2001 divorced Living alone 80
4 C f 15-12-1995 widowed house owner 150
5 D m 26-07-1985 cohabitating appartment 105
6 E M 24-03-2003 married own house 180
7 F x 04-04-1979 married leasing an apartment 95
8 G f 17-02-1998 not married appartment 110
9 H m 12-01-2005 unmarried living with parents 75

```

This suggests that the clipboard memory content has a Tab-delimited format. You can test this assumption by placing your cursor after the “A” and then move the cursor: it jumps to before “m”. In Notepad++ text editor (available on the course web) you can choose View=>Show symbol=>Show white space and TAB and you actually see spaces as points and TABs as → arrows:

```

1 id->sex->dob->marital->living->sqmeter
2 A->m->01-01-1900->married->apartment · lease->999
3 B->F->20-02-2001->divorced->Living · alone->80
4 C->f->15-12-1995->widowed->house · owner->150
5 D->m->26-07-1985->cohabitating->appartment->105
6 E->M->24-03-2003->married->own · house->180
7 F->x->04-04-1979->married->leasing · an · apartment->95
8 G->f->17-02-1998->not · married->appartment->110
9 H->m->12-01-2005->unmarried->living · with · parents->75

```

EpiData will correctly interpret the difference between a TAB delimiter and one or more space characters.

Reading data from clipboard into EpiData Analysis and save it as an EpiData file

In EpiData Analysis we prepare the basics to read data from clipboard and save them to an EpiData *.REC file:

```

cls
close
logclose

```

```

read /cb
savedata "spreadsheet_data_1.rec" /replace

```

The command is remarkably simple: read with the option / cb (for clipboard). It makes of course sense to save it right away from memory to a data file with an intermediary name of your choice. Also, save the program as “b_ex04_task.pgm”.

Switch to the sheet “epidata” and copy the rectangular data A1:F9 to the clipboard, then switch back to EpiData analysis and run all with F9. The output should then read:

```

. read /cb
Loading data from clipboard, please wait...
First line: (clipboard) id sex dob marital living sqmeter
Separator: Tab (tab: 45)(semicolon: 0)(comma: 0)(space: 9) Lines: 9 (incl. field names)
File name :from clipboard
Fields: 6 Total records: 8 Included: 8
. savedata "spreadsheet_data_1.rec" /replace
Saving data to: c:\epidata_course\spreadsheet_data_1.rec
6 Fields 8 Records

```

If we browse, we get:

	id	sex	dob	marital	living	sqmeter
1	A	m	01/01/1900	married	apartment lease	999
2	B	F	20/02/2001	divorced	Living alone	80
3	C	f	15/12/1995	widowed	house owner	150
4	D	m	26/07/1985	cohabitating	apartment	105
5	E	M	24/03/2003	married	own house	180
6	F	x	04/04/1979	married	leasing an apartment	95
7	G	f	17/02/1998	not married	apartment	110
8	H	m	12/01/2005	unmarried	living with parents	75

From the Variables window (press F3, if not shown spontaneously), we check whether all the variables are of the type that we wanted them to be, most notably that dates are date variables and numbers are integer or float variables:

```

id      S id
sex     S sex
dob     D dob
marital S marital
living   S living
sqmeter I sqmeter

```

This is the case here, but this should not just be assumed: we may encounter that a date field has become a text field because the spreadsheet column hadn't been formatted correctly or that one single date among all had a wrong format. It is crucial to ensure that all is in order in respect to the type of variable.

It is immediately apparent in the small dataset here that all records have an identifier. However, we better check it formally as we cannot use records that cannot be checked against the original. We make a binary variable:

```
cls  
close  
read "spreadsheet_data_1.rec"  
  
cls  
gen i hasid=1  
if id=. then hasid=0  
freq hasid /m
```

and get:

hasid	
	N
1	8
Total	8

We are thus satisfied that all records have an identifier. That none is a duplicate will be done later again in another form, but this is how we would do that:

```
cls  
sort id  
gen i seq=1  
if id=id[_n-1] then seq=seq[_n-1]+1  
freq seq
```

and we get:

seq	
	N
1	8
Total	8

No identifier occurs more than once and every record has an identifier. What we did here was to sort the records by ID to ensure that any identical identifiers would be next to each other. Then we make a variable SEQ and give it in every record the default value 1. A powerful capability of EpiData Analysis is to look at records before or after the current position of the process. EpiData Analysis proceeds record by record through a data file, from top to bottom. The current record has the position [_n]. There is no need to give the designation for the current record, but in fact id and id[_n] are both valid designations for the identifier in the current position. id[_n-1] designates the identifier of the record immediately preceding that of the current record, while id[_n+5] would be the identifier of the fifth record forward from the current one. Our commands above thus say “if the record before the current one has the same identifier as the current one, then add 1 to the value of seq in the current record to the value that seq has in the record before the current record”. As a result, all seq values must be 1 if there are no multiples of the identifier. We will make extensive use of this positioning when we deal with the analysis of a relational database. For the time being we

can asterisk the frequencies and drop the two variables we just created, and give a label to the variable ID:

```
cls  
close  
read "spreadsheet_data_1.rec"  
  
cls  
gen i hasid=1  
if id=. then hasid=0  
* freq hasid /m  
  
cls  
sort id  
gen i seq=1  
if id=id[_n-1] then seq=seq[_n-1]+1  
* freq seq  
  
drop hasid seq  
  
label id "Unique identifier"
```

It is still common practice to use string (text) fields for the values of categorical variables like sex or housing arrangement, etc, as was also done here in the spreadsheet. For instance, a frequency of the variable SEX gives here:

sex	
	N
f	2
F	1
m	3
M	1
x	1
Total	8

The value “x” was our doing to denote the value in records with no information on the variable SEX. In this spreadsheet, there was no control over the way how the value should be written and we have thus 4 instead of the 2 known sexes. Of course, in EpiData it is easy to exert control to avoid such errors (it’s also possible in a spreadsheet). In any case, there are issues with using text values for categorical variables. There are good reasons why it is nowadays standard to use systematically numeric coding for all variables unless there is a compelling reason to use a string (such as always for identifiers as nobody uses them to count something). The main reason for numeric coding is efficiency. Evaluation of text fields in analysis is “expensive” as text must be evaluated, then converted into numbers internally: computers don’t add up letters, they add up numbers. Text values take processing time in excess of what is required if all calculation can be done directly with numbers. Numeric coding is also efficient for file size and thus another factor for processing speed: to be unambiguous and clear, the required length of a field containing a text value for a categorical field is often more than 1, while with numeric coding it can commonly be just 1 (allowing up to 10 strata).

In the following, we will convert all categorical text fields to fields with numeric coding with unambiguous metadata for the labels.

Starting with the variable SEX, we can use the EpiData function to convert all text values (m and M, and f and F) to lower case to get more swiftly from the current 5 to the desired 3 required values:

```
cls  
freq sex  
define sexn #  
sexn=9  
if lower(sex)="f" then sexn=1  
if lower(sex)="m" then sexn=2  
tables sexn sex
```

The function is LOWER(variablename). To do this systematically, we start with a frequency of the current variable, and end with a cross-tabulation of the new against the old variable to check that all assignments are correct. We could then drop the two respective commands, but we prefer to leave them with a preceding asterisk denoting it as a comment. This is a comment in the script to assure ourselves that we had checked. Then we drop the old variable and label the new one properly, followed by a final frequency of the new variable to assure ourselves that all is in order:

```
cls  
* freq sex  
define sexn #  
sexn=9  
if lower(sex)="f" then sexn=1  
if lower(sex)="m" then sexn=2  
* tables sexn sex  
drop sex  
rename sexn to sex  
label sex "Patient's sex"  
labelvalue sex /1="Female"  
labelvalue sex /2="Male"  
labelvalue sex /9="Not recorded"  
* freq sex
```

The output is now neat and clean without ambiguity:

Patient's	
sex	N
Female	3
Male	4
Not recorded	1
Total	8

The variable DOB only needs a proper variable label:

```
label dob "Date of birth"
```

The variable LIVING has obviously been made free text with the resulting short-comings:

living	N
apartment lease	1
appartment	2
house owner	1
leasing an apartment	1
Living alone	1
living with parents	1
own house	1
Total	8

We have 7 values in 8 records. Such coding could be very tedious and is obviously pretty useless for analysis as such. There is no other way than actually going through the actual original values one by one and trying to put them into reasonable categories. We could be forgetting one or the other value as it is so tedious. To have an alert for omissions, we make a default value (choosing here "8") that should not remain if all re-assignments are proper. We also make use of the function SUBSTR(variablename, beginning position, ending position) to avoid copying all of it. The latter is unfortunately often required, but not in the relatively simple case here:

```

cls
* freq living
define livingn #
livingn=8
if living="xxx" then livingn=9
if substr(living,1,4) ="apar" then livingn=1
if substr(living,1,4) ="appa" then livingn=1
if substr(living,12,4)="apar" then livingn=1
if substr(living,1,4) ="hous" then livingn=2
if substr(living,5,4) ="hous" then livingn=2
if substr(living,13,4)="pare" then livingn=3
if substr(living,8,4) ="alon" then livingn=4
* tables livingn living
drop living
rename livingn to living
label living "Housing arrangements"
labelvalue living /1="Apartment"
labelvalue living /2="House"
labelvalue living /3="Parents"
labelvalue living /4="Other"
labelvalue living /9="Not recorded"

```

The recoded field has somewhat more appeal than the original:

Housing arrangements	N
Apartment	4
House	2
Parents	1
Other	1
Total	8

We finalize recoding with the field MARITAL analogously and provide the field SQMETER with just a variable label so that in the end we have a neatly recoded dataset:

	id	dob	sqmeter	sex	living	marital
1	A	01/01/1900	999	Male	Apartment	Married
2	B	20/02/2001	80	Female	Other	Divorced
3	C	15/12/1995	150	Female	House	Widowed
4	D	26/07/1985	105	Male	Apartment	Cohabitating
5	E	24/03/2003	180	Male	House	Married
6	F	04/04/1979	95	Not recorded	Apartment	Married
7	G	17/02/1998	110	Female	Apartment	Single
8	H	12/01/2005	75	Male	Parents	Single

Exporting the spreadsheet to a delimited text file and importing that file into EpiData

We mentioned above that copying the data in the spreadsheet to clipboard and exporting the data to a delimited text file is essentially the same approach. We showed that the copying to the clipboard is copying a tab-delimited file to the clipboard that EpiData Analysis understands to be of this format with the option /cb. EpiData Analysis also allows importing text files that are physically on disk and have a name and an extension that identifies them as a text file. Text files have the extension *.txt, but this extension does not define the nature of the delimiter, i.e. the separator between the values for separate variables. In EpiData Entry 3.1 we can import files with the extension *.txt and tell the software which delimiter the file actually has (i.e. Tab, comma, semi-colon). The issue is of importance because the choice of the delimiter must always be something that is not used in the data: for instance, a tab will not be used between words of the value in a text field, as we use a space character to separate words. In the US and many other countries, the decimal separator is the full stop. In many other countries, the decimal separator is a comma. In our Windows® software we make in our set-up the definitions and our spreadsheet software will then adhere to these rules. Not everybody actually makes the effort to properly set up the display rules for the display of numbers. As a result, we must be very careful to check whether what we demand is then actually what we get.

A point in case is the most commonly used delimited text file format, the comma-separated values, with the extension *.csv. Fortunately, it seems, the issue is solved correctly by spreadsheet software such as Excel® or LibreOffice Calc:

- If the Windows set-up is to use a full stop as the decimal separator, then the delimiter between variables will be a comma when exporting to *.csv.
- If the Windows set-up is to use a comma as the decimal separator, then the delimiter between variables will be a semicolon when exporting to *.csv.

EpiData Analysis will be able to correctly identify the delimiter either way (but always check!). We will thus export to *.csv. Once you have exported, examine it in the text editor.

When using a full stop as decimal separator, we get:

```
1 id,sex,dob,marital,living,sgmeter
2 A,m,01-01-1900,married,apartment · lease,999
3 B,F,20-02-2001,divorced,Living · alone,80
4 C,f,15-12-1995,widowed,house · owner,150
5 D,m,26-07-1985,cohabitating,appartment,105
6 E,M,24-03-2003,married,own · house,180
7 F,x,04-04-1979,married,leasing · an · apartment,95
8 G,f,17-02-1998,not · married,appartment,110
9 H,m,12-01-2005,unmarried,living · with · parents,75
```

When using a comma as decimal separator, we get:

```
1 id;sex;dob;marital;living;sgmeter
2 A;m;01-01-1900;married;apartment · lease;999
3 B;F;20-02-2001;divorced;Living · alone;80
4 C;f;15-12-1995;widowed;house · owner;150
5 D;m;26-07-1985;cohabitating;appartment;105
6 E;M;24-03-2003;married;own · house;180
7 F;x;04-04-1979;married;leasing · an · apartment;95
8 G;f;17-02-1998;not · married;appartment;110
9 H;m;12-01-2005;unmarried;living · with · parents;75
```

We note that this works correctly even if – like in our case – there is not even a real number (float variable) in the data set. Important is here to note again that the two approaches are the same, except that one is tab-delimited, the other comma (or semicolon) delimited. Neither will work if the content is not properly formatted!

More complexity: two specimens with a discordant result taken at the same time

In the workbook “b_ex04_2_original.xlsx” we have a bit added complexity. The unit of observation here is not one patient but one specimen. Specimens are taken at different visits of individual patients. Furthermore, it is possible that more than one specimen is taken at one given visit. We added the case where two different specimens yield a different result at one given visit. What we wish to do is to 1) import the data from the spreadsheet and then 2) manipulate the EpiData file so that each observation time per patient has a single hierarchically dominant result and that the way to go about it would not be limited to just two specimens as in our example.

We proceed as before by creating a second sheet “epidata” in the workbook that contains the EpiData-compatible reformatted data from the “original” sheet. It is easiest to open the “b_ex04_1.pgm” program in EpiData Analysis and save it right away as “b_ex04_2.pgm”. The rectangular file is then copied to clipboard and read and saved in EpiData Analysis as before. All the variable manipulations that exist are retained with perhaps minor modifications, and the additional ones are added.

Special consideration is given to the recoding of the specimen result. We have more than one specimen in a given month per patient and they may be discordant such as in:

PATID	MM	SPECSEQ	RESULT
A	1	1	negative
A	1	2	positive
A	1	3	no result

What we want to have is a single RESULT per patient-month. We also want this to be hierarchical, i.e. that any result overrides no result, and a positive result overrides a negative result, and that at the end of our procedure the hierarchically highest result is written into the first of the sequences, i.e. that the above becomes:

PATID	MM	SPECSEQ	RESULTN
A	1	1	positive
A	1	2	positive
A	1	3	no result

Then we can retain only the first record in the sequence, i.e. SPECSEQ=1 to have only one result per patient-month. There are of course different approaches to solving this, but we propose hierarchical numeric recoding, i.e.:

Result	Value
No result	0
Negative	1
Positive	2

We make first a variable that combines patient identifier and month to a new identifier (e.g. PATMMID):

```
gen s(3) patmmid=patid+"-"+mm
label patmmid "Patient-month identifier"
```

We then sort by this new identifier and within it by month, and number the sequence similar as we did before, i.e.:

```
sort patmmid mm
gen i seq=1
if patmmid[_n-1]=patmmid then seq=seq[_n-1]+1
```

Thus, we should get something of the type:

PATMMID	SEQ	MM	SPECID	RESULT
A-1	1	1	X123	negative
A-1	2	1	Y321	positive
A-1	3	1	A412	no result
A-2	1	2	B123	negative
B-0	1	0	C123	positive

The sequence of the RESULT is irrelevant as the hierarchical coding suffices to get things right in the next step. In this next step we reshuffle the hierarchically highest to the first in the sequence:

```
define resultn #
resultn=result
if result[_n+1]>result then resultn=result[_n+1]
drop result
rename resultn to result
label result "DST result"
labelvalue result /0="No result"
labelvalue result /1="Susceptible"
labelvalue result /2="Resistant"
```

and we will get something of the type:

PATMMID	SEQ	MM	SPECID	RESULT
A-1	1	1	X123	positive
A-1	2	1	Y321	positive
A-1	3	1	A412	no result
A-2	1	2	B123	negative
B-0	1	0	C123	positive

After this the specimen identifier is incorrectly paired with the result and can be dropped together with the sequence counter and we have a data set in which PATMMID is a unique identifier for patient-month (which we must check).

Tasks:

- o *The B_EX04_TASK.XLSX is a real data set from a study on laboratory drug susceptibility test results from several countries / jurisdictions. Create a sheet satisfying EpiData format requirements. Copy the rectangular selection to the clipboard and read it into EpiData Analysis and save it to an EpiData *.REC file. Note that it might be tricky to deal properly with some variables that may prevent correct reading of the rectangular file!*
- o *The data set has a patient identifier, IDCODE. This identifier is unique for the patient in a given jurisdiction, but it is not unique for the data set. Specimens might be taken at the start of treatment or at any month on treatment. One patient in the data set thus may have several results. Make a variable IDCODE2 by adding the month to IDCODE. This will make it unique for a given patient in a given month. Actually – not quite: there may be more than one specimen in a given month, but only one result should count. The IDCODE2 could also come from two different patients if they are from a different jurisdiction. Make a third identifier so that you get in total three: 1) taking jurisdiction, patient and month of treatment, 2) taking jurisdiction and patient and 3) taking patient into account.*
- o *Sort the data set so that all specimens for a given patient-month are next to each other, then recode all variable pertaining to isoniazid in a hierarchical manner, so that it becomes easy to assign the value that will be retained to the first specimen for that patient-month. Hierarchy: high-level resistance>low-level resistance>susceptible>no result. Then select to retain only the first specimen per patient-month.*
- o *Recode the various variables for isoniazid drug susceptibility test results (phenotypic and genotypic results). Recode to a single variable for the isoniazid result. Discuss the hierarchy that you wish to assign to this end.*

Solution to Exercise 4: From a spreadsheet to an EpiData file

At the end of this exercise you should be able to:

- a. Master the import of spreadsheet content into an EpiData file
- b. Recode the imported content to an analyzable file

Task:

- o The B_EX04_TASK.XLSX is a real data set from a study on laboratory drug susceptibility test results from several countries / jurisdictions. Create a sheet satisfying EpiData format requirements. Copy the rectangular selection to the clipboard and read it into EpiData Analysis and save it to an EpiData *.REC file. Note that it might be tricky to deal properly with some variables that may prevent correct reading of the rectangular file!*

Solution:

The tricky part was perhaps to deal with the original sheet variable IDCODE. If the values were left unchanged, EpiData may misread it as a date variable with erroneous dates that did not exist and in consequence the import failed. We solved it by unambiguously making it to a text field. The value:

2014-01-006

was made to be:

x-2014-01-006

by the following code:

```
=IF(original!G2<>" ",CONCATENATE("x-", original!G2), "xxx")
```

The preceding “x-” identified the value unambiguously to a string field for EpiData Analysis. The extraneous “x-” was then easily stripped in EpiData Analysis with:

```
gen s(11) idcode0=substr(idcode2,3,11)
```

Task:

- o The data set has a patient identifier, IDCODE. This identifier is unique for the patient in a given jurisdiction, but it is not unique for the data set. Specimens might be taken at the start of treatment or at any month on treatment. One patient in the data set thus may have several results. Make a variable IDCODE2 by adding the month to IDCODE. This will make it unique for a given patient in a given month. Actually – not quite: there may be more than one specimen in a given month, but only one result should count.*

The IDCODE2 could also come from two different patients if they are from a different jurisdiction. Make a third identifier so that you get in total three: 1) taking jurisdiction, patient and month of treatment, 2) taking jurisdiction and patient and 3) taking patient into account.

Solution:

This is relatively simple to accomplish. We chose this approach:

```

cls
* freq country
define countryn #
if country="A" then countryn=1
if country="B" then countryn=2
if country="C" then countryn=3
if country="D" then countryn=4
if country="E" then countryn=5
if country="F" then countryn=6
if country="G" then countryn=7
if country="H" then countryn=8
drop country
rename countryn to country
label country "Name of country / jurisdiction"
labelvalue country /1="Jurisdiction A"
labelvalue country /2="Jurisdiction B"
labelvalue country /3="Jurisdiction C"
labelvalue country /4="Jurisdiction D"
labelvalue country /5="Jurisdiction E"
labelvalue country /6="Jurisdiction F"
labelvalue country /7="Jurisdiction G"
labelvalue country /8="Jurisdiction H"

cls
* Create derived identifiers
gen s(16) idcoden=country+substr(idcode,2,12)+"-"+mmtxt
drop idcode2 idcode
rename idcoden to idcode2
label idcode2 "Country-patient-month identifier"
cls
gen s(13) idcode=substr(idcode2,1,13)
label idcode "Country-patient identifier"
cls
gen s(11) idcode0=substr(idcode2,3,11)
label idcode0 "Patient identifier"

```

Task:

- o Sort the data set so that all specimens for a given patient-month are next to each other, then recode all variable pertaining to isoniazid in a hierarchical manner, so that it becomes easy to assign the value that will be retained to the first specimen for that patient-month. Hierarchy: high-level resistance>low-level resistance>susceptible>no result. Then select to retain only the first specimen per patient-month.*

Solution:

First, we determined the frequency of the number of specimens per patient-month, using a standard approach which you will be using very often as this question arises very frequently:

```
cls  
* Identify multiple specimens per patient-month  
sort idcode2 mm  
gen i specseq=1  
if idcode2=idcode2[_n-1] then specseq=specseq[_n-1]+1  
* freq specseq  
* =>  
*      N  
* 1      582  
* 2      26  
* 3      5  
* 4      4  
* 5      2  
* 6      2  
* Total 621  
* => Up to 6 specimens for 1 patient in 1 month
```

It is not necessary to know in this specific situation that at most 6 specimens are available per patient months, but it will be of key importance to know it for the next step. We show this here for isoniazid phenotypic result at 0.2 mg/L, i.e. the variable H02. Currently, these are the original (spreadsheet-coded) results:

```
NT      490  
R      126  
S      5  
Total 621
```

As we recommended a hierarchical numerical coding, we create a new numeric variable H02N and then “reshuffle the values” hierarchically to the first record within IDCODE2:

```
cls  
define h02n #  
h02n=0  
if h02="S" then h02n=1  
if h02="R" then h02n=2  
if idcode2[_n+1]=idcode2 and h02n[_n+1]>h02n then h02n=h02n[_n+1]  
if idcode2[_n+2]=idcode2 and h02n[_n+2]>h02n then h02n=h02n[_n+2]  
if idcode2[_n+3]=idcode2 and h02n[_n+3]>h02n then h02n=h02n[_n+3]  
if idcode2[_n+4]=idcode2 and h02n[_n+4]>h02n then h02n=h02n[_n+4]  
if idcode2[_n+5]=idcode2 and h02n[_n+5]>h02n then h02n=h02n[_n+5]  
drop h02  
rename h02n to h02  
label h02 "INH result at 0.2 mg/L"
```

Before we drop h02 we browse to see all relevant variables, picking the case with 6 specimens in a given patient-month:

idcode2	mm	specseq	h02	h02n
1-2014-01-011-00	0	1	NT	2
1-2014-01-011-00	0	2	NT	2
1-2014-01-011-00	0	3	R	2
1-2014-01-011-00	0	4	R	2
1-2014-01-011-00	0	5	R	2
1-2014-01-011-00	0	6	NT	0

Note that the importance is that the first line of data (that is the record with SPECSEQ=1) must be correct in that it takes the hierarchically highest among all results from the total of specimens for the given patient-month. The highest value for H02 is R which translates numerically to 2 and this is correctly appearing in the first record of the sequence. It is thereby irrelevant that the last value is 0 (it is logical that it retains the default because there is no other record afterwards with the same IDCODE2) because after completing this approach for each variable (on isoniazid, then other drugs), all that is retained are the records in which SPECSEQ=1:

```
select specseq=1
```

Task:

- o Recode the various variables for isoniazid drug susceptibility test results (phenotypic and genotypic results). Recode to a single variable for the isoniazid result. Discuss the hierarchy that you wish to assign to this end.*

Solution:

This might be a somewhat arbitrary classification and will surely required expert input. We chose here the following hierarchy (this is specific for isoniazid, but must be defined for each individual drug, and for other drugs this may differ):

Phenotypic result > genotypic result

In the absence of a phenotypic result, the genotypic result counts

If there are genotypically mutations in both the *katG* and *inhA* promoter gene, then the resistance is high-level, whatever the result of phenotypic testing may be. Thus, we coded:

```
define inh #
inh=9
if inhlevel=0 then inh=inhmolsum
if inhlevel=1 then inh=1
if inhlevel=2 then inh=2
if inhlevel=3 then inh=2
if inhlevel=4 then inh=3
if inhmolsum=3 then inh=3
if inha=2 and katg=2 then inh=3 // already done, but make sure
label inh "Resistance to INH"
```

```
labelvalue inh /0="No result"  
labelvalue inh /1="Susceptible"  
labelvalue inh /2="Low-level resistance"  
labelvalue inh /3="High-level resistance"
```

The entire b_ex04_solution.pgm reads as:

```
* Part B, Exercise 4
* Import spreadsheet data and create identifiers
* EpiData course
* Author: Hans L Rieder
* First version: 05 Feb 2018
* Current version: 02 Mar 2018

*****
* 1) Read data from spreadsheet and save to EpiData file
cls
close
logclose

* read /cb
* savedata "b_ex04_task_in.rec" /replace

*****
* 2) Read EpiData file, make basic checks and create new
* identifiers:
* idcode : idcode [original, derived]
* idcode0: country-idcode [original extended]
* idcode2: country-idcode-month [original extended]
cls
close
logclose

read "b_ex04_task_in.rec"

cls
* Manually inspect for typing errors and correct
if substr(idcode,1,4)="x- 2" then \
    idcode=substr(idcode,1,2)+substr(idcode,4,11)
if substr(idcode,7,1)<>"-" then \
    idcode=substr(idcode,1,6)+"-"+substr(idcode,7,6)

cls
* Exclude records without valid identifier
* <= they cannot be used
gen i hasid=1
if lower(substr(idcode, 1,3))="xxx" then hasid=0
if lower(substr(idcode,11,3))="xxx" then hasid=0
select hasid=1
drop hasid

cls
* freq mm /m
define mmn ##
mmn=99
mmn=integer(mm) if mm<>"Unknown"
drop mm
rename mmn to mm
label mm "Month of treatment"
```

```

cls
* Create text month with leading zero
* to allow correct alphabetical sorting
gen s(2) mmxt=mm
if mm<10 then mmxt="0"+mm
select mm<>99

cls
* freq country
define countrn #
if country="A" then countrn=1
if country="B" then countrn=2
if country="C" then countrn=3
if country="D" then countrn=4
if country="E" then countrn=5
if country="F" then countrn=6
if country="G" then countrn=7
if country="H" then countrn=8
drop country
rename countrn to country
label country "Name of country / jurisdiction"
labelvalue country /1="Jurisdiction A"
labelvalue country /2="Jurisdiction B"
labelvalue country /3="Jurisdiction C"
labelvalue country /4="Jurisdiction D"
labelvalue country /5="Jurisdiction E"
labelvalue country /6="Jurisdiction F"
labelvalue country /7="Jurisdiction G"
labelvalue country /8="Jurisdiction H"

cls
* Create derived identifiers
gen s(16) idcoden=country+substr(idcode,2,12)+"-"+mmxt
drop idcode2 idcode
rename idcoden to idcode2
label idcode2 "Country-patient-month identifier"
cls
gen s(13) idcode=substr(idcode2,1,13)
label idcode "Country-patient identifier"
cls
gen s(11) idcode0=substr(idcode2,3,11)
label idcode0 "Patient identifier"

label spnr "Specimen number"

cls
* Reduce dataset size for faster processing
keep idcode0 idcode idcode2 country mm spnr \
    h02 h10 h50 inhlevel katg inha inhmol inhmolsum

cls
* Identify multiple specimens per patient-month
sort idcode2 mm
gen i specseq=1
if idcode2=idcode2[_n-1] then specseq=specseq[_n-1]+1

```

```

* freq specseq
* =>
*      N
* 1      582
* 2      26
* 3      5
* 4      4
* 5      2
* 6      2
* Total 621
* => Up to 6 specimens for 1 patient in 1 month

savedata "temp_01.rec" /replace

*****
* 3) Recode to priority on first of multiple specimens
* => Isoniazid

* Hierarchy (code numerically from 0 to highest):
*  High level resistance > low level resistance
*  Resistant > Susceptible
*  Susceptible > No result

cls
close
logclose

read "temp_01.rec"

cls
define h02n #
h02n=0
if h02="S" then h02n=1
if h02="R" then h02n=2
if idcode2[_n+1]=idcode2 and h02n[_n+1]>h02n then h02n=h02n[_n+1]
if idcode2[_n+2]=idcode2 and h02n[_n+2]>h02n then h02n=h02n[_n+2]
if idcode2[_n+3]=idcode2 and h02n[_n+3]>h02n then h02n=h02n[_n+3]
if idcode2[_n+4]=idcode2 and h02n[_n+4]>h02n then h02n=h02n[_n+4]
if idcode2[_n+5]=idcode2 and h02n[_n+5]>h02n then h02n=h02n[_n+5]
drop h02
rename h02n to h02
label h02 "INH result at 0.2 mg/L"

cls
define h10n #
h10n=0
if h10="S" then h10n=1
if h10="R" then h10n=2
if idcode2[_n+1]=idcode2 and h10n[_n+1]>h10n then h10n=h10n[_n+1]
if idcode2[_n+2]=idcode2 and h10n[_n+2]>h10n then h10n=h10n[_n+2]
if idcode2[_n+3]=idcode2 and h10n[_n+3]>h10n then h10n=h10n[_n+3]
if idcode2[_n+4]=idcode2 and h10n[_n+4]>h10n then h10n=h10n[_n+4]
if idcode2[_n+5]=idcode2 and h10n[_n+5]>h10n then h10n=h10n[_n+5]
drop h10
rename h10n to h10

```

```

label h10 "INH result at 1.0 mg/L"

cls
define h50n #
h50n=0
if h50="S" then h50n=1
if h50="R" then h50n=2
if idcode2[_n+1]=idcode2 and h50n[_n+1]>h50n then h50n=h50n[_n+1]
if idcode2[_n+2]=idcode2 and h50n[_n+2]>h50n then h50n=h50n[_n+2]
if idcode2[_n+3]=idcode2 and h50n[_n+3]>h50n then h50n=h50n[_n+3]
if idcode2[_n+4]=idcode2 and h50n[_n+4]>h50n then h50n=h50n[_n+4]
if idcode2[_n+5]=idcode2 and h50n[_n+5]>h50n then h50n=h50n[_n+5]
drop h50
rename h50n to h50
label h50 "INH result at 5.0 mg/L"

labelvalue h02-h50 /0="No result"
labelvalue h02-h50 /1="Susceptible"
labelvalue h02-h50 /2="Resistant"

cls
define inhleveln #
inhleveln=0
if inhlevel="-" then inhleveln=1 // Susceptible!
if inhlevel="S" then inhleveln=1 // Susceptible!
if inhlevel="H0,2" then inhleveln=2
if inhlevel="H1" then inhleveln=3
if inhlevel="H5" then inhleveln=4
if idcode2[_n+1]=idcode2 and inhleveln[_n+1]>inhleveln then
inhleveln=inhleveln[_n+1]
if idcode2[_n+2]=idcode2 and inhleveln[_n+2]>inhleveln then
inhleveln=inhleveln[_n+2]
if idcode2[_n+3]=idcode2 and inhleveln[_n+3]>inhleveln then
inhleveln=inhleveln[_n+3]
if idcode2[_n+4]=idcode2 and inhleveln[_n+4]>inhleveln then
inhleveln=inhleveln[_n+4]
if idcode2[_n+5]=idcode2 and inhleveln[_n+5]>inhleveln then
inhleveln=inhleveln[_n+5]
drop inhlevel
rename inhleveln to inhlevel
label inhlevel "INH result summary level"
labelvalue inhlevel /0="No result"
labelvalue inhlevel /1="Susceptible"
labelvalue inhlevel /2="Resistant at 0.2 mg/L"
labelvalue inhlevel /3="Resistant at 1.0 mg/L"
labelvalue inhlevel /4="Resistant at 5.0 mg/L"

cls
define katgn #
katgn=0
if katg="-" then katgn=1
if lower(substr(katg,1,4))="wild" then katgn=1
if lower(substr(katg,1,3))="del" then katgn=2
if (substr(katg,1,3)) = "315" then katgn=2
if lower(substr(katg,1,3))="mix" then katgn=2
if lower(substr(katg,1,3))="mut" then katgn=2

```

```

if idcode2[_n+1]=idcode2 and katgn[_n+1]>katgn then katgn=katgn[_n+1]
if idcode2[_n+2]=idcode2 and katgn[_n+2]>katgn then katgn=katgn[_n+2]
if idcode2[_n+3]=idcode2 and katgn[_n+3]>katgn then katgn=katgn[_n+3]
if idcode2[_n+4]=idcode2 and katgn[_n+4]>katgn then katgn=katgn[_n+4]
if idcode2[_n+5]=idcode2 and katgn[_n+5]>katgn then katgn=katgn[_n+5]
drop katg
rename katgn to katg
label katg "INH katG result"
labelvalue katg /0="No result"
labelvalue katg /1="No mutation"
labelvalue katg /2="Mutation"

cls
define inhan #
inhan=0
if lower(substr(inha,11,4))="wild" then inhan=1
if lower(substr(inha,15,4))="wild" then inhan=1
if lower(substr(inha, 1,1))="c" then inhan=2
if lower(substr(inha, 1,1))="g" then inhan=2
if lower(substr(inha, 1,3))="del" then inhan=2
if lower(substr(inha, 1,3))="mut" then inhan=2
if lower(substr(inha,11,3))="mut" then inhan=2
if idcode2[_n+1]=idcode2 and inhan[_n+1]>inhan then inhan=inhan[_n+1]
if idcode2[_n+2]=idcode2 and inhan[_n+2]>inhan then inhan=inhan[_n+2]
if idcode2[_n+3]=idcode2 and inhan[_n+3]>inhan then inhan=inhan[_n+3]
if idcode2[_n+4]=idcode2 and inhan[_n+4]>inhan then inhan=inhan[_n+4]
if idcode2[_n+5]=idcode2 and inhan[_n+5]>inhan then inhan=inhan[_n+5]
drop inha
rename inhan to inha
label inha "INH inha result"
labelvalue inha /0="No result"
labelvalue inha /1="No mutation"
labelvalue inha /2="Mutation"

cls
define inhmoln #
inhmoln=0
if lower(substr(inhmol,1,1))="s" then inhmoln=1
if lower(substr(inhmol,1,1))="s?" then inhmoln=0
if lower(substr(inhmol,1,2))="rb" then inhmoln=2
if lower(substr(inhmol,1,2))="rh" then inhmoln=3
if idcode2[_n+1]=idcode2 and inhmoln[_n+1]>inhmoln then
inhmoln=inhmoln[_n+1]
if idcode2[_n+2]=idcode2 and inhmoln[_n+2]>inhmoln then
inhmoln=inhmoln[_n+2]
if idcode2[_n+3]=idcode2 and inhmoln[_n+3]>inhmoln then
inhmoln=inhmoln[_n+3]
if idcode2[_n+4]=idcode2 and inhmoln[_n+4]>inhmoln then
inhmoln=inhmoln[_n+4]
if idcode2[_n+5]=idcode2 and inhmoln[_n+5]>inhmoln then
inhmoln=inhmoln[_n+5]
drop inhmol
rename inhmoln to inhmol
label inhmol "INH inhmol result"
labelvalue inhmol /0="No result"
labelvalue inhmol /1="Susceptible"

```

```

labelvalue inhmol /2="Low level resistance"
labelvalue inhmol /3="High level resistance"

cls
define inhmolsumn #
inhmolsumn=0
if lower(substr(inhmolsum,1,1))="-" then inhmolsumn=1
if lower(substr(inhmolsum,1,2))="hb" then inhmolsumn=2
if lower(substr(inhmolsum,1,2))="hh" then inhmolsumn=3
if idcode2[_n+1]=idcode2 and inhmolsumn[_n+1]>inhmolsumn then
  inhmolsumn=inhmolsumn[_n+1]
if idcode2[_n+2]=idcode2 and inhmolsumn[_n+2]>inhmolsumn then
  inhmolsumn=inhmolsumn[_n+2]
if idcode2[_n+3]=idcode2 and inhmolsumn[_n+3]>inhmolsumn then
  inhmolsumn=inhmolsumn[_n+3]
if idcode2[_n+4]=idcode2 and inhmolsumn[_n+4]>inhmolsumn then
  inhmolsumn=inhmolsumn[_n+4]
if idcode2[_n+5]=idcode2 and inhmolsumn[_n+5]>inhmolsumn then
  inhmolsumn=inhmolsumn[_n+5]
drop inhmolsum
rename inhmolsumn to inhmolsum
label inhmolsum "INH inhmolsum result"
labelvalue inhmolsum /0="No result"
labelvalue inhmolsum /1="Susceptible"
labelvalue inhmolsum /2="Low level resistance"
labelvalue inhmolsum /3="High level resistance"

savedata "temp_02.rec" /replace

*****
* 4) Recode to priority on first of multiple specimens
* => Next drug

cls
close
logclose
read "temp_02.rec"

* Do this for each drug
* At the end of the process, the hierarchically
* highest value will be in the sequentially
* first SPNR if there are multiple SPNRs
* => keeping only the first SPNR suffices:

select specseq=1

* Create a single result for isoniazid

define inh #
inh=9
if inhlevel=0 then inh=inhmolsum
if inhlevel=1 then inh=1
if inhlevel=2 then inh=2
if inhlevel=3 then inh=2
if inhlevel=4 then inh=3

```

```

if inhmolsum=3 then inh=3
if inha=2 and katg=2 then inh=3 // already done, but make sure
label inh "Resistance to INH"
labelvalue inh /0="No result"
labelvalue inh /1="Susceptible"
labelvalue inh /2="Low-level resistance"
labelvalue inh /3="High-level resistance"

savedata "temp_03.rec" /replace

*****
* 5) Any other recoding

cls
close
logclose
read "temp_03.rec"

* xxx

drop spnr specseq idcode0 idcode2
drop h02 h10 h50 inhlevel
drop katg inha inhmol inhmolsum

sort country idcode mm

savedata "b_ex04_task_out.rec" /replace

*****
* Clean up and erase temporary session files

set echo=off
close
define yesno # global
yesno=?Delete all temporary files: 1=yes 0=no?
imif yesno=1 then
cls
type "Be patient ... you will be alerted upon completion" /h2
erase "temp_01.chk"
erase "temp_01.rec"
erase "temp_02.chk"
erase "temp_02.rec"
erase "temp_03.chk"
erase "temp_03.rec"
cls
type "All temporary files erased" /h2
else
type "All temporary files retained" /h2
endif
set echo=on

*****
cls

```

```
close  
read "b_ex04_task_out.rec"
```