Empirical labour market analysis

Workshop material, 13th version

Issued: February 10, 2020

bence.papp@uni-corvinus.hu

Content

Content ..... **i**
Aim and review of the course ..... **ii**
Exercise 1 – Getting started with STATA and basic statistics **1**
Importing, editing and basic attributes of the current database 1
Labelling variables and values, saving 3
Writing to the output, built-in functions 5
Generating and replacing new variables, conditioning and comparing 5
Query statistics, line fitting, regression, ANOVA 6
Appending and merging with other databases 8
Sorting and conditioning 10
Writing do files (batch or script files) 11
Exercise 2 – Cross-sectional analysis of the labour market **12**
Labour Force Survey 12
Generating age groups 14
Indicator for employment, comparing with CSO's version 15
Indicator for unemployment, comparing with CSO's version 16
Indicator for activity and working age 17
Weighing for inferring from the sample 18
The basic rates: activity, employment, unemployment 19
Visualizing our results 20
Exercise 3 – Time series analysis of the labour market **22**
Creating new database holding results of calculations 22
Writing cycles – the for loop 23
Generating the time series 24
Application 1 – Report generator for unemployment rate **27**
Getting acquainted with the calculating do file 27
The controlling and invoking VBA code 27
Application 2 – Micro simulation **34**
Formatting, describing and exploring the source databases 34
Process the data, the core of the simulator 36

Aim and review of the course

The mission of this course is to give insight to basics of two main things. The first is **measurement** of the labour market and the second is the **analysis** based on the data gained through the measurement. Measurement means not only the proper gauge and scale but the knowledge and the understanding of the subject of the measurement. As measurement covers more than its name suggests so does analysis. Analysis comprises model application to the segment of reality under examination, raising adequate questions, choosing suitable instrument — mathematical or econometrical tools — and the correct interpretation of the results.

As economists are expected to master the skills and capacity of modelling, measurement, analysis, estimation, interpretation, algorithmization and the usage of certain software so it is in association with every profession who are up to scrupulously work on a field adjacent to economics. Every attendee of this workshop seminar is going to get closer to fulfil this requirement while acquiring the skills to handle a **prominent** and **recognized software**, STATA.

The essence and main landmarks of this course can be circumscribed with the element of the table of contents of this document. It starts with making the students familiar with STATA while collecting data via a survey and repeating some basic econometric tools in practice. Exercises will be done on a variety of high quality databases of International Labour Organization, Eurostat and European Social Survey. After the introductory part the course continues with an insight into the **Hungarian National Labour Force Survey** and its usage for cross-sectional analysis of the labour market. Beside cross-sectional analysis time series analysis shall not be abandoned. For this reason was the third exercise composed. Ending the course of workshops there are two applications which present the applicability of STATA for more complex tasks with respect to labour issues.

Exercise 1 – Getting started with STATA and basic statistics

We will collect data through a survey among the students of the group. Then we are going to format the database, merge the actual database with the results of a former survey, and calculate basic statistics and get to know the basics of graphical display.

Importing, editing and basic attributes of the current database

```
cd "S:\Empirical\WD"
```

Sets the working directory (WD) of STATA to a certain folder. Hereinafter there will be no need to give the full path to STATA if you want to work with files within this folder.

```
dir
```

List the content of the working directory.

```
clear
```

Clear the memory of STATA. All observations and variables will be lost. But this command is crucial if you want to load new dataset, because that could be done only if there is nothing in the memory.

```
insheet using questionnaire.csv,
delim(";")
```

Importing data from comma separated value format file. The option `delim` determines the value separator (the character between cells) in the file being imported to Stata. In the example a semi-colon.

```
help insheet
```

Right at the beginning we must get to know how to access the manual of Stata and its commands. Let us have a look what kind of options does the `insheet` command have.

```
insheet using questionnaire.csv,
clear
```

After comma you can state **options** for the commands. In this case `insheet` has a `clear` named option, which will clear memory before importing.

```
save questionnaire.dta
```

Save the data in Stata's memory onto the hard drive more specifically into the WD. The file will have Stata format.

```
use questionnaire.dta
```

In advance let us know how to load data from a native Stata data file into the memory. Type `use` and as the **parameter** of the command give the name of the Stata file.

```
set more off
```

Setting the `more` parameter off, which means that the output of a command shows up all at once in one block and not divided into pages.

```
describe
describe name
```

Describing the dataset in the memory. Basic information about variables and number of observations.

```
d
```

The short version of the command `describe`. Almost every command has abbreviation. Let us look into the help.

```
help describe
help
```

Opens the help of stata

```
d, short
```

Displays only the attributes of the database. Description of the variables is not included.

```
d, simple
```

Only the variable names.

```
inspect hg
codebook hg
```

Summarized information about a variable and its content. The two commands give similar output but there are also differences between them. Try both with the same numerical and string type variables!

```
list
```

Listing the whole database. In case of many variables and immense observations the output will not be easily human-readable. What information do we have here?

```
edit
```

Opens the database in an independent new window. Excel-like look and feel. You can copy data from Excel directly into this table.

Labelling variables and values, saving

```
label variable name "Full name"
```

It gives the label "Full name" to the variable named "name".
Variable names are typically short and terse. Labels are descriptive, informative. In commands you refer to the variable with its name (not with the label).

Press the button **PageUp** and you can move backward in the command history.

```
label variable neptun "Neptun code"

label variable sex "Gender"

label variable hg "Body height"

label variable pl "Palm length"

label variable pw "Palm width"

label variable by "Year of birth"

label variable bm "Month of birth"

label variable bd "Day of birth"

label variable paddr "Permanent
address"

label variable dist "Paddr distance
from Bp"

label variable hschool "Type of high
school"

label variable res "Residence while
learning"

label variable ttime "Travel time"

label variable wgroup "Workshop
group"

label variable program "Degree
course"
```

Giving label all of the variables.

You can copy the variable name from the **variable window** if you click on it.

You can select the former (or any previous) command also from the **command history window** by clicking on one of them.

```
d

d p*
```

Exploring the new labels of the variables.

```
d p?
```

You can use wildcard characters (e.g. asterisk and question mark).

```
tabulate hschool
```

Frequency table on type of high school.

The categories are codes. We want to label the categories in order to make them more informative.

```
label define l_sex 0 "male" 1
"female"

label define l_hschool 1 "grm.sch."
2 "ec.sec.voc.sch." 3
"oth.sec.voc.sch." 4 "other"

label define l_res 1 "Pest" 2 "Buda"
3 "Pest county" 4 "Alföld(east)" 5
"Dunántúl(west)" 6 "Abroad"
```

Define a value label sets.

```
label values sex l_sex

label values hschool l_hschool

label values res l_res
```

Assign the value label sets to the proper variables.

```
tab hschool

tab sex

tab l_res
```

Now the categories are already labelled.

```
label dir

label list l_res

label list _all

labelbook l_res

labelbook
```

Information about value label sets.

```
save wgroup1, replace

dir
```

Saves the database in STATA format. `Replace` means: overwrite existing file if necessary.

Dir reports that the new file came into being within the WD.

```
label drop l_sex
```

Deleting a value label set (l_sex). It is inevitable in case you want to edit or rewrite a label set. Stata lets you modify a set only in a complicated way; it is easier to drop (erase) the old one and create a new one.

Writing to the output, built-in functions

```
display "Hello"
```

Displays a string on the output window.

```
di 3
```

Displays a number.

```
di 3+8
di sin(2*_pi)
```

Calculates the operation and displays the result.

```
di "3+8"
di "sin(2*_pi)"
```

Displays the operation as a string.

```
di mdy(9,1,2019)
di mdy(1,1,1960)
di mdy(12,30,1959)
```

`mdy` returns with the number of days since 1 January 1960 until day given as argument.

The result, obviously, could be negative as well.

Generating and replacing new variables, conditioning and comparing

```
generate new_var = 0
generate byte new_var2 = 0
generate float new_var3 = 0.0
```

Generates a new variable and initializes it with the value 0. Initialization is compulsory!

Optionally you can prescribe the **type** of variable.

```
gen birth_date = mdy(bm,bd,by)
label var birth_date "Date of birth"
```

Generates a new variable for every observation and sets its initial value to the complete birth date formed from the month, day and year.

```
list birth_date
```

Lists the variables given in the argument list for all observations.

```
format birth_date %d
```

Change the display format of the birth_date variable to date format. First UK style.

```
format birth_date %dM.D,CY
```

Then USA style.

```
format birth_date %dCY-M-D
```

Lastly Hungarian style.

```
list birth_date
```

```
gen s_age_2019 = 2019-by
```

Generates simple age proxy of every observation's age in 2019.

```
gen age_2019 = s_age_2019
```

Generates precise age derived from the age proxy.

```
replace age_2019 = s_age_2019-1 if mdy(bm,bd,2019) > mdy(9,1,2019)
```

Decrement age_2019 by one if the person's birthday is not over on the current day.

`replace` is for replacing the values of an existing variable. **You cannot do this with `generate`.** `generate` is for generating non-existing variables. If you insist using generate you should delete the variable first with the `drop` command.

`if` clause is for conditioning a command. Here the `replace` command. It stipulates that replacement should be carried out only if the observation's birthday is not over on the current day.

```
compare s_age_2019 age_2019
```

Compares two variables for every observation.

```
save wgroup1, replace
```

Saving for safety reasons.

Query statistics, line fitting, regression, ANOVA

```
summarize hg
```

```
summarize hg, detail
```

Getting basic statistics about the body height. With the `detail` option we can request more profound statistics.

```
_pctile hg, n(100)
```

```
di r(r34)
```

Quantiles more profoundly. The first command calculates quantiles with the cardinality given by the `n()` option (this case 100, so the quantiles will be percentiles). The second command displays one of the results stored in the `r vector` (results' vector) — in this case the 34th percentile.

```
tabulate sex
```

Plain frequency table.

```
tabulate sex, sum(hg)
```

Summary statistics about body height broken down by gender.

```
tabulate res sex
```

```
tab res sex, row
```

```
tab res sex, row nofreq
```

```
tab res sex, col
```

```
tab res sex, cell
```

Two-way cross table (contingency table). The first variable gives the values for the vertical axis and the second variable gives the values for the horizontal axis.

Adds row relative frequencies.

Row relative frequencies without absolute frequencies.

Add column relative frequencies.

Add cell relative frequencies.

```
tab res sex, sum(ttime)
```

Combination of contingency table and summary statistics.

```
histogram hg
```

Calculates and display histogram over height.

```
histogram hg, bin(3)
```

Histogram with forced number of intervals.

```
histogram hg, bin(3)
xlabel(150(20)210)
```

Formatting the y-axis scale labels.

```
gen palmsur = pl*pw
```

Generate the surface of palm for everyone.

```
reg palmsur hg sex
```

Make regression for palm's surface with height and sex dummy.

Note the R squared parameter.

```
gen hg_sq = hg^2
```

Generate the square of body height.

```
reg palmsur hg_sq sex
```

Now make the regression again but this time with the square of the height.

R squared is better, the model fits better.

```
graph twoway (lfit palmsur hg)
(scatter palmsur hg), name(g1)
```

Visualizing the formerly revealed relationship.

Linear and quadratic fitting.

```
graph twoway (qfit palmsur hg_sq)
(scatter palmsur hg_sq), name(g2)
```

```
graph dir
```

List all graphs which are in the memory.

```
graph display g1
```

Displaying certain graph.

```
graph display g2
```

```
graph drop g1 g2
```

Deleting graphs.

```
one hg sex, tab
```

One-way ANOVA on height by sex.

According to F statistics there is significant difference in height by gender.

Appending and merging with other databases

```
describe using wgroup2
```

Describe database which is not in the memory. Let us memorize the number of observations.

```
d
```

Counting the number of observations in the actual database. Note it.

```
append using wgroup2
```

Append new database to the memory.

```
d
```

Count the number of observation in the resultant database. It must be the sum of the former two numbers.

```
tab neptun
```

Tabulate by Neptun code. This code is unique, and there is two observations with the same code.

```
list name neptun wgroup if neptun == "IRYX2F"
```

It may be the teacher who is twice among the observations. Listing with restriction to a certain neptun code. Neptun code is a string so you have to put it between double quotation marks.

```
use wgroup1, clear
```

Load in the data of our group again.

```
merge 1:1 neptun using wgroup2
```

Merge the database in the memory (master) with the database currently loaded (using) by the variable named neptun.

```
list name neptun wgroup if name == "Papp Bence"
```

Now there is only one person to one Neptun code.

```
tab _merge
```

A new variable named _merge has been generated. It informs us about the observations' former location.

If you want to process new merging it will be unsuccessful until you delete or rename this variable, because every merge wants to generate new `_merge` and it cannot be done unless the former is vanished.

```
rename _merge location
```

Give a more apt name to _merge.

```
list name location
```

Who was in the master, who was in the using database and who was in both?

```
list name wgroup
```

The . (dot) values means **missing observation**. **Infelicitously it also means infinite.** As a consequence you have to be aware of this when forming conditions in an if clause.

Let us replace one's body height with . , and then list the names of those who are taller than 170 cm.

```
list name if wgroup == .
```

Filter out the missing observations.

```
list name if mi(wgroup)
```

Either with the . value or with built-in function.

```
label data "Data of two students' groups (year of 2020 and 2019)"
```

Adds caption to the database.

```
save wrgoups, replace
```

Sorting and conditioning

```
list name hg
```

Examine the effect of the `sort` command.

```
sort hg
```

It produces an ascending order.

```
list name hg
```

```
gsort hg
```

Ascending and descending ordering. Nota bene the usage of the minus sign.

```
gsort -hg
```

```
list name hg in 1/5
```

The first 5 (tallest) member of the group.

```
list name hg in -3/L
```

The last 3 (shortest) member of the group.

```
list name if ttime < 10 & hschool == 1 & (res == 1 | res == 2)
```

List the names of members who travel less than 10 minutes to the university and who attended grammar school and who live during education either in Pest or Buda.

Logical relations (operators):

AND	-	&
OR	-	|
NOT	-	!
EQUAL	-	==
(nota bene: double equal sign is for equality checking, single equal sign is for assignment!)		
NOT EQUAL	-	!=
GREATER THAN OR EQUAL	-	>=
LESS THAN OR EQUAL	-	<=

```
sum hg if sex == 1
```

What are the average heights of females and males?

```
sum hg if sex == 0
```

Writing do files (batch or script files)

```
doedit
```

Open the do editor in a new window. We will work here with finishing the do file.

```
* Single line comment type 1
// Single line comment type 2
/* Comment line 1
Comment line 2 */
```

You can add comment to the do file. These lines will not be executed. They are mainly for documentation.

```
clear
cd "S:\Empirical\WD"
use wgroups
```

Initial commands of the do file.

```
log using log2019.log, replace text
```

Open log file to write the following results into it.

```
tab sex
sum hg ttime
```

Make computations.

```
log close
```

Closing the log file.

Save the do file with the name of "practice_01.do".

```
do practice_01.do
```

Revert to the command window and type the `do` command with the filename of our do file. It will execute it.

Check out the log file in the WD.

Exercise 2 – Cross-sectional analysis of the labour market

Using the Hungarian Labour Force Survey we will have a brief insight to the results of comprehensive data acquisition and catch a glimpse of its method. We will generate the basic and most important indicators and categories for analysis.

Labour Force Survey

```
use LFS(66th wave).dta
```

Open the 66th wave of the Hungarian LFS. Wave means a quarter, and the first quarter was 1992Q1. Let us calculate and then check which year and which quarter do we have at hand!

```
d county area hid person
```

Let us peruse the database!

```
d county-person
```

Start with the first four variables. We can do it in the way we have already learnt, or with the hyphen operator (-) which is for setting a range of consecutive variables.

In this example the second `describe` command has every variable from county to person as parameters.

The range operator (hyphen) could be used for other command as well (e.g `tabulate` or `summarize`).

```
d, s
```

How many people are in the sample?

```
count
```

The `count` command is also good for counting every observation, and more …

```
cou if kor>14 & kor<75
cou if kor>74
cou if kor<15
```

We can use `count` with conditions.

```
list hid person in 1/20
```

Let us have a deeper look into the database.

`hid` uniquely identifies households and `person` the individuals within.

```
codebook hid person
```

There is no missing observation for household identifier and person number.

```
sort hid person

gen myid = 1000000 + _n
```

In association with the household identifier one could be curious about how to generate a series of unique numbers.

Sort data firstly by household identifier and secondly by person number within the household.

We can use special variable `_n` that holds the row number in which the observation under calculation can be found.

Generating unique identifiers starting from 1000000 for each observation.

```
replace myid = 1 in 13
replace myid = wei[13] in 12
```

If we are talking about generating by row numbers, here is how to access variable values by row number. Using the `square brackets` or the `in` qualifier are the tools for this task.

Taking the second `replace` command we address the 12th value of `myid` and replace it with the 13th value of `wei`.

```
tab w1hour
tab absent_c
tab mionem
tab search_b
tab avail
```

And there are the key variables which make us able to identify the labour market statuses.

Who worked at least 1 hour in the previous week? The question is set only to people in working age.

```
d weight
```

Weight variable to be able to estimate for the population from the sample.

```
d educ_d educH

tab educ_d educH

d absent_c absentH

tab absent_c absentH
```

Harmonized variables to ease comparison between countries.

```
tab1 hcitiz magyar allev

tab Magyar, missing

tab allev, missing
```

Let us check the data concerning the citizenship of the observed person.

`tab1` makes tables according to each variable in its argument respectively.

There could be missing observations because of the jump in the questionnaire.

The `missing` option make `tab` count also the missing values and put into the table.

Use the `display` command as a calculator to check whether the values add up the entire dataset.

Generating age groups

```
// Age group generator do file.
```

Start a do file with this header.

```
gen agegroup = 0
```

Generate a polychotomous variable for coding the age groups.

```
replace agegroup = 1 if age > 14
replace agegroup = 2 if age > 19
replace agegroup = 3 if age > 24
replace agegroup = 4 if age > 29
replace agegroup = 5 if age > 39
replace agegroup = 6 if age > 49
replace agegroup = 7 if age > 59
replace agegroup = 8 if age > 69
replace agegroup = 9 if age > 74
```

The group number 1 gathers everybody who is older than 14 years and younger than 20.

The group number 0 consists of the people who are younger than 15.

Everybody who is elder than 74 are convened into the group number 9.

```
label var agegroup "Age groups"
label define l_agegr 0 "0-14" 1 "15-
19" 2 "20-24" 3 "25-29" 4 "30-39" 5
"40-49" 6 "50-59" 7 "60-69" 8 "70-
74" 9 "75 or more"
label values agegroup l_agegr
```

Labelling the age group coding variable.

```
// End of do file
```

Save and close the do file.

```
gen agegroup2 =
recode(age,14,19,24,29,39,49,59,69,
74,75)
```

The easier way for the same task with the help of a built-in function. As the arguments of `recode` you have to give the variable which you want to stratify and the upper limit of the nascent groups.

```
tab1 agegr*
```

Review the two categorizations.

Indicator for employment, comparing with CSO's version

```
gen employed=0 if kor>14 & kor<75
```

Generate a new variable but only if the person's age is greater than 14 and less than 75 (he or she is in the working age group). Older and younger people will get a missing value (. — the dot).

This variable will code whether a person is employed or not.

```
replace employed=1 if w1hour==1 |
absent_c == 1
```

Replace the employed named variable with 1 if the person has worked at least 1 hour in the last week or was temporarily away from his or her job.

```
label var employed "Employed or not
employed - YES/NO dichotomous
variable"
```

Give a proper label for our new variable.

```
label dir

label list yn yesno
```

Looking for value label set.

```
label values employed yn
```

"yn" will do perfectly.

```
tab employed
```

Check out what have we done.

```
compare employed csoe1
```

Compares our employed variable with the one which was generated by the CSO (Central Statistical Office).

There is a slight difference: there are few people who are employed according to our indicator and not by the official. Let us find out the reason!

```
list mionem employed csoe1 if
employed>csoe1
```

The variable mionem informs us about whether the person are away from her/his work more than 3 month and gets at least the half of her/his salary or not.

If not than the person cannot be classified into the employed class. It is true for every 8 person in question. Let us modify our variable!

```
d mionem

label list mionemen
```

Query the label value set's name belonging to mionem. Then find out the value which belongs to "away more than 3 month and gets less than half of salary". We have to have it because if conditions can be composed only with values (not with labels).

```
tab mionem, nolab
```

Alternative way for revealing the values behind the labels. Tabulates without labels.

```
numlabel mionemen, add

tab mionem

numlabel mionemen, remove

tab mionem
```

Other solution for disclosing the numbers assigned to the value labels.

You can add and remove the values from the labels.

```
replace employed = 0 if mionem == 3
```

The needed modification.

```
compare employed csoe1
```

The comparison reports that there is no more divergence.

Indicator for unemployment, comparing with CSO's version

```
tab1 search_b meth* avail
```

Tabulate the key variables for defining the state of unemployment.

To be unemployed one shall not be occupied (I.), shall seek job (II.) and shall be able to take the job (III.). If someone does not seek job but has already found a job and will begin in short time the II. condition does not bound.

Not every search method counts as accepted method.

```
gen unemployed = 0 if kor>14 &
kor<75
```

Generate our unemployed variable for the working age class.

```
replace unemployed = 1 if (search_b
== 1 | search_b == 2) & avail == 1
```

Replace it according to the rule above.

```
compare unemployed csou
```

Comparing with the CSO's version. There is again a minuscule divergence. There are some unemployed according to our computation who are not considered as unemployed by the CSO. Let us search for the roots again!

```
drop unemployed

gen unemployed = 0 if kor>14 &
kor<75

replace unemployed = 1 if (
(search_b == 1  & (metha == 1 |
methb == 1 | methc ==1 | methd ==1 |
methe ==1 | methf ==1 | methi ==1 |
methj ==1 | methk ==1 | methm == 1
)) | search_b == 2) & avail == 1
```

Recreate the unemployed variable, but now we will filter out certain search methods (g, h and l).

```
compare unemployed csou
```

Comparison for checking our compliance.

```
label var unemployed "Unemployed or
not unemployed - YES/NO dichotomous
variable"

label values unemployed yn
```

Labelling our new variable.

```
gen UE = 0 if employed == 1 |
unemployed == 1

replace UE = 1 if unemployed == 1

label var UE "Unemployed - defined
over actives"

label values UE yn
```

Generate a second version of unemployed called UE which will fit better for calculating the unemployment rate.

The relevant difference between unemployed and UE is that the former is defined over active aged and the latter is defined over actives.

Indicator for activity and working age

```
gen active = 0 if kor>14 & kor<75
```

Generating binomial variable for actives.

```
replace active = 1 if employed == 1
| unemployed == 1

label var active "Active or not
active - YES/NO dichotomous
variable"

label values active yn
```

```
gen workage = 0

replace workage = 1 if kor>14 &
kor<75

label var workage "Working age? -
YES/NO dichotomous variable"

label values workage yn
```

Generating binomial variable for working age.

```
save "LFS66_processed", replace
```

Saving our work.

Weighing for inferring from the sample

```
d weight

codebook weight
```

The weight variable. Every observation has a weigh meaning how many other people she or he represents in the population (Hungary).

```
tab1 employed unemployed active
workage
```

The labour market statuses and their numbers for our sample.

```
tab1 employed unemployed active
workage [weight = weight]
```

The labour market statuses and their numbers' estimate for the population.

An error message appears: we shall use integer weights.

```
gen rweight=round(weight,1)
```

Generate a rounded weight derived from the original weight variable.

```
tab1 employed unemployed active
workage [weight = rweight]
```

Now it works!

```
tab county [w=rw]
```

```
tab sex [w=rw]
```

```
tab educH [w=rw]
```

```
tab agegroup [w=rw]
```

```
tab agegroup sex [w=rw], nofr row
```

```
tab employed sex [w=rw]
```

Examine our labour market from different aspect with descriptive statistics.

The basic rates: activity, employment, unemployment

```
sum active [w=weight]
```

The activity (participation) rate: the share of actives among working aged.

The mean of this variable adds up the activity rate because it was defined over the group of active aged people (everybody else has missing value for this variable) and who is active has one and who is not active got zero.

Note that the command `summarize` does not prerequire integer weights!

```
sum employed [w=wei]
```

The employment rate: the proportion of employed among the working aged.

```
sum UE [w=wei]
```

The unemployment rate: the number of unemployed divided by the number of actives.

Nota bene, we intendedly defined UE (in contrast to the variable unemployed) on the basis of active people (and not on the domain of active age people).

```
sum unemployed [w=wei]
```

```
sum workage [w=wei]
```

The mean of unemployed adds the share of unemployed people among working age population.

The mean of workage adds the proportion of the entire population who are in working age.

```
tab educH [w=wei], sum(UE)
```

Calculate the unemployment rate broken down by educational achievements.

```
tab educH [w=wei], sum(UE) nost nofr
noobs
```

The same but without the standard deviation, the frequencies and the observations.

```
tab educH, sum(UE) nost nofr noobs
```

The same without weighing. What is the reason of the difference? Somehow certain people are overrepresented in the sample.

Do the same comparison for `employed`: the weighted and unweighted frequencies!

```
tab sex [w=wei], sum(UE) nost nofr
noobs

tab agegroup [w=wei], sum(UE) nost
nofr noobs

tab county [w=wei], sum(UE) nost
nofr noobs
```

The unemployment rate grouped by the main stratifying variables.

```
tab educH sex [w=wei], sum(UE) nost
nofr noobs

tab educH sex [w=wei], sum(employed)
nost nofr noobs

tab educH sex [w=wei], sum(active)
nost nofr noobs
```

The main rates in contingency tables.

```
tabstat active employed UE [w=wei],
s(mean) by(educH)
```

All important rates in one table by educational attainment.

Visualizing our results

```
graph hbar employed [w=wei],
over(county)
```

A bar chart with the mean of the given variable (employed) grouped by counties.

```
graph hbar employed [w=wei],
over(county, sort(1))
```

Sorting the bars according to their length in ascending order.

```
graph hbar employed [w=wei],
over(county, sort((mean) UE))
```

Ordering by the unemployment rate of the counties.

```
graph hbar employed [w=wei],
over(county, sort((mean) UE) des)
```

Ordering by the unemployment rate of the counties in descending order.

```
graph bar UE [w=wei], over(agegroup,
label(angle(vertical))) over(sex)
nofill
```

Draw bars for unemployment rate by age groups and gender.

Without `nofill` the empty categories would be displayed as well.

Without the label option the labels would be displayed horizontally and will overlap each other.

```
graph bar UE [w=wei], over(agegroup,
label(angle(vertical))) over(sex,
relabel(0 "na" 1 "male" 2 "female"))
nofill
```

Add new labels for genders omitting the value from it. Shows off better.

```
graph bar UE [w=wei], over(agegroup,
label(angle(vertical))) over(sex,
relabel(0 "na" 1 "male" 2 "female"))
nofill ylabel(0(0.02)0.1
0.1(0.1)0.4, angle(horizontal)
labsize(small)) ytitle("Unemployment
rate")
```

Set the labels for the y axis so as to compare easier the shorter bars.

Add a more correct title for the value axis.

Exercise 3 – Time series analysis of the labour market

Continuing the analysis of the Hungarian labour market through the LFS we turn to another aspect namely the dynamic of phenomena. Using the concept of the main indicators learned in the previous section we will assemble time series reporting of the changes of labour market.

Creating new database holding results of calculations

```
use LFS66_processed, clear
```

Open the previously saved database.

```
graph bar employed UE active,
over(agegroup,
label(angle(vertical))) over(sex,
relabel(0 "na" 1 "male" 2 "female"))
nofill ylabel(#6,angle(horizontal)
labsize(small)) ytitle("Basic
indicators") legend(label(1
"Employment rate") label(2
"Uemployment rate") label(3
"Activity rate")) title("Hungary's
Labour Market 2008 2nd quarter")
```

Just for revision have a look at the main indicators.

```
collapse UE
```

It makes a new database with only one variable and with only one observation which is a statistic (mean by default) of the given parameter (UE).

```
list
```

See what we have gotten.

```
use LFS66_processed, clear
```

Load our source database again.

```
collapse (count) UE
```

Let us count the number of defined (non-missing) observations for UE and make a new table for it.

```
collapse (mean) UEr=UE ACr=active
(rawsum) NR=wei if workage==1
[w=wei]
```

Put unemployment rate and activity rate with new variable names UEr and ACr into a new table and in addition write as the third variable, named NR, the estimated number of people in working age.

`rawsum` ignores the weight variable.

```
collapse (mean) UEr=UE ACr=active
Er=employed (count) CUE=UE
CAC=active CE=employed (rawsum)
NR=wei if workage==1 [w=wei], by(sex
educH)
```

Extend the previous table with the number of non-missing observation in the sample and break down firstly by gender and secondly by educational attainment.

```
sum NR

di r(sum)
```

Check whether the sum of NR adds up the total number of people in working age.

After calling the `summarize` command STATA stores the sum of all observation's value belonging to the variable in the **r system matrix** under the row named sum.

Writing cycles – the for loop

```
// Do file for basic cycles

foreach i of numlist 1 2 5 43 {

di `i'

}

// End of do file
```

Open the do editor and write the for loop into it. Execution advisably should be carried out from the do editor.

The statement starts with the word **`foreach`**.

Then the name of the **cycle variable** shall be given.

It is followed by the key word **`of`**.

That follows the declaration of what kind of list do we want to put the element on which the loop will have to go through on. **`Numlist`** is a list consisting of numbers.

Then we define the list's elements.

The trunk of the loop starts with an **opening curly bracket**.

The **content of the trunk** could be any of STATA's command in any number.

The loop ends with a **closing curly bracket**.

To refer to the actual content of the cycle variable you have to put the name of the cycle variable between a special brackets (open with ` (AltGr+7) and close with ' (Shift+1)).

```
foreach i of numlist 6/12 {
```

Other types of number list's element definition.

```
di `i'

}

foreach i of numlist 1(2)11 {

di `i'

}
```

```
foreach i of varlist UE active
employed {

di "The mean of `i'"

quietly sum `i' [w=wei]

di r(mean)

}
```

A for loop for skimming through a list of variables.

A variable list is declared by the word **`varlist`**.

With `quietly` we suppress the terminal output of `summarize` and then query only the mean from the results.

Generating the time series

```
// Time series generator do file
```

Open the do editor and start a do file.

```
set more off
```

Turn out paging — do not bother ourselves with clicking next after every page in the output window.

```
foreach i of numlist 1(4)85 {

use alfs`i'_comp, clear
```

The cycle will go through LFS-s as from 1992q1 until 2013q1.

```
gen byte active=0 if korH>14 &
korH<75

replace active=1 if csoe1==1 |
csou==1

gen byte UE=0 if active==1

replace UE=1 if csou==1

gen byte m_act = active if sex == 1

gen byte f_act = active if sex == 2

gen byte m_emp = csoe1 if sex == 1
```

Generating the main labour market state indicators for every quarter.

```
gen byte f_emp = csoe1 if sex == 2

gen byte m_ue = UE if sex == 1

gen byte f_ue = UE if sex == 2
```

```
collapse active csoe1 UE m_act-f_ue
[w=weight]
```

Make a new table with the rates.

```
global date = 1992+(`i'-1)/4

gen str6 date = "$date" + "q1"

order date, before(active)
```

Jot down the current year/quarter.

Put the date variable in front of the columns.

```
if `i' > 1 append using Main_rates

save Main_rates, replace

}
```

If it is not the first year then append the data of the former year to it.

Update the output table anyway.

```
sort date
```

Sorting the table by date.

Generating serial numbers — preparing for plotting.

```
gen x=_n
```

```
global xlabel1 = ""

foreach i of numlist 1/10 {

local year = `i'+1991

global xlabel1 = `" $xlabel1 `i'
"`year'q1" "'

}

global xlabel2 = ""

foreach i of numlist 11/22 {

local year = `i'+1991

global xlabel2 = `" $xlabel2 `i'
"`year'q1" "'

}
```

Define two macros (xlabel1 and xlabel2) for holding the time labels for the graph.

Two is required because one could not hold so long text we need.

```
twoway (bar UE x, fcolor("200 200
200") ylabel(0(0.02)0.14)
barwidth(0.5)) (line f_ue x, lw(1))
(line m_ue x, lw(1)) ,
legend(label(1 "Overall ue rate")
label(2 "Women ue rate") label(3
"Men ue rate")) xlabel($xlabel1
$xlabel2, angle(vertical))
xtitle("") ytitle("Unemployment
rates")
```

Plotting the overall unemployment rate and minor rates by gender.

Application 1 – Report generator for unemployment rate

STATA could be used from other application (e.g. MS Word) as an object. It makes us room to connect to STATA from another program (e.g. VBA) without having to open it. Windows mediate between the programs; but of course it has to know about the object — you have to register STATA (execute STATA with parameter: "SataSE.exe /Regserver" or "SataSE.exe /Register"). We will write a terse but pithy VBA code which uses STATA to calculate unemployment rate from LFS and its change from year to year and incorporate these data into a report.

Getting acquainted with the calculating do file

```
use lfs$sDate.dta, clear
```

Open an LFS survey which belongs to the year/quarter given by the global macro called sDate (starting date).

sDate is set by the calling application, in our case the VBA code from MS Word (it will be discussed in the next section).

```
gen active = 1 if csou == 1 | csoe
== 1
```

Generate the binomial variable for actives.

```
sum csou if active == 1 [w=weight]
```

Calculate the unemployment rate.

```
scalar ue1 = r(mean)
```

Save the unemployment rate into the storage named ue1.

```
use lfs$eDate.dta, clear

gen active = 1 if csou == 1 | csoe
== 1

sum csou if active == 1 [w=weight]

scalar ue2 = r(mean)
```

Do the very same with the other year's (per quarter of course) LFS. The global macro for that is eDate (ending date).

Store the respective unemployment rate into ue2.

The controlling and invoking VBA code

```
Sub STATA()
```

Start a subroutine in MS Word.

```
Dim startingDate As String:
startingDate = "2008q2"
```

Declare a new string with the content regarding the starting date of the analysis.

```
Dim endingDate As String: endingDate
= "2009q2"
```

Declare a new string with the content regarding the ending date of the analysis.

```
Dim WDpath As String: WDpath =
"""g:\ENG\Teaching
materials\Automatic_Unemp_Report\"""
```

String for the WD of STATA.

```
Dim stataobj
```

Declare a new object and set it to STATA as an object.

```
Set stataobj =
CreateObject("stata.StataOLEApp")
```

Simple speaking from here on stataobj will represent STATA.

```
Dim return_code As Integer
```

Storage for the return code of different functions. If there will be problems during executions it could help us debugging.

```
Selection.WholeStory

Selection.Delete

For i = 1 To 30

ActiveDocument.Paragraphs.Add

Next i
```

Put 30 paragraphs into a bank new Word document. Later we are going to edit them.

```
With ActiveDocument.Paragraphs(1)

.Range.Text = "Report on
Unemployment"

.LineSpacing = 24

.Alignment = wdAlignParagraphCenter

.Range.Font.Bold = True

.Range.Font.Size = 24

End With
```

Edit and format the title.

```
With ActiveDocument.Paragraphs(2)

.Range.Text = "Change from " +
startingDate + " to " + endingDate

.LineSpacing = 12

.Alignment = wdAlignParagraphCenter

.Range.Font.Bold = False

.Range.Font.Italic = True

.Range.Font.Size = 16

End With
```

Edit and format the content about the time horizon of the analysis.

```
With ActiveDocument.Paragraphs(3)

.Range.Text = "Made by Kiss Pista"

.LineSpacing = 12

.Alignment = wdAlignParagraphCenter

.Range.Font.Bold = False

.Range.Font.Italic = True

.Range.Font.Size = 16

End With
```

Edit the author's paragraph.

```
With ActiveDocument.Paragraphs(4)

.Range.Text = "Date of creation: " &
Format(Now, "dd.mm.yyyy")

.LineSpacing = 50

.Alignment = wdAlignParagraphCenter

.Range.Font.Bold = False

.Range.Font.Italic = True

.Range.Font.Size = 16

End With
```

Edit the information about the date of reporting.

```
return_code = stataobj.DoCommand("cd
" + WDpath)

If return_code <> 0 Then

MsgBox "Error while setting WD."

Exit Sub

End If
```

Command STATA to change its WD to the predefined WD.

If the return code is not zero, then a problem occurred. A dialog box will show up to inform us and then terminate the execution of the subroutine.

```
return_code =
stataobj.DoCommand("global sDate = "
+ """ + startingDate + """)

If return_code <> 0 Then

MsgBox "Error while setting starting
date macro."

Exit Sub

End If
```

Setting the starting date macro for STATA. Recall that it is used in the before-mentioned do file.

```
return_code =
stataobj.DoCommand("global eDate = "
+ """ + endingDate + """)

If return_code <> 0 Then

MsgBox "Error while setting ending
date macro."

Exit Sub

End If
```

Setting the ending date macro for STATA. Recall that it was the other important message to be passed to STATA.

```
return_code = stataobj.DoCommand("do
calculations.do")

If return_code <> 0 Then

MsgBox "Error while executing do
file."

Exit Sub

End If
```

Execute the calculating do file.

```
Dim ue1 As Double

ue1 = stataobj.ScalarNumeric("ue1")
* 100
```

Get the unemployment rate for the first analysed quarter. Transform it to percentage.

```
Dim ue2 As Double

ue2 = stataobj.ScalarNumeric("ue2")
* 100
```

Get the unemployment rate for the second analysed quarter. We transform it also to percentage.

```
Dim rText(7) As String
```

Declare an array of strings to make it hold different report text alternatives to make our work more variegated.

```
Dim rTextNumber As Integer
```

Declare an integer to store the chosen number of text in the text array.

```
rText(1) = "I am very sorry to
announce that the unemployment rate
increased. The increment is " &
Format(ue2 - ue1, "#0.0") & "
percentage point (from " &
Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(2) = "I regret reporting that
the unemployment rate increased. The
increment is " & Format(ue2 - ue1,
"#0.0") & " percentage point (from "
& Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(3) = "It is to be pitied that
the unemployment rate increased. The
increment is " & Format(ue2 - ue1,
"#0.0") & " percentage point (from "
& Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(4) = "It is to the great
delight of me to report that the
unemployment rate fell. The
decrement is " & Format(ue1 - ue2,
"#0.0") & " percentage point (from "
& Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(5) = "I am pleased to jot down
that the unemployment rate fell. The
```

Fill the array with texts.

The first three is for the bad news which means that the unemployment rate lifted.

The next three is for good news which means quite the contrary in terms of unemployment rate.

The seventh is a neutral message for neutral situation (no change in unemployment rate).

All of them embeds the rates acquired from STATA.

```
decrement is " & Format(ue1 - ue2,
"#0.0") & " percentage point (from "
& Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(6) = "I am happy to assert
that the unemployment rate fell. The
decrement is " & Format(ue1 - ue2,
"#0.0") & " percentage point (from "
& Format(ue1, "#0.00") & " to " &
Format(ue2, "#0.00") & ")."

rText(7) = "Nothing changed in
association with unemployment rate.
It is and it was " + Format(ue1,
"#0.0") + " percentage point."
```

According to the fact whether the unemployment rate increased or decreased or did not changed we randomly chose a text from the string array.

```
If ue2 > ue1 Then

  Randomize

  rTextNumber = Int(3 * Rnd) + 1

ElseIf ue2 < ue1 Then

  Randomize

  rTextNumber = Int(3 * Rnd) + 4

Else

  rTextNumber = 7

End If
```

In the last edited paragraph we insert the chosen text with the results of the calculation.

```
With ActiveDocument.Paragraphs(5)

.Range.Text = rText(rTextNumber)

.LineSpacing = 12

.Alignment = wdAlignParagraphLeft

.Range.Font.Bold = False

.Range.Font.Italic = False

.Range.Font.Size = 12

End With
```

```
Set stataobj = Nothing
```

After finishing our work we have to terminate the object and free memory.

```
End Sub
```

The end of the subroutine.

Application 2 – Micro simulation

With formerly built-up databases of an imaginary labour market and with the help of STATA we will write a simulator which matches open positions with job seekers. The databases comprise 1908 corporations with 20640 open positions and 81349 people who seek after job. The corporations as well as the seekers have demands in association with the contingently latter job. Based on a simple algorithm we will assign workers to positions and estimate job vacancy and unemployment. We would be able to simulate the impact of different measures affecting this labour market.

Formatting, describing and exploring the source databases

```
clear
```

Clear the memory, just in case.

```
set more off
```

Turn off the paging behaviour of STATA. Otherwise you will have to always press the “next” button to see the next page of the output.

```
insheet using
"distances_of_cities.csv",
delim(";")
```

Import a database which holds distances between the biggest towns of Hungary.

```
mkmat budapest bkscsaba  debrecen
     eger gyr  kaposvr   kecskemt ///

     miskolc   nyregyhza pcs
     salgtarjn szeged    szkesfehrvr
     szekszrd ///

     szolnok   szombathely
     tatabnya  veszprm   zalaegerszeg
     ajka baja ///

     cegld     dunajvros rd   gdll
     gyngys    gyula     hajdbszrmny
     hdmezvsrhely ///

     kazincbarcika  kiskunflegyhza
     kiskunhalas    mosonmagyarvr
     nagykanizsa ///

     oroshza   zd   ppa  sopron
     szentes   vc, matrix(CityDist)
```

Create a matrix from these data in order to later be easily able to refer these distances.

```
matrix list CityDist
```

List the matrix, just to see how it looks like.

```
di CityDist[2,4]
```

What is the distance between Békéscsaba and Eger?

```
insheet using "offerers.csv",
delim(";") clear

label values corpres CityNames

labe var corpres "Corporation
Residence"

labe var corpname "Corporation Name"

labe var corpid "Corporation ID"

save offerers.dta, replace
```

Import the database of the corporates who want to hire workers for their open positions.

Format, make variable labels and save the database in STATA format.

```
insheet using "positions.csv",
delim(";") clear

labe var posid "Position ID"

labe var corpid "Corporate ID"

labe var corpwtime "Working Time
Required"

labe var corpquali "Qualification
Required"

labe var corpcrea "Creativity
Required"

labe var corpteam "Team Spirit
Required"

labe var corpcomm "Communication
skills Required"

labe var maxsal "Maximum Salary"

save positions.dta, replace
```

Import the database of the open positions.

Format and save it.

```
insheet using "seekers.csv",
delim(";") clear

capture label drop CityNames

label define CityNames 1 "Budapest"
     2    "Békéscsaba"   3
     "Debrecen"     4    "Eger"    5
     "Győr"    6    "Kaposvár"     7
```

Import the database of the job seekers.

Format, make variable labels, create value labels then assign it, lastly save the database in STATA format.

```
      "Kecskemét"    8    "Miskolc" 9
      "Nyíregyháza"  10   "Pécs"    11
      "Salgótarján"  12   "Szeged"  13
      "Székesfehérvár"    14
      "Szekszárd"    15   "Szolnok" 16
      "Szombathely"  17   "Tatabánya"
      18   "Veszprém"     19
      "Zalaegerszeg" 20   "Ajka"    21
      "Baja"    22   "Cegléd"  23
      "Dunaújváros"  24   "Érd"     25
      "Gödöllő" 26   "Gyöngyös"     27
      "Gyula"   28   "Hajdúböszörmény"
      29   "Hódmezővásárhely"  30
      "Kazincbarcika"     31
      "Kiskunfélegyháza"  32
      "Kiskunhalas"  33
      "Mosonmagyaróvár"   34
      "Nagykanizsa"  35   "Orosháza"
      36   "Ózd"     37   "Pápa"    38
      "Sopron"  39   "Szentes" 40
      "Vác"

label values seekres CityNames

labe var seekid "Seeker ID"

labe var seekname "Seeker Name"

labe var seekres "Seeker Residence"

labe var seektdist "Seeker Travel
Distance Wanted"

labe var seekwtime "Working Time
Wanted"

labe var seekquali "Seeker
Qualification"

labe var seekcre "Seeker Creativity"

labe var seekteam "Seeker Team
Spirit"

labe var seekcomm "Seeker
Communication Skills"

labe var minsal "Minumum Salary"

save seekers.dta, replace
```

Process the data, the core of the simulator

```
noisily capture mkdir results
```

Create a new folder where we will save the matching tables for every corporate/position. For every corporate we are going to create a unique folder and add every eligible seeker.

Capture - Skip if there is an error

Noisily - do not suppress error message

```
clear

gen str9 seekid = ""

gen int offers = 0

save "results\\findings", replace
```

Make a table which will hold every seeker and how many offers they got.

```
clear

gen str9 seekid = ""

gen str9 posid = ""

save "results\\matchings", replace
```

Matching table for which position to which seeker is assigned (the algorithm is simple: every position gets the eligible seeker with the lowest minimum wished salary).

```
insheet using
"distances_of_cities.csv",
delim(";") clear

mkmat budapest bkscsaba  debrecen
     eger gyr  kaposvr   kecskemt ///

     miskolc   nyregyhza pcs
     salgtarjn szeged    szkesfehrvr
     szekszrd ///

     szolnok   szombathely
     tatabnya  veszprm   zalaegerszeg
     ajka baja ///

     cegld     dunajvros rd   gdll
     gyngys    gyula     hajdbszrmny
     hdmezvsrhely ///

     kazincbarcika  kiskunflegyhza
     kiskunhalas    mosonmagyarvr
     nagykanizsa ///

     oroshza   zd   ppa  sopron
     szentes   vc, matrix(CityDist)
```

Generate (town-to-town) distance matrix.

```
use positions.dta, clear
```

Open positions database.

```
merge m:1 corpid using offerers.dta
```

Merge positions with offerers (corp.) date in order to have the corporation seat as well.

```
sort posid
```

Sorting by position ID and save

```
save posANDofferers.dta, replace
```

```
count
local N_positions = r(N)
```

Counting how many positions there are. We will have to find a worker (seeker) for every one of them.

```
scalar pos_nomatch = 0
```

Storage for the number of positions which did not get neither one worker.

```
forvalues i = 1/`N_positions' {
```

Go through all of the positons.

```
di `i'
```

Monitoring the process: where are we?

```
quietly{
```

Suppress terminal output

```
local corpname =
subinstr(corpname[`i'],".","",.)
```

Saving the current position's parameters into local macros (storages).

```
local posid = posid[`i']
```

In corporate name substitute ever dot with null string. It is crucial because later we will make folder with these names and dots would make problem.

```
local corpcrea = corpcrea[`i']
local corpteam = corpteam[`i']
local corpcomm = corpcomm[`i']
local corpwtime = corpwtime[`i']
local corpquali = corpquali[`i']
local maxsal = maxsal[`i']
local corpres = corpres[`i']
```

```
di "`corpname'"
```

Monitoring: which corp. is under process?

```
noisily capture mkdir
"results\\`corpname'"
```

Create a new subfolder with the name of the corporation. It could be that there is already a

subfolder with the same name, because this may not be the first cycle.

```
use seekers.dta, clear
```

Loading the seekers' database so as to be able to select appropriate ones.

```
keep if minsal<`maxsal' &
seekcre>=`corpcrea' & seekcomm >=
`corpcomm' & seekteam>=`corpteam' &
seekwtime == "`corpwtime'" &
seekquali == "`corpquali'" &
CityDist[seekres,`corpres'] <
seektdist
```

Filter out the eligible job seekers.

```
count

if r(N) == 0 {

scalar pos_nomatch = pos_nomatch + 1

}
```

Check whether there is at least one eligible worker.

```
else{
```

If there is then continue the procedure

```
save "results\\tmp.dta", replace

outsheet using
"results\\`corpname'\\`posid'.csv",
delimiter(";") replace
```

Save the results in a temporary folder, because it will be needed later.

And as a csv file in the corporation's folder.

```
keep seekid minsal

g str9 posid = "`posid'"

merge 1:1 seekid using
"results\\matchings.dta", update
```

Compare the actual eligible seekers with the ones who have already won a position (in the matching table).

```
keep if posid == "`posid'" & _merge
!= 5
```

Drop those seekers who are already assigned to a position or not concerned for this position.

```
sort minsal

keep if _n==1
```

Keep only the seeker who has the lowest salary requirement.

```
drop minsal _merge
```

```
append using
"results\\matchings.dta"

save "results\\matchings.dta",
replace
```

Extend the matching table based on the lowest salary requirement.

```
use "results\\tmp.dta", clear

keep seekid
```

Load the previously saved temporary result with the eligible seekers.

```
merge 1:1 seekid using
"results\\findings.dta"
```

Merge the table with the number of offers they got.

```
replace offers = 1 if mi(offers)
```

If it is the first offer of hers then change the missing value to one.

```
replace offers = offers + 1 if
_merge == 3
```

If it is not the first then increment.

```
drop _merge

save "results\\findings.dta",
replace
```

Update the table (seekers and number of offers, respectively).

```
}
```

End of the else branch.

```
use posANDofferers.dta, clear
```

Load the table according to which we make the search for eligible seekers.

```
if `i' == 10 continue, break
```

Restriction on the number of positions to which we are looking for seekers. Break will terminate the loop.

```
}
```

End of quietly.

```
}
```

End of for loop.

```
di "Positions which did not get
enquirer(seeker):"

di pos_nomatch
```

Display how many positions have no chance to be fulfilled.