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.

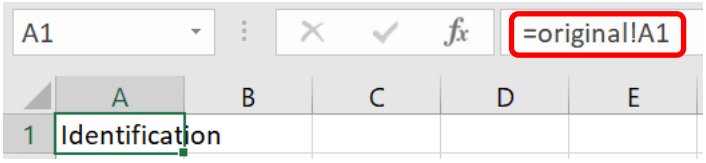
Formatting the spreadsheet correctly

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:



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

	A	B	C	D	E	F	G
1	Identificat	sex	date of bir	marital sta	Living arra	housing	surface in sqm
2	A	m	0	married	apartmen	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	sqmeter

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:

```
id  sex dob marital living  sqmeter
A   m   01-01-1900  married apartment lease 999
B   F   20-02-2001  divorced    Living alone    80
C   f   15-12-1995  widowed house owner 150
D   m   26-07-1985  cohabitating    appartment  105
E   M   24-03-2003  married own house   180
F   x   04-04-1979  married leasing an apartment    95
G   f   17-02-1998  not married appartment  110
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:

```
id→sex→dob→marital→living→sqmeter
A→m→01-01-1900→married→apartment·lease→999
B→F→20-02-2001→divorced→Living·alone→80
C→f→15-12-1995→widowed→house·owner→150
D→m→26-07-1985→cohabitating→appartment→105
E→M→24-03-2003→married→own·house→180
F→x→04-04-1979→married→leasing·an·apartment→95
G→f→17-02-1998→not·married→appartment→110
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	appartment	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	appartment	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:

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

When using a comma as decimal separator, we get:

```
id;sex;dob;marital;living;sqmeter
A;m;01-01-1900;married;apartment lease;999
B;F;20-02-2001;divorced;Living alone;80
C;f;15-12-1995;widowed;house owner;150
D;m;26-07-1985;cohabitating;appartment;105
E;M;24-03-2003;married;own house;180
F;x;04-04-1979;married;leasing an apartment;95
G;f;17-02-1998;not married;appartment;110
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-limited, 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 that 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:

- ***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!***
- ***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.***
- ***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.***
- ***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.***