

# Exercise set for introductory courses in microdata.no

## Objective:

*To familiarize yourself with the basic parts of microdata.no, focusing on building data sets in order to produce descriptive statistics as well as regression analysis.*

*Extra exercise (11): Collection of longitudinal information (data set with several observations per unit)*

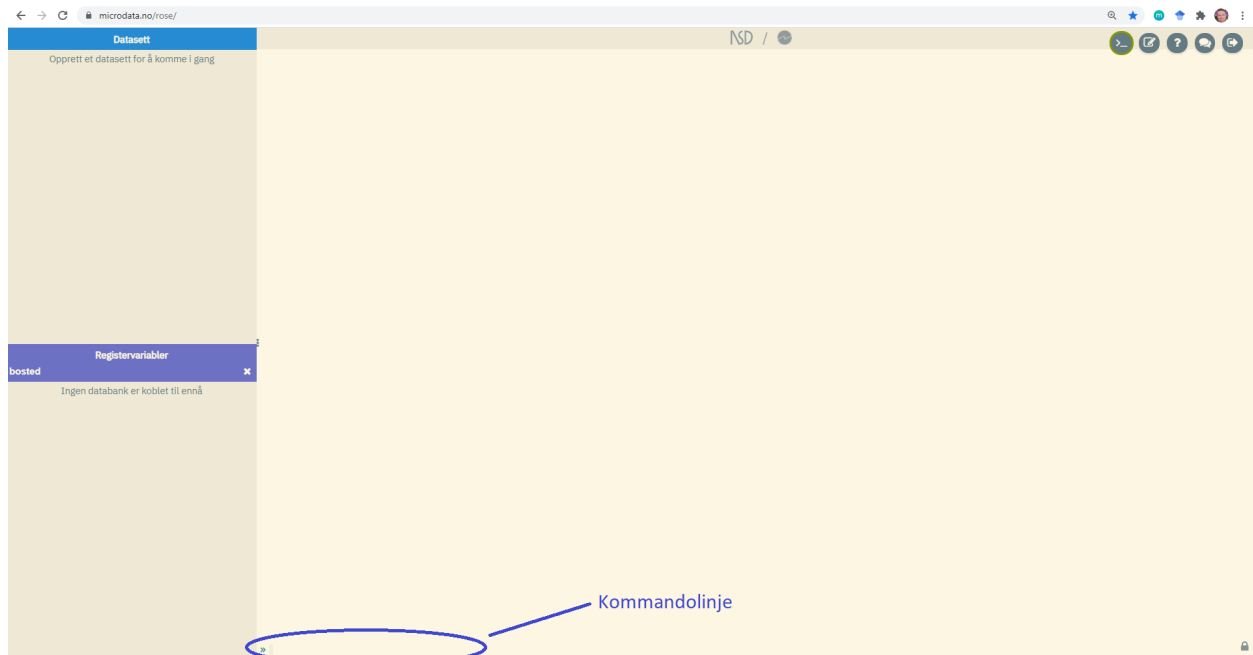
## Login / start-up of microdata.no:

1. Enter the web address “microdata.no/en” in the url field in your browser. Use Chrome or Firefox, or any browser other than Internet Explorer.
2. Hold the mouse cursor over the “Login” button and select "Analysis Tool"
3. Select login option and log in the same way as with an online bank - the most common is BankID on mobile (have mobile available)

The screenshot shows the homepage of microdata.no. A blue circle highlights the address bar containing 'microdata.no/en', with a blue arrow pointing up to it. Another blue circle highlights the 'LOGIN' button in the top navigation bar, with a blue arrow pointing left to it. The main content area features the title 'microdata.no – data access without application' and a list of bullet points: 'No applications', 'Instant access', 'Time series from 1964', 'Export function: Create a dataset and apply for it', and 'Self-service: The institutions register their users themselves'. There are also buttons for 'BIBLIOGRAPHY', 'VARIABLE OVERVIEW', and 'COURSE CALENDAR AND TUTORIALS'. At the bottom, logos for Sikt and Statistisk sentralbyrå are visible, along with a footer containing the text 'Take a microdata.no course in the spring term' and the date '26. January 2022 by sssu.ab@ssb.no'.

We start by working in the analysis area of microdata.no. This is the space that will meet you once you have logged in. Note that before you start working, the contents of the windows are empty.

At the very bottom you will find the command line. This is where you enter the commands you want to execute. Lesson 7) in the exercise set is to save all your work as a script. This allows you to automatically store, systematize, edit and run sets of commands.



### **Helpful hints:**

The command `help` will generate an overview of all available commands. `help` in combination with a command name will present useful information about the current command.

The key combination `<ctrl> + <z>` can be used to undo and go one step back. So if you make a mistake, e.g. encode a variable incorrectly, you can reverse this by going one step back and reencode. You can in principle click `<ctrl> + <z>` many times and undo everything you have done.

The "up arrow" key on the keyboard makes it possible to recall previously used commands. These can then be adjusted and run as new commands.

To see what format the values of a variable have, click on the variable name in the variable list for your dataset in the upper left of the work area. There you will also find a code overview and basic statistics for the specific variable. You can do the same in the variable list at the bottom left. This contains all available variables in the total database (requires that you first use the command `require` to connect to the database).

Variables can be deleted or renamed as needed. This is done by using the commands `drop <variable>` and `rename <name_old> <name_new>` respectively.

*In this exercise set, we will study how wage income has developed in recent years, divided into different demographic groups of the population. We will also take a closer look at selected professions.*

*The focus is on building up datasets for analysis and creating informative descriptive statistics. Finally, we demonstrate how we uncover statistical relationships between variables, and run a simple regression analysis to find causal relationships related to wage income.*

Solutions can be found on the last pages of the exercise set.

## 1) Connect to the variable database

Always start by connecting to the database of variables that you will import into your local dataset in the next steps.

As of today, there is only one available database: Statistics Norway's database. It will eventually be possible to connect to databases that also contain data from other data owners.

Use the following command in the command line to connect to Statistics Norway's database:

```
require no.ssb.fdb:12 as ds
```

(The last part of the syntax, "as ds" is used to create a short name / alias for the database. You are free to choose other names if you wish.)

## 2) Create an empty dataset

The next step is to create a local dataset that you will fill with variables from Statistics Norway's database. Use the following command and select an appropriate name for the dataset (replace <dataset name> with a separate name):

```
create-dataset <dataset name>
```

Your dataset will first be empty. When you add variables with the command `import`, you will see in the upper left window that the dataset is filled with more and more variables, for each `import` command you execute.

### 3) Define population

After creating a dataset, it is time to define the population we are going to study. This can also be done later, but it is recommended to do this as early as possible since the work with data facilitation and analysis then becomes less resource / time consuming (the smaller the data set / population, the faster the response).

Populations are defined by:

- Retrieving variables to be used as population criterias
- Dropping units (individuals) that fall outside the criteria

The command `import` is used to retrieve variables into your dataset from the database. The variable information will be automatically linked to the respective units (individuals) in your population using a built-in identification number, and the data will be organized in a cross-sectional way (one record per unit):

```
import <database alias>/<variable name> <YYYY-MM-DD> [as <alias>]
```

*<database alias>* is to be replaced by the short name you have created for Statistics Norway's database (ds). *<variable name>* is to be replaced by the name of the variable you need to import (remember capital letters), and *<YYYY-MM-DD>* with the measurement date, e.g. 2019-12-31. It is appropriate to use separate names for variables that are imported. This is done by entering `as` plus an optional name at the end of the expression.

When importing variables with fixed information (gender, country of birth, date of birth, etc.), the measurement date is not required.

Example 1: `import ds/NUDB_BU 2019-07-31 as edu`

Example 2: `import ds/BEFOLKNING_FOEDSELS_AAR_MND as birthdate`

- a) It is a good idea to start with a universal variable, since the system uses a left-join principle: The first variable defines your population (which you can later trim down to the desired size).

Start by importing the variable `BEFOLKNING_FOEDSELS_AAR_MND` (date of birth in numeric format `YYYYMM`) into your dataset. Use the command `import`. This is a fixed information, and it is not required to specify a measurement date.

After the import command is run, you can see in the window at the top left that the dataset consists of the date of birth variable plus an individual key variable. By clicking on the variable name, a box with metadata for this variable appears.

Variables with the symbol “123” in front have a numeric format, while “abc” means that the variable has a text format (alphanumeric).

In the same box you will also see the number of records in your dataset (determined by the population of the first imported variable).

You have now created a data set that consists of the entire Norwegian population. Get to know your population by studying the age composition. For this you first need to create a new age variable based on the variable you have imported. Use the command `generate` and measure age per 2019. The variable alias of the date of birth variable that you imported is used as input.

Suggestion:

```
generate age = 2019 - int(birthdate / 100)
```

*(The last part of the formula extracts the first four digits from a 6-digit date of birth, i.e. the year. We only want the integer from the division. Therefore we use the function `int()`. The difference between the stated year 2019 and the year of birth is then age measured per 2019.)*

b) Study the age distribution. Use the command `histogram`:

```
histogram age, discrete
```

The option `discrete` ensures all age cohorts get their own column in the bar chart. This is recommended when using a histogram to look at discrete values (and not intervals of continuous values).

Note that the age range extends well over 100. The reason for this is that your population is not properly filtered. Your dataset contains all the people in the database who have ever lived, including dead people.

c) You need a variable that indicates whether people are resident, dead, or emigrated to remove people you do not want to include in your analysis. For this you use the variable `ds/BEFOLKNING_STATUSKODE`. This variable is

alphanumeric, and takes the values '1', '3' and '5' which mean respectively resident, emigrated and died. Also note that the variable is a so-called cross-sectional variable that only has values for a fixed date per year. It varies when the different cross-sectional variables are measured, but in this case 1/1 is the fixed measurement date (in the command line, valid dates are suggested in the autocomplete menu that appears when entering the command, and you can also check valid measurement dates by clicking on the variable in the database list at the bottom left, possibly in the variable overview you find via the login page).

First import the variable and use the measurement date 2019-01-01. Then use the command `keep if` to keep only those who were resident in Norway on this date (you remove people who died before 2019-01-01 or who had emigrated on this date).

Suggestion:

```
import ds/BEFOLKNING_STATUSKODE 2019-01-01 as regstat
keep if regstat == '1'
```

Use the histogram command again to see what the age distribution now looks like (remember the option `discrete`).

You can also use the command `summarize` to retrieve summary statistics for age (average, standard deviation, median etc): `summarize age`

- d) In this exercise set we will study wage income, and for this purpose it is suitable to further limit the population to people aged 31-49 (when most have finished their studies and have started working).

Use the command `keep if` or `drop if` in combination with standard IF conditions to trim down the dataset to contain the age group 31-49.

Suggestion:

```
keep if age > 30 & age < 50
```

Look at the number of units in your dataset. It has now been reduced from 10,626,204 to 1,368,922 individuals.

In logical IF-expressions, the following characters are used: > (greater than), < (less than), == (equals), != (not equal to), >= (greater than or equal to), <= (less than or equal to), | (or), & (and).

Note that only a simple = is used unless it is in the context of an IF-condition. Example:

```
generate male = 1 if gender == '1'
```

The function `int()` converts numbers into integers.

The `substr()` function extracts substrings from variables with values in text format (alphanumeric), where you specify resp. variable name, position for the first character to be extracted, and the number of positions to be read. Example of extracting the first character from a text value: `pos1 = substr(varx,1,1)`

Missing is referred to by `sysmiss()`, where the variable is specified inside the parenthesis, e.g. “`drop if sysmiss(income)`” states that units with missing value for the variable `income` are to be deleted.

Numeric variables: Values are entered without quotation marks, e.g. 1, 3, 10000

Alphanumeric variables: Values have text format and are specified with single quotation marks, e.g. '1', '3', '10000' (double quotation marks may also be used).

#### 4) Import analysis variables (fill the dataset with variables)

You now have a pre-defined population. The next step is to add variables that you need for your analysis. For this purpose, use the command `import`.

Most data in the database are organized as longitudinal/event data, i.e. variable values are updated continuously. Optional measurement dates are then specified when using the `import` command.

Fixed information such as gender, country of birth, date of birth etc: Measurement date is not required.

Some variables in the database, so-called cross-sectional variables, are taken from statistical sources and are only measured on fixed dates per year. In such cases, you need to specify a specific measurement date suggested by the system (a list of valid dates will occur in the auto suggestion pop-up list when entering a date). You will also find valid measurement dates in the system's variable overviews.

A fourth variant is annual accumulated values that show the same annual value regardless of which date is chosen within a year. This mainly applies to financial information such as income, wealth, debt, etc. In this case, you may choose a date freely within the relevant year.

Import the following variables using the command `import` (also remember the short name of the database):

- a) BEFOLKNING\_KJOENN (gender, fixed information)
- b) BEFOLKNING\_FODELAND (country of birth, fixed information)
- c) NUDB\_BU per 2019-07-31 (highest level of education, 6-digit NUS education code, event information)
- d) BEFOLKNING\_KOMMNR\_FAKTISK per 2019-01-01 (4-digit municipal code of residence, cross-sectional information)
- e) INNTEKT\_WLONN per 2015-12-31 (annual wage income, accumulated value => declared value will be the same no matter which date is chosen within the current year)
- f) INNTEKT\_WLONN per 2016-12-31
- g) INNTEKT\_WLONN per 2017-12-31
- h) INNTEKT\_WLONN per 2018-12-31
- i) INNTEKT\_WLONN per 2019-12-31

NB! People with 0 in wage income => missing value in the database. But this is fine in our analysis, since we are only interested in making statistics for positive incomes. If you wish to include people with zero income, this is done as follows:

```
replace salary = 0 if sysmiss(salary)
```

## 5) Run simple descriptive statistics

Now that we have the population and variables ready, we can start and create descriptive statistics. There are a number of tools available.

To retrieve statistical numbers for numeric variables, usually the `summarize` command is used. To retrieve frequencies / counts divided into different categories, the command `tabulate` is used. You may also combine `tabulate` and `summarize` to display statistical figures divided into categories (e.g. `tabulate gender, summarize(salary)`).



It is also possible to create graphical representations such as pie chart (`piechart`), bar chart (`barchart`), histogram (`histogram`), anonymized plot chart (`hexbin`) and flow chart (`sankey`).

Descriptive statistics can be easily copied and exported into Excel or Google spreadsheets for further processing and creation of your own graphical representations. In this way, you also get to see all decimals, and not just the standard display defined within the system. Graphical output from `microdata.no` can only be exported as an image file and can only be edited in an image editing program such as "Paint".

We start by creating one-dimensional statistics on wage developments for our population. Use the command `summarize` to show how wage income has changed over the period 2015-2019. You can list all the variables you want to measure in a single command.

Suggestion:

```
summarize salary15 salary16 salary17 salary18 salary19
```

The figures can be graphically presented as a bar chart through the command `barchart`. The syntax has the same logic compared to `summarize`. However, you need to specify what kind of statistics you want to show for the bar charts.

Suggestion:

```
barchart (mean) salary15 salary16 salary17 salary18 salary19
```

In addition to average (mean) you can also show the number of individuals with valid values (in this case the number of individuals with positive values) by entering `count` instead of `mean`.

Suggestion:

```
barchart (count) salary15 salary16 salary17 salary18 salary19
```

By entering `median`, you get median values instead. Try this too!

Also study the salary distribution in your population for 2019. Use the command `histogram` and use the variable `salary19`. Tip: You can use the option `freq` to display frequencies as the unit of measure on the y-axis instead of the default view. The option `normal` can also be used to add a standard normal distribution for reference use. All options can be used in combination.

Suggestion:

```
histogram salary19, freq normal
```

## 6) Running two-dimensional descriptive statistics

It is possible to create statistics and graphical presentations with several dimensions. We need to adapt variables for this purpose. We want to examine how wages are distributed between gender, country of birth and level of education. Gender does not need to be recoded, as it only has two categories. But for country of birth and level of education, on the other hand, we need to recode into fewer categories. If not, the statistics will be confusing.

### a)

Let's start by creating a dummy variable for country of birth: Norwegian (= 1) and non-Norwegian (= 0). Call the variable "norwegian". Be sure to always code dummy variables as numeric values, otherwise they will not be able to be used in regression analyzes.

Tip:

- i) Norwegian  $\Leftrightarrow$  country of birth = '000'
- ii) Immigrant background  $\Leftrightarrow$  other codes including missing (persons with temporary residence)

When generating categorical variables, e.g. dummy variables, the simplest method is to first assign all individuals one of the values (use `generate`), and then use `replace` to change the value based on given criterias so that the classification is correct. You can use as many `replace` commands as you want to complete the entire recoding of a given variable, but only one `generate` command.

An alternative to `generate/replace` is to use the more advanced command `recode`. Note that the latter requires variables to be in a numeric format.

Example A:

```
generate male = 1
replace male = 0 if gender == '2'
```

Example B:

```
generate norwegian = 0
replace norwegian = 1 if country == '000'
```

Example C:

```
destring gender
recode gender (2 = 0)
rename gender male
```

Use the command `tabulate` to check the coding of the dummy variable "norwegian". Also use the option `cellpct` for percentage display.

Also try the command `piechart` to create a pie chart of the Norwegian vs non-Norwegian distribution.

## b)

A simple way to aggregate education categories is to pull out the first digit of the standardized NUS code (hierarchical code ranging from 0 (lowest) to 8 (highest)).

Create an aggregated division of education level, i.e. 1st digit level:

Extract the first digit of the 6-digit education code using the `subst()` function.

Tip: `substr(edu, 1, 1)` extracts the first digit from the alphanumeric variable `edu`. Input parameter no. 2 indicates the starting position, while parameter no. 3 indicates the number of positions to be read.

Use the command `tabulate` to show the coding of education. Feel free to use the option `cellpct`.

Transform the aggregated education level variable from alphanumeric to numeric format so that it can be used for ordering and mathematical purposes. Use the command `destring` for this purpose, followed by the name of the variable you want to transform.

You can create alternative divisions of education level, instead of using the standard 1st digit NUS division (= highest NUS level). If you need e.g. 3 categories, you can do it like this:

```
import NUDB_BU 2011-06-01 as edu
generate edulevel = substr(edu,1,1)
destring edulevel, force
replace edulevel = 1 if edulevel <= 3
replace edulevel = 2 if edulevel == 4 | edulevel == 5
replace edulevel = 3 if edulevel > 5
```

Note that `NUDB_BU` is alphanumeric. To make it easier to recode, the `destring` command is used to convert the values to numeric format (the `force` option is used to ensure that any values that contain characters other than numbers is given the value `sysmiss`). Thus, one can use `<`, `>`, `<=`, `>=` etc, which is not possible for alphanumeric values. Remember that leading zeros are removed by numerical conversion. For example, '0301' becomes 301 when converted to numeric format.

Alternatively, you can use the command `recode` instead of `generate / replace`:

```
import NUDB_BU 2011-06-01 as edu
generate edulevel = substr(edu,1,1)
destring edulevel, force
recode edulevel (0/3 = 1) (4/5 = 2) (6/8 = 3)
```

The last line can alternatively be expressed as follows:

```
recode edulevel (0 1 2 3 = 1) (4 5 = 2) (6 7 8 = 3)
```

Now that we have the variables ready, we can start creating multidimensional statistics.

For displaying frequencies / numbers divided into different categories, use `tabulate` and then add more than one variable to the expression. The more variables, the more dimensions. Two variables give a two-way table etc. This can again be combined with `summarize` as an option for displaying statistical measurements instead of number, e.g. `tabulate gender marital_status, summarize(salary)`.

**c)**

Start by creating summary statistics on wage income for the years 2015-2019, but only for those with education level < 2. Use the command `summarize`.

Do the same, but this time for those with an education level > 6.

Then make a table that shows the average wage income for 2019 by gender.

Suggestion:

```
tabulate gender, summarize(salary19)
```

Do the same, but divide by respectively country of birth (Norwegian) and level of education (1-digit) instead of gender.

Make a bar chart that shows the average wage income over the years 2015-2019, broken down by gender. Use the command `barchart` with the option `over()`.

Suggestion:

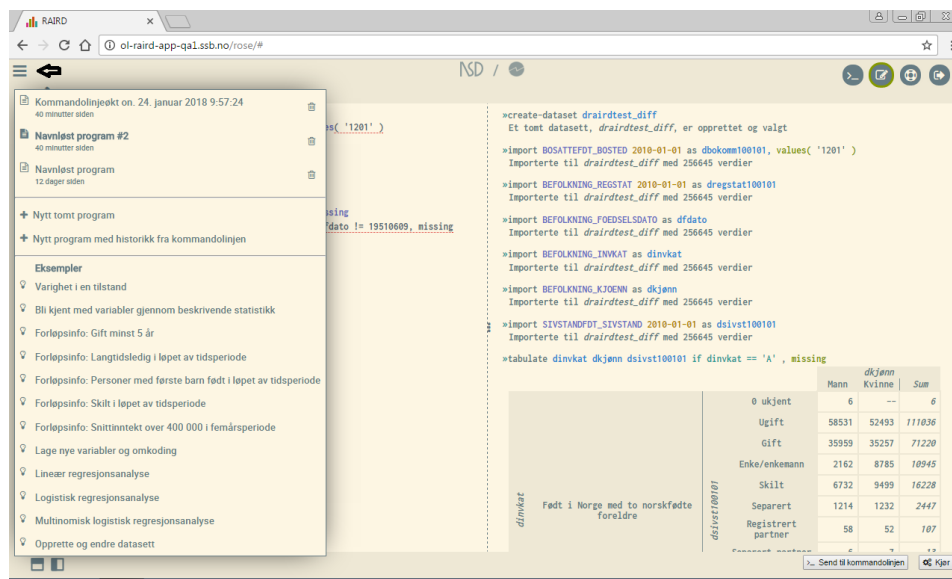
```
barchart(mean) salary15 salary16 salary17 salary18 salary19,
over(gender)
```

Do the same, but divide by respectively country of birth (Norwegian) and level of education instead of gender.

## 7) Save your work into a script

A script can be used to make various adjustments in the analysis. All command lines can then be run again in a block.

- Click on the script button (button no. 2 in the upper right corner) to enter the script area
- Click on the menu button at the top left
- Scroll down to and click on "Nytt program med historikk fra kommandolinjen" ("New program with history from the command line")



The screenshot shows the RAIRD web interface. On the left, a menu is open with the option "Nytt program med historikk fra kommandolinjen" selected. The main area displays a script editor with the following code:

```
>create-dataset drairdtest_diff
Et tomt datasett, drairdtest_diff, er opprettet og valgt

>import BOSATTEFDT_BOSTED 2010-01-01 as dbokomm100101, values( '1201' )
Importerte til drairdtest_diff med 256645 verdier

>import BEFOLKNING_REGSTAT 2010-01-01 as dregstat100101
Importerte til drairdtest_diff med 256645 verdier

>import BEFOLKNING_FOEDELSDATO as dfdato
Importerte til drairdtest_diff med 256645 verdier

>import BEFOLKNING_INVKAT as dinvkat
Importerte til drairdtest_diff med 256645 verdier

>import BEFOLKNING_KJOENN as dkjenn
Importerte til drairdtest_diff med 256645 verdier

>import SIVSTANDFDT_SIVSTAND 2010-01-01 as dsivst100101
Importerte til drairdtest_diff med 256645 verdier

>tabulate dinvkat dkjenn dsivst100101 if dinvkat == 'A' , missing
```

Below the script, a table is displayed with the following data:

	Mann	dkjenn Kvinner	Sum
0 ukjent	6	---	6
Ugift	58531	52493	111036
Gift	35959	35257	71220
Enke/enkemann	2162	8785	10945
Skilt	6732	9499	16228
Separert	1214	1232	2447
Registrert partner	58	52	107

- All your work is now embedded in a script.
- Enter a name for your script in the field just above the script area. Your script will now be saved with this name and can be found again by clicking the menu button in the upper left.
- As an alternative to a)-e) you can just type `save ' . . . . . '` in the command line, where you enter a suitable name for the script inside the single quotation marks. Then go into the script window and continue working on your script.
- You can now run your work as a block by clicking on "Kjør" ("Execute") or "Send til kommandolinjen" ("Send to command line"), or alternatively make adjustments first and then run again. As long as your script has a name declared, it will be auto-saved when you work in it, just like in Google Docs.

Note: The system remembers previously executed command sequences. So if you run the same script twice in a row, the second run will only retrieve the previous result. If you adjust something or add new lines at the end of the script, the executed sequences are retrieved from memory, and only the new one is executed. If, on the other hand, lines are changed at the very beginning of a script, all the sequences are considered new, and everything is run all over again.

NB! When a script is run using the “Send til kommandolinjen” button, everything is sent to the work area, and whatever may have been there before will be overwritten by the new execution. Therefore, make sure that your work in the work area is always taken care of via scripts.

NB! You can make backups by copying scripts into your own documents. If you intend to paste the content into a new script afterwards, you must make sure to use a clean text format via programs such as "Notepad" etc. (text that is copied into the script editor directly from programs such as Word etc. can create curl since the text then contains various formats that can create problems when running scripts).

Are you on target for lesson 7? Then you can try the slightly more advanced lessons 8-11.

You can choose whether you want to continue working in the script editor or in the command line in the next exercises. It is recommended to work in the script editor and run the commands from there due to the ability to store and systematize your work. The editor can, among other things, be used to copy and paste so that you do not have to use so much manual programming. The command line, on the other hand, is best suited for exploratory work and for getting acquainted with variables and syntax.

## **8) Create refined statistics: Take a closer look at selected occupational groups**

Now that we have mastered descriptive statistics, we can go a step further and look at how wage income varies across occupational groups.

### **a)**

For this we first need to import a variable that indicates occupational codes (4-digit alphanumeric code): REGSYS\_ARB\_YRKE\_STYRK08.

The occupational variable is a so-called cross-sectional variable that is measured per 16/11 each year. Import this into your dataset and use the measurement date 2019-11-16. Appropriate alias can be `profession19`.

The occupational variable has many codes, so it becomes too detailed for our purpose. We therefore choose to select interesting professions we want to look at:

- Leader professions (codes starting with '1')
- General practitioners ('2211')
- Medical specialists ('2212')
- Nurses ('2223')
- Teachers (codes starting with '23' )
- IT developers (codes starting with '25')
- Agricultural professions (codes starting with '61')

In addition, we must specify a rest group that contains all other occupations, as well as a category for people who are not working.

Use the commands `generate` and `replace` to create a new variable that consists of a division with the selected occupations listed above. Call the variable `prof_gr`.

Tip:

- Start with the command `generate` and set all values for the new variable `prof_gr` to 9 (rest group)
- In the next line, use `replace` to replace with the value 1 in cases where `profession19` starts with '1' (leaders)
- Repeat the process for each occupational code you want to extract and replace with ever higher codes until you get to 7
- Finally, you replace the values for `prof_gr` with 999 if `profession19` has missing value (`replace prof_gr = 999 if sysmiss(profession19)`)

*(If this becomes too difficult, it is allowed to look at the solutions at the back of the exercise set)*

**b)**

Value labels can be used to show what the different values mean, so that the statistics become more readable and understandable. For this, the commands `define-labels` and `assign-labels` are used.

Create understandable labels for your various occupational codes using `define-labels`, and associate them with the appropriate variable with `assign-labels`.

Note that labels that contain special characters or spaces must be specified with single quotation marks. If you are unsure, you can just use quotation marks consistently. Then it will always be right. In the example below, the labels "Male" and "Female" are used. These do not contain any special characters and can be nicely specified without quotation marks. The values must always be entered in the format they have (with quotation marks when alphanumeric values, without otherwise).

Example:

```
define-labels genderlabel 1 Male 2 Female
assign-labels gender genderlabel
```

c)

Make an overview of the number of people with the different occupations using the command `tabulate`. Use the new variable `prof_gr`. Also use the option `cellpct` to display the percentage distribution.

Make a table that shows the average wage income in 2019 divided into the selected occupations using the command-combination `tabulate/summarize`.

Create a bar chart for a graphical representation of the same numbers using the command `barchart`. Use the option `over()` to split into columns based on occupation.

Finally, create a bar chart that shows the wage development over 2015-2019 divided into the selected occupations using `barchart`. Also use the option `over()`.

## 9) Use information about parents in analysis

Microdata.no has, among other things, information about the father's and mother's identity number available in the database. These can be used to add all sorts of information about mom and dad. In the steps below, we demonstrate how this is done.

- a) Import father and mother ID numbers into your dataset. The variables are called resp. `BEFOLKNING_FAR_FNR` and `BEFOLKNING_MOR_FNR`. This is fixed information and you should therefore not specify a measurement date. Use resp. `father_fnr` and `mother_fnr` as aliases.
- b) Create a new dataset using the command `create-dataset`, which you use to add parent information in the usual way. Call it `parents`.



- c) You are automatically sent to the new `parents` data set. Import the `INNTEKT_WLONN`, `REGSYS_ARB_YRKE_STYRK08` and `NUDB_BU` variables into this dataset in the same way as you did in your original dataset. Also use the same measurement dates, for the wage variable you only import per 2019-12-31. Also arrange the education variable in the same way (1st digit) and give appropriate names that indicate father, e.g. `salary19_father`, `profession19_father`
- d) Copy / clone the father variables using the command `clone-variables` and call them accordingly, but enter "mother" instead of "father". Example:  
`clone-variables salary19_father -> salary19_mother`
- e) You now have a double set of salary, occupational and educational variables for resp. father and mother which you will next connect to your original dataset. Use the command `merge` for this purpose, and use resp. father's and mother's ID numbers as merge keys. Example: `merge salary19_father profession19_father edulevel_father into totalpop on father_fnr`
- f) Move over to your original data set and create statistics:
- i) Statistics on own salary, father's salary and mother's salary in 2019 using `summarize`
  - ii) Check if there is a correlation between own salary and father's salary. Do the same with mother's salary, and also check father's and mother's salary against each other. Use the command `correlate` and list the two variables to be compared, e.g. `correlate salary19 salary19_father`
  - iii) Do the same as ii) with own, mother's and father's level of education.
  - iv) Do as in ii) and check own salary vs. father's salary for men (`if gender == '1'`). Also check own salary vs. mother's salary for women (`if gender == '2'`). Do the same for the level of education.
- g) Use the occupational variable for father and extract selected occupations as in 8a). Use a variable name that indicates father, e.g. `prof_gr_father`. If you work in the script editor, you can copy the command lines you have used before and adjust. Then it goes faster. This requires that you first transfer your work to the script area. See 7) on how to do this. Use the labels you have already created, and then link them to the father's occupation codes using `assign-labels`. You do not need to make the same labels twice.
- h) Run the following command to show the connection between father and own profession (runs only for the selected professions, excluding "other" and "without job"):

```
tabulate prof_gr_father prof_gr if prof_gr_father < 9,  
rowpct
```

Do the same, but this time only for men (gender == '1'):

```
tabulate prof_gr_father prof_gr if prof_gr_father < 9 &  
gender == '1', rowpct
```

*(If the steps above are too difficult, take a look at the solutions at the back. You are allowed to take a look !)*

## 10) Regression analysis

a) Carry out a linear regression analysis in which the effects of various background characteristics on wage income in 2019 are analyzed using the command `regress`. Use variables you have created earlier in the exercise set:

- Wage income per 2019 (depending variable)
- Father's wage income per 2019
- Age
- Norwegian
- Male (create a dummy variable for men based on the variable `gender`)
- Oslo (create a dummy variable that is set equal to 1 if `residence == '0301'`)
- Higher education (create a dummy variable for high level of education based on the variable for highest completed education, `level >= 7 =>` the value 1, the rest 0)

NB! Remember that the dependent variable must be listed first in the regression expression.

Suggestion:

```
generate male = 0  
replace male = 1 if gender == '1'
```

```
generate oslo = 1 if residence == '0301'  
replace oslo = 0 if residence != '0301'  
tabulate oslo, cellpct
```

```

generate high_edu = 1 if edulevel >= 7
replace high_edu = 0 if edulevel >= 0 & edulevel < 7

regress salary19 age norwegian oslo male high_edu
salary19_father

```

- b) Use the command `regress-predict` to retrieve the residual values from the regression model above. Tip: You specify the variables in exactly the same way as in the `regression` expression in a). Also specify the option `residuals()` to indicate that you want to generate a new variable that contains the residuals (inside the parentheses you enter a name of the variable with the residuals, e.g. `res`)
- c) Study the residuals using graphical tools such as `histogram`. Use the name of the residual variable from b) as input.
- d) To check if the residuals are normally distributed, use the `histogram` command (with the residual variable as input) with the option `normal`.

### 11) Extra exercise: Retrieve information based on events over time - longitudinal extracts

In addition to the common cross-sectional data structure (one measurement date per variable) that you do using the command `import`, you can also retrieve events that occur over a time span. This can be used to identify units where time is used as an extra dimension.

In this lesson, we will try to identify individuals in our population who divorced during the previous year, i.e. in 2018.

- a) Use the command `create-dataset` and create a new dataset with a suitable name, e.g. `events`.
- b) Use the command `import-event` to retrieve all records for the variable `ds/SIVSTANFDT_SIVSTAND` which is found within the date interval 2018-01-01 to 2018-12-31. The variable can be given an appropriate name, e.g. `siv_events`
- c) Use the command `keep if` to keep only records that contain the value for the "divorced" status, i.e. the value '4'.
- d) You must now aggregate the data into person level with one record per individual. You then use the command `collapse(count)` and specify the event based variable `siv_events` followed by a comma to specify the option,

and finally the condition `by (PERSONID_1)`. The measurement statistics `count` counts the number of records (i.e. with the status "divorced" since we have removed all other records), and replaces the original value with this number for the variable `siv_events`.

- e) Use the command `rename` to rename `siv_events` into e.g. `divorces`.
  - f) Use the command `merge` to merge the variable `divorces` into your individual data set (`totalpop`).
  - g) Use the command `use` to move to the `totalpop` dataset. Create a dummy variable that takes the value 1 for people with values on `divorces`  $\geq 1$ . Others are given the value 0. Use the commands `generate` and `replace` in the same way as the procedure earlier in the exercise set.
  - h) Check how many in your population have the status "divorced" during 2018 by using the command `tabulate`. Use the dummy variable from g) as input.
-

## Solutions (syntax):

1)

```
require no.ssb.fdb:12 as ds
```

2)

```
create-dataset totalpop
```

3a)

```
import ds/BEFOLKNING_FOEDSELS_AAR_MND as birthdate  
generate age = 2019 - int(birthdate/100)
```

3b)

```
histogram age, discrete
```

3c)

```
import ds/BEFOLKNING_STATUSKODE 2019-01-01 as regstat  
keep if regstat == '1'  
histogram age, discrete  
summarize age
```

3d)

```
keep if age > 30 & age < 50
```

4)

```
import ds/BEFOLKNING_KJOENN as gender  
import ds/BEFOLKNING_FODELAND as country  
import ds/NUDB_BU 2019-07-31 as edu  
import ds/BEFOLKNING_KOMMNR_FAKTISK 2019-01-01 as residence  
import ds/INNTEKT_WLONN 2015-12-31 as salary15  
import ds/INNTEKT_WLONN 2016-12-31 as salary16  
import ds/INNTEKT_WLONN 2017-12-31 as salary17  
import ds/INNTEKT_WLONN 2018-12-31 as salary18  
import ds/INNTEKT_WLONN 2019-12-31 as salary19
```

5)

```
summarize salary15 salary16 salary17 salary18 salary19
```

```
barchart (mean) salary15 salary16 salary17 salary18 salary19
```

```
barchart (count) salary15 salary16 salary17 salary18 salary19
barchart (median) salary15 salary16 salary17 salary18 salary19
```

```
histogram salary19, freq
histogram salary19, freq normal
```

```
6a)
generate norwegian = 0
replace norwegian = 1 if country == '000'
tabulate norwegian
tabulate norwegian, cellpct
piechart norwegian
```

```
6b)
generate edulevel = substr(edu,1,1)
tabulate edulevel, cellpct
destring edulevel
```

```
6c)
summarize salary15 salary16 salary17 salary18 salary19 if edulevel < 2
summarize salary15 salary16 salary17 salary18 salary19 if edulevel > 6
```

```
tabulate gender, summarize(salary19)
tabulate norwegian, summarize(salary19)
tabulate edulevel, summarize(salary19)
```

```
barchart (mean) salary15 salary16 salary17 salary18 salary19, over(gender)
barchart (mean) salary15 salary16 salary17 salary18 salary19, over(norwegian)
barchart (mean) salary15 salary16 salary17 salary18 salary19, over(edulevel)
```

```
8a)
import ds/REGSYS_ARB_YRKE_STYRK08 2019-11-16 as profession19
```

```
generate prof_gr = 9
replace prof_gr = 1 if substr(profession19,1,1) == '1'
replace prof_gr = 2 if profession19 == '2211'
replace prof_gr = 3 if profession19 == '2212'
replace prof_gr = 4 if profession19 == '2223'
replace prof_gr = 5 if substr(profession19,1,2) == '23'
replace prof_gr = 6 if substr(profession19,1,2) == '25'
replace prof_gr = 7 if substr(profession19,1,2) == '61'
```

```
replace prof_gr = 999 if sysmiss(profession19)
```

8b)

```
define-labels proflabel 1 Leaders 2 'General practitioners' 3 'Medical specialists' 4 Nurses 5  
Teachers 6 'IT developers' 7 'Agricultural professions' 9 Other 999 'Not working'  
assign-labels prof_gr proflabel
```

8c)

```
tabulate prof_gr  
tabulate prof_gr, cellpct  
tabulate prof_gr, summarize(salary19)  
barchart (mean) salary19, over(prof_gr)  
barchart (mean) salary15 salary16 salary17 salary18 salary19, over(prof_gr)
```

9a)

```
import ds/BEFOLKNING_FAR_FNR as father_fnr  
import ds/BEFOLKNING_MOR_FNR as mother_fnr
```

9b)

```
create-dataset parents
```

9c)

```
import ds/INNTEKT_WLONN 2019-12-31 as salary19_father  
import ds/REGSYS_ARB_YRKE_STYRK08 2019-11-16 as profession19_father  
import ds/NUDB_BU 2019-07-31 as edu_father  
generate edulevel_father = substr(edu_father,1,1)  
destring edulevel_father
```

9d)

```
clone-variables salary19_father -> salary19_mother  
clone-variables profession19_father -> profession19_mother  
clone-variables edulevel_father -> edulevel_mother
```

9e)

```
merge salary19_father profession19_father edulevel_father into totalpop on father_fnr  
merge salary19_mother profession19_mother edulevel_mother into totalpop on mother_fnr
```

9f)

```
use totalpop  
summarize salary19 salary19_father salary19_mother  
correlate salary19 salary19_father  
correlate salary19 salary19_mother
```

```
correlate salary19_father salary19_mother
correlate edulevel edulevel_father
correlate edulevel edulevel_mother
correlate edulevel_father edulevel_mother
```

```
correlate salary19 salary19_father if gender == '1'
correlate salary19 salary19_mother if gender == '2'
correlate edulevel edulevel_father if gender == '1'
correlate edulevel edulevel_mother if gender == '2'
```

9g)

```
generate prof_gr_father = 9
replace prof_gr_father = 1 if substr(profession19_father,1,1) == '1'
replace prof_gr_father = 2 if profession19_father == '2211'
replace prof_gr_father = 3 if profession19_father == '2212'
replace prof_gr_father = 4 if profession19_father == '2223'
replace prof_gr_father = 5 if substr(profession19_father,1,2) == '23'
replace prof_gr_father = 6 if substr(profession19_father,1,2) == '25'
replace prof_gr_father = 7 if substr(profession19_father,1,2) == '61'
replace prof_gr_father = 999 if sysmiss(profession19_father)
```

```
assign-labels prof_gr_father proflabel
```

9h)

```
tabulate prof_gr_father prof_gr if prof_gr_father < 9, rowpct
tabulate prof_gr_father prof_gr if prof_gr_father < 9 & gender == '1', rowpct
```

10a)

```
generate male = 0
replace male = 1 if gender == '1'
```

```
generate oslo = 1 if residence == '0301'
replace oslo = 0 if residence != '0301'
tabulate oslo, cellpct
```

```
generate high_edu = 1 if edulevel >= 7
replace high_edu = 0 if edulevel >= 0 & edulevel < 7
```

```
regress salary19 age norwegian oslo male high_edu salary19_father
```

10b)

```
regress-predict salary19 age norwegian oslo male high_edu salary19_father, residuals(res)
```



10c)  
histogram res

10d)  
histogram res, normal

11a)  
create-dataset events

11b)  
import-event ds/SIVSTANDFDT\_SIVSTAND 2018-01-01 to 2018-12-31 as siv\_events

11c)  
keep if siv\_events == '4'

11d)  
collapse (count) siv\_events, by(PERSONID\_1 )

11e)  
rename siv\_events divorces

11f)  
merge divorces into totalpop

11g)  
use totalpop  
generate divorced2018 = 0  
replace divorced2018 = 1 if divorces >= 1

11h)  
tabulate divorced2018

---