User Guide for microdata.no

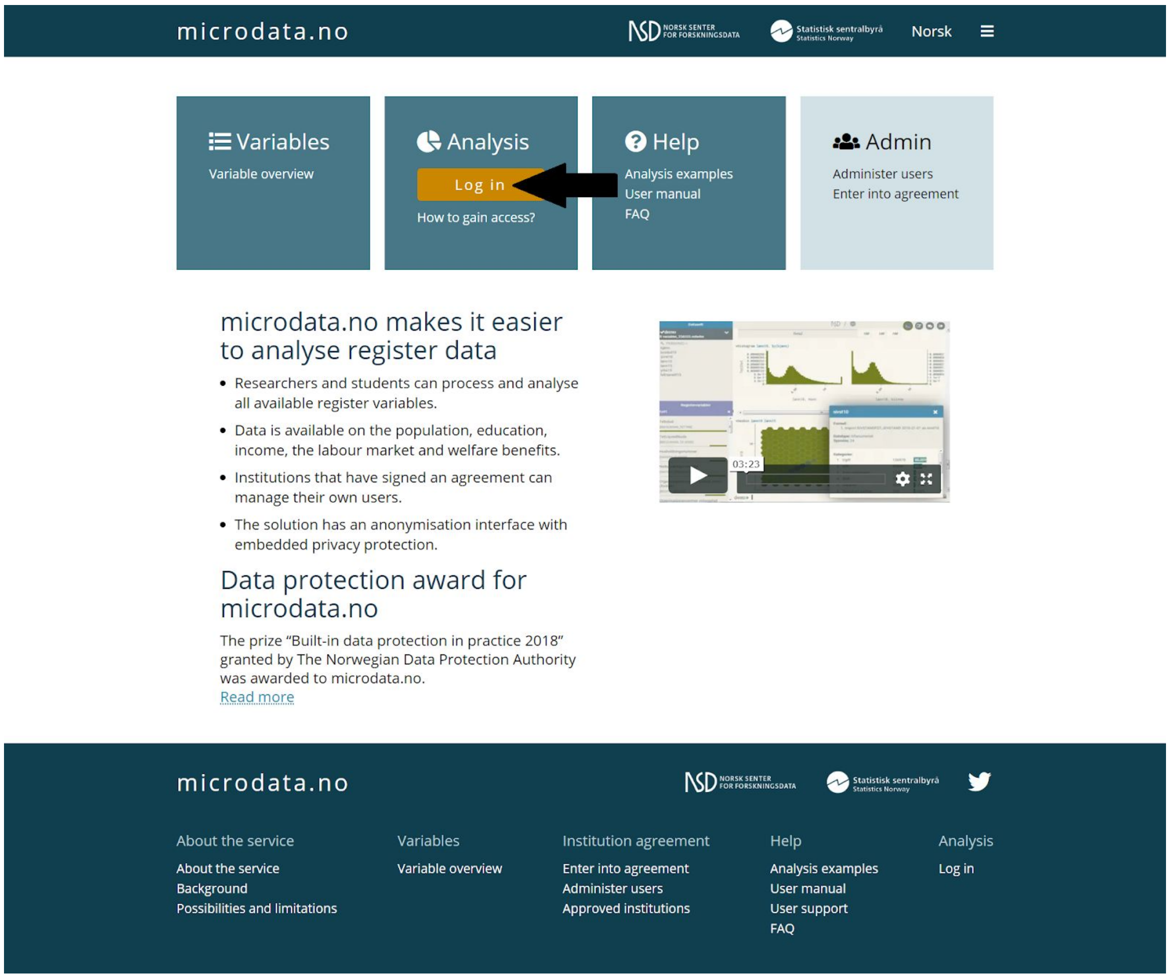
Content list

1.About the user interface

Microdata.no is a web-based analysis system. It is recommended to use browsers such as Chrome and Firefox for the best user experience. Internet Explorer and Edge may cause errors in the screen becoming blank and / or inability to log in, and are therefore not recommended.

Login to microdata.no is done via the following website:

https://microdata.no/en/



microdata.no makes it easier to analyse register data

- Researchers and students can process and analyse all available register variables.
- Data is available on the population, education, income, the labour market and welfare benefits.
- Institutions that have signed an agreement can manage their own users.
- The solution has an anonymisation interface with embedded privacy protection.

Data protection award for microdata.no

The prize "Built-in data protection in practice 2018" granted by The Norwegian Data Protection Authority was awarded to microdata.no.
Read more

What you see after login is a website consisting of an analysis area (work space). This is used to explore variables and test commands:

By clicking on variables one can get acquainted with the content, value format, validity period, etc. Through commands that are run separately, variables can be imported into datasets for further processing and analysis. In addition to descriptive statistical possibilities, one can also perform advanced statistical operations such as regression analyzes, etc. See chapters 1.1-1.4 for more information on the work space.

After exploring and familiarizing with the data you want to use in analyzes, it is strongly recommended to use the script functionality to systematize the analysis work. This is especially true if one intends to carry out a more comprehensive analysis (beyond basic descriptive statistics for a few variables). Using scripts has many advantages over working exclusively in the analysis area through the use of single commands:

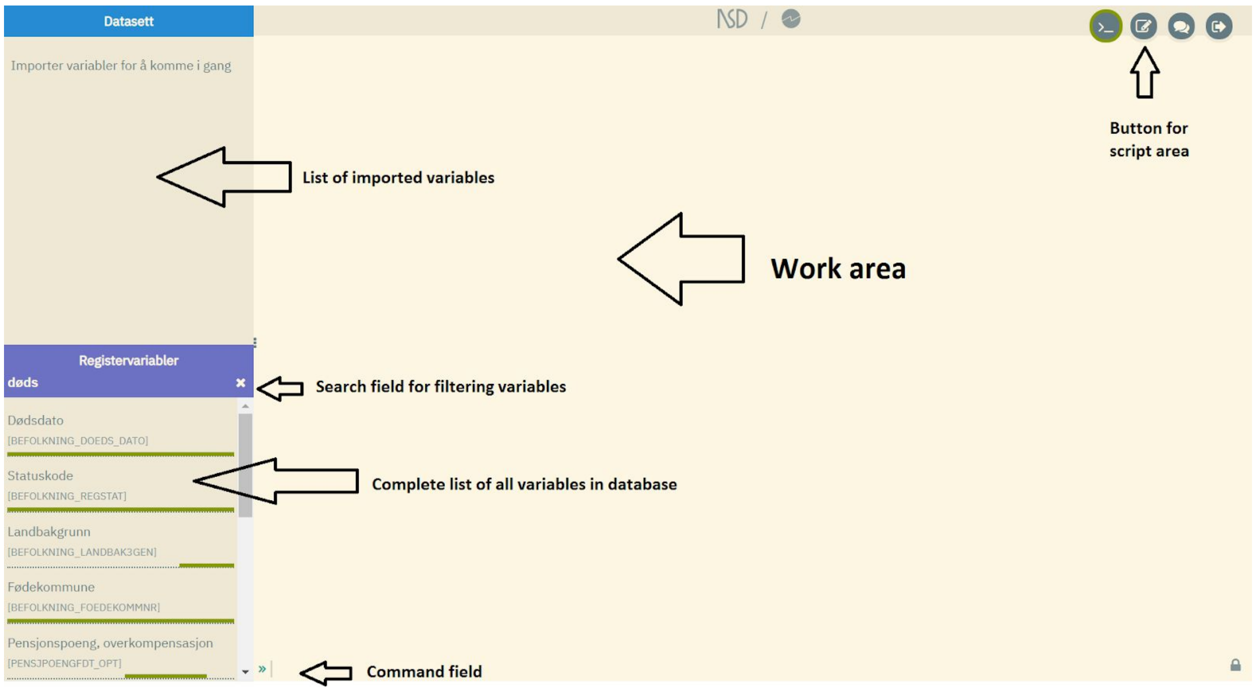
a) One can construct command sequences to be run in one operation. The results of script executions are displayed continuously as commands are run and the execution is stopped if errors are detected (syntax or logical errors)
b) Command sequences can be edited and rerun
c) Much easier to identify errors in a long chain of command sequences
d) Works as documentation / log of work
e) Various work sessions can be saved with their separate names for later reuse
f) The contents of a script can be copied into separate (external) documents for extra backup

Work already performed in the analysis area can easily be transferred into a script for further editing. This can be done in two ways:

- In the script window there is a menu button at the top left. There you can select “Nytt skript med historikk fra kommandolinjen” ("New script with command line history").
- Use the command `history` in the work space. This returns a chronological list of all commands run, which can be copied into the script window by using the <ctrl> + <c> keyboard combination

For more information on using scripts, see chapter 1.5.

1.1 The analysis area (work space)

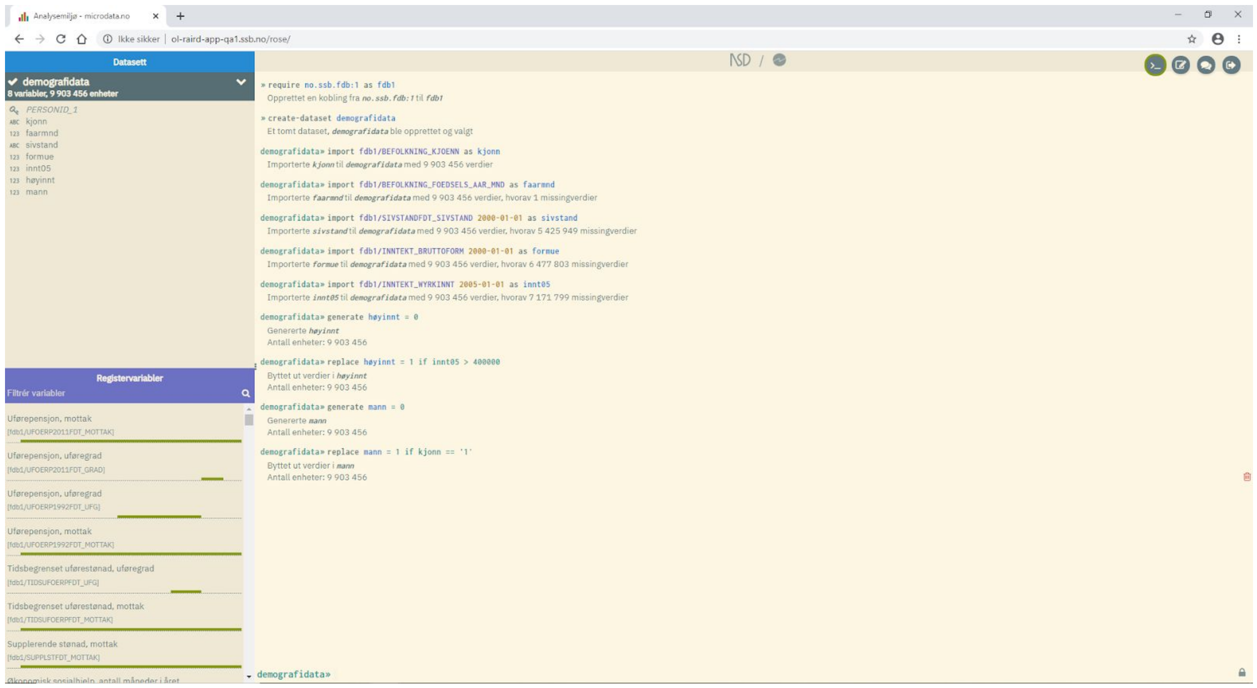


The analysis area consists of the following:

- Work area (largest space at the far right)
- Overview of available variables (area at the bottom left - this space is blank until you have connected to a data bank, ref. chapter 2.1)
- Overview of variables extracted/imported into own work datasets (area at the top left)
- Command line (bottom of work area)

Custom datasets are built up by retrieving (importing) and if necessary processing (and developing new) variables from the data directory. Datasets are stored in the user's work space and do not disappear without the user deleting them. See Chapter 2 on how to create datasets and import variables.

After the analysis area has been filled up with imported variables, it may look like this:


Analysemiljø - microdata.no
Ikke sikker | ol-raird-app-qa1.ssb.no/rose/
Datasett
Registervariabler
Filtrér variabler

demografidata
8 variabler, 9 903 456 enheter

PERSONID_1
kjonn
faarmnd
sivstand
formue
innt05
høyinnt
mann

Uførepensjon, mottak [fdb1/UFOERP2011FDT_MOTTAK]
Uførepensjon, uføregrad [fdb1/UFOERP2011FDT_GRAD]
Uførepensjon, uføregrad [fdb1/UFOERP1992FDT_UFG]
Uførepensjon, mottak [fdb1/UFOERP1992FDT_MOTTAK]
Tidsbegrenset uførestønad, uføregrad [fdb1/TIDSUFOERPFDT_UFG]
Tidsbegrenset uførestønad, mottak [fdb1/TIDSUFOERPFDT_MOTTAK]
Supplerende stønad, mottak [fdb1/SUPPLSTFDT_MOTTAK]
Økonomisk sosialhjelp, antall måneder i året

```
» require no.ssb.fdb:1 as fdb1
  Opprettet en kobling fra no.ssb.fdb:1 til fdb1

» create-dataset demografidata
  Et tomt dataset, demografidata ble opprettet og valgt

demografidata» import fdb1/BEFOLKNING_KJOENN as kjonn
  Importerte kjonn til demografidata med 9 903 456 verdier

demografidata» import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
  Importerte faarmnd til demografidata med 9 903 456 verdier, hvorav 1 missingverdier

demografidata» import fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
  Importerte sivstand til demografidata med 9 903 456 verdier, hvorav 5 425 949 missingverdier

demografidata» import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as formue
  Importerte formue til demografidata med 9 903 456 verdier, hvorav 6 477 803 missingverdier

demografidata» import fdb1/INNTEKT_WYRKINNT 2005-01-01 as innt05
  Importerte innt05 til demografidata med 9 903 456 verdier, hvorav 7 171 799 missingverdier

demografidata» generate høyinnt = 0
  Genererte høyinnt
  Antall enheter: 9 903 456

demografidata» replace høyinnt = 1 if innt05 > 400000
  Byttet ut verdier i høyinnt
  Antall enheter: 9 903 456

demografidata» generate mann = 0
  Genererte mann
  Antall enheter: 9 903 456

demografidata» replace mann = 1 if kjonn == '1'
  Byttet ut verdier i mann
  Antall enheter: 9 903 456

demografidata»
```

In the example above, a dataset named “demografidata” ("demographic data") has been created, containing 8 variables and 9 903 456 units (individuals). Amongst the variables, you will find the personal identification key PERSONID_1. This is a system variable that is always included with the import of variables. It specifies a unique person identifier that is used as a key for linking variables together on individual level. The system performs an automatic merging of variables according to the "left join" principle, so if you are only interested in using data on individual cross-sectional level, then you do not need to deal with this variable.

In some cases, using the PERSONID_1 identification key may be applicable. It is possible to generate summary statistics based on events over time through the command `collapse` and even control at which unit level one wants the aggregated data to be available (this is described in more detail in chapter 3.4). The command `merge` makes it possible to merge different datasets created by the user, where the identification key may also be relevant to use (see section 2.8).

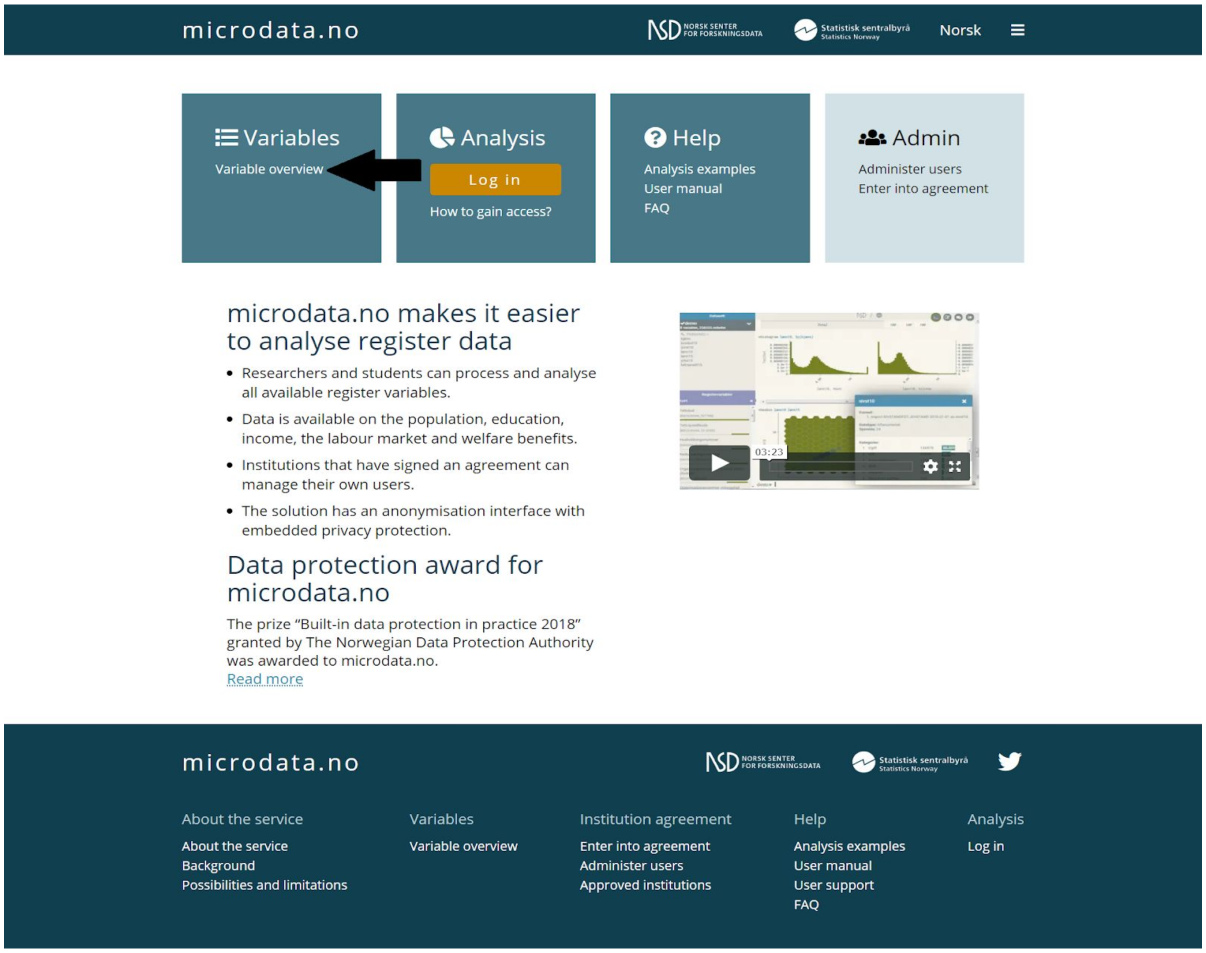
Note that both `collapse` and `merge` allow aggregation and merging on various unit levels respectively. However, the system requires that the merging / aggregation key holds a minimum number of value categories (over 1000). Eg. one cannot aggregate or merge datasets at gender level, marital level or residence level as the number of categories are too low.

The work area displays a log containing the commands that have been executed and the resulting response, be it tables, figures and other feedback.

To indicate that the user is now working on the dataset “demografidata>>” ("demographic data>>"), “demografidata>>” is now displayed instead of ">>" at the bottom (command line). If multiple datasets are created, the window at the top left will contain several variable views correspondingly. To work on different datasets, one can switch between them by using the command `use <dataset>`. The leading text in the command line will indicate what dataset the user has moved to.

1.2 Variables and variable definitions

The microdata.no analysis system has a wide range of demographic, educational, economic, employment and social security variables in the database. The variables are comprehensively described in the variable overview found on the opening page (https://microdata.no/discovery):


microdata.no

Norsk

Variables
Variable overview

Analysis
Log in
How to gain access?

Help
Analysis examples
User manual
FAQ

Admin
Administer users
Enter into agreement

microdata.no makes it easier to analyse register data

- Researchers and students can process and analyse all available register variables.
- Data is available on the population, education, income, the labour market and welfare benefits.
- Institutions that have signed an agreement can manage their own users.
- The solution has an anonymisation interface with embedded privacy protection.

Data protection award for microdata.no

The prize “Built-in data protection in practice 2018” granted by The Norwegian Data Protection Authority was awarded to microdata.no.
Read more

microdata.no

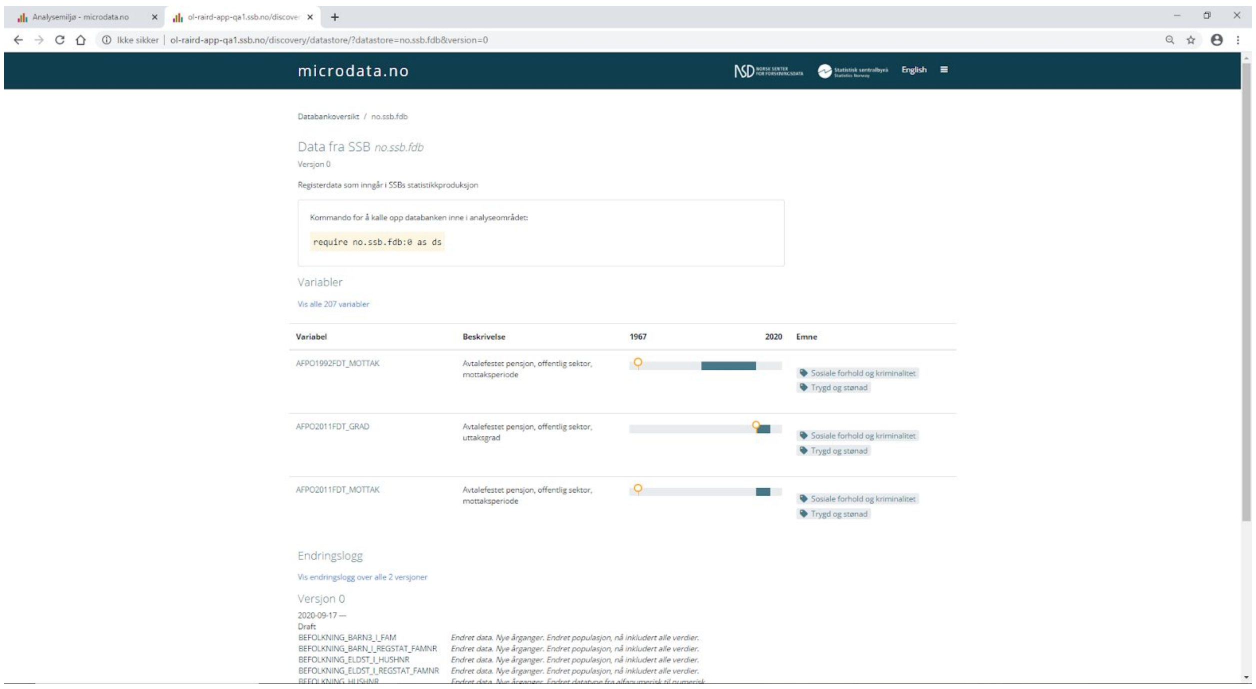
About the service
About the service
Background
Possibilities and limitations

Variables
Variable overview

Institution agreement
Enter into agreement
Administer users
Approved institutions

Help
Analysis examples
User manual
User support
FAQ

Analysis
Log in


Analysemiljø - microdata.no
ol-raird-app-qa1.ssb.no/discove
Ikke sikker
ol-raird-app-qa1.ssb.no/discovery/datastore/?datastore=no.ssb.fdb&version=0
microdata.no
English
Databankoversikt / no.ssb.fdb

Data fra SSB *no.ssb.fdb*

Versjon 0

Registerdata som inngår i SSBs statistikkproduksjon

Kommando for å kalle opp databanken inne i analyseområdet:

```
require no.ssb.fdb:0 as ds
```

Variabler

Vis alle 207 variabler

Variabel	Beskrivelse	1967 2020	Emne
AFPO1992FDT_MOTTAK	Avtalefestet pensjon, offentlig sektor, mottaksperiode		Sosiale forhold og kriminalitet Trygd og stønad
AFPO2011FDT_GRAD	Avtalefestet pensjon, offentlig sektor, uttaksgrad		Sosiale forhold og kriminalitet Trygd og stønad
AFPO2011FDT_MOTTAK	Avtalefestet pensjon, offentlig sektor, mottaksperiode		Sosiale forhold og kriminalitet Trygd og stønad

Endringslogg

Vis endringslogg over alle 2 versjoner

Versjon 0

2020-09-17 —

Draft

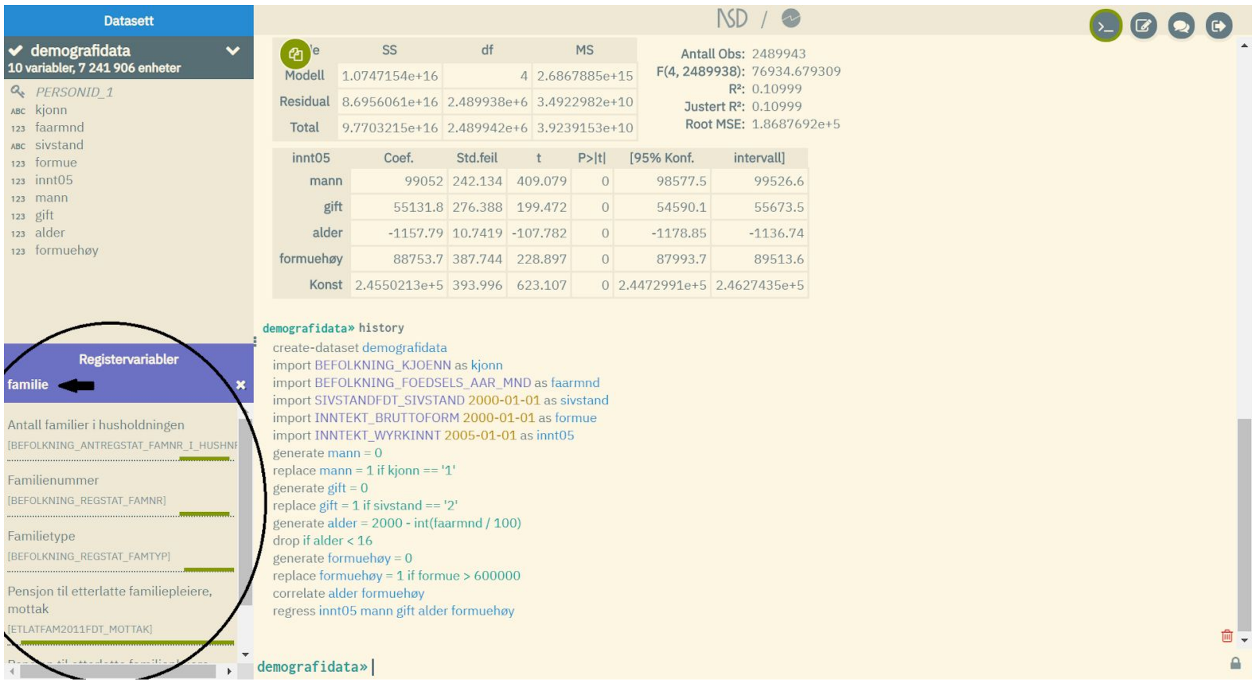
BEFOLKNING_BARN3_I_FAM	*Endret data. Nye årganger. Endret populasjon, nå inkludert alle verdier.*
BEFOLKNING_BARN_I_REGSTAT_FAMNR	*Endret data. Nye årganger. Endret populasjon, nå inkludert alle verdier.*
BEFOLKNING_ELDST_I_HUSHNR	*Endret data. Nye årganger. Endret populasjon, nå inkludert alle verdier.*
BEFOLKNING_ELDST_I_REGSTAT_FAMNR	*Endret data. Nye årganger. Endret populasjon, nå inkludert alle verdier.*
BEFOLKNING_HUSHNR	*Endret data. Nye årganger. Endret datatype fra alfanumerisk til numerisk*

The variable list may be expanded so that it shows all variables in the data bank (over 200 in Statistics Norway's data bank). Improvements will come in the near future, such as search functionality and subject divisions.

At the bottom you will find an overview of all versions for the specific data bank, as well as change log. By clicking on the version names, you will be sent to a new page showing the content of this particular version.

Note that variable information is currently only available in Norwegian.

As the illustration below shows, a complete variable overview is also found at the bottom left of the analysis area. Here it is possible to filter the variable list by entering parts of a variable name in the search field. The filter works both against variable description and the name itself. In this way, it becomes easier to find the variable in question.


Datasett
demografidata
10 variabler, 7 241 906 enheter
PERSONID_1
kjonn
faarmnd
sivstand
formue
innt05
mann
gift
alder
formuehøy
Registervariabler
familie
Antall familier i husholdningen
[BEFOLKNING_ANTREGSTAT_FAMNR_I_HUSHNF
Familienummer
[BEFOLKNING_REGSTAT_FAMNR]
Familietype
[BEFOLKNING_REGSTAT_FAMTYP]
Pensjon til etterlatte familiepleiere, mottak
[ETLATFAM2011FDT_MOTTAK]
NSD /

'e	SS	df	MS
Modell	1.0747154e+16	4	2.6867885e+15
Residual	8.6956061e+16	2.489938e+6	3.4922982e+10
Total	9.7703215e+16	2.489942e+6	3.9239153e+10

Antall Obs:	2489943
F(4, 2489938):	76934.679309
R²:	0.10999
Justert R²:	0.10999
Root MSE:	1.8687692e+5

innt05	Coef.	Std.feil	t	P>|t|	[95% Konf.	intervall]
mann	99052	242.134	409.079	0	98577.5	99526.6
gift	55131.8	276.388	199.472	0	54590.1	55673.5
alder	-1157.79	10.7419	-107.782	0	-1178.85	-1136.74
formuehøy	88753.7	387.744	228.897	0	87993.7	89513.6
Konst	2.4550213e+5	393.996	623.107	0	2.4472991e+5	2.4627435e+5

```
demografidata» history
  create-dataset demografidata
  import BEFOLKNING_KJOENN as kjonn
  import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
  import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
  import INNTEKT_BRUTTOFORM 2000-01-01 as formue
  import INNTEKT_WYRKINNT 2005-01-01 as innt05
  generate mann = 0
  replace mann = 1 if kjonn == '1'
  generate gift = 0
  replace gift = 1 if sivstand == '2'
  generate alder = 2000 - int(faarmnd / 100)
  drop if alder < 16
  generate formuehøy = 0
  replace formuehøy = 1 if formue > 600000
  correlate alder formuehøy
  regress innt05 mann gift alder formuehøy

demografidata» |
```

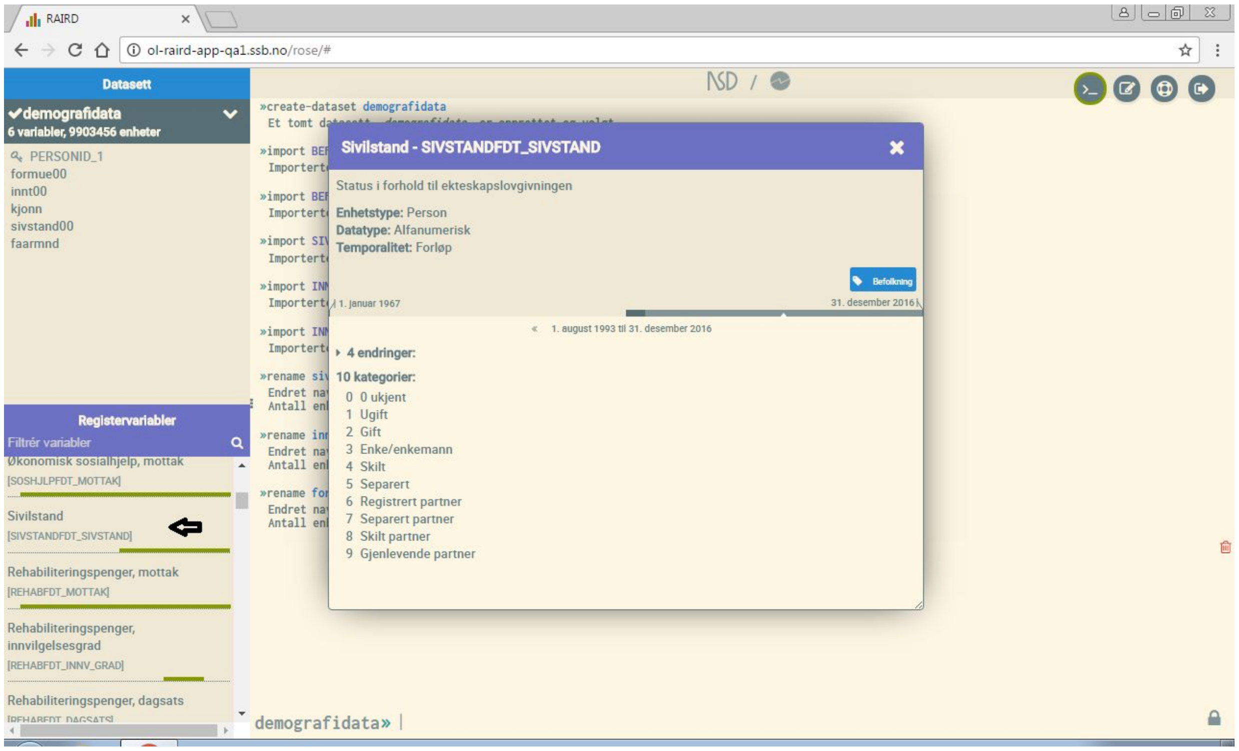
All variables are displayed with an associated timeline that marks the validity period(s), i.e. which time span is covered. Variables in microdata.no are three-dimensional - they contain time. By "clicking" on variables in the list, descriptive statistics and other information such as variable type can be retrieved.

In the example below, this is exemplified for the variable “Sivilstand” ("Marital status"). The variable is presented in a separate movable and resizable window. It provides detailed information about the variable:

- Key information: Variable name, variable label, variable description, variable type

- Detailed interactive timeline that allows for studies of changes in coding over time: Changes in the coding are displayed through different colors illustrating the time periods to which they apply. Clicking on the different fields in the timeline will bring up a list of the codes that were valid during the current period. In the example, the field that applies to August 1, 1993 - December 31, 2016 is marked, and a list of 10 categories then appears

- Information about changes: In the example, “4 endringer” ("4 revisions") is shown. This is the number of revisions compared to the previous time period. By clicking on "4 endringer", a list of the new codes will appear



```
»create-dataset demografidata
 Et tomt d
»import BEF
 Importert
»import BEF
 Importert
»import SIV
 Importert
»import INN
 Importert
»import INN
 Importert
»rename siv
 Endret nav
 Antall enh
»rename inn
 Endret nav
 Antall enh
»rename for
 Endret nav
 Antall enh
```

Sivilstand - SIVSTANDFDT_SIVSTAND

Status i forhold til ekteskapslovgivningen

Enhetstype: Person
Datatype: Alfanumerisk
Temporalitet: Forløp

Befolkning

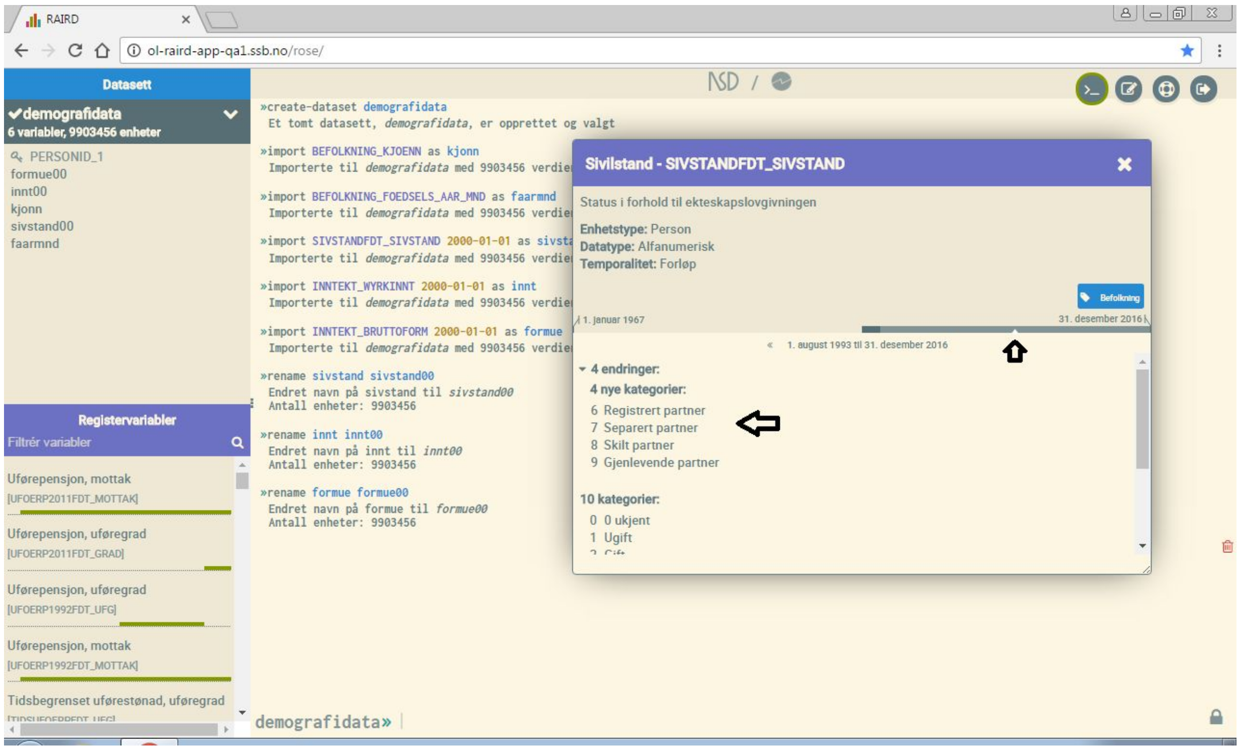
1. januar 1967 — 31. desember 2016

« 1. august 1993 til 31. desember 2016

4 endringer:

10 kategorier:

0 0 ukjent
1 Ugift
2 Gift
3 Enke/enkemann
4 Skilt
5 Separert
6 Registrert partner
7 Separert partner
8 Skilt partner
9 Gjenlevende partner



```
»create-dataset demografidata
 Et tomt datasett, demografidata, er opprettet og valgt

»import BEFOLKNING_KJOENN as kjonn
 Importerte til demografidata med 9903456 verdie

»import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
 Importerte til demografidata med 9903456 verdie

»import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivsta
 Importerte til demografidata med 9903456 verdie

»import INNTEKT_WYRKINNT 2000-01-01 as innt
 Importerte til demografidata med 9903456 verdie

»import INNTEKT_BRUTTOFORM 2000-01-01 as formue
 Importerte til demografidata med 9903456 verdie

»rename sivstand sivstand00
 Endret navn på sivstand til sivstand00
 Antall enheter: 9903456

»rename innt innt00
 Endret navn på innt til innt00
 Antall enheter: 9903456

»rename formue formue00
 Endret navn på formue til formue00
 Antall enheter: 9903456
```

Sivilstand - SIVSTANDFDT_SIVSTAND

Status i forhold til ekteskapslovgivningen

Enhetstype: Person
Datatype: Alfanumerisk
Temporalitet: Forløp

Befolkning

1. januar 1967 — 31. desember 2016

« 1. august 1993 til 31. desember 2016

4 endringer:

4 nye kategorier:

6 Registrert partner
7 Separert partner
8 Skilt partner
9 Gjenlevende partner

10 kategorier:

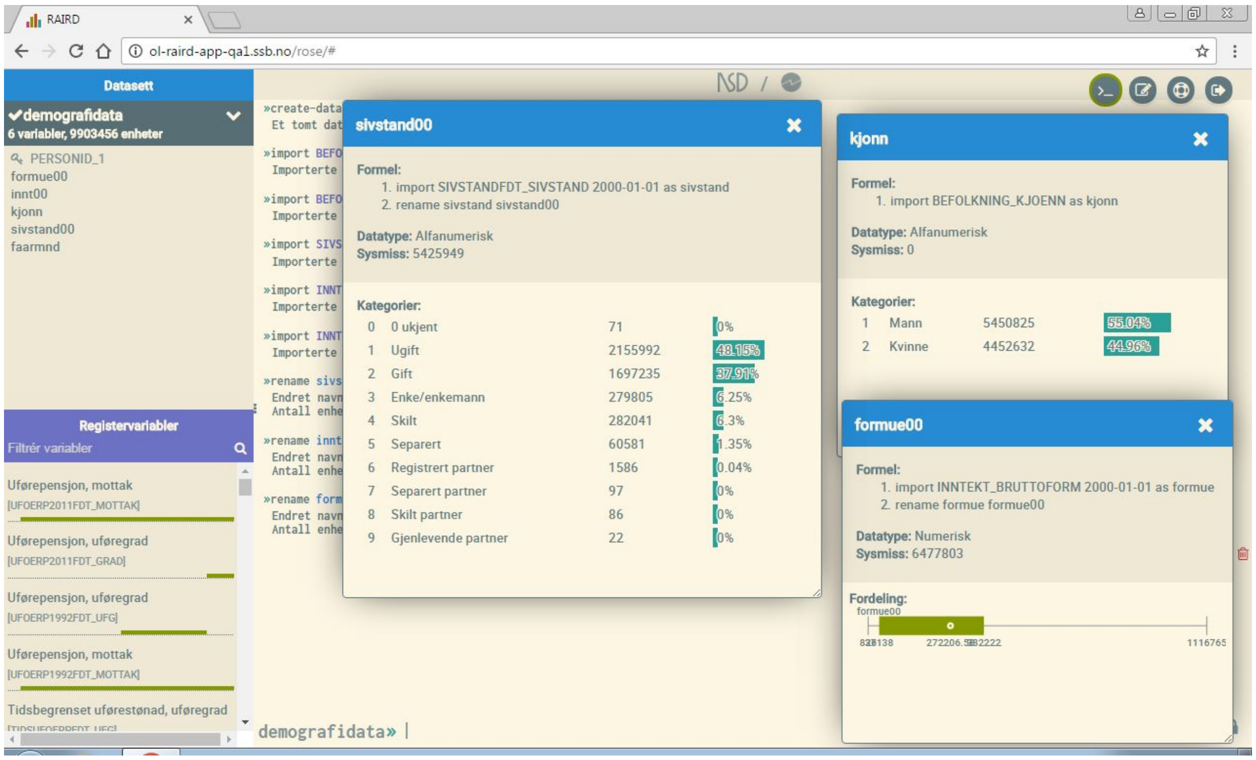
0 0 ukjent
1 Ugift

For variables imported to the user's dataset ("demografidata"), a slightly different type of information appears that may be useful when dealing with many different variables. This information adjusts continuously as changes are made to the variables, and appears in separate pop-ups when clicking on variables in the list of your current dataset:

- Formula: At the top of the window, a "creation history" is presented. This is a tool for insight on how a variable has been created or re-encoded

- Key information: Variable type and number of units with value for missing data (sysmiss)

- Frequency distribution and simple statistics: For categorical variables, frequency distribution is displayed, while for continuous variables, a standard boxplot is displayed with a box representing the two middle quartiles plus average and minimum/maximum value (so-called whiskers). If values become unreadable due to overlap, the pop-up window can be expanded.



demografidata
6 variabler, 9903456 enheter

PERSONID_1
formue00
innt00
kjonn
sivstand00
faarmnd

Uførepensjon, mottak [UFOERP2011FDT_MOTTAK]
Uførepensjon, uføregrad [UFOERP2011FDT_GRAD]
Uførepensjon, uføregrad [UFOERP1992FDT_UFG]
Uførepensjon, mottak [UFOERP1992FDT_MOTTAK]
Tidsbegrenset uførestønad, uføregrad [TIDSUFOERPFDT_UFG]

```
»create-data
 Et tomt dat
»import BEFO
 Importerte
»import BEFO
 Importerte
»import SIVS
 Importerte
»import INNT
 Importerte
»import INNT
 Importerte
»rename sivs
 Endret navn
 Antall enhe
»rename innt
 Endret navn
 Antall enhe
»rename form
 Endret navn
 Antall enhe

demografidata» |
```

sivstand00

Formel:
1. import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
2. rename sivstand sivstand00

Datatype: Alfanumerisk
Sysmiss: 5425949

Kategorier:

0	0 ukjent	71	0%
1	Ugift	2155992	48.15%
2	Gift	1697235	37.91%
3	Enke/enkemann	279805	6.25%
4	Skilt	282041	6.3%
5	Separert	60581	1.35%
6	Registrert partner	1586	0.04%
7	Separert partner	97	0%
8	Skilt partner	86	0%
9	Gjenlevende partner	22	0%

kjonn

Formel:
1. import BEFOLKNING_KJOENN as kjonn

Datatype: Alfanumerisk
Sysmiss: 0

Kategorier:

1	Mann	5450825	55.04%
2	Kvinne	4452632	44.96%

formue00

Formel:
1. import INNTEKT_BRUTTOFORM 2000-01-01 as formue
2. rename formue formue00

Datatype: Numerisk
Sysmiss: 6477803

Fordeling:
formue00
83138 272206.5 382222 1116765

1.3 The Command field

The command field is a central part of the user interface, where commands may be entered for execution. This includes importing/retrieving variables to the work space, creating descriptive variable statistics, commands for processing and encoding variables, administrative help commands (`help`, `history`, `clear` etc) or analysis. See Appendix A for a complete list of available commands.

The microdata.no analysis system has a built-in self-filling solution that suggests relevant commands based on what is typed. In most cases, it is sufficient to enter a few characters before the system is able to suggest the desired command. By pressing the <tab> key on the keyboard, the command is inserted correctly.

Most commands assume that additional information is entered, and self-filling also works in such contexts. The command `import` needs additional information on variable name and measurement time in order to be executed. If you want to import the variable *kjønn (gender)*, you can start by entering the letter "k". This will result in a list of all variables with the letter "k" in the name - in practice there will be many to choose from. When typing the next letter "j", the system will list all variable names containing the combination "kj". After entering "ø", the letter combination will be sufficiently unique for the system to reduce the number of alternatives into a few, and the user can select the correct variable with the arrow keys on the keyboard and then use the *<tab>* key. Following the variable selection, a measurement date need to be specified given that the variable does not contain fixed information such as "gender". The default value for the system is the last used date. If this is ok, use the *<tab>* key. If not, enter an optional new date instead. The system will also suggest relevant options at the end. For `import`, the option `as` may be used. This may be used to create an alias for the particular variable. The register variables in the database often have inconvenient and long names that may be renamed using the `as` option for more understandable/readable analysis and statistics outputs/prints.

1.4 Useful Commands

If you are wondering what kind of commands to use, you can start with `help`. This will present a list of all available commands. For those familiar with the analysis software Stata, most of the commands will be recognisable. The microdata.no analysis system uses a Stata-like syntax, which also means that commands are written in English.

If you want full information about a particular command, this can be solved by entering `help` followed by the command name, e.g. `help import`. An explanation with examples of use is then displayed. A complete list of commands and functions is shown in Appendices A and B.

Another useful command is `history.` It lists all commands that have been used in a session. This list can be copied and pasted into the system's script editor, or into a private document. This is an easy way to copy your work.

The command `clear` may be used to delete all content in the workspace if you want to start all over again. This should therefore be used with caution!

An alternative to `clear` may be to use the known key combination <*Ctrl*> + <*Z*>. This is a widely used "regret-functionality", where the last operation is made undone (takes you one step back). This may be done unlimited amount of steps, until only an empty work area remains. The key combination <*Ctrl*> + <*Y*> may be used to redo the last "undo". Apple users may use the combinations <*Cmd*> + <*Z*> and <*Cmd*> + <*Y*>.

A useful tool when using the command field is to use the <*arrow-key up*> on the keyboard. This will scroll you through all previous commands used in chronological order until you find the one you want to reuse. This will save you the work of having to manually enter the same command syntax several times. Often there is a need to run different variants of the same command syntax (e.g. with slightly different variables or parameters). In such cases, an already used command may be selected from the list to be rerun after some small adjustments.

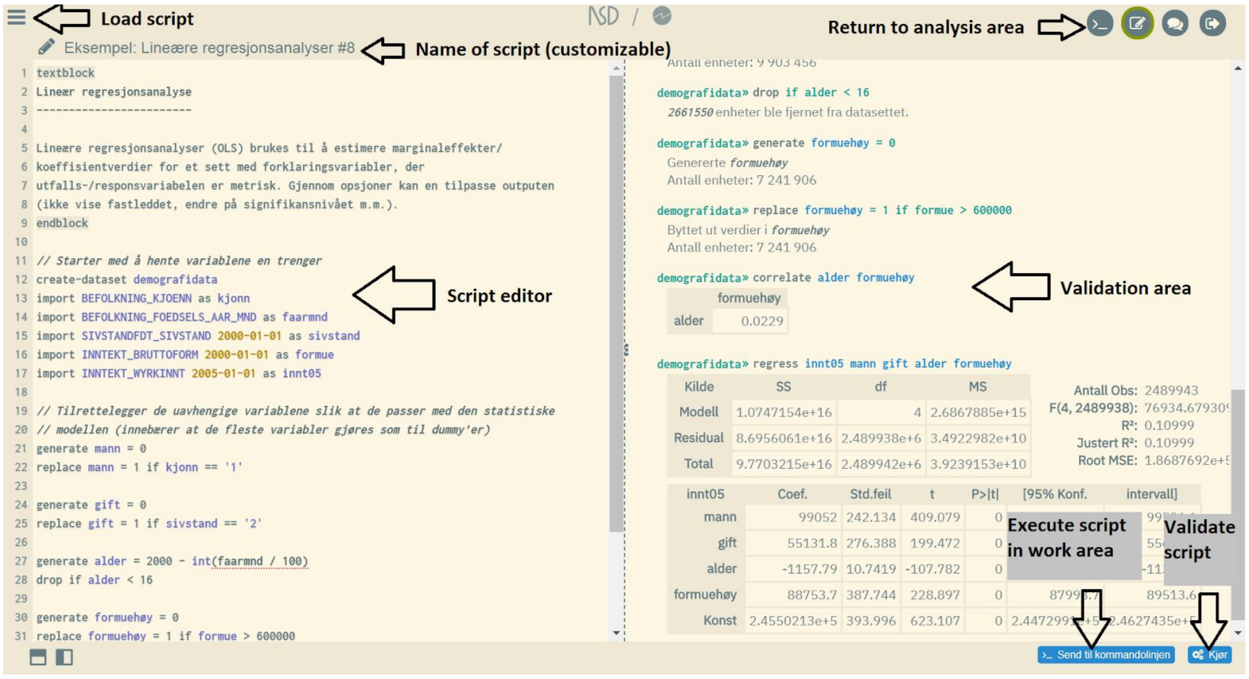
1.5 The Script editor

The script editor lets users enter sequences of commands that can be run together as a script. There are two options for executing a script: At the bottom right there are two buttons. "Kjør" ("Run") will run through the script on the right side of the script area and perform a validation (checks if everything is correct). The second button "Send til kommandolinjen" ("Send to command line") sends all commands to the work space and executes them there.

All command lines in a script are executed sequentially, and the result will continuously be displayed while running. In case of any errors in the syntax, the execution will be stopped where the error is located. In such cases, the error may be corrected and the script rerun.

When making adjustments to a script and performing a rerun, the work space will be replaced by the result of the new script run (using the "Kjør" button will only test the script and no changes will be made to the work space). Be sure not to overwrite your work if this is not preserved in the form of scripts.

Note that the system "remembers" previously run command sequences. So when executing exactly the same command script over again without changes, it will only take a few seconds to retrieve the result of the run. This principle also applies if preliminary parts of a script have been run earlier, where only the last part has been edited. Then the system will "remember" what has previously been run and only use resources to work its way through the part of the script that is changed.



```
textblock
Lineær regresjonsanalyse
------------------------

Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
koeffisientverdier for et sett med forklaringsvariabler, der
utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
(ikke vise fastleddet, endre på signifikansnivået m.m.).
endblock

// Starter med å hente variablene en trenger
create-dataset demografidata
import BEFOLKNING_KJOENN as kjonn
import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
import INNTEKT_BRUTTOFORM 2000-01-01 as formue
import INNTEKT_WYRKINNT 2005-01-01 as innt05

// Tilrettelegger de uavhengige variablene slik at de passer med den statistiske
// modellen (innebærer at de fleste variabler gjøres som til dummy'er)
generate mann = 0
replace mann = 1 if kjonn == '1'

generate gift = 0
replace gift = 1 if sivstand == '2'

generate alder = 2000 - int(faarmnd / 100)
drop if alder < 16

generate formuehøy = 0
replace formuehøy = 1 if formue > 600000
```

```
  Antall enheter: 9 903 456

demografidata» drop if alder < 16
  2661550 enheter ble fjernet fra datasettet.

demografidata» generate formuehøy = 0
  Genererte formuehøy
  Antall enheter: 7 241 906

demografidata» replace formuehøy = 1 if formue > 600000
  Byttet ut verdier i formuehøy
  Antall enheter: 7 241 906

demografidata» correlate alder formuehøy
            formuehøy
  alder        0.0229

demografidata» regress innt05 mann gift alder formuehøy
  Kilde     SS             df           MS                   Antall Obs:  2489943
  Modell    1.0747154e+16            4  2.6867885e+15   F(4, 2489938):  76934.67930
  Residual  8.6956061e+16  2.489938e+6  3.4922982e+10               R²:  0.10999
  Total     9.7703215e+16  2.489942e+6  3.9239153e+10        Justert R²:  0.10999
                                                               Root MSE:  1.8687692e+

  innt05          Coef.  Std.feil        t  P>|t|  [95% Konf.     intervall]
  mann            99052   242.134  409.079      0                 99
  gift          55131.8   276.388  199.472      0                 55
  alder        -1157.79   10.7419 -107.782      0                 -11
  formuehøy     88753.7   387.744  228.897      0     8799 .7     89513.6
  Konst     2.4550213e+5  393.996  623.107      0  2.447299  +5  2.4627435e+
```

1.5.1 Executing parts of a script

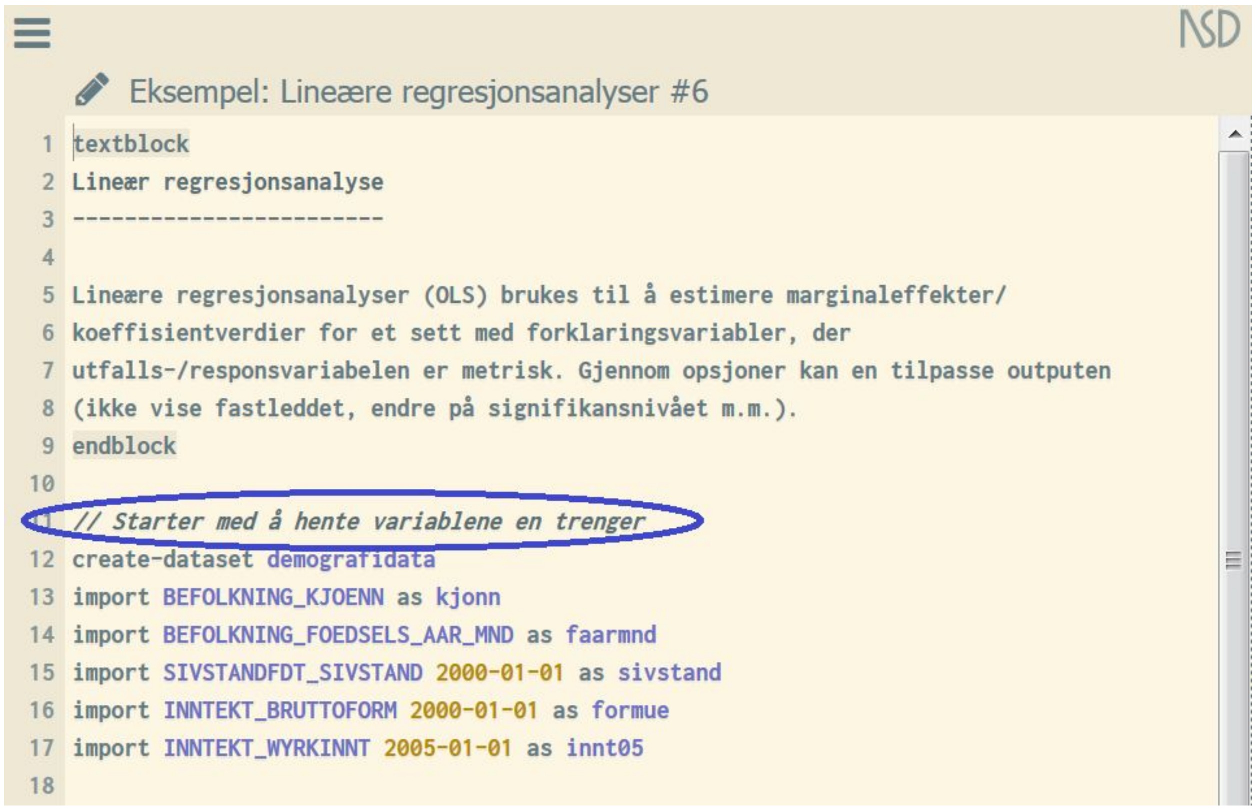
It is possible to execute only parts of a script. This may be done in two ways:

A) Mark out individual lines by defining them as help-text/comments

- Enter the characters "//" in front of the relevant lines you want to keep out. The system interprets everything that comes behind "//" as help-text/comments, and it will therefore be skipped from the execution

- In the next steps, the help-text marking may be removed gradually for more and more command lines until the entire script is completely executed. Note that the command lines previously executed are kept in the memory and will not be rerun. Only those lines where the "//" characters are removed will actually be rerun.

Example: Using comment/help-text functionality



```
textblock
Lineær regresjonsanalyse
------------------------

Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
koeffisientverdier for et sett med forklaringsvariabler, der
utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
(ikke vise fastleddet, endre på signifikansnivået m.m.).
endblock

// Starter med å hente variablene en trenger
create-dataset demografidata
import BEFOLKNING_KJOENN as kjonn
import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
import INNTEKT_BRUTTOFORM 2000-01-01 as formue
import INNTEKT_WYRKINNT 2005-01-01 as innt05

```

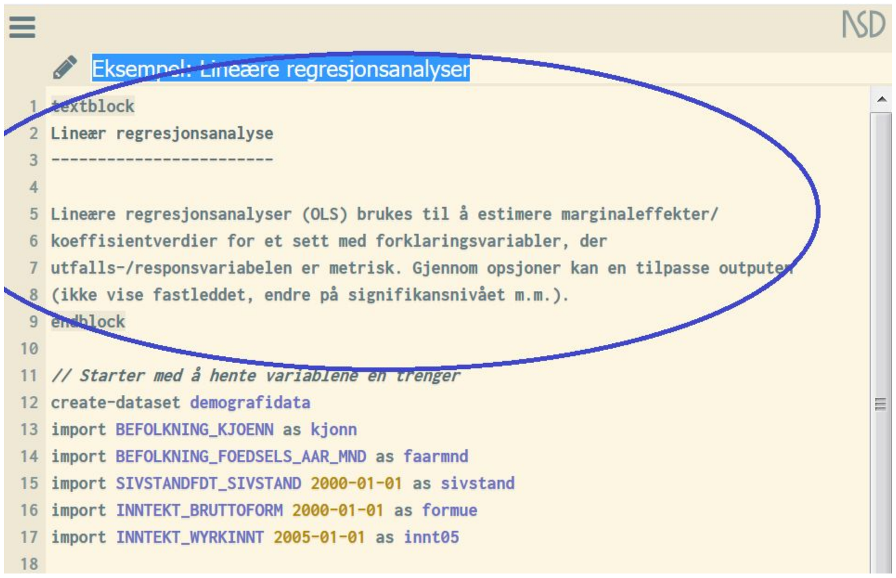
B) Selecting larger parts of a script by defining blocks as help-text

- Enter the keywords `textblock` and `endblock` before and after a block of command respectively. Everything in between will then be kept out of the execution. Note that the purpose of `textblock` and `endblock` is to enter comments for analyzes performed in the analysis area. The analysis area can therefore be filled up with the command lines that were defined as comments in the script, and it may look a bit messy. But as fewer and fewer lines are kept out from the execution, the analysis area will gradually contain less of this

- The advantage of `textblock` and `endblock` is that it is less time consuming if the number of lines to be marked out is extensive

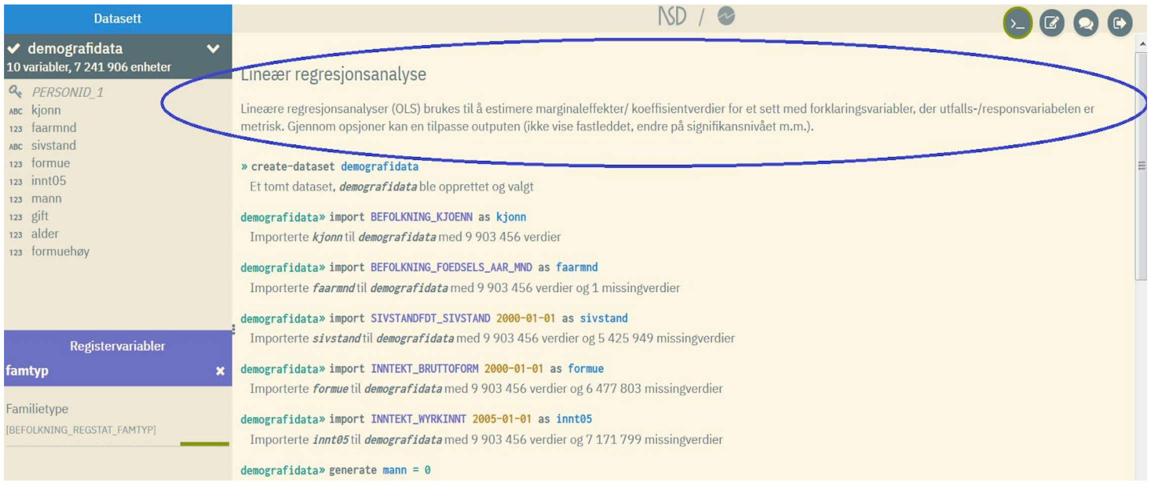
- Just like using the characters "//", the system will "remember" what has been run previously and will jump right down to the part of the script where the "text block mark" has been removed. Note that this does not apply when the textblock-functionality has been used on the preliminary parts of the script

Example: Using textblocks in scripts



Eksempel: Lineære regresjonsanalyser

```
textblock
Lineær regresjonsanalyse
------------------------

Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
koeffisientverdier for et sett med forklaringsvariabler, der
utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
(ikke vise fastleddet, endre på signifikansnivået m.m.).
endblock

// Starter med å hente variablene en trenger
create-dataset demografidata
import BEFOLKNING_KJOENN as kjonn
import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
import INNTEKT_BRUTTOFORM 2000-01-01 as formue
import INNTEKT_WYRKINNT 2005-01-01 as innt05

```



Lineær regresjonsanalyse

Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/ koeffisientverdier for et sett med forklaringsvariabler, der utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen (ikke vise fastleddet, endre på signifikansnivået m.m.).

```
» create-dataset demografidata
  Et tomt dataset, demografidata ble opprettet og valgt

demografidata» import BEFOLKNING_KJOENN as kjonn
  Importerte kjonn til demografidata med 9 903 456 verdier

demografidata» import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
  Importerte faarmnd til demografidata med 9 903 456 verdier og 1 missingverdier

demografidata» import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
  Importerte sivstand til demografidata med 9 903 456 verdier og 5 425 949 missingverdier

demografidata» import INNTEKT_BRUTTOFORM 2000-01-01 as formue
  Importerte formue til demografidata med 9 903 456 verdier og 6 477 803 missingverdier

demografidata» import INNTEKT_WYRKINNT 2005-01-01 as innt05
  Importerte innt05 til demografidata med 9 903 456 verdier og 7 171 799 missingverdier

demografidata» generate mann = 0
```

1.5.2 Script editor advantages

There are many advantages of actively using the script editor to execute sequences of commands:

- Works as a work backup
    - When naming a script, it is stored in the system and may be retrieved again later
    - Scripts may be saved as text format by copying over to a text document externally. This serves as extra security. If the work is lost for some reason, it may be retrieved from your external document file and pasted into the editor for rerunning. In this way all the analysis work will be recreated as it was originally.

Note: External script storage should be done in plain text format of the type ".txt" through applications such as Notepad etc. More advanced word processing tools such as Word and Google Doc perform text formatting that allows some characters to be altered, e.g. singular quotation marks. When this is then copied and pasted back into the microdata.no script editor, the characters may not be recognized and the system could actually shut down as a consequence.

- Scripts are a way of systematizing and recollecting your work. The order of command sequences can be adjusted, and other adjustments may also be performed, such as adding comments/help-text (see chapter 1.5.1) which makes it easier to recollect and easier for third-parties to understand what the purpose of the various operations is.

- Script works as a log of work (can be added to analysis reports to document your work)

- It is easy to make adjustments on an analysis. If there is a need to do things a little differently, the script may be edited and rerun. Edited scripts may be stored with new names. This makes it easier to document and compare results

- Using the scripting capability actively makes it easier to collaborate with others. Scripts may be sent in text format to other colleagues, e.g. via email

- Scripts may be edited in the same way as in Google Doc or other word processing programs such as Word: It is possible to edit by cutting, copying and pasting text, as well as marking text-blocks and moving them around as needed

- The system "remembers" previously run scripts given that they are unaltered. It will only take a few seconds to reproduce a previously run result. Note that if some parts in a script are altered and rerun, the system will treat it as a completely new set of commands, and it will take much longer time to execute it. If, on the other hand, there is a need to adjust command lines only at the end of a script, the system will jump right down and only use resources to process this part. The rest are retrieved from the memory.

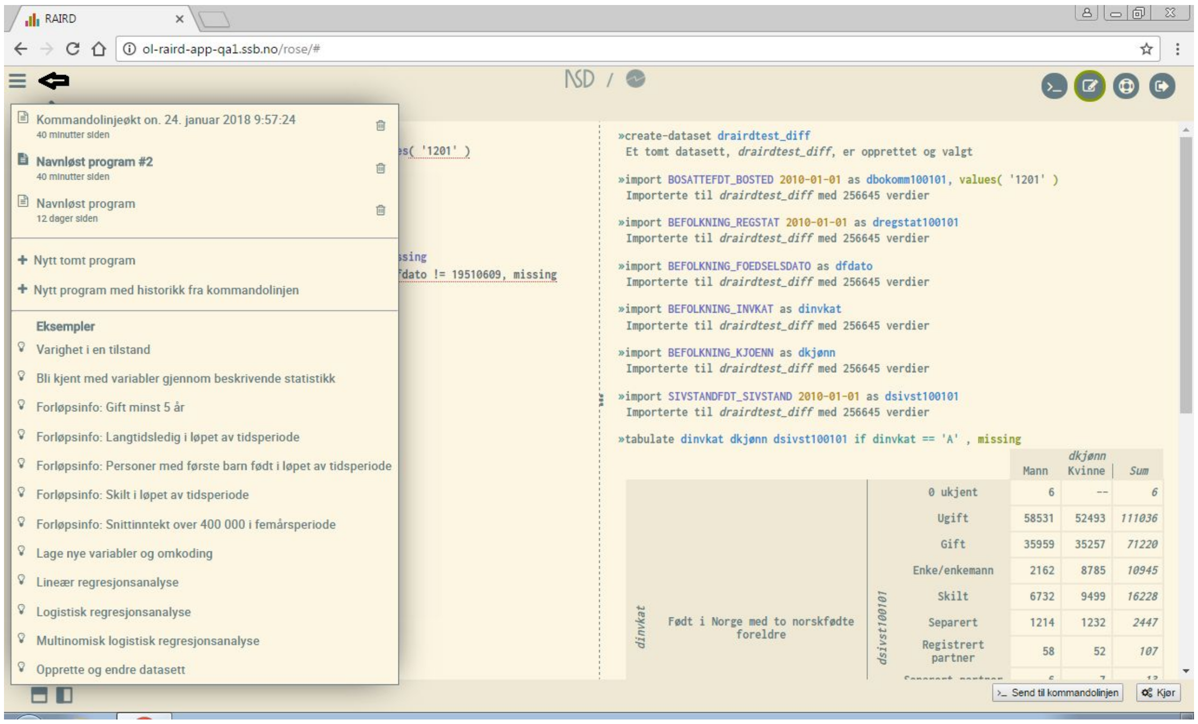
1.5.3 Organization of scripts

The image below shows the various possibilities for organizing scripts (programs). The menu button at the top left of the script window allows you to create a completely new (empty) program, or to retrieve all commands used in the active work window: “Nytt program med historikk fra kommandolinjen” ("New program with command line history").

The contents of active scripts are stored continuously with a default name (similar to Google Doc). By entering a custom title at the top of the script, where it says "Untitled program", the default name will be replaced by this. Any work done on the script will then automatically be saved with this name.

It is possible to store as many scripts as you wish by naming them with new names. The system will also periodically store the current (active) script at regular intervals (backup).

At the bottom of the program menu there are examples of command sequences for different types of tasks. They can be used as active scripts that can be run directly. They can also be edited and saved with new names (can be used as a template).



Kommandolinjeøkt on. 24. januar 2018 9:57:24
40 minutter siden

Navnløst program #2
40 minutter siden

Navnløst program
12 dager siden

+ Nytt tomt program
+ Nytt program med historikk fra kommandolinjen

Eksempler
- Varighet i en tilstand
- Bli kjent med variabler gjennom beskrivende statistikk
- Forløpsinfo: Gift minst 5 år
- Forløpsinfo: Langtidsledig i løpet av tidsperiode
- Forløpsinfo: Personer med første barn født i løpet av tidsperiode
- Forløpsinfo: Skilt i løpet av tidsperiode
- Forløpsinfo: Snittinntekt over 400 000 i femårsperiode
- Lage nye variabler og omkoding
- Lineær regresjonsanalyse
- Logistisk regresjonsanalyse
- Multinomisk logistisk regresjonsanalyse
- Opprette og endre datasett

```
es( '1201' )
ssing
fdato != 19510609, missing
```

```
»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_FOEDSELSDATO 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 dkjønn
 Importerte til drairdtest_diff med 256645 verdier

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

»tabulate dinvkat dkjønn dsivst100101 if dinvkat == 'A' , missing
```

				dkjønn		
				Mann	Kvinne	*Sum*
dinvkat	Født i Norge med to norskfødte foreldre	dsivst100101	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*

1.5.4 Troubleshooting using scripts

It is not easy to avoid errors in the command syntax when running scripts. This can be misspellings or logical errors that cannot be executed. The system will then stop running the script where the error is located and mark the appropriate line. An error message will also be provided.

What to do:

i) Check what may have gone wrong. Pay special attention to the line marked with errors. Run the part of the script that do not contain errors, i.e. until the line containing the error (se Chapter 1.5.1 on how to run portions of a script)

ii) Double check whether the syntax is correct, that the variable name is correctly spelled, that the date of import is valid (a variable may not have data for the current measurement time)

iii) Use statistical tools like the commands `tabulate` or `summarize.` See if there are any errors in the way the relevant variables are encoded. Also check if the value format is correct (numerical or alphanumerical)

iv) Dummy variables or categorical variables where at least one of the categories has few observations can lead to undesirable analysis results:
   - When performing regression analyzes, only units with valid values across all the included variables will be analyzed
   - Even if all dummy variables to be used in a regression analysis initially have sufficient numbers of observations for both values 0 and 1, this may not be the case if a number of units are kept out of the regression analysis due to missing values
   - The analysis may be stopped and an error message given. A solution can be to recategorize: Recode the variables in question in order to transfer units from the most populated categories into those with the least number of observations. Another solution may be to drop the problematic variables from the analysis

2. Creating and changing datasets

Data analyzes in microdata.no require that users start by connection to a data bank, then creating an empty dataset in order to import the required variables. Variable imports are done through the use of import-commands (see chapters 2.3.1, 2.3.2 and 2.4).

If you need to adjust the population or remove variables, the commands `drop` or `keep` are relevant to use (see chapters 2.6 and 2.7). Variables for deletion may be specified in the command expression. If no variable is specified, whole records are deleted conditioned on the IF-expression followed (on the same line).

2.1 Connect to data bank

When logging in to the analysis area for the first time, or when starting a completely new session, all the windows are blank. In order to create datasets, it is necessary to connect to available data banks. Normally, it is sufficient to connect to the data bank containing a substantial collection of register data offered by Statistics Norway. Microdata.no will in the near future offer data banks containing data also from other data sources.

It is optional which version of the data bank to connect to. The newest version is recommended, as it contains the newest variables and updates. By connecting to earlier versions, it is possible to compare possible effects caused by version differences. When new data measurements are added to an existing variable, this may inflict upon prior sequences of events. Therefore, new data bank versions will always be created in such cases.

In the variable overview found on the main web page, you will find a list of all available data banks and data bank versions, including all variables contained. Chapter 1.2 describes this in more detail.

Command for connection to data bank:

```
    require <databank:version> as <alias>
```

Example:

```
    require no.ssb.fdb:1 as fdb1
```

2.2 Creating a dataset

To be able to work with data and analyzes in microdata.no, one must always start by creating a dataset. This is done by typing the following command in the command field:

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

Example:

```
create-dataset mydataset
```

In many cases it is sufficient to create one dataset only, but in principle one can create as many as possible. An example where it is applicable to create several datasets is when working with variables organized differently (having different unit levels). In addition to a dataset on individual level (one record per individual), a user may also need to analyze other datasets organized with events as units (several records per individual).

2.3 Retrieving variables into a dataset

The next step is to fill your dataset with relevant variables.

All underlying variables in microdata.no are basically organized in the same way; as events:

individual id-number x value x start date x stop date

Microdata.no further distinguishes between four types of variables:

1) Event variables with sequences of events (each observation represents a state change, i.e. the variable changes value, with varying start and stop dates)

2) Constant variables with only one observation per unit (fixed information such as gender, date of birth, country of birth)

3) Statistical status variables measured at fixed times (start date = stop date, and only the value at this particular time in question is known)

4) Variables with cumulative annual aggregates that apply from 1/1 to 31/12 for each year (annual income etc.)

Datasets are built by using the command `import`, with a measurement date usually specified (the exception is variables with constant values such as gender).

Chapter 2.3.1 shows in detail how to use the command `import` to import variables into a dataset in the case with individuals as unit level. Chapter 2.3.2 shows alternative imports of variables with events as units (persons are represented by several observations over time, depending on the number of events that have occurred). In this case, the relevant command to use is `import-event`.

2.3.1 Datasets containing cross-sectional data

The command `import` is used for imports of the following types of information:

- Fixed information
- Cross-sectional information on specific custom dates (custom extractions from event variables)
- Status information on regularly predetermined measurement dates
- Yearly aggregated information

The name of the variable to import into your dataset is required, in addition to the measurement date. The system suggests relevant variables and dates through a self-filling feature that minimizes the chance of writing errors.

For each time the `import` is run, a new variable will be added to the work dataset (automatically linked with the respective individuals). The resulting dataset will consist of one observation per unit (individual), with an optional number of variables.

When importing fixed (constant value) variables, the measurement date is excluded from the expression:

```
import <variable> as <alias>
```

However, all other variables require a measurement date on the format *YYYY-MM-DD:*

```
import <variable> <measurement date> as <alias>
```

For statistical status variables, the regularly predetermined measurement dates must be used since the values in principle only will apply to these particular dates (one does not know the actual change dates for such variables). The microdata.no analysis system will in such cases suggest the relevant statistics dates through the self-filling function in order to help the user as much as possible. When importing cross-sectional data on custom dates, the last used date is proposed. For variables with annual cumulative measures, it is the annual value for the year in question that is imported, therefore the specific date does not matter as long as the correct year is used (January 1. or December 31. the particular year, the result will be the same).

Example: Data matrix using import (4 variables)

ID	**Variable 1**	**Variable 2**	**Variable 3**	**Variable 4**
`123456`	`1`	`200000`	`0301`	`1`
`135791`	`1`	`410000`	`0301`	`1`
`147036`	`2`	`515000`	`1201`	`sysmiss`
`159371`	`2`	`309011`	`1101`	`sysmiss`
`160505`	`2`	`357000`	`1101`	`1`
`173951`	`2`	`399000`	`0301`	`3`

Important:

The command `import` performs practically two operations:

a) Retrieves values for a given variable
b) Links the variable values onto the existing dataset through a so-called "left-join" merging

Linking/merging through “left-join” means that only values for units (individuals) in the existing dataset are imported. Therefore, it is important to start by importing a variable with a limited number of missing values, such as gender, country background or date of birth. On the other hand, if the first variable imported into your dataset is a variable that indicates sickness absence on a given date, your population will be defined by these individuals with a valid sickness absence measurement and it will not be possible to retrieve information about other people in later stages. In other words, the first import-step will define the population of the current dataset.

Note that the sample population may be trimmed along the way in the process of building up a dataset, through the commands `drop` and `keep`, c.f. chapter 2.6.

Units/individuals in an existing dataset that have a missing value for an imported variable will still be included in the sample, but will have a so-called sysmiss-values for the spesific variable (see chapter 2.6).

If you have a clear idea of which units (individuals) should be included in an analysis population, you may want to trim the population as early as possible. This could provide significant improvements in how fast the system works.

If the first variable imported into your dataset is universal, e.g. "Gender", your dataset will consist of most of the individuals from the total database, including people who are dead, emigrated or unborn at the specific time measurement. This can be solved by first importing the variable "BEFOLKNING_REGSTAT" measured at the corresponding measurement date, and then keeping all individuals who take the value '1' (= resident in Norway). Example:

```
require no.ssb.fdb:1 as fdb1
create-dataset demographics
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_REGSTAT 2005-01-01 as regstat05
keep if regstat05 == '1'
```

2.3.2 Datasets containing event information

In addition to retrieving information measured at selected dates, users may also make calculations based on events over time. E.g. one may be interested in finding individuals who got married, became unemployed, or who were unemployed for over 6 months during a given time-span. The command `import-event` can be used for this purpose. It performs a variable import where *all* records (= events) per unit (= individual) are retrieved over a specified time span. In addition to the variable name, two time-points are required in the expression: Start- and stop-date. All events that have happened between the two dates will be retrieved to your dataset, i.e. all events that overlap the time-interval. The dataset will contain a varying number of records per unit (individual) depending on how frequent change-events have occured.

Note that only one event-based variable can be imported into a given dataset, and that such a dataset can not contain other variables. If there is a need to work on several event-based variables, one dataset needs to be created for each variable. Syntax expression for import of event-based information:

```
create-dataset <dataset>
import-event <variable> <start date> to <stop date> as <alias>
```

Example: Data matrix using import-event (time interval: 2000-01-01 - 2003-01-01)[1]

ID	**Start**	**Stop**	**Variable**
123456	2000-01-01	2000-05-30	1
123456	2000-05-31	2001-12-31	4
123456	2002-01-01	2003-08-15	2
135791	2000-04-10	2002-03-03	2
135791	2002-03-04	2002-11-11	3
147036	2002-02-28	2004-07-16	1

[1] Note: All events overlapping the time-period 2000-01-01 - 2003-01-01 are retrieved

2.3.3 Cross-sectional vs. event-based datasets

Unlike cross-sectional datasets that are built through the command `import,` this is not possible through the command `import-event.` Data extraction at the event level will always result in different numbers of records per individual, and it will make little sense to link such extracts into a common data set. A new dataset must therefore be created for each event-based data extraction (see section 2.3.2).

The purpose of event-organized data extraction is, as mentioned, to make calculations based on events over time through the command `collapse`. This will transform the event-based dataset into a unit-level data set (one record per individual) with the aggregated statistical measure being the new variable value (measured over the specified time span), allowing the variable to be linked with cross-sectional datasets for further analysis.

Chapter 2.8 describes the method for linking datasets together.

2.4 Datasets containing regular time measurements (panel data)

To be able to perform advanced regression analyzes in the form of panel data analysis, data must be organized in a different way compared to regular regression analyzes. Panel data are datasets in which each unit takes values for all included variables measured over a specified number of times. This has the advantage that the time component can be included in analyzes, and the databases become much larger, often resulting in analyzes of better quality.

There is a large battery of panel data analysis techniques, the distinction goes on which assumptions are made about the variability of the variables over time. Common variants used are fixed effect and random effect analyzes. This analysis form will be reviewed in chapter 5.6.

Data to be used in panel data analysis must be imported as follows:

```
create-dataset <dataset>
import-panel <variable list> <measurement date list> as <alias>
```

Example: Data matrix using import-panel (3 variables, 3 measurements)

ID	**Time**	**Variable 1**	**Variable 2**	**Variable 3**
123456	2000-01-01	1	200000	0301
123456	2001-01-01	1	210000	0301
123456	2002-01-01	2	215000	1201
135791	2000-01-01	2	305011	1101
135791	2001-01-01	2	301000	1101
135791	2002-01-01	3	299000	0301
147036	2000-01-01	1	150000	2030
147036	2001-01-01	1	159000	2030
147036	2002-01-01	3	199000	0301

Note:

- Panel datasets quickly become very large, since all units / individuals in the data set are measured T times, where T stands for the number of measurements. This is especially true if you import many variables as well

- A good practice when creating panel datasets is to first create a population of appropriate size, then duplicate this and finally import panel data into the empty data set of the duplicate population.

Example: Create population, duplicate units into new data set, and finally import panel data for the given population (= residents in Oslo per January 1., 2010, aged 18-39)[1]

```
require no.ssb.fdb:1 as fdb1

create-dataset population
import fdb1/BOSATTEFDT_BOSTED 2010-01-01 as residence
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
generate age = 2010 - int(birth_year_month/100)
keep if age >= 18 & age < 40 & residence == '0301'

clone-units population paneldata

use paneldata
import-panel fdb1/INNTEKT_WLONN fdb1/SIVSTANDFDT_SIVSTAND
fdb1/BOSATTEFDT_BOSTED 2011-12-31 2012-12-31 2013-12-31 2014-12-31
2015-12-31
```

[1] Panel datasets are created using a single import-panel command. Multiple batches cannot be imported into the same panel dataset. Nor is it possible to mix common cross-sectional data and / or event-based data with panel data.

2.5 How to navigate between datasets

The following command syntax can be used to navigate between datasets:

```
use <dataset>
```

However, when a new dataset is created through the command `create-dataset,` you will move automatically into this dataset.

2.6 Population filtering

A filter option may be used to specify which values to be included in your dataset, f.x. if only females are to be imported:

```
import BEFOLKNING_KJOENN as gender, values ( '2' )
```

Alternatively, variables may first be imported as usual, followed by a trim procedure using the commands `drop` or `keep`:

```
import BEFOLKNING_KJOENN as gender
drop if gender == '1'
```

In microdata.no, if-expressions may be used in many contexts, also in relation to the `drop` and `keep` commands. The following common logical operators are possible to use:

- Larger than	>
- Less than	<
- Equal to	==
- Larger than or equal	>=
- Less than or equal	<=
- Not equal	!=
- Or	|
- And	&

The following expression will remove individuals under 18 years from your population:

```
keep if age >= 18
```

Values for missing data can be assigned as follows:

```
sysmiss(<variable>)
```

Individuals with no data on wage income can be removed from your population as follows:

```
drop if sysmiss( wage )
```

It is also possible to create random samples from a dataset. The command `sample` may be used for such purposes. For more about syntax and examples, use the command `help sample`.

2.7 Removing variables from datasets

In an analysis situation there is often a need to remove some of the variables first imported, as they are considered irrelevant. For instance, some variables are used solely for the purpose of deriving values for new variables, and are considered redundant after the particular operation is finished.

Streamlining a dataset is done through the command `drop`, where the name of the redundant variable is specified:

```
drop <variable>
```

As we have seen, the drop-command can be used both to remove units (= rows in the data matrix), see chapter 2.6, **and** variables (= columns in the data matrix).

2.8 How to link datasets

Datasets are usually built up through the command `import,` which adds one and one variable measured at specified custom dates. Such variables have the same unit type, i.e. persons represented by the unique identifier `PERSONID_1.` The linking is done automatically through the analysis system, so that the user only has to deal with the `import-`command, specifying variable name, measurement date and optional alias.

The analysis system makes it also possible to analyze data measured over other unit types such as event level, municipal level, family level, course level, etc. Data with unit types other than persons cannot be imported directly into an individual level dataset. They must first be processed into the appropriate unit level given by the variable to be used as a link key in the target data set. Only then the datasets can be linked together using the command `merge.`

Data at a lower unit level than person, e.g. event or course level, must be aggregated to the person level using the command `collapse()` before using `merge.`

Data at a higher unit level than person, e.g. municipality or family level, however, can be connected to the personal data set using the corresponding variable in the target data set (using it as a link key).

See chapter 2.10 for examples on how to link information on parents, families and courses into your individual level dataset. The latter illustrates the interconnections of data at a lower level than persons (course data are information on ongoing education represented by the relevant course/subject taken at a given time, where people can take several courses simultaneously). Data on parents and families illustrate interconnections of data at a *higher* level than person.

2.9 Examples: Creating and revising a dataset

```
textblock
Start by importing the necessary variables

First, a connection to the data bank is made, then a dataset called demographics is created. All
imports and processing will take place in this particular dataset until actively changing dataset through
the command `use <dataset>`

Start- and stop dates are also provided through the import-command
endblock

require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 as maritalstate
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth

// Rename variables by adding postfix referring to year
rename maritalstate maritalstate00
rename wealth wealth00

// Drop variable gender from dataset
drop gender

// Keep only married persons
keep if maritalstate00 == '2'
```

2.10 Examples: How to link data with unit levels other than individual level

Example: Linking of parental information

```
textblock
How to use parental information in analyzes
----------------------------------------------------------

The database contains variables for father’s and mother’s birth identification number respectively,
which makes it possible to link parental information with an individual based dataset.

The command `merge` makes it possible to link datasets. The unit identification variable of the source
dataset is used by default, unless customized through an "on"-option".

In this example, separate datasets are made for fathers and mothers, which are linked to a personal
level dataset via the key-link variables `fnr_far` and `fnr_mor`.
endblock

//Connect to data bank
require no.ssb.fdb:1 as fdb1

// Make an individual level dataset with links to father and mother
create-dataset persondata
import fdb1/INNTEKT_WYRKINNT 2010-01-01 as income
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/NUDB_BU 2010-01-01 as edu
import fdb1/BEFOLKNING_FAR_FNR as fnr_father
import fdb1/BEFOLKNING_MOR_FNR as fnr_mother

// Retrieve information on father, and link to individual level dataset
create-dataset fatherdata
import fdb1/INNTEKT_WYRKINNT 2010-01-01 as income_father
import fdb1/NUDB_BU 2010-01-01 as edu_father
merge income_father edu_father into persondata on fnr_father

// Retrieve information on mother, and link to individual level dataset
create-dataset motherdata
import fdb1/INNTEKT_WYRKINNT 2010-01-01 as income_mother
import fdb1/NUDB_BU 2010-01-01 as edu_mother
```

```
merge income_mother edu_mother into persondata on fnr_mother

// Perform regular linear regression analysis for testing of causality between personal and parental
income

use persondata
generate male = 0
replace male = 1 if gender == '1'

destring edu, force
generate highedu = 0
replace highedu = 1 if edu >= 700000 & edu < 999999
replace highedu = edu if sysmiss(edu)

destring edu_father, force
generate highedu_father = 0
replace highedu_father = 1 if edu_father >= 700000 & edu_father < 999999
replace highedu_father = edu_father if sysmiss(edu_father)

destring edu_mother, force
generate highedu_mother = 0
replace highedu_mother = 1 if edu_mother >= 700000 & edu_mother < 999999
replace highedu_mother = edu_mother if sysmiss(edu_mother)

summarize income income_father income_mother
histogram income_father, percent
histogram income_mother, percent
correlate income_father income_mother
tabulate highedu_father highedu_mother

regress income male income_father income_mother highedu highedu_father highedu_mother
```

Example: Linking of family level data

```
textblock
Aggregere opplysninger til familienivå
----------------------------------------

Individuals can be linked to a family number that can be used to aggregate into family-level
information. Individuals belonging to the same family will be registered with the same family number
consisting of the person ID of the oldest person in the family.

In this example, a personal dataset is first created and then filtered down to persons in families
consisting of married couples with small children (code 2.1.1). Then, demographic information is
retrieved.

Family income is information at family level, i.e. unit = family. Therefore, a new dataset must be
created for this purpose (datasets cannot consist of variables with different unit types). Occupational
income is then imported at person level, and next the command `collapse (sum)` is used to sum the
incomes at family level (`by (famnr)`). The result is a dataset with family as unit.

Finally, family income is linked to the person dataset through the command `merge`.
endblock

//Connect to data bank
require no.ssb.fdb:1 as fdb1

// Create a dataset for persons in families consisting of married couples with small children
create-dataset persondata
import fdb1/BEFOLKNING_REGSTAT_FAMTYP 2010-01-01 as famtype
tabulate famtype
keep if famtype == '2.1.1'

// Add various demographic information
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
generate age = 2010 - int(birth_year_month/100)
import fdb1/BOSATTEFDT_BOSTED 2010-01-01 as residence
generate county = substr(residence,1,2)
import fdb1/BEFOLKNING_BARN_I_HUSH 2010-01-01 as children
```

```
// Create dataset for generating total occupational income per family => unit = family
create-dataset famdata
import fdb1/BEFOLKNING_REGSTAT_FAMNR 2010-01-01 as famnr
import fdb1/INNTEKT_WYRKINNT 2010-01-01 as work_income
collapse (sum) work_income, by(famnr)
rename work_income fam_income

// Link family income to person dataset (unit = persons)
merge fam_income into persondata on PERSONID_1

// Produce family statistics. The family number consists of the person ID of the oldest person in the
family. By removing individuals with missing income, the result is a dataset with family as unit. All
personal information will apply to the oldest person in the family

use persondata
drop if sysmiss(fam_income)

rename age age_oldest
rename gender gender_oldest

define-labels countytxt '01' 'Østfold' '02' 'Akershus' '03' 'Oslo' '04' 'Hedmark' '05' 'Oppland' '06'
'Buskerud' '07' 'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder' '11' 'Rogaland' '12'
'Hordaland' '14' 'Sogn and Fjordane' '15' 'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'

assign-labels county countytxt

tabulate county

histogram age_oldest, discrete
histogram children, percent

tabulate children
tabulate children, cellpct
tabulate children gender_oldest

summarize fam_income
barchart (mean) fam_income, by(county)
barchart (mean) fam_income, by(children)
histogram fam_income, freq
histogram fam_income, by(children) percent
```

Example: Linking/extraction of course level data

```
textblock
Retrieve information about ongoing education (courses)
-------------------------------------------------

Information about ongoing education (so-called course data) exist with courses as unit level (with
associated course identifier). Courses are given by the combination person x course type, where each
individual can in practice be represented with several course types at the same time.

Since the course data does not have persons as unit, these cannot be imported into the personal
dataset in the usual way, but must be linked by using the command `merge`.

First, add a link between the course ID and the person ID in the course dataset, and then aggregate to
person level using the `collapse`-command. Finally, link the aggregated course data to the personal
dataset.

In this example, a personal dataset is first created consisting of persons resident in Norway (regstatus
== '1') per 2010-01-01. Thereafter, history of ongoing education is retrieved for the period 2010-2012,
where only data on higher education (master or higher, level 7 and 8) is kept. The `collapse (count)`
command is used to count the number of observations with ongoing education per individual over the
period 2010-2012, and the result is then linked to the personal dataset for further analysis.

NB! Note that the variable `course type` after using `collapse` will consist of values for the current
statistics being generated, in this case the number of observations (`count`).
endblock

//Connect to data bank
require no.ssb.fdb:1 as fdb1

// Create person-level dataset for residents in Norway per 2010-01-01 with variable gender
create-dataset persondata
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_REGSTAT 2010-01-01 as regstatus
keep if regstatus == '1'

// Retrieve persons taking higher education in the period 2010-2012
create-dataset coursedata
import-event fdb1/NUDB_KURS_NUS 2010-01-01 to 2012-01-01 as coursetype
destring coursetype, force
```

```
keep if coursetype >= 700000 & coursetype < 999999

// Add a link between kurs-ID and person-ID to the course dataset
create-dataset link_course_person
import fdb1/NUDB_KURS_FNR as fnr
merge fnr into coursedata

// Produce statistics (collapse) over number of events per individual containing information on higher
education, and link with person dataset
use coursedata
collapse (count) coursetype, by(fnr)
rename coursetype courses
merge courses into persondata

// Generate table showing the number of people taking higher education during 2010-2012
use persondata
generate edu_high = 0
replace edu_high = 1 if courses >= 1
tabulate edu_high gender
```

3. Variable adaptations

Most variables imported into a dataset need to be recoded before further analysis. Further, there is usually a need to create separate variables based on the imported information. This is done through the commands `generate`, `replace` and `recode`.

3.1 Creating new variables and recoding: generate/replace

The command `generate` is a tool for generating new variables. It requires name of variable and what values it should have. This can be a specific value or a value based upon an equation/ formula. If-conditions are used to indicate which cases/units are to receive a value.

Note that `generate` can only be used to specify one value. If you want to specify more values (based on other conditions), the `replace`-command can be used to complete the process.

The `generate`-command can also be used to copy other variables: `generate` *<new variable>* = *<old variable>*. This too can be combined with if-conditions.

Example on how to code the dummy "male" derived from the source variable BEFOLKNING_KJOENN (contains information on gender, where the alphanumerical value `'1'` represents males):

```
import BEFOLKNING_KJOENN as gender
generate male = 1
replace male = 0 if gender != '1'
```

There are many possible ways to create logical conditions, all of which will give the same result. The dummy variable “male” could also be coded as follows:

```
import BEFOLKNING_KJOENN as gender
generate male = 0
replace male = 1 if gender == '1'
```

Note the following when recoding or generating new variables:

- "=" are used to set values through the commands `generate` or `replace`. However, "==" are used in relation to logical if-expressions.

- Values for alphanumerical variables need to be specified with singular quotation marks (`'1', '2', …` etc), while numerical values are specified without quotation marks (`1, 2, ….` etc).
  - The value format are found by looking at the specific variable on the top left (the dataset window) or bottom left (registry database window).

- Code for missing data are specified the following way: `sysmiss(<variable>)`

  - Example (removing units with missing data on "gender":

    ```
    import BEFOLKNING_KJOENN as gender
    generate male = 1
    replace male = 0 if gender != '1'
    drop if sysmiss( gender )
    ```

- The following logical operators may be used in if-expressions:

Larger than	>
Less than	<
Equal to	==
Larger than or equal to	>=
Less than or equal to	<=
Not equal to	!=
Or	|
And	&

- Dummy variables need to be numerical of methodically reasons, and must also take the values 1 and 0. A dummy variable cannot take only the value 1 as this will give unwanted results or error messages when performing regression analysis. In practice, one must therefore be careful to code all units that do not have the "success" value with the value 0 (see example at the top of the previous page)

- When using dummy variables in if-expressions, there is no need to specify the value 1.

  - Example: The expression `tabulate sivilstatus if male == 1` will give the exact same result as `tabulate sivilstatus if male`

- If the purpose of the adaptation of the variables is to perform regression analyzes, categorical values should be coded in numerical form. If not, there is a risk that the system will not accept the variable input, and an error message may occur when running commands such as `regress`, `logit`, etc.

- For methodological reasons, categorical variables should usually be arranged as dummy variables such as in the example of the variable "mann" above. This also applies to multi-category variables (more than two categories) such as "Education level". In such cases, a set of dummy variables which, in combination, corresponds to the multi-category variable need to be created (in practice, each category minus the reference/base category need to be represented by separate dummy variables, where the estimates are interpreted relative to the reference category). The process of creating sets of dummy variables can however be automated by using the prefix "i." in front of the variable name in the regression expression. Then the lowest value is automatically used as the reference value.

- Missing values: Be aware that all units where at least one of the included variables has a missing value are excluded from regression runs. Variables with many missing values that are not recoded will then result in the regression analysis being performed on a much smaller data set than planned. This is something one should be aware of during the facilitation. In the gender example, there will typically be few units/individuals with missing value, but there may be other variables that indicate e.g. social security benefits such as "disability". Here, a majority will have missing value, and only those who are disabled will have a valid value. In such cases one should code in the following manner:

```
import PENSJONER_UFOERGRAD 2010-01-01 as disabilitydegree
generate disabled = 1
replace disabled = 0 if sysmiss( disabilitydegree )
```

- Missing Values for income variables: This will typically refer to all people with income = 0. If these need to be included, they should be recoded into 0’s:

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

3.2 Variable recoding: recode

The command `recode` can be used to recode variables, as an alternative to `replace`. This is a useful tool when recoding many values/categories within a single variable. The command makes it possible to complete the full set of recodings in a single command expression, which contributes to shorter processing time. It is also possible to recode multiple variables at a time, using `recode`.

Example on coding the variable "male" derived from `BEFOLKNING_KJOENN` (contains data on gender where male = '1' and female = '2') using `recode`:

```
import BEFOLKNING_KJOENN as male
destring male, force
recode male (2 = 0)
```

Note that `recode` only applies to numerical variables. Therefore, alphanumerical variables (string variables) first need to be converted into numerical value format by using the command `destring,` as shown in the example above. The option `force` is recommended, as it ensures that the conversion is carried out also in cases where single values containing non-numerical characters occur (these are set to missing value). See chapter 3.7 for more information on `destring`.

Examples on how to encode groups of numbers for the variables *var1* and *var2*:

- `recode var1 var2 (1 2 3 = 0)` *(values 1-3 recoded into 0)*
- `recode var1 var2 (1/7 = 0)` *(values 1-7 recoded into 0)*
- `recode var1 var2 (1/7 = 0) (nonmissing = 1) (missing = 99)`
  (other valid values recoded into 1, missing values recoded into 99)
- `recode var1 var2 (1/7 = 0) (* = 99)` (all other values recoded into 99)

3.3 Use of functions

In addition to the basic mathematical operators

```
=, +, -, /, *, ( , ),
```

microdata.no gives access to a large number of functions to be utilised in order to generate variables. A specific example is the case of recoding residency from municipality into county level. As data on residency takes alphanumerical values on municipality level by default (= four-digit code where the first two-digits represent county number whereas the last two-digits specifies the municipality within the county), the function `substr()` is needed in order to retrieve the first two digits representing counties:

```
generate county = substr(residency,1,2)
```

The input parameters “1” and “2” inside the substr()-expression are referring to the starting position and the number of characters to read respectively. The municipality of Bergen are represented by the value '1201'. Retrieving the first two digits will result in the value ‘12’ which represents the county of Hordaland.

Another typical use case for `substr()` is when there is a need for information on educational level on a higher aggregated level than the default 6-digit code level. Using an educational division on 1- or 2-digit level is very common. This function will suit as a very useful tool for such a purpose.

Other important functions are `round()`, `int()`, alternatively `floor()`. These are useful for the purpose of transforming decimal numbers into integers or to retrieve subvalues. `round()` rounds decimal numbers the regular way, while `int()` and `floor()` rounds downwards. If e.g. there is a need to retrieve the birth year from the numerical variable `yearmonth` (year and month on the format *YYYYMM*), the following expression can be used:

```
generate byear = int( yearmonth / 100)
```

This expression will generate birth year by dividing by 100 and keeping the integer number (skipping the decimal digits). In practice, this operation retrieves the first four digits from a numerical 6-digit value. For example, to retrieve the value 2010 from the numerical value 201006 in order to calculate age per 2013, the following expression can be used:

```
generate age = 2013 - int( yearmonth / 100)
```

If birth date is represented by an 8-digit numerical number (*YYYYMMDD*), the expression need to be adjusted as follows:

```
generate age = 2013 - int( birthdate / 10000)
```

Appendix B presents a list of all available functions. Note that each function have requirements regarding which type of variable formats they are suited for, e.g. `substr()` requires alphanumerical values only.

3.4 How to generate time-aggregated values - collapse

The command `collapse` is a tool for statistical measurements aggregated over a specified time span. Examples may be calculations of a state duration measured over a given time interval, retrieval of status in a given time interval, retrieval of number of occurrences in given states in a given time interval, or summation of values over a given time interval.

This is done on event-organized data sets (see chapter 2.3.2) through the following command:

```
     collapse (<aggregate measure>) <dataset>, by(<unit-id>)
```

Type of aggregation is required as input in the parenthesis following `collapse,` and then the name on an event organized dataset. Aggregation type may be as follows:

- Max
- Min
- Mean
- Count
- Sum

The option `by(`*`<unit-id>`*`)` is used to specify which unit type to aggregate over. This will usually be individuals, given by the unit identification number contained by the key variabel `PERSONID_1`.

Example 1: Calculate the number of times the individuals have changed their marital status during 2000-2005

```
require no.ssb.fdb:1 as fdb1
create-dataset maritalevent
import-event fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 to 2005-01-01 as
     maritalperiod
collapse ( count ) maritalperiod, by(PERSONID_1)
rename maritalperiod maritalstates
replace maritalstates = maritalstates - 1
tabulate maritalstates
```

Example 2: Calculate the number of divorces per individual during 2000-2005

```
require no.ssb.fdb:1 as fdb1
create-dataset maritalevent
import-event fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 to 2005-01-01 as
     maritalperiod
keep if maritalperiod == '4'
collapse ( count ) maritalperiod, by(PERSONID_1)
rename maritalperiod divorces
tabulate divorces
```

Note that the variable `maritalperiod` initially contains data on marital status (each new record represents a change in marital status). However, through the steps in the examples, the variable is transformed from containing event level data into containing the `count`-value measured over the 2000-2005 period for the specific unit level (= individual). Thus, following the `collapse`-procedure, the variable `maritalperiod` will now contain the number of marital statuses measured per individual over the period (example 1) or the number of divorces per individual measured over the same period (= the number of records containing the value ‘4’ which represents the status “divorced”) (example 2).

NB! In order to be able to continue working with the aggregated value generated through `collapse`, the dataset needs to be linked with the other variables placed in the main analysis dataset built through the `import`-procedure (see chapter 2.3.1). See chapter 2.8 on how to do this.

3.5 Renaming variables

It is appropriate to have understandable and intuitive variable names, and renaming variables is fully possible. This is easily done through the following command:

```
rename <variable_name_old> <variable_name_new>
```

As microdata.no variables are strongly time-related, a good rule will be to include time indication in the name (e.g. year). Example:

```
rename sivstand sivstand00
```

3.6 Using labels

Tabulations and other statistical output become more understandable by attaching text-labels to the various categorical variable values. Microdata.no makes it possible to define a set of value labels to be attached to all variables sharing the same type of categorization:

```
define-labels <label-set name> <value1> <label1> <value2>
<label2> … <valuen> <labeln>

assign-labels <variable> <label-set name>
```

Labels are first made using `define-labels`, and then they are attached to the relevant variable(s) in the next stage.

Example of a categorical (alphanumerical) residency variable at county level (variable `county`). The label set named "countystring" can, through the command `assign-labels`, be attached to as many variables as possible, given that they share the same type of value set (it is not necessary to create the same set of labels several times):

```
define-labels countystring '01' 'Østfold' '02' 'Akershus' '03'
'Oslo' '04' 'Hedmark' '05' 'Oppland' '06' 'Buskerud' '07'
'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder'
'11' 'Rogaland' '12' 'Hordaland' '14' 'Sogn and Fjordane' '15'
'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'

assign-labels county countystring
```

Note: If the variable contains numerical values, the values in the `define-labels` expression need to be specified without singular quotes. However, it is recommended for the text-labels to have singular quotations either way as this makes it possible to use all sorts of signs and characters including space.

NB! Text-labels must not contain commas, as this will result in error messages when executing (the comma character is reserved by the system to indicate the use of options)!

3.7 Changing value format from alphanumerical (text) into numerical

Many variables available through microdata.no contain alphanumerical values (text format). However, the command `destring` can convert such values into numerical format. By default, the variable will be overwritten by the new format (a separate variable will not be created):

```
destring <variable/variable list> [, <options>]
```

If there are values containing non-numerical characters, e.g. “,”, “.”, “-”, “nkr”, “$”, then the conversion will not complete unless the options `force` or `ignore()` are used.

The option `force` will force the system to convert into numerical values no matter what, where values containing non-numerical characters will be given the value for missing value: `sysmiss`

The option `ignore()` makes it possible to define which characters/symbols to be ignored during the conversion process. This could be useful for values formatted by hyphens, commas, thousands separators etc. The following example will ignore dots, commas and hyphens that occur in values for the variable *var1*:

```
destring var1, ignore('.,-')
```

Alphanumerical values containing commas as decimal separators ('2,1', '10000,00' etc) may be converted directly into numerical values while keeping the decimals. By doing so, the converted values will be using dots as decimal separator. The option `dpcomma` can be used for such operations.

3.8 Example

```
// Connect to data bank
require no.ssb.fdb:1 as fdb1

// Retrieve required variables
create-dataset demographics
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth
import fdb1/BOSATTEFDT_BOSTED 2000-01-01 as residence

// Generate age in 2000 from birthyear
generate age = 2000 - int( birth_year_month / 100 )

// Generate dummy variable for male by using gender variable
generate male = 0
replace male = 1 if gender == '1'

// Group wealth into four intervals
generate wealthint = 1
replace wealthint = 2 if wealth > 150000
replace wealthint = 3 if wealth > 250000
replace wealthint = 4 if wealth > 400000

// Designate wealth in 1000 nkr
generate wealth1000 = wealth / 1000

// Recode from municipality to county level
generate county = substr(residence,1,2)

// Add value labels for counties (=> nicer descriptive output)

define-labels countystring '01' 'Østfold' '02' 'Akershus' '03' 'Oslo' '04' 'Hedmark' '05' 'Oppland' '06'
'Buskerud' '07' 'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder' '11' 'Rogaland' '12'
'Hordaland' '14' 'Sogn and Fjordane' '15' 'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'

assign-labels county countystring

tabulate county
```

4. Descriptive variable statistics

Microdata.no provides various techniques for data exploration. The most basic and useful tools are frequency tabulations (one-way or cross tables) and summary statistics (for numerical/metrical variables). It is also possible to visualize through histograms, barcharts, piecharts or anonymised scatterplots (hexbinplots).

The microdata.no analysis system currently has the following commands available for the production of descriptive statistics:

- Tabulate
- Summarize
- Boxplot
- Hexbin
- Piechart
- Histogram
- Barchart
- Sankey

Through various options, alternative representations of the same distributions may be displayed, and specified units can also be filtered out from the tables/figures through if-conditions.

4.1 Tabulate - frequency tables

The command `tabulate` is a tool for creating frequency tables, and is the most common statistics command to map out data/variables and to produce descriptive statistics.

The command can be applied to all categorical variables. These are often alphanumerical, however it is quite possible to create frequency tables for numerical variables as well, as long as the number of values does not become too extensive.

The standard display for tables generated through `tabulate` is cells containing frequency numbers (number of units), which can be one-way, two-way, or multi-dimensional. By default, any labels attached to a variable value set are shown in the leading columns and table header, and missing values will be omitted from the table basis.

Through the use of options the table presentations can be customized:

- View percentages instead of frequencies
- Show figures in leading columns and table header without value labels
- View tables with missing values included
- Create volume tables that show summary values (average, sum, etc.) for any variable within each cell
- Conduct a chi-test (tests for deviation from a completely random bivariate distribution) through a `chi2`-option

Like most microdata.no commands, `tabulate` may be used in combination with if-conditions to control which units to be included in the specific table, i.e. dataset populations do not necessarily have to be trimmed in advance of statistical executions.

Syntax expression:

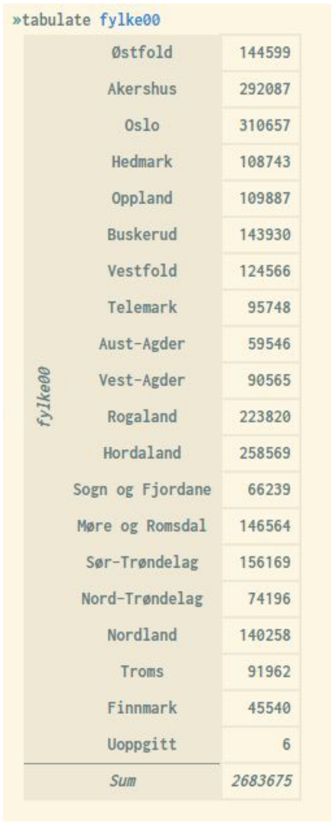
```
tabulate <variable/variable list> [, <options>]
```

Tip:

Tables generated through the command `tabulate` can be exported to other programs such as Excel, Word, Google Sheets, etc. This is done by clicking on a "copy"-icon that pops up when the mouse cursor is over the table. Then use the key combination <*Ctrl*> + <*C*> and paste into the desired document. This solution is also applicable for other types of output, such as `regress`.

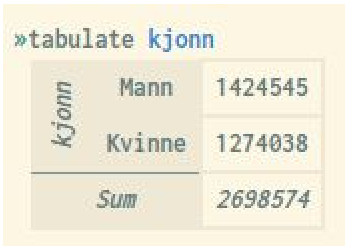
4.1.1 One-way frequency tables

Example of frequency table for the variables "residence county per 2000-01-01" and gender respectively:



»tabulate fylke00

fylke00	
Østfold	144599
Akershus	292087
Oslo	310657
Hedmark	108743
Oppland	109887
Buskerud	143930
Vestfold	124566
Telemark	95748
Aust-Agder	59546
Vest-Agder	90565
Rogaland	223820
Hordaland	258569
Sogn og Fjordane	66239
Møre og Romsdal	146564
Sør-Trøndelag	156169
Nord-Trøndelag	74196
Nordland	140258
Troms	91962
Finnmark	45540
Uoppgitt	6
Sum	*2683675*



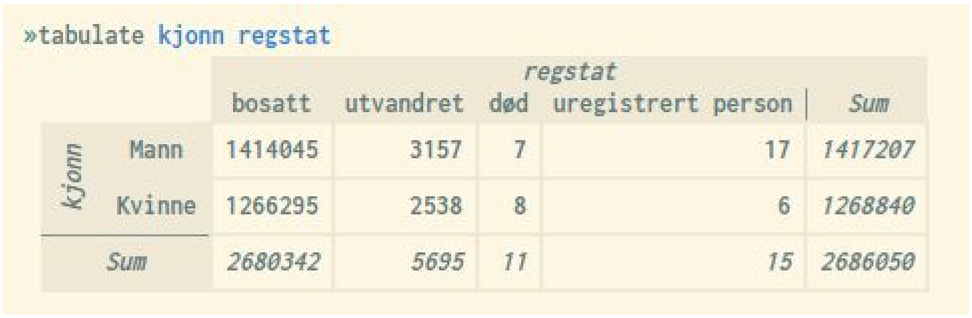
»tabulate kjonn

kjonn	
Mann	1424545
Kvinne	1274038
Sum	*2698574*

4.1.2 Multi-dimensional frequency tables

Multi-dimensional frequency tables are tables that shows distributions over 2 or more variables. They contain cell frequency values, row sums, column sums, and total sums (sum of all inner cells).

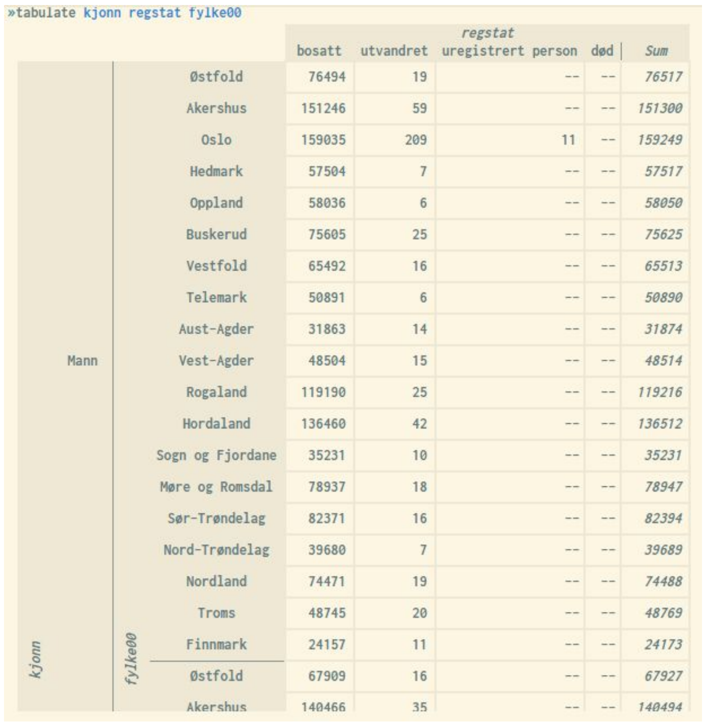
Example of frequency table showing distributions over gender and registry status:



»tabulate kjonn regstat

	regstat				
kjonn	bosatt	utvandret	død	uregistrert person	*Sum*
Mann	1414045	3157	7	17	*1417207*
Kvinne	1266295	2538	8	6	*1268840*
Sum	*2680342*	*5695*	*11*	*15*	*2686050*

Example of three-dimensional frequency table showing distributions over gender, registry status, and residence county (note that the illustration does not show the whole table):



```
»tabulate kjonn regstat fylke00
```

kjonn	fylke00	regstat: bosatt	utvandret	uregistrert person	død	*Sum*
Mann	Østfold	76494	19	--	--	*76517*
	Akershus	151246	59	--	--	*151300*
	Oslo	159035	209	11	--	*159249*
	Hedmark	57504	7	--	--	*57517*
	Oppland	58036	6	--	--	*58050*
	Buskerud	75605	25	--	--	*75625*
	Vestfold	65492	16	--	--	*65513*
	Telemark	50891	6	--	--	*50890*
	Aust-Agder	31863	14	--	--	*31874*
	Vest-Agder	48504	15	--	--	*48514*
	Rogaland	119190	25	--	--	*119216*
	Hordaland	136460	42	--	--	*136512*
	Sogn og Fjordane	35231	10	--	--	*35231*
	Møre og Romsdal	78937	18	--	--	*78947*
	Sør-Trøndelag	82371	16	--	--	*82394*
	Nord-Trøndelag	39680	7	--	--	*39689*
	Nordland	74471	19	--	--	*74488*
	Troms	48745	20	--	--	*48769*
	Finnmark	24157	11	--	--	*24173*
	Østfold	67909	16	--	--	*67927*
	Akershus	140466	35	--	--	*140494*

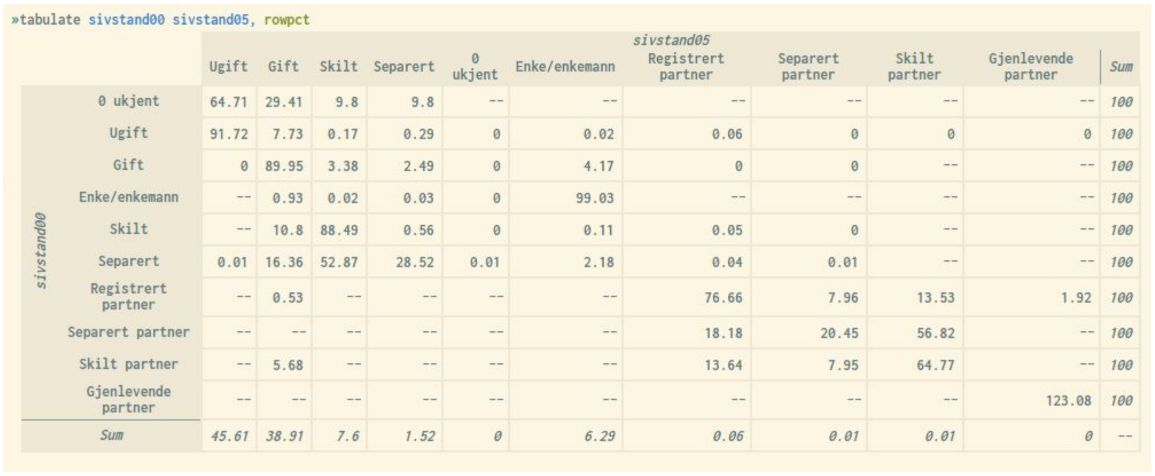
4.1.3 Frequency tables using percentages

The `tabulate`-command can also be used to show frequency distributions through percentages. This is done through the following options:

- `rowpct` row percentages (share of row total)
- `colpct` column percentages (share of column total)
- `cellpct` cell percentages (share of total summed over all inner cells)
- `freq` frequency values (default, only used in combination with percentage presentations)

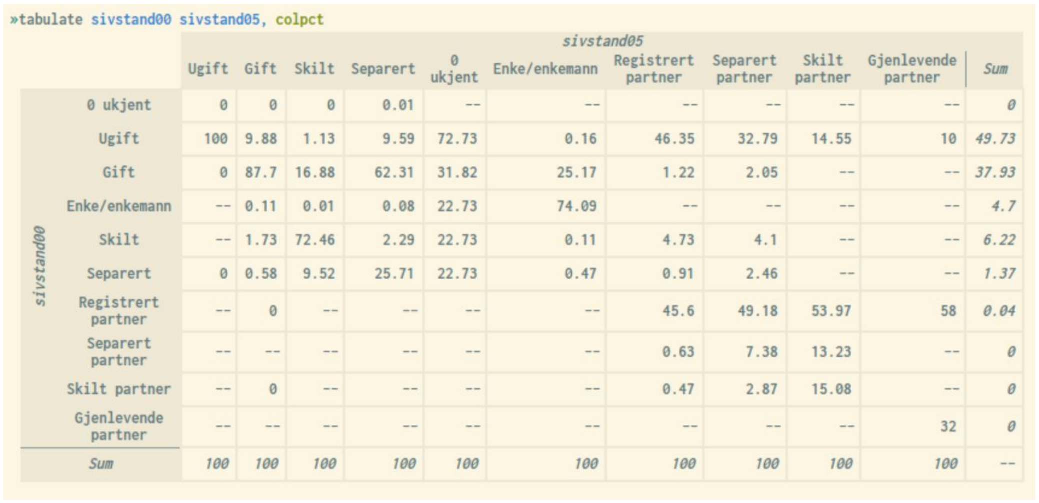
More options can be combined in the same command expression, e.g. to show both frequencies and row percentages in the same table (see example #4 below).

Example:



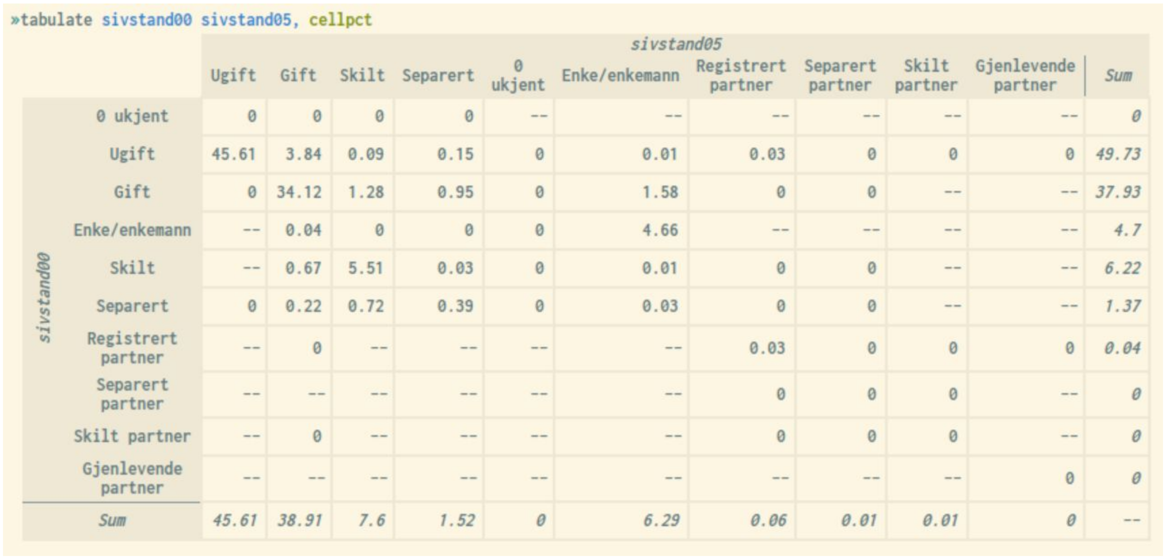
```
»tabulate sivstand00 sivstand05, rowpct
```

sivstand00	sivstand05										
	Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	*Sum*
0 ukjent	64.71	29.41	9.8	9.8	--	--	--	--	--	--	*100*
Ugift	91.72	7.73	0.17	0.29	0	0.02	0.06	0	0	0	*100*
Gift	0	89.95	3.38	2.49	0	4.17	0	0	--	--	*100*
Enke/enkemann	--	0.93	0.02	0.03	0	99.03	--	--	--	--	*100*
Skilt	--	10.8	88.49	0.56	0	0.11	0.05	0	--	--	*100*
Separert	0.01	16.36	52.87	28.52	0.01	2.18	0.04	0.01	--	--	*100*
Registrert partner	--	0.53	--	--	--	--	76.66	7.96	13.53	1.92	*100*
Separert partner	--	--	--	--	--	--	18.18	20.45	56.82	--	*100*
Skilt partner	--	5.68	--	--	--	--	13.64	7.95	64.77	--	*100*
Gjenlevende partner	--	--	--	--	--	--	--	--	--	123.08	*100*
Sum	*45.61*	*38.91*	*7.6*	*1.52*	*0*	*6.29*	*0.06*	*0.01*	*0.01*	*0*	--



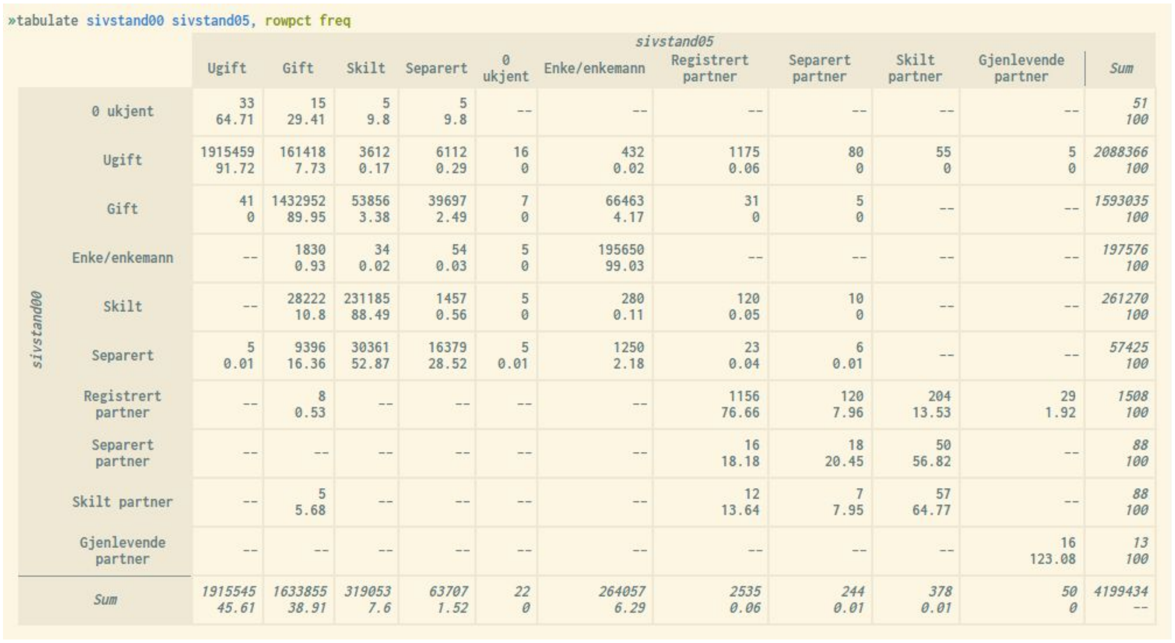
```
»tabulate sivstand00 sivstand05, colpct
```

sivstand00	sivstand05										
	Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	*Sum*
0 ukjent	0	0	0	0.01	--	--	--	--	--	--	*0*
Ugift	100	9.88	1.13	9.59	72.73	0.16	46.35	32.79	14.55	10	*49.73*
Gift	0	87.7	16.88	62.31	31.82	25.17	1.22	2.05	--	--	*37.93*
Enke/enkemann	--	0.11	0.01	0.08	22.73	74.09	--	--	--	--	*4.7*
Skilt	--	1.73	72.46	2.29	22.73	0.11	4.73	4.1	--	--	*6.22*
Separert	0	0.58	9.52	25.71	22.73	0.47	0.91	2.46	--	--	*1.37*
Registrert partner	--	0	--	--	--	--	45.6	49.18	53.97	58	*0.04*
Separert partner	--	--	--	--	--	--	0.63	7.38	13.23	--	*0*
Skilt partner	--	0	--	--	--	--	0.47	2.87	15.08	--	*0*
Gjenlevende partner	--	--	--	--	--	--	--	--	--	32	*0*
Sum	*100*	*100*	*100*	*100*	*100*	*100*	*100*	*100*	*100*	*100*	--



```
»tabulate sivstand00 sivstand05, cellpct
```

sivstand00	sivstand05										
	Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	*Sum*
0 ukjent	0	0	0	0	--	--	--	--	--	--	*0*
Ugift	45.61	3.84	0.09	0.15	0	0.01	0.03	0	0	0	*49.73*
Gift	0	34.12	1.28	0.95	0	1.58	0	0	--	--	*37.93*
Enke/enkemann	--	0.04	0	0	0	4.66	--	--	--	--	*4.7*
Skilt	--	0.67	5.51	0.03	0	0.01	0	0	--	--	*6.22*
Separert	0	0.22	0.72	0.39	0	0.03	0	0	--	--	*1.37*
Registrert partner	--	0	--	--	--	--	0.03	0	0	0	*0.04*
Separert partner	--	--	--	--	--	--	0	0	0	--	*0*
Skilt partner	--	0	--	--	--	--	0	0	0	--	*0*
Gjenlevende partner	--	--	--	--	--	--	--	--	--	0	*0*
Sum	*45.61*	*38.91*	*7.6*	*1.52*	*0*	*6.29*	*0.06*	*0.01*	*0.01*	*0*	--



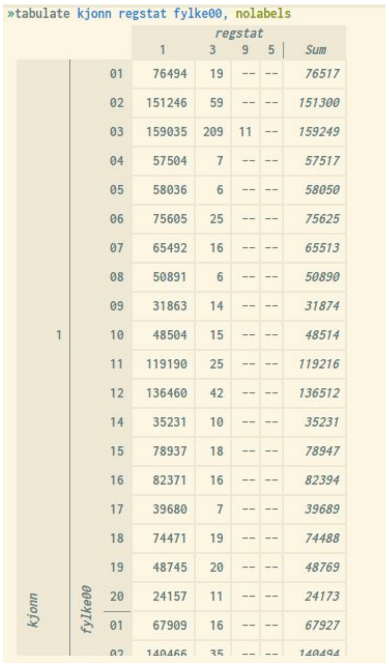
```
»tabulate sivstand00 sivstand05, rowpct freq
```

sivstand00	sivstand05: Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	*Sum*
0 ukjent	33 64.71	15 29.41	5 9.8	5 9.8	--	--	--	--	--	--	*51 100*
Ugift	1915459 91.72	161418 7.73	3612 0.17	6112 0.29	16 0	432 0.02	1175 0.06	80 0	55 0	5 0	*2088366 100*
Gift	41 0	1432952 89.95	53856 3.38	39697 2.49	7 0	66463 4.17	31 0	5 0	--	--	*1593035 100*
Enke/enkemann	--	1830 0.93	34 0.02	54 0.03	5 0	195650 99.03	--	--	--	--	*197576 100*
Skilt	--	28222 10.8	231185 88.49	1457 0.56	5 0	280 0.11	120 0.05	10 0	--	--	*261270 100*
Separert	5 0.01	9396 16.36	30361 52.87	16379 28.52	5 0.01	1250 2.18	23 0.04	6 0.01	--	--	*57425 100*
Registrert partner	--	8 0.53	--	--	--	--	1156 76.66	120 7.96	204 13.53	29 1.92	*1508 100*
Separert partner	--	--	--	--	--	--	16 18.18	18 20.45	50 56.82	--	*88 100*
Skilt partner	--	5 5.68	--	--	--	--	12 13.64	7 7.95	57 64.77	--	*88 100*
Gjenlevende partner	--	--	--	--	--	--	--	--	--	16 123.08	*13 100*
Sum	*1915545 45.61*	*1633855 38.91*	*319053 7.6*	*63707 1.52*	*22 0*	*264057 6.29*	*2535 0.06*	*244 0.01*	*378 0.01*	*50 0*	*4199434 --*

4.1.4 Frequency tables and category labels

Labels are used by default for variable values in the leading column and table header. However, this can be turned off by using the option `nolabels,` showing only the values.

Example:



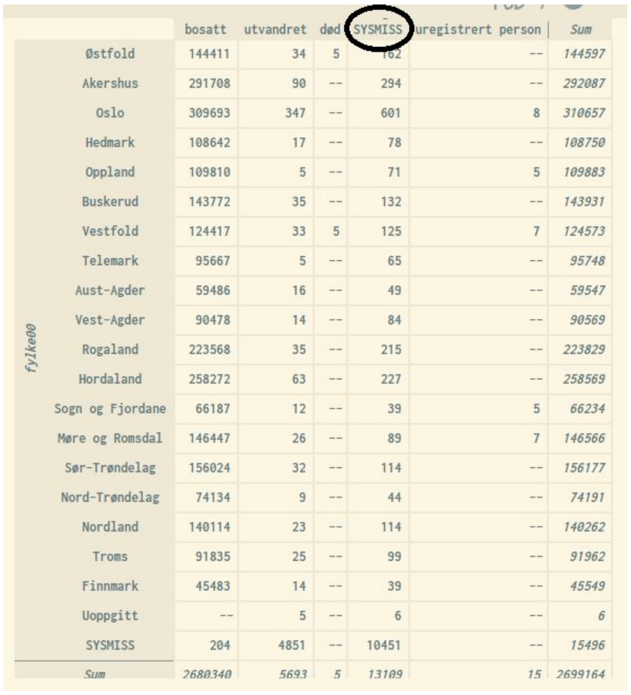
```
»tabulate kjonn regstat fylke00, nolabels
```

kjonn	fylke00	regstat: 1	3	9	5	*Sum*
1	01	76494	19	--	--	*76517*
1	02	151246	59	--	--	*151300*
1	03	159035	209	11	--	*159249*
1	04	57504	7	--	--	*57517*
1	05	58036	6	--	--	*58050*
1	06	75605	25	--	--	*75625*
1	07	65492	16	--	--	*65513*
1	08	50891	6	--	--	*50890*
1	09	31863	14	--	--	*31874*
1	10	48504	15	--	--	*48514*
1	11	119190	25	--	--	*119216*
1	12	136460	42	--	--	*136512*
1	14	35231	10	--	--	*35231*
1	15	78937	18	--	--	*78947*
1	16	82371	16	--	--	*82394*
1	17	39680	7	--	--	*39689*
1	18	74471	19	--	--	*74488*
1	19	48745	20	--	--	*48769*
1	20	24157	11	--	--	*24173*
	01	67909	16	--	--	*67927*
	02	140466	35	--	--	*140494*

4.1.5 Frequency tables and missing values

By default, missing values are not included during the `tabulate`-calculations, unless the `missing`-option is used.

Example:

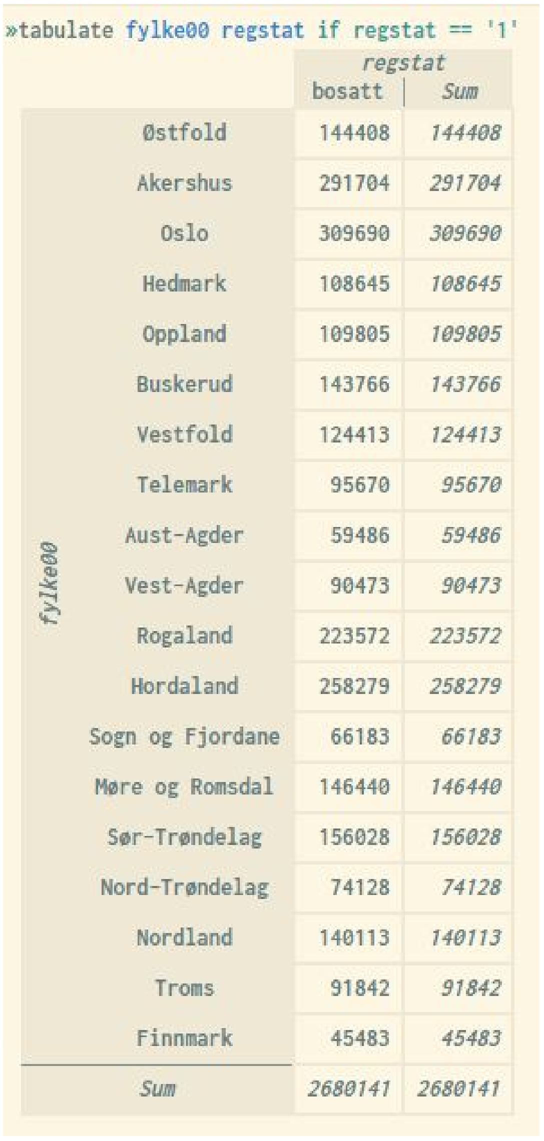


fylke00	bosatt	utvandret	død	SYSMISS	uregistrert person	*Sum*
Østfold	144411	34	5	162	--	*144597*
Akershus	291708	90	--	294	--	*292087*
Oslo	309693	347	--	601	8	*310657*
Hedmark	108642	17	--	78	--	*108750*
Oppland	109810	5	--	71	5	*109883*
Buskerud	143772	35	--	132	--	*143931*
Vestfold	124417	33	5	125	7	*124573*
Telemark	95667	5	--	65	--	*95748*
Aust-Agder	59486	16	--	49	--	*59547*
Vest-Agder	90478	14	--	84	--	*90569*
Rogaland	223568	35	--	215	--	*223829*
Hordaland	258272	63	--	227	--	*258569*
Sogn og Fjordane	66187	12	--	39	5	*66234*
Møre og Romsdal	146447	26	--	89	7	*146566*
Sør-Trøndelag	156024	32	--	114	--	*156177*
Nord-Trøndelag	74134	9	--	44	--	*74191*
Nordland	140114	23	--	114	--	*140262*
Troms	91835	25	--	99	--	*91962*
Finnmark	45483	14	--	39	--	*45549*
Uoppgitt	--	5	--	6	--	*6*
SYSMISS	204	4851	--	10451	--	*15496*
Sum	*2680340*	*5693*	*5*	*13109*	*15*	*2699164*

4.1.6 Frequency table filtering

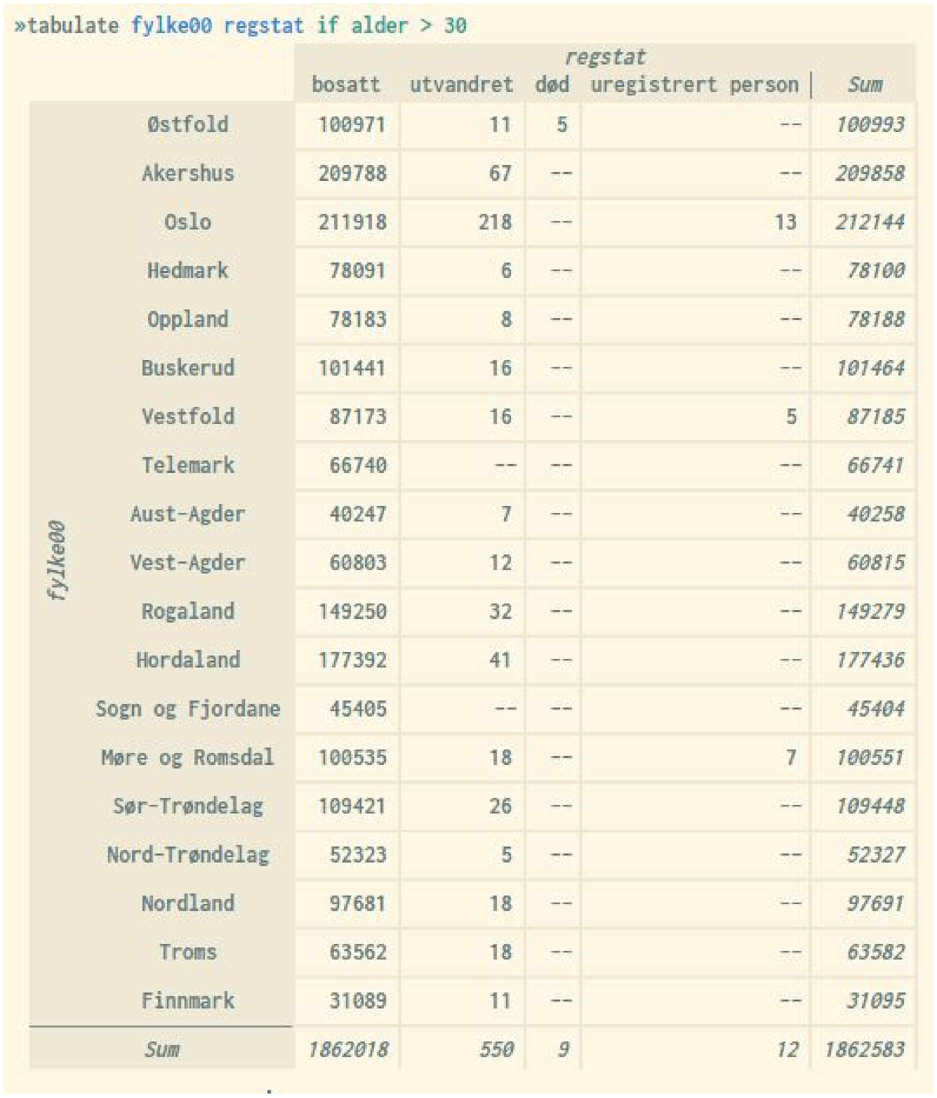
Frequency tables can be generated for sub-populations through the use of if-conditions, i.e. there is no need to trim the dataset in advance.

Examples:



```
»tabulate fylke00 regstat if regstat == '1'
```

		regstat	
		bosatt	Sum
fylke00	Østfold	144408	*144408*
	Akershus	291704	*291704*
	Oslo	309690	*309690*
	Hedmark	108645	*108645*
	Oppland	109805	*109805*
	Buskerud	143766	*143766*
	Vestfold	124413	*124413*
	Telemark	95670	*95670*
	Aust-Agder	59486	*59486*
	Vest-Agder	90473	*90473*
	Rogaland	223572	*223572*
	Hordaland	258279	*258279*
	Sogn og Fjordane	66183	*66183*
	Møre og Romsdal	146440	*146440*
	Sør-Trøndelag	156028	*156028*
	Nord-Trøndelag	74128	*74128*
	Nordland	140113	*140113*
	Troms	91842	*91842*
	Finnmark	45483	*45483*
Sum		*2680141*	*2680141*



```
»tabulate fylke00 regstat if alder > 30
```

		regstat				
		bosatt	utvandret	død	uregistrert person	*Sum*
fylke00	Østfold	100971	11	5	--	*100993*
	Akershus	209788	67	--	--	*209858*
	Oslo	211918	218	--	13	*212144*
	Hedmark	78091	6	--	--	*78100*
	Oppland	78183	8	--	--	*78188*
	Buskerud	101441	16	--	--	*101464*
	Vestfold	87173	16	--	5	*87185*
	Telemark	66740	--	--	--	*66741*
	Aust-Agder	40247	7	--	--	*40258*
	Vest-Agder	60803	12	--	--	*60815*
	Rogaland	149250	32	--	--	*149279*
	Hordaland	177392	41	--	--	*177436*
	Sogn og Fjordane	45405	--	--	--	*45404*
	Møre og Romsdal	100535	18	--	7	*100551*
	Sør-Trøndelag	109421	26	--	--	*109448*
	Nord-Trøndelag	52323	5	--	--	*52327*
	Nordland	97681	18	--	--	*97691*
	Troms	63562	18	--	--	*63582*
	Finnmark	31089	11	--	--	*31095*
Sum		*1862018*	*550*	*9*	*12*	*1862583*

4.1.7 Volume tables

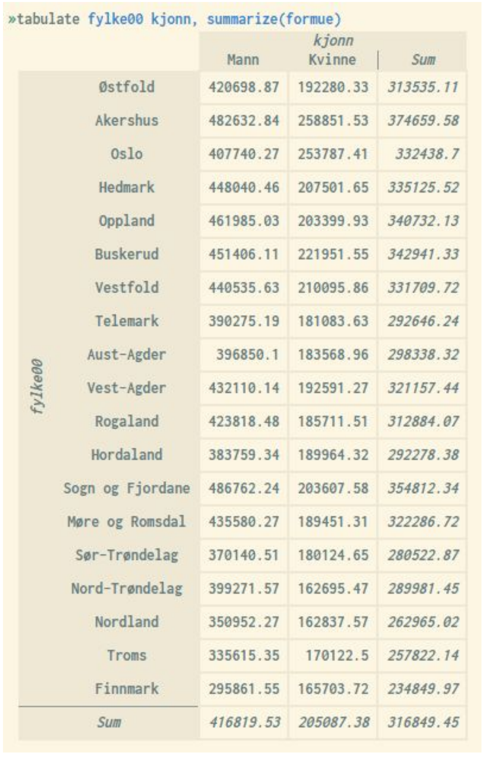
The `tabulate`-command can also be used to generate volume tables: Instead of frequencies or frequency percentages, each cell will show summary statistics as specified for an optional variable. The following option will produce volume tables in combination with the `tabulate`-command:

```
summarize(<variable>)
```

By default, means are shown. However, this can be altered by using the following extra options in the `tabulate`-expression:

- `mean`	Mean value (default)
- `max`	Maximum value
- `min`	Minimum value
- `sum`	Sum
- `std`	Standard deviation
- `p25`	25%-quartile
- `p50`	50%-quartile
- `p75`	75%-quartile

Example of volume table showing mean wealth by residence county and gender:



```
»tabulate fylke00 kjonn, summarize(formue)
```

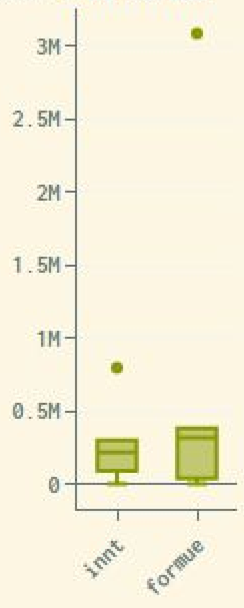
		kjonn		
		Mann	Kvinne	*Sum*
fylke00	Østfold	420698.87	192280.33	*313535.11*
	Akershus	482632.84	258851.53	*374659.58*
	Oslo	407740.27	253787.41	*332438.7*
	Hedmark	448040.46	207501.65	*335125.52*
	Oppland	461985.03	203399.93	*340732.13*
	Buskerud	451406.11	221951.55	*342941.33*
	Vestfold	440535.63	210095.86	*331709.72*
	Telemark	390275.19	181083.63	*292646.24*
	Aust-Agder	396850.1	183568.96	*298338.32*
	Vest-Agder	432110.14	192591.27	*321157.44*
	Rogaland	423818.48	185711.51	*312884.07*
	Hordaland	383759.34	189964.32	*292278.38*
	Sogn og Fjordane	486762.24	203607.58	*354812.34*
	Møre og Romsdal	435580.27	189451.31	*322286.72*
	Sør-Trøndelag	370140.51	180124.65	*280522.87*
	Nord-Trøndelag	399271.57	162695.47	*289981.45*
	Nordland	350952.27	162837.57	*262965.02*
	Troms	335615.35	170122.5	*257822.14*
	Finnmark	295861.55	165703.72	*234849.97*
	Sum	*416819.53*	*205087.38*	*316849.45*

4.2 Summarize and boxplot - metrical statistics

The commands `summarize` and `boxplot` are tools for generating summary statistics for metrical/continuous variables. Like other statistical commands in microdata.no, if-conditions may be used to generate statistics for sub-populations (trimming of population in advance is not necessary).

Examples are presented below, showing summary statistics for the variables income and wealth measured per 2000-01-01. The command `boxplot` shows a graphical presentation using a standard boxplot (a box representing the two middle quartiles, plus mean, minimum, and maximum values):

```
»summarize innt formue
 Variabel     mean          std        count    min     max       25%      50%      75%
   innt    216734.7462  157690.3886  2699162   1265   794619     89175   216523   299527
  formue   315841.0182  476449.5303  2591219     33  3084837   36844.5   169486   380881

»summarize formue if alder > 50
 Variabel     mean          std        count   min     max        25%         50%         75%
  formue   525248.1495  611764.7615  671330   305   3084837   149883.75   341365.5   641339.75

»boxplot innt formue
```

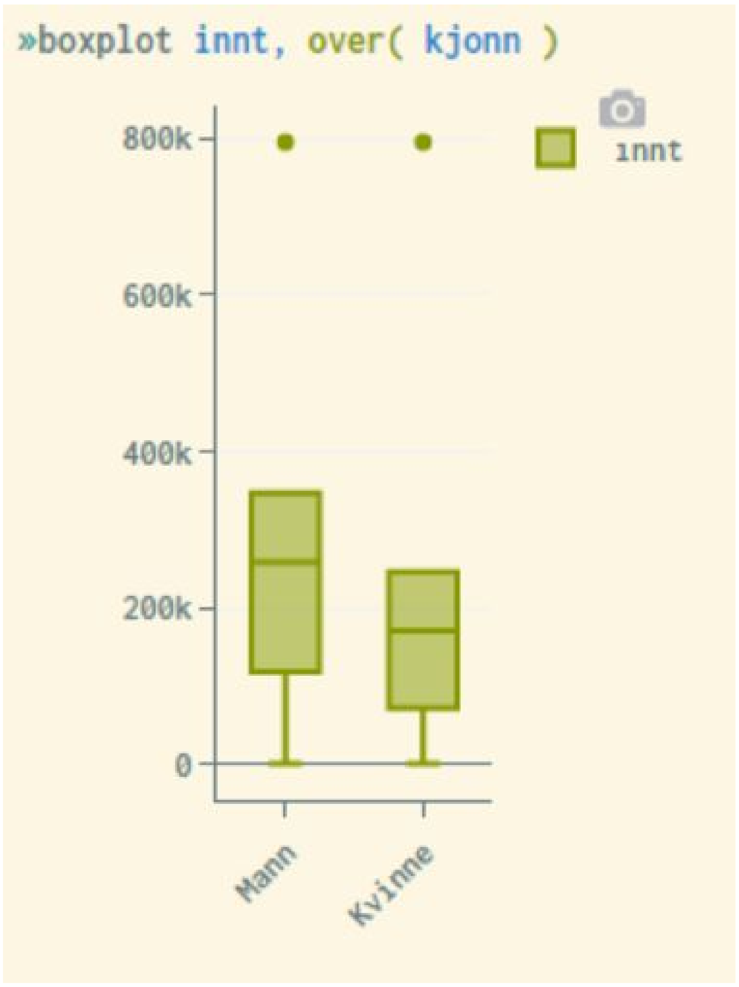


By holding the mouse cursor over the various boxplot areas, the corresponding values will be shown.

The command `boxplot` gives the opportunity to show separate figures for specified categories represented by a custom variable:

```
    boxplot <variable1>, over ( variable2 )
```

Example of boxplot measuring income per 2000-01-01 by gender:



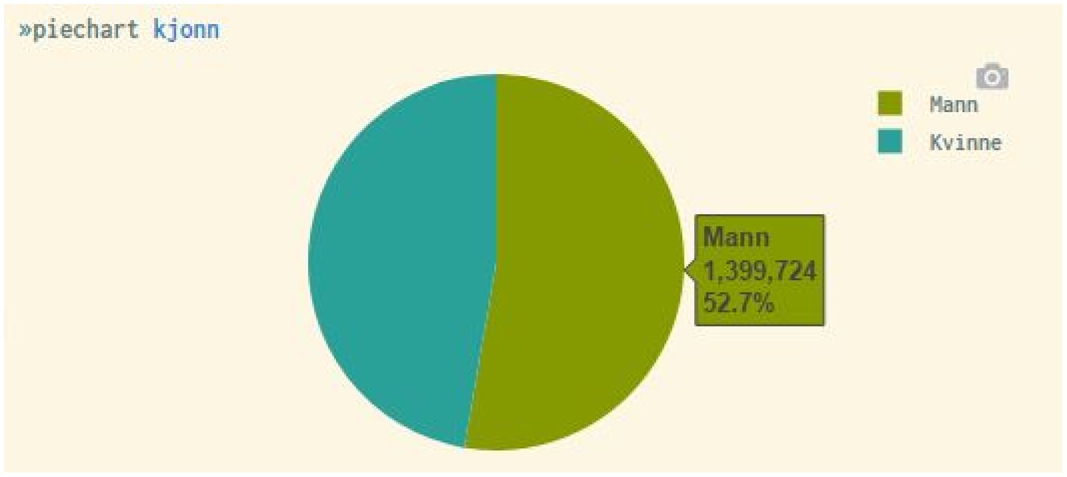
```
»boxplot innt, over( kjonn )
```

4.3 Piecharts

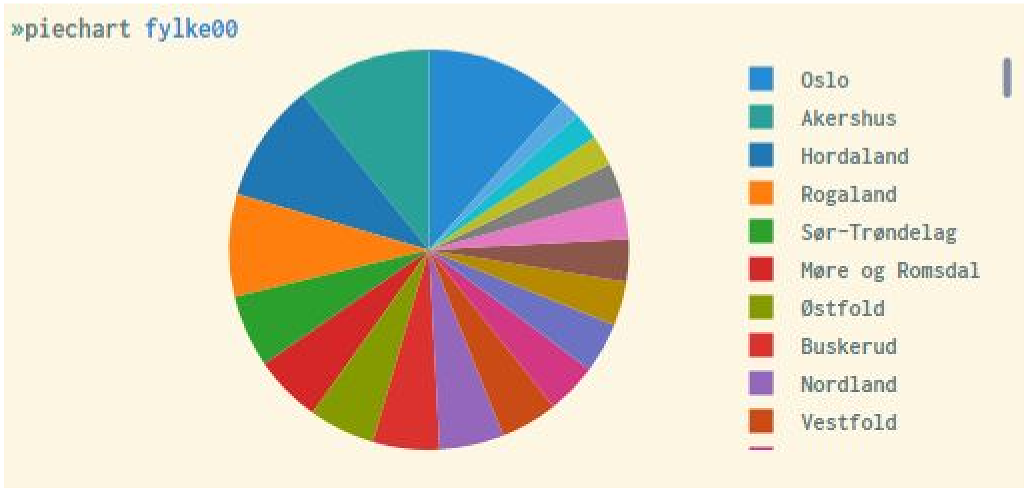
The following command are used to produce piecharts for categorical variables:

```
piechart <variable>
```

By holding the mouse cursor over the various piechart areas, the corresponding figures will be shown.



```
»piechart kjonn
```



```
»piechart fylke00
```

4.4 Histogram - graphical frequency presentation

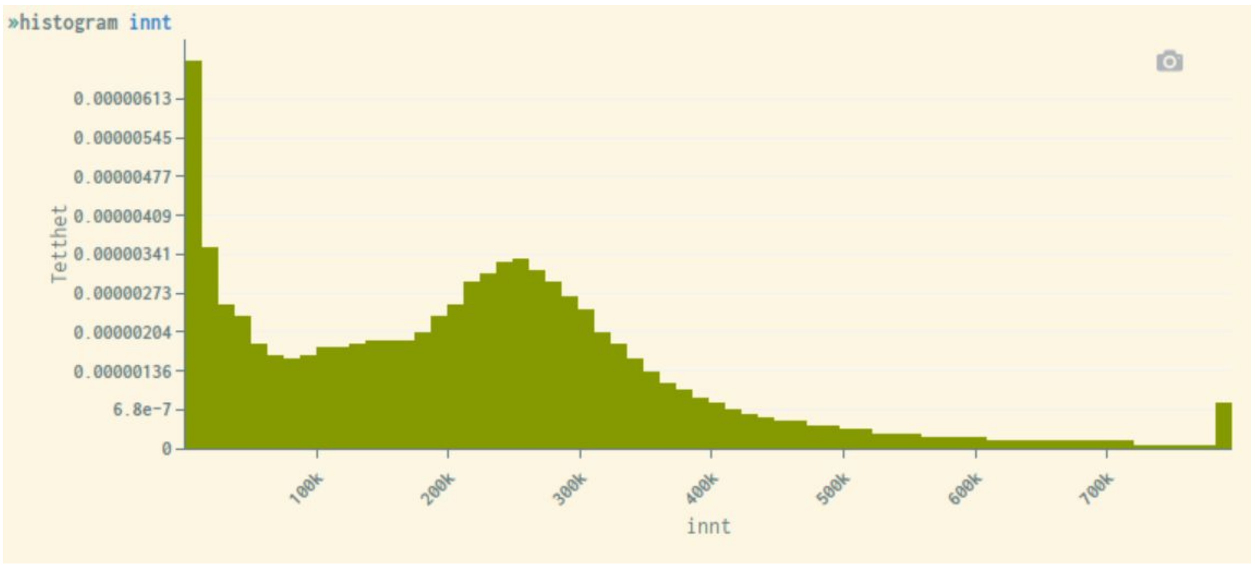
Histograms are graphical representations of univariate distributions for continuous variables (e.g., income). Each bar represents the frequency value for the corresponding predetermined variable interval. Through the options `bin()` and `width()`, it is possible to adjust and specify the number of bars and the interval width respectively. This is illustrated in the examples below.

The default display shows density as the frequency value. This can also be customized through options, in order to change the unit of measurement on the y-axis into actual frequency (number), proportion, or percentage. The following options can be used for: `freq, fraction, percent`

People with very high or very low income can easily be identified if the range of values becomes too narrow, which is problematic in terms of privacy. Therefore, the system performs a top/ bottom coding where the 1% highest and 1% lowest values are replaced by the respective limit values. Thus, the first and last bars will always be much higher than the neighboring bars, as illustrated in the examples below. This top/bottom coding is discussed in detail in Appendix C.

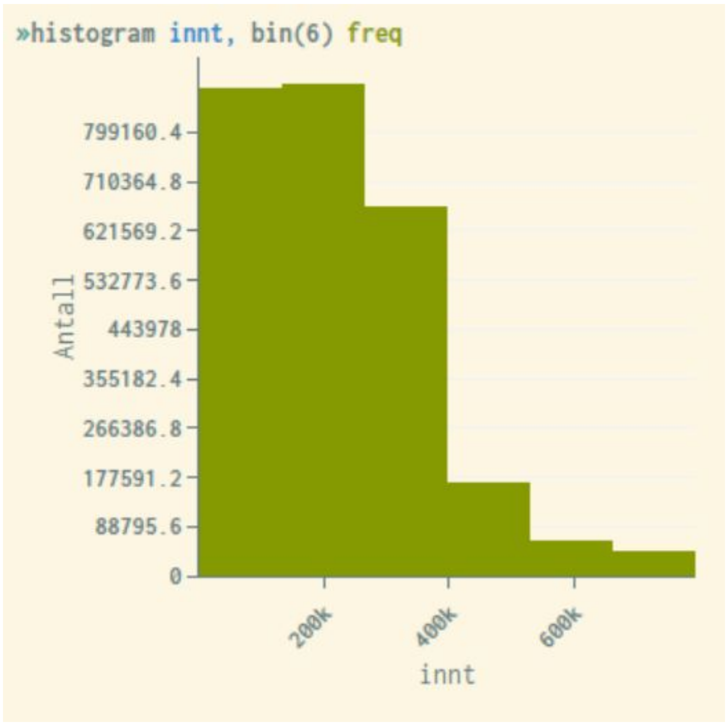
By holding the mouse cursor over the various bars in the diagram, the respective bar intervals and frequency values will be shown.

Example:



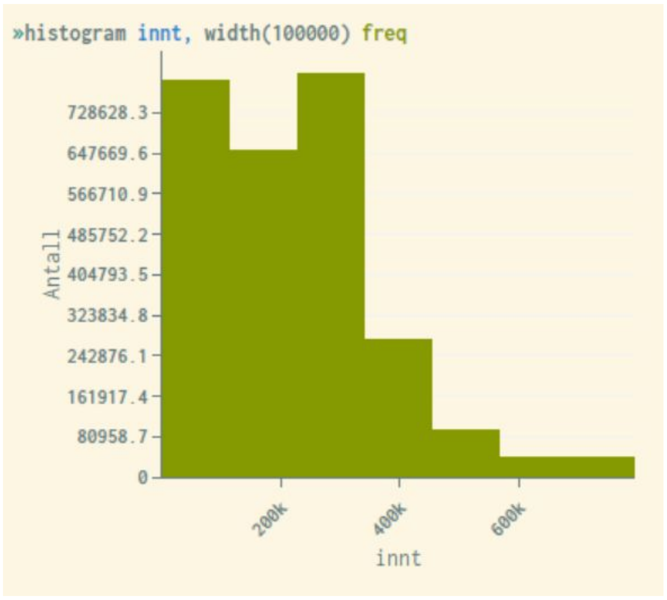
```
»histogram innt
```

Histogram showing income distributed over 6 intervals, and frequency numbers on the y-axis (each bar has the same income interval width):



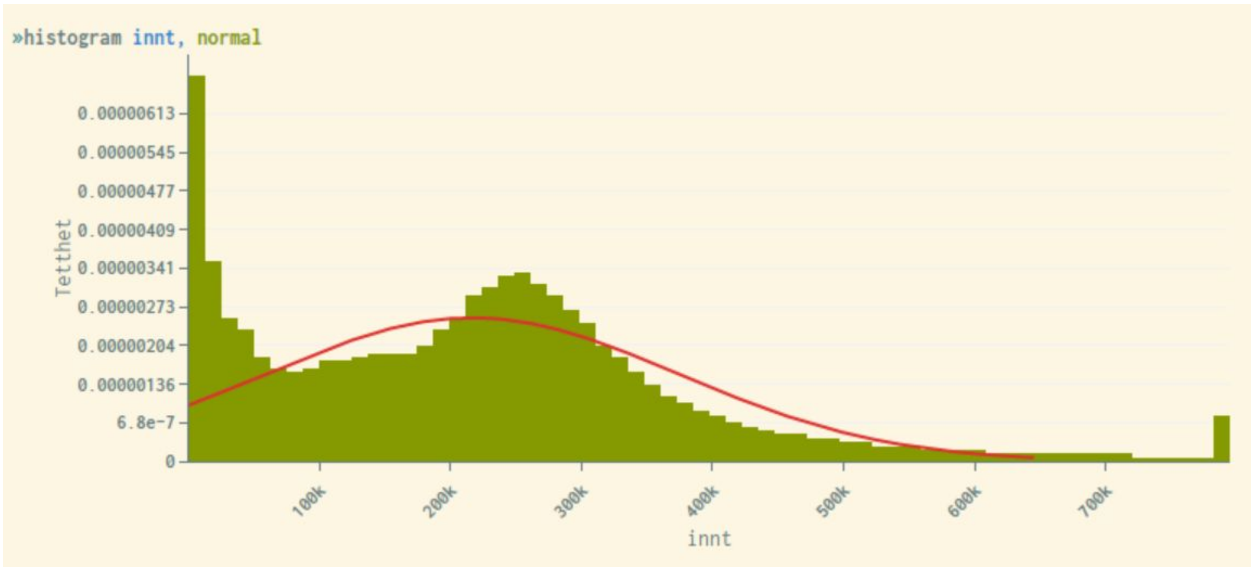
```
»histogram innt, bin(6) freq
```

Histogram showing income where each interval width are set to 100'000:



```
»histogram innt, width(100000) freq
```

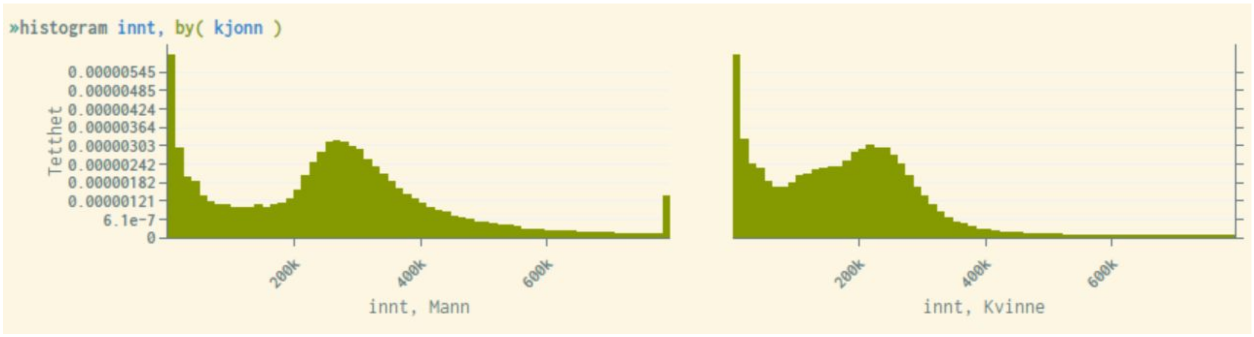
Through the option `normal`, a normal distributed curve is placed over the bars in the figure. This is helpful to study the degree of deviation from a normal distribution:



```
»histogram innt, normal
```

Histograms can be displayed over distributions for another variable that must be categorical, e.g. gender. This is done through the option `by(<variable>)`.

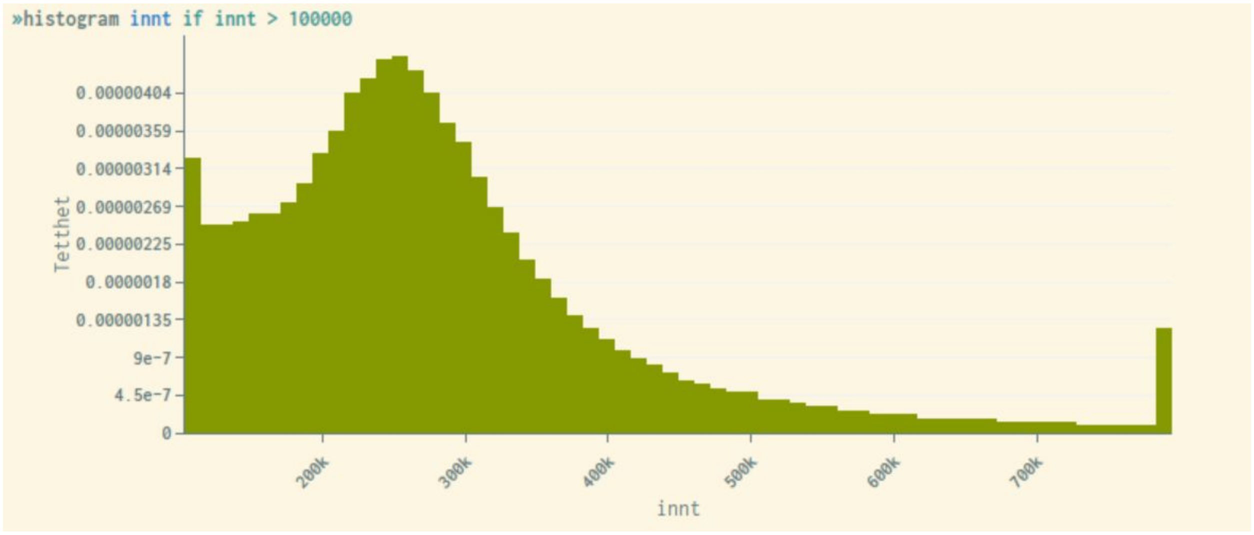
Example:



```
»histogram innt, by( kjonn )
```

Like other statistical representations in microdata.no, filtering can be performed through if-conditions, where the histogram is shown only for a sub-population.

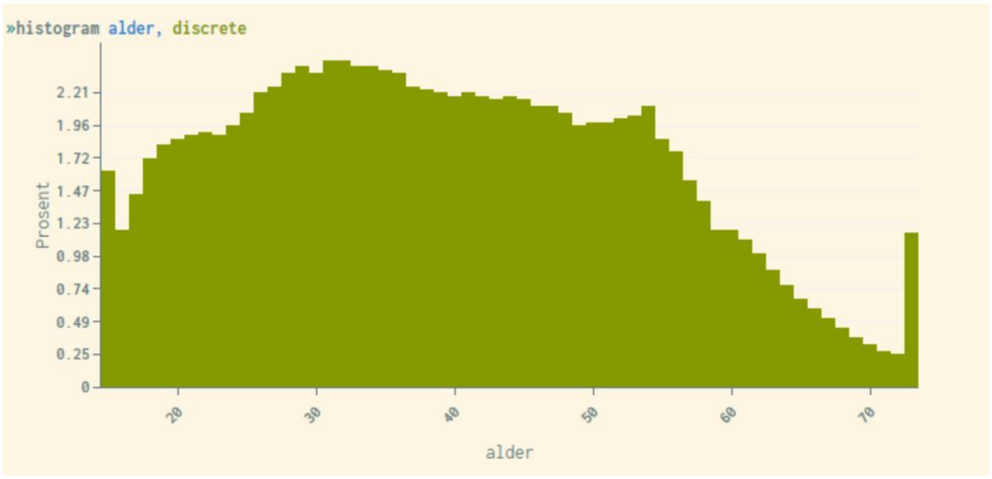
Example showing histogram only for individuals with an income above 100,000 nkr:



```
»histogram innt if innt > 100000
```

As mentioned, the histogram by default will divide into a predetermined number of bars/ intervals. Through the option `discrete` this can be adjusted to display a bar for each individual value. This is not appropriate for metric variables of economic nature (number of bars becomes very high). However, for continuous variables with a limited number of values, this representation may be useful. Examples of variables may be "age", percentages, or amounts that are rounded to the nearest 10,000 or 100,000.

Example of using the option `discrete` for the variable "age" (note that the system also in this case ensures that the first and last bars are top/bottom coded, since people of very low/high age are relatively easy to identify):



```
»histogram alder, discrete
```

4.5 Barcharts

The command `barchart` is a tool for making standard barchart diagrams. A variable or set of variables need to be specified, in addition to the statistical measurement to be performed. The option `over()` makes it possible to distribute the bars over one or more optional categorical variables, e.g. gender.

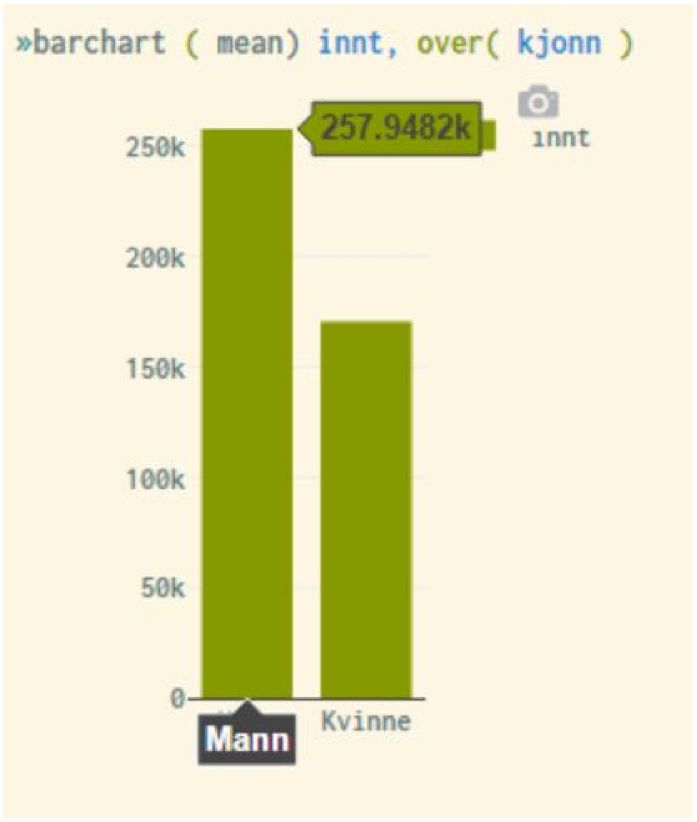
Like other microdata.no graphical displays, it is possible to hold the mouse cursor over the various areas in the diagram to show corresponding values.

Syntax:

```
barchart(<statistical measure>) <variable list>[, over(<variable list>)]
```

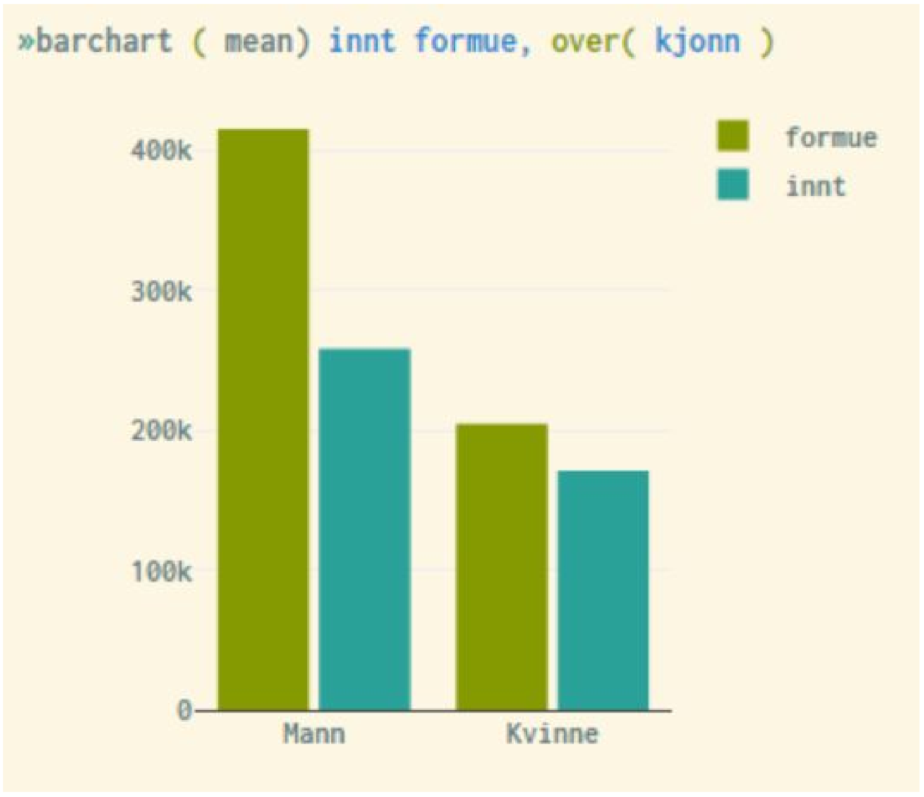
Stacked barcharts can be made through the option `stack`.

Example of barcharts measuring mean income distributed over gender:



```
»barchart ( mean) innt, over( kjonn )
```

Example of barchart measuring mean income and wealth, distributed over gender:



```
»barchart ( mean) innt formue, over( kjonn )
```

4.6 Hexbin - anonymized scatterplot

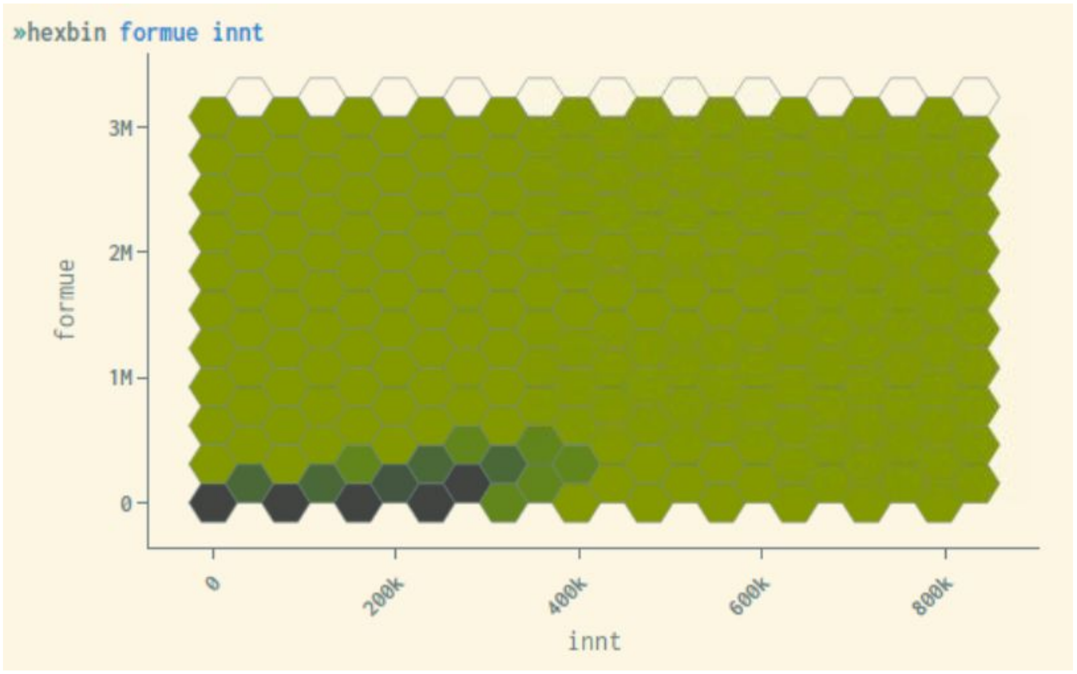
Hexbin diagrams are basically anonymized scatterplots where the two-dimensional area is divided into a given number of hexagons. The colour of each hexagon represents the density of observations in the specific interval of x- and y-values. The darker the colour, the more observations are observed in this particular area.

Hexbin diagrams produce a graphical presentation of the distribution of units between two continuous variables, and are not suitable for categorical variables.

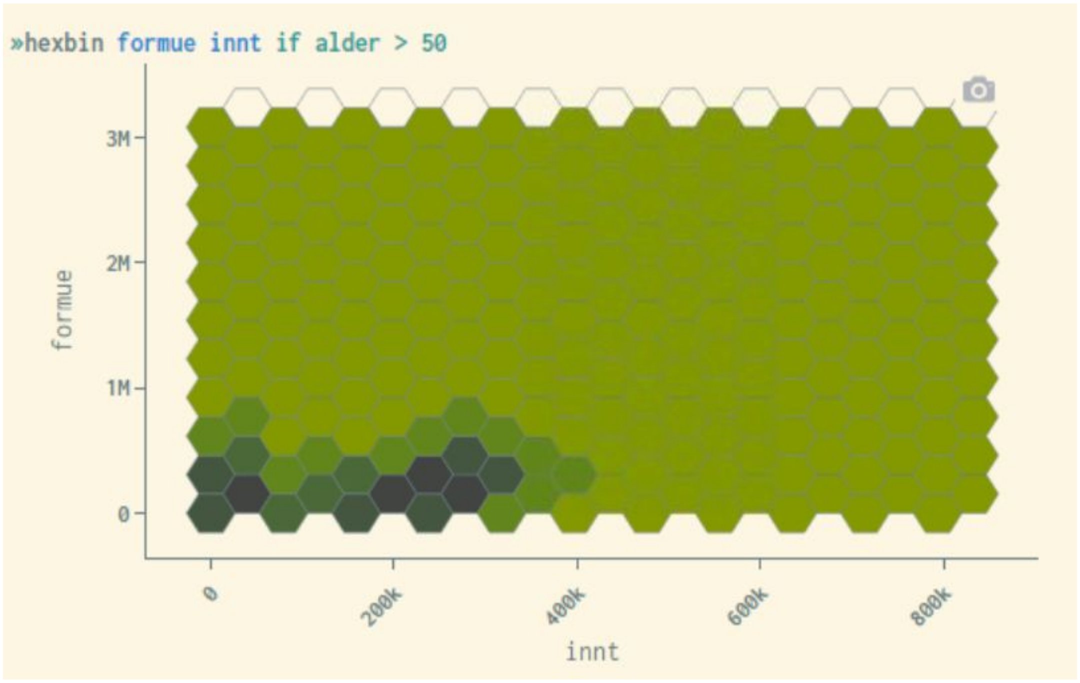
By holding the mouse cursor over the various areas in the diagram, the corresponding values will be shown.

Like other statistical presentations, if-conditions may be used in combination with the hexbin-expression to show diagrams for sub-populations. Also, the number of hexagons and intervals may be customized through options.

Examples:



```
»hexbin formue innt
```



```
»hexbin formue innt if alder > 50
```

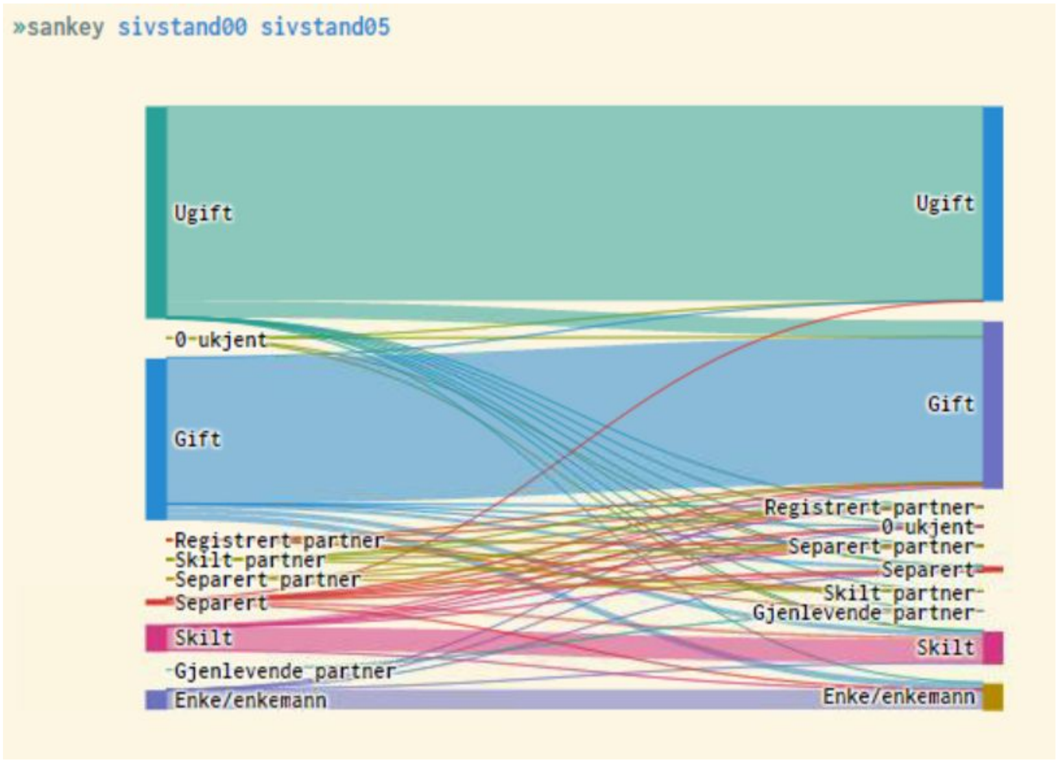
4.7 Sankey - transition diagrams

Sankey diagrams are a way to visualize transitions between statuses. In microdata.no, this can be used to get an overview of units' (individual's) movements between two time-measurements, either for the same variable or for different variables. Movements between different types of states (e.g. jobsearch status -> job status) can be viewed, or changes in distributions for the same variable over time (e.g. residence00 -> residence05 or maritalstate00 -> maritalstate05).

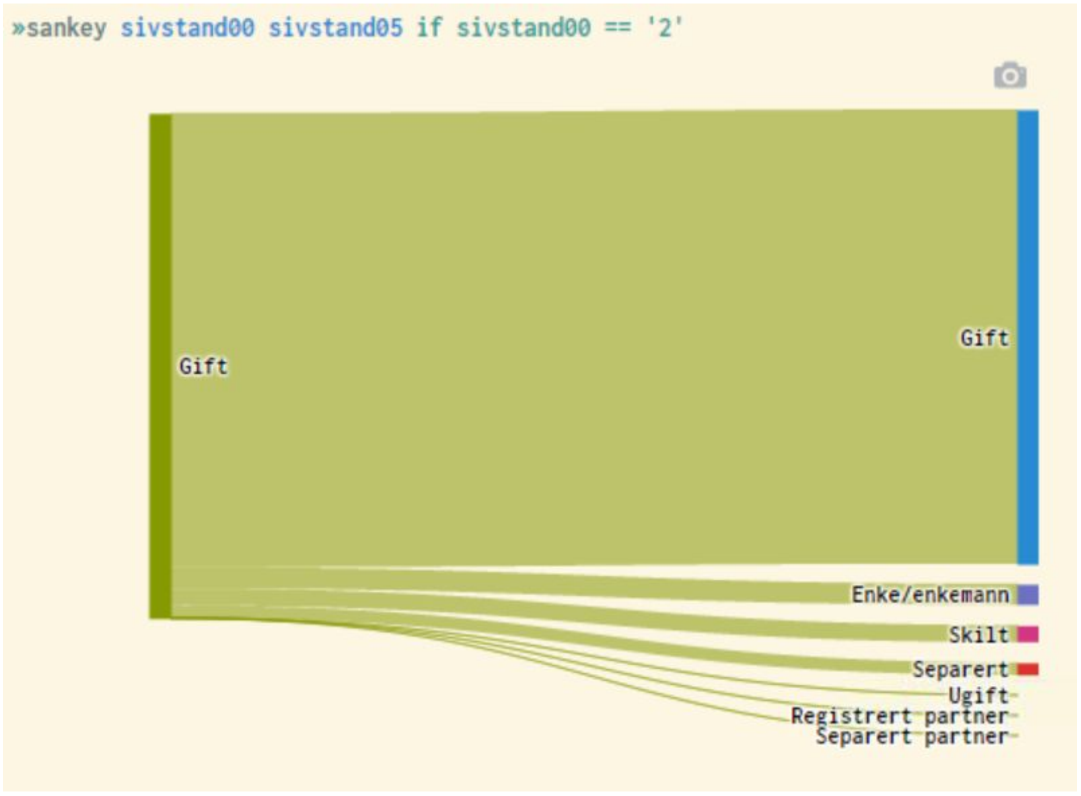
The transition visualization requires two categorical cross-sectional variables to be used. The number of categories should not be too large, as the chart can quickly become unreadable. This can be solved by converting into fewer categories or by using an if-condition that controls which transitions to study.

By holding the mouse cursor over a transition field, the corresponding number of units are shown.

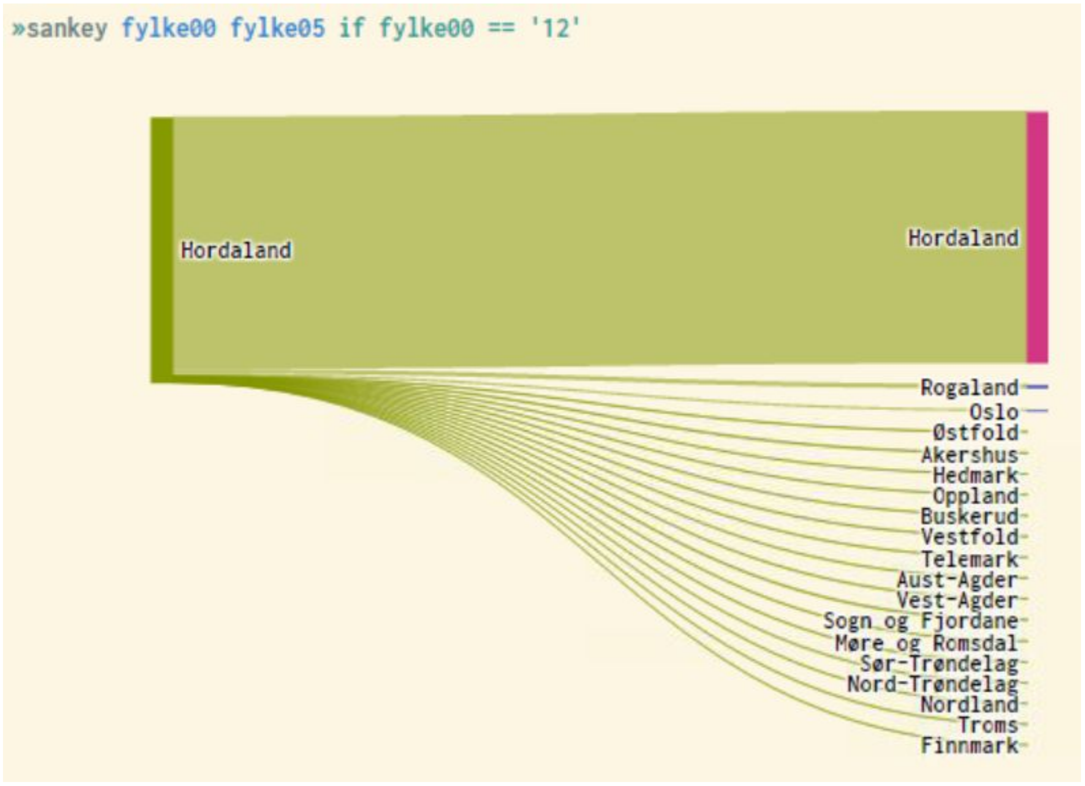
Examples:



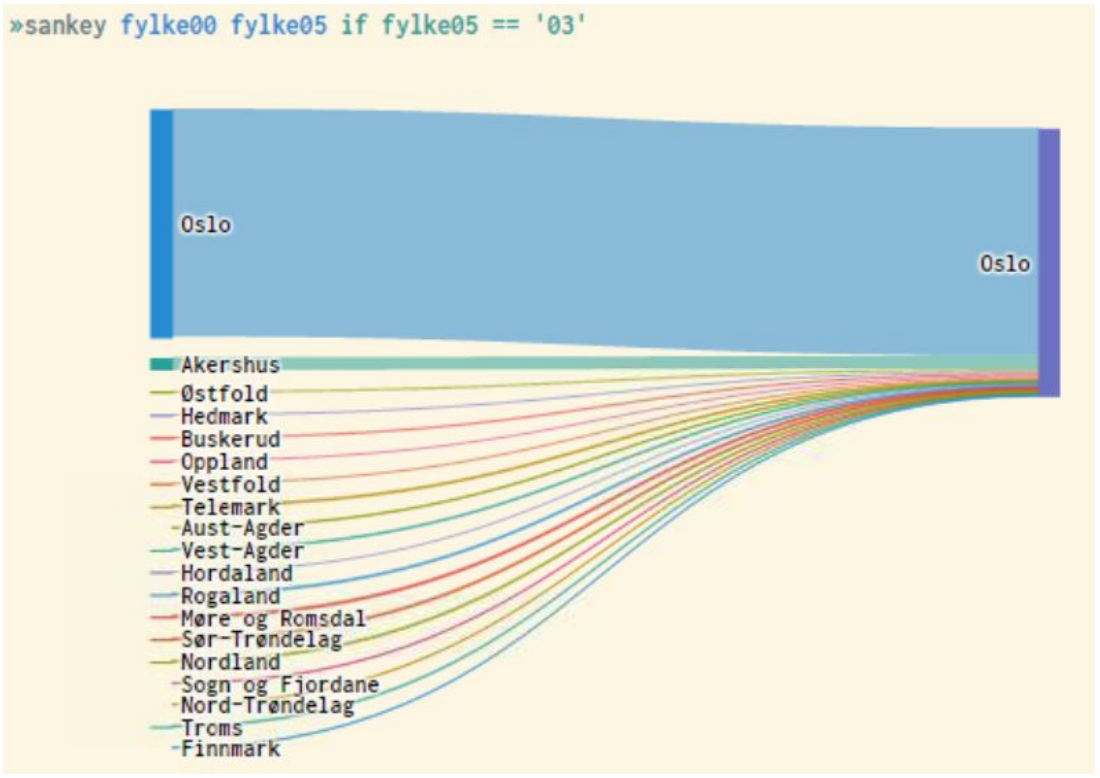
```
»sankey sivstand00 sivstand05
```



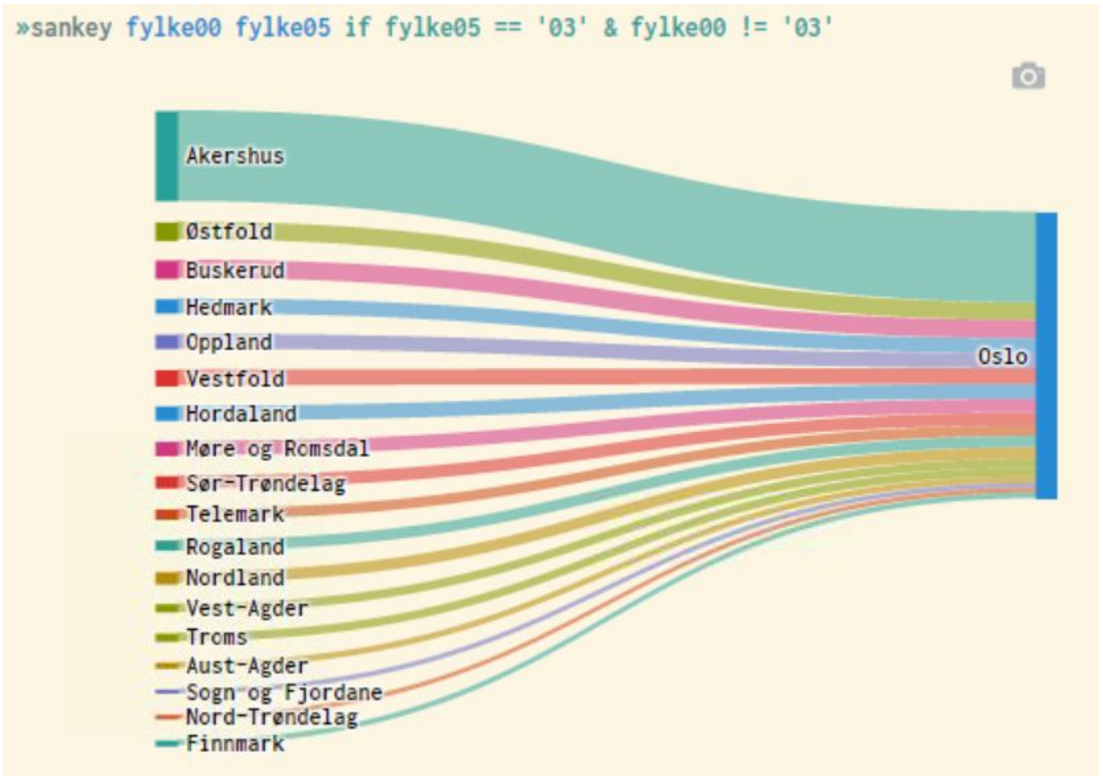
```
»sankey sivstand00 sivstand05 if sivstand00 == '2'
```



```
»sankey fylke00 fylke05 if fylke00 == '12'
```



```
»sankey fylke00 fylke05 if fylke05 == '03'
```



```
»sankey fylke00 fylke05 if fylke05 == '03' & fylke00 != '03'
```

4.8 Examples

The scripts below can be used to recreate the descriptive statistics examples presented in chapter 4. These will also be available as executable scripts in microdata.no.

4.8.1 Tabulate

```
require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/INNTEKT_WYRKINNT 2000-01-01 as income
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 as maritalstate00
import fdb1/SIVSTANDFDT_SIVSTAND 2005-01-01 as maritalstate05
import fdb1/BOSATTEFDT_BOSTED 2000-01-01 as residence00
import fdb1/BEFOLKNING_REGSTAT 2000-01-01 as regstat

// Recode from municipality to county level
generate county00 = substr(residence00 ,1, 2)


// Generate descriptive statistics

// Frequency tabulation - one-way and two-way
// The best way to map out discrete variables is by using frequency tabulations. They show the
number of units within each category, as well as giving an overview of possible categories used by the
specific variable(s). Frequency statistics may be shown not only for single variables through one-way
tabulations, but also for combinations of two or more variables in the same cross-table. This will give
insight of the distribution of frequencies controlled for values of the other variables

define-labels countystring '01' 'Østfold' '02' 'Akershus' '03' 'Oslo' '04' 'Hedmark' '05' 'Oppland' '06'
'Buskerud' '07' 'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder' '11' 'Rogaland' '12'
'Hordaland' '14' 'Sogn and Fjordane' '15' 'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'

assign-labels county00 countystring

tabulate county00
tabulate gender
tabulate gender regstat
tabulate gender regstat county00

// Cross-table with categorical values only (no labels)
tabulate gender regstat county00, nolabels

// Cross-table with missing values
tabulate county00 regstat, missing

// Cross-table only for residents of Norway per 1.1.2000
tabulate county00 regstat if regstat == '1'

// Cross-table only for persons over 30 years
generate age = 2000 - int(birth_year_month/100)
tabulate county00 regstat if age > 30

// Percentage tabulation
tabulate maritalstate00 maritalstate05, rowpct
tabulate maritalstate00 maritalstate05, colpct
tabulate maritalstate00 maritalstate05, cellpct
tabulate maritalstate00 maritalstate05, rowpct freq

// The tabulate-command may also be used to produce volume tables through a summarize-option.
This will show statistics such as means, sums etc for a specific variable distributed over the
combinations of categories of the chosen tabulate-variables

tabulate county00 gender, summarize(wealth)
```

4.8.2 Summarize and boxplot

```
// Summary statistics for metrical or continuous variables

// The command summarize is used to generate summary statistics for metrical or continuous
variables. Values shown are mean, quartiles a.o. The command boxplot presents the same figures
graphically through a standard boxplot diagram

require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/INNTEKT_WYRKINNT 2000-01-01 as income
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import fdb1/BOSATTEFDT_BOSTED 2000-01-01 as residence00

// Recoding from municipality to county level
generate county00 = substr(residence00,1,2)

// Generate age per 2000
generate age = 2000 - int(birth_year_month/100)

summarize income wealth
summarize wealth if age > 50
summarize wealth if residence00 == '0301'

boxplot income wealth
boxplot income, over( gender )
```

4.8.3 Histogram and barchart

```
// Histogram and barchart

require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/INNTEKT_WYRKINNT 2000-01-01 as income
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month

// Generate age per 2000
generate age = 2000 - int(birth_year_month/100)

// Histogram (frequency distributions)
// This is a way of presenting frequency distributions for metrical/continuous variables graphically,
where the values are grouped into appropriate intervals and the corresponding frequencies are
represented by bars. The total area of all the bars will sum into 1 by default, unless customized
through options. Options are also a tool for choosing the division of values (number of bars),
displaying a normal distribution curve as reference etc.

histogram income
histogram income, freq
histogram income, fraction
histogram income, percent
histogram income, normal
histogram income, bin(6) freq
histogram income, width(100000) freq

histogram income, by( kjonn )
histogram income if income > 100000


// By using a discrete-option, histograms may also illustrate the value distribution for discrete
variables. Each category/value will then be represented by separate bars

histogram age, discrete


// Barcharts
// Such diagrams are useful for statistical presentations of continuous/metrical variables in a lucid
manner. Several variables may be combined in the diagram, in order to break down the numbers
based on categorical characteristics (gender, educational level etc)

barchart ( mean) income, over( gender )
barchart ( mean) income wealth, over( gender )
```

4.8.4 Piechart and hexbin-plot

```
require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/INNTEKT_WYRKINNT 2000-01-01 as income
import fdb1/INNTEKT_BRUTTOFORM 2000-01-01 as wealth
import fdb1/BEFOLKNING_KJOENN as gender
import fdb1/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import fdb1/BOSATTEFDT_BOSTED 2000-01-01 as residence00

// Recoding from municipality to county level
generate county00 = substr(residence00,1,2)

define-labels countystring '01' 'Østfold' '02' 'Akershus' '03' 'Oslo' '04' 'Hedmark' '05' 'Oppland' '06'
'Buskerud' '07' 'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder' '11' 'Rogaland' '12'
'Hordaland' '14' 'Sogn and Fjordane' '15' 'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'
```

```
assign-labels county00 countystring

// Generate age per 2000
generate age = 2000 - int(birth_year_month/100)

// Piecharts
// This is a useful way of presenting the percentage shares of values for discrete variables graphically

drop if age < 16

piechart gender
piechart county00


// Hexbinplot
// This can be seen as an anonymized scatterplot diagram, suitable for continuous/metrical variables,
where the density of the plots are represented by colours (the darker, the more plots in the particular
hexbin)

hexbin wealth income
hexbin wealth income if age > 50
```

4.8.5 Sankey-diagram

```
// Transitions diagrams (Sankey)

require no.ssb.fdb:1 as fdb1

create-dataset demographics
import fdb1/SIVSTANDFDT_SIVSTAND 2000-01-01 as maritalstate00
import fdb1/SIVSTANDFDT_SIVSTAND 2005-01-01 as maritalstate05
import fdb1/BOSATTEFDT_BOSTED 2000-01-01 as residence00
import fdb1/BOSATTEFDT_BOSTED 2005-01-01 as residence05

// Recoding from municipality to county level
generate county00 = substr(residence00,1,2)
generate county05 = substr(residence05,1,2)

define-labels countystring '01' 'Østfold' '02' 'Akershus' '03' 'Oslo' '04' 'Hedmark' '05' 'Oppland' '06'
'Buskerud' '07' 'Vestfold' '08' 'Telemark' '09' 'Aust-Agder' '10' 'Vest-Agder' '11' 'Rogaland' '12'
```

```
'Hordaland' '14' 'Sogn and Fjordane' '15' 'Møre and Romsdal' '16' 'Sør-Trøndelag' '17' 'Nord-Trøndelag'
'18' 'Nordland' '19' 'Troms' '20' 'Finnmark' '99' 'Unknown'

assign-labels county00 countystring
assign-labels county05 countystring

sankey county00 county05 if county00 == '12'
sankey county00 county05 if county05 == '03'
sankey county00 county05 if county05 == '03' & county00 != '03'

sankey maritalstate00 maritalstate05
sankey maritalstate00 maritalstate05 if maritalstate00 == '2'
```

5. Advanced analysis

In addition to descriptive functionalities, microdata.no makes it possible to perform advanced analysis such as regression analysis. Presently, the following advanced analysis tools are available in microdata.no:

- correlate
- anova
- regress
- logit / probit
- mlogit
- regress-panel

More functionality will be added consecutively, based on feedback from statistical users. In principle, all functionality available in STATA may also be implemented in microdata.no.

5.1 Correlate - correlation measures

The command `correlate` is a tool for analyzing statistical correlations between variables. Values ranging from -1 to 1 are reported in a correlation matrix for the specified variables, where minus and plus-values implicate negative and positive correlation respectively. The value 0 indicates no correlation. The closer to +/- 1, the stronger is the estimated correlation.

Syntax:

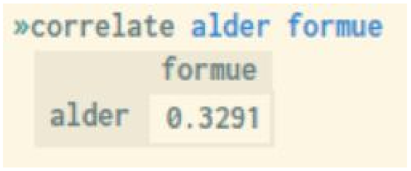
```
correlate <variable list> [if] [, options]
```

If no variable is specified, a correlation matrix for all variables in the dataset is presented.

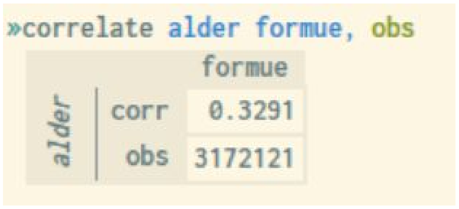
The following options may be used to present alternative measures:

- covariance	Show covariance instead of correlation coefficient
- pairwise	Pairwise presentation
- obs	Show number of observations behind each correlation coefficient
- sig	Show significance value for each correlation coefficient

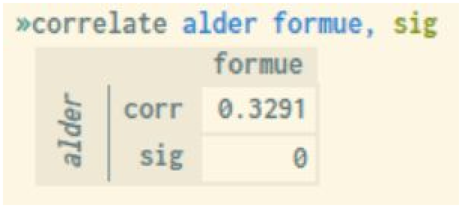
Examples:



```
»correlate alder formue
        formue
alder   0.3291
```



```
»correlate alder formue, obs
              formue
alder  corr   0.3291
       obs    3172121
```



```
»correlate alder formue, sig
              formue
alder  corr   0.3291
       sig         0
```

5.2 Anova

Anova-tests can be viewed as a simplified regression analysis, in which the focus lies in whether the mean value of a dependent continuous variable is different in two or more independent groups given by another categorical variable. One example is to test whether the average salary is different for people with low, medium, and high education (using an independent variable where the level of education is divided into three groups).

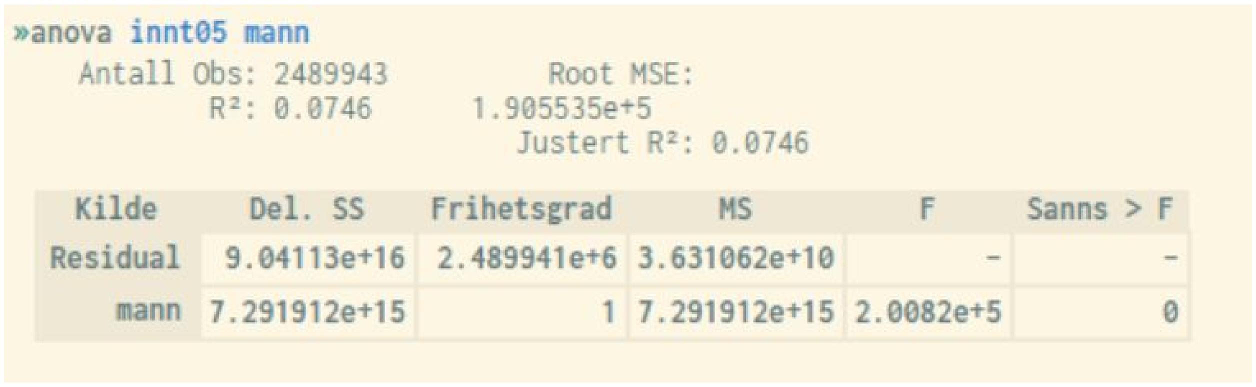
An Anova-test can check if there are significant differences between at least two of the groups (given by the independent variable), but it does not indicate which group(s) this applies to. For such purposes, regression analyses need to be performed (see section 5.3).

Syntax:

```
    anova <variable> <variable list> [if] [, options]
```

If testing two variables only, i.e. one dependent and one independent variable, a one-way Anova-test is performed. It is also possible to test a dependent variable against two independent categorical variables, also called a two-way Anova-test.

Example:



```
»anova innt05 mann
   Antall Obs: 2489943         Root MSE:
         R²: 0.0746       1.905535e+5
                            Justert R²: 0.0746
```

Kilde	Del. SS	Frihetsgrad	MS	F	Sanns > F
Residual	9.04113e+16	2.489941e+6	3.631062e+10	-	-
mann	7.291912e+15	1	7.291912e+15	2.0082e+5	0

5.3 Regress - ordinary least squares estimation

The command `regress` is a tool for performing ordinary least square estimations (OLS) where the dependent variable takes continuous/metrical values such as income.

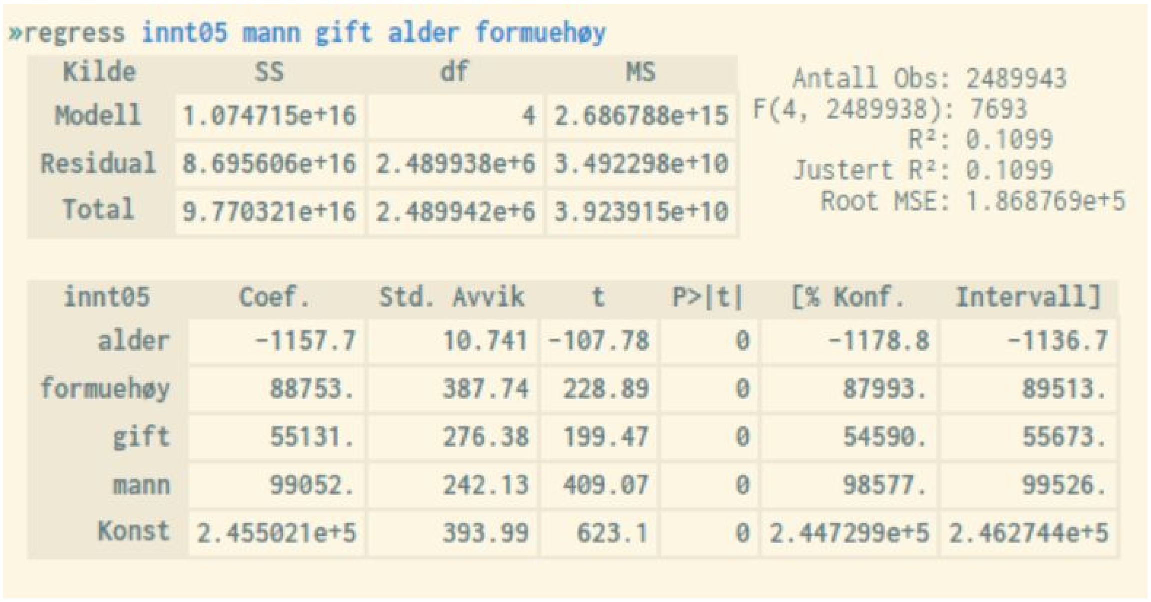
In short, the model involves estimating (possible) marginal effects from a set of independent variables (explanatory variables) on the dependent variable (response variable). "Marginal effect" is a measure of how much the dependent variable is estimated to increase in value, caused by an increase by one unit of measure in the respective independent variable.

The most important thing to look at when interpreting the result of a regression is the explanatory force of:

a) The model as a whole
b) Each variable

This is done by studying the significance values "*Adjusted* R^2" and "*P*> | *t* |" respectively.

Below is an example of the result from a regression analysis performed in microdata.no. The numbers in the lower part are linked to the different variables, while the numbers at the top refer to the analysis model as a whole.



```
»regress innt05 mann gift alder formuehøy
```

Kilde	SS	df	MS
Modell	1.074715e+16	4	2.686788e+15
Residual	8.695606e+16	2.489938e+6	3.492298e+10
Total	9.770321e+16	2.489942e+6	3.923915e+10

```
  Antall Obs: 2489943
F(4, 2489938): 7693
          R²: 0.1099
  Justert R²: 0.1099
    Root MSE: 1.868769e+5
```

innt05	Coef.	Std. Avvik	t	P>|t|	[% Konf.	Intervall]
alder	-1157.7	10.741	-107.78	0	-1178.8	-1136.7
formuehøy	88753.	387.74	228.89	0	87993.	89513.
gift	55131.	276.38	199.47	0	54590.	55673.
mann	99052.	242.13	409.07	0	98577.	99526.
Konst	2.455021e+5	393.99	623.1	0	2.447299e+5	2.462744e+5

Justert R^2 (*Adjusted R^2*) is an overall measure of how much of the observed variance in the dependent variable is explained by the sum of the independent variables, expressed in percent. The scale ranges from 0 to 1, with values near 1 being optimal. In practice, values will never reach 1 when analyzing socioeconomic individual data due to random noise and unobserved causal relationships. Typical values will therefore usually be in the range of 0 - 0.5.

The R^2 value will always increase for each additional independent variable added to the regression model. This does not necessarily mean that the model is getting better, especially if the variables added are not statistical significant. *Justert R^2* takes this into account and will only increase in value if the extra variables are significant. Therefore, it is recommended study this measure in particular.

If *Justert R^2* shows a *lower* value by adding an additional independent variable, this will indicate that the selected variable may have a relatively high degree of correlation with some of the other independent variables, i.e. multicollinearity. This is certainly something you should avoid.

P > |*t*| or the *p-values* (in column 4 in the lower regression output table) indicate the probability that the *t*-value appears as a result of pure randomness. In order to say that a variable is significant, the associated *p*-value must be lower than 0.05 at a 5% significance level. Values close to or equal to 0 are ideal.

In short, the value *t* (column 3) is a standardized measure of the coefficient value (= the marginal effect), c.f. values in the *Coef.*-column (column 1), where limit values of +/- 1.96 correspond to a 5% significance level. Thus, values exceeding 1.96 with positive or negative sign will be considered significant at a 5% level (5% level is a common operational limit).

Also the 95% confidence interval values presented in the two rightmost columns in the lower main table are useful to study, as they are quite intuitive. If the interval includes the value 0, one can rule out that the coefficient in question shows a significant relationship between the associated independent variable and the response variable.

The coefficient values in column 1 are only relevant for significant variables, and show the marginal effect on the response variable of a unit's increase in the value of the associated independent variable.

The illustration above shows that all variables are significant with a good margin (high t-values). *Alder* (Age) has a negative effect on income, while the other variables have a positive effect. *Konst* refers to the constant, i.e. the starting value of the response variable when all independent variables take the value 0, which is of no particular interpretive importance.

Furthermore, values of *Justert* R^2 at 0.1099 is not so bad considering that the model has only 4 explanatory variables.

If *Justert* R^2 shows a very low value, extra independent variables of relevance should possibly be added, or the model design may not be in accordance with the statistical requirements of the empirical data and should be revaluated. If the value goes *down* by adding an extra variable (instead of up), this may indicate multicollinearity (interrelationship between independent variables). It is therefore important to carefully plan which independent variables to use in an explanatory model. Moreover, it is of special importance to have a minimum degree of mutual correlation between the independent variables. If in doubt, correlation matrices may be used to reveal such dependencies through the following command:

```
    corr <variable list>
```

This command is reviewed in chapter 5.1.

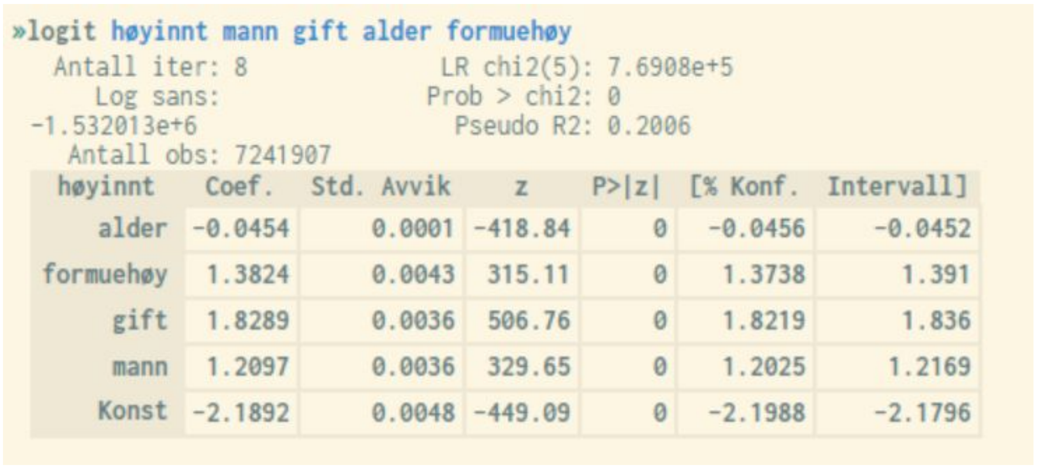
5.4 Logit and probit - logistic regression analysis

Logistic regression analysis is a tool for estimating the probability of "success" (one condition in front of another) or to end up in one of several possible states.

The commands `logit` and `probit` can be used to perform a logistic analysis where the dependent variable is a categorical variable with 2 possible outcomes (dummy variable). Examples may be job/non-job, retired/non-retired etc. Logit models assume that the probability of "success" follows a logarithmic (log) distribution, while the probit variant assumes a normal distribution. The two distributions are virtually the same, and the results will therefore be approximately the same. However, Logit is the most widely used model, and that is the one we focus on in the examples below.

The result of `logit` provides a table of common values such as coefficients, standard deviations, z-values, p-values, and confidence intervals. The numbers in the main table are linked to the different variables, while the numbers at the top refer to the analysis model as a whole (indicate the model's quality/explanatory power).

Example:



```
»logit høyinnt mann gift alder formuehøy
   Antall iter: 8              LR chi2(5): 7.6908e+5
      Log sans:               Prob > chi2: 0
 -1.532013e+6                   Pseudo R2: 0.2006
    Antall obs: 7241907
```

høyinnt	Coef.	Std. Avvik	z	P>|z|	[% Konf.	Intervall]
alder	-0.0454	0.0001	-418.84	0	-0.0456	-0.0452
formuehøy	1.3824	0.0043	315.11	0	1.3738	1.391
gift	1.8289	0.0036	506.76	0	1.8219	1.836
mann	1.2097	0.0036	329.65	0	1.2025	1.2169
Konst	-2.1892	0.0048	-449.09	0	-2.1988	-2.1796

In the example above, the dependent variable “høyinnt” (high income) is coded as follows:

```
generate høyinnt = 0
replace høyinnt = 1 if innt05 > 400000
```

Like ordinary linear regression analysis (see chapter 5.3), some numbers are more important to study than others. The P-value, *Prob> chi2,* indicates how good the statistical model is, i.e. it is an estimation of the explanatory power of the sum of all independent variables. The closer to 0 the better, and values should be below 0.05.

Pseudo R2 is a variant of *Justert R^2 (Adjusted R^2)* reported by ordinary linear regression analyzes, indicating how much of the variance in the response variable is explained by the independent variables (scale from 0 to 1 where highest possible values are ideal). However, this overall measure should be interpreted with great caution, as in many cases it indicates a value that is either artificially high or low. *Prob> chi2* is therefore recommended for logistic regression models.

The p-values of the variables, *P* > |*z*|*,* correspond to *P* > |*t*| in ordinary linear regression analysis. The limit value here is also 0.05 if operating with a significance level of 5% (commonly used). Reported values below this limit imply that the associated variable is significant at a 5% level.

Studies of *z*-values or associated p-values give the same conclusions. The z-value is a standardized version of the coefficient value, which has an expectation equal to 0 and where values exceeding +/- 1.96 imply that the corresponding variable has a significant influence on the likelihood of "success". Positive values indicate positive effect, and vice versa.

The confidence interval given by the two rightmost columns can be interpreted in the same way as for ordinary linear regression analysis, i.e. if it includes the value 0, this indicates zero significance.

As can be seen in the example above, all explanatory variables are significant with a good margin (high z-values). "Alder" (Age) has a negative effect on the probability of ending up in a high income group, while the other variables have a correspondingly positive effect. Furthermore, the model's P-value is equal to 0, which shows that we have a good explanatory model.

5.5 Mlogit - multinomial logistic regression analysis

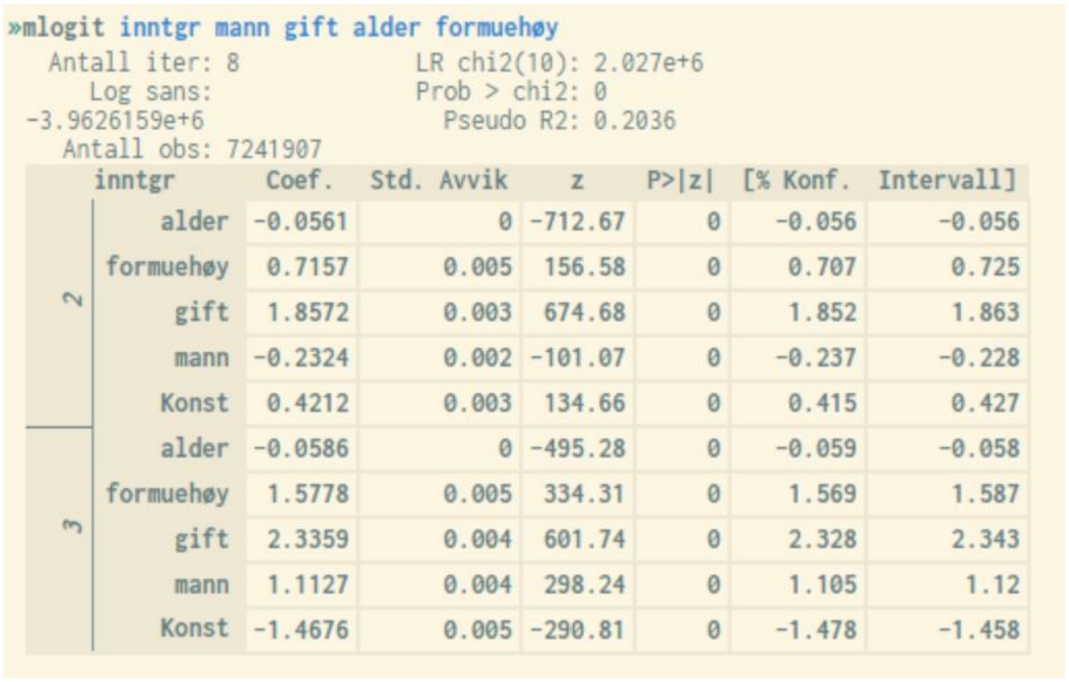
It is possible to analyze logistic regression models even when the dependent variable has more than two possible outcomes. Multinomial models can be used for such purposes.

Syntax expression:

```
     mlogit <response variable> <independent variables> [,
<options>]
```

In the reported result, the main table is more extensive compared to common (binomial) logistic models. It contains a set of coefficients, standard errors, z-values etc for each possible outcome minus the reference outcome. If e.g. three outcomes, only two sets of values are shown, where all are relative to the reference outcome. Therefore, they need to be interpreted in comparison with the probability of ending up in the reference outcome. The various reported measures are reviewed in chapter 5.4.

Example:



```
»mlogit inntgr mann gift alder formuehøy
   Antall iter: 8            LR chi2(10): 2.027e+6
      Log sans:              Prob > chi2: 0
 -3.9626159e+6                 Pseudo R2: 0.2036
    Antall obs: 7241907
```

	inntgr	Coef.	Std. Avvik	z	P>|z|	[% Konf.	Intervall]
2	alder	-0.0561	0	-712.67	0	-0.056	-0.056
	formuehøy	0.7157	0.005	156.58	0	0.707	0.725
	gift	1.8572	0.003	674.68	0	1.852	1.863
	mann	-0.2324	0.002	-101.07	0	-0.237	-0.228
	Konst	0.4212	0.003	134.66	0	0.415	0.427
3	alder	-0.0586	0	-495.28	0	-0.059	-0.058
	formuehøy	1.5778	0.005	334.31	0	1.569	1.587
	gift	2.3359	0.004	601.74	0	2.328	2.343
	mann	1.1127	0.004	298.24	0	1.105	1.12
	Konst	-1.4676	0.005	-290.81	0	-1.478	-1.458

In the example above, the dependent variable “inntgr” (income category) is coded as follows:

```
generate inntgr = 1
replace inntgr = 2 if innt05 > 200000
replace inntgr = 3 if innt05 > 400000
```

5.6 Regress-panel - panel data regression analysis

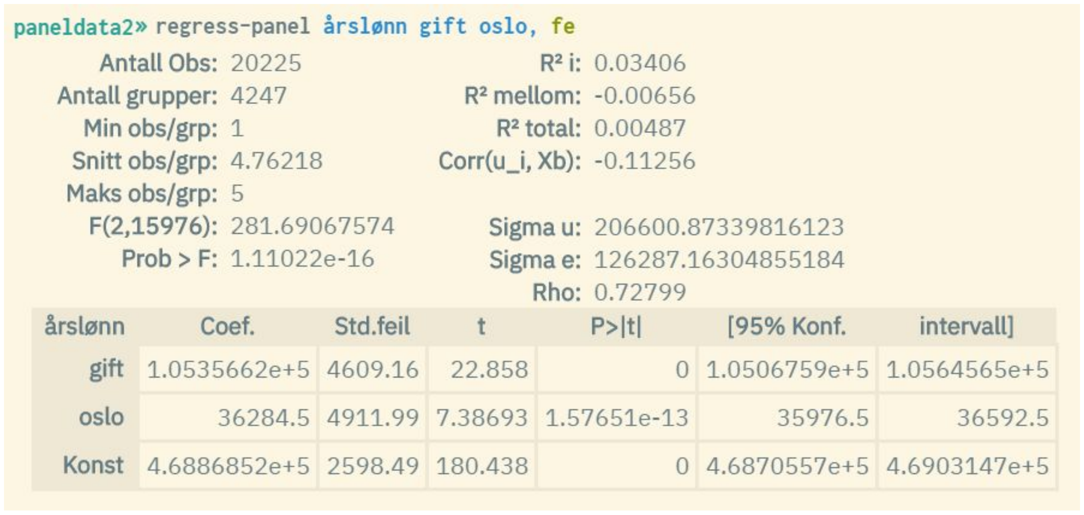
Panel data are datasets where each unit takes values for all variables measured over a specified set of measurement dates. This has the advantage that time becomes a component in analyzes. In addition, the data base becomes much larger, usually leading to analyzes of better quality.

See chapter 2.4 on how to create datasets for panel data analysis. A syntax script example is also presented there.

There is a large battery of panel data analyzes that can be used, depending on what assumptions are made about the variability of the various variables over time. Common variants used are fixed effect and random effect analyzes.

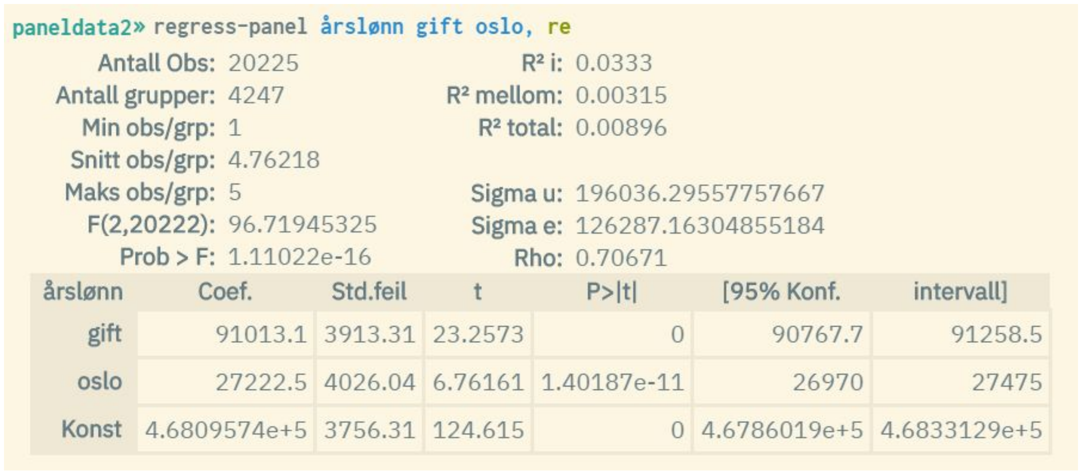
In the example below, annual salary (annual wage income) is used as a dependent variable, and dummy variables for marital status = married, and residence = Oslo are used as explanatory variables. In addition, five measurement dates are used: December 31. in the years 2011-2015. Population = all persons who completed a master's degree during the autumn semester 2010.

Example 1: Panel regression with fixed effects



```
paneldata2» regress-panel årslønn gift oslo, fe
      Antall Obs: 20225                       R² i: 0.03406
  Antall grupper: 4247                   R² mellom: -0.00656
     Min obs/grp: 1                        R² total: 0.00487
    Snitt obs/grp: 4.76218             Corr(u_i, Xb): -0.11256
    Maks obs/grp: 5
     F(2,15976): 281.69067574               Sigma u: 206600.87339816123
       Prob > F: 1.11022e-16                Sigma e: 126287.16304855184
                                                Rho: 0.72799
 årslønn        Coef.   Std.feil        t          P>|t|   [95% Konf.      intervall]
    gift 1.0535662e+5    4609.16   22.858              0  1.0506759e+5  1.0564565e+5
    oslo      36284.5    4911.99  7.38693    1.57651e-13       35976.5       36592.5
   Konst 4.6886852e+5    2598.49  180.438              0  4.6870557e+5  4.6903147e+5
```

Example 2: Panel regression with random effects (same dataset as example 1)



```
paneldata2» regress-panel årslønn gift oslo, re
      Antall Obs: 20225                       R² i: 0.0333
  Antall grupper: 4247                   R² mellom: 0.00315
     Min obs/grp: 1                        R² total: 0.00896
    Snitt obs/grp: 4.76218
    Maks obs/grp: 5                         Sigma u: 196036.29557757667
     F(2,20222): 96.71945325                Sigma e: 126287.16304855184
       Prob > F: 1.11022e-16                    Rho: 0.70671
 årslønn        Coef.   Std.feil        t          P>|t|   [95% Konf.      intervall]
    gift      91013.1    3913.31  23.2573              0       90767.7       91258.5
    oslo      27222.5    4026.04  6.76161    1.40187e-11         26970         27475
   Konst 4.6809574e+5    3756.31  124.615              0  4.6786019e+5  4.6833129e+5
```

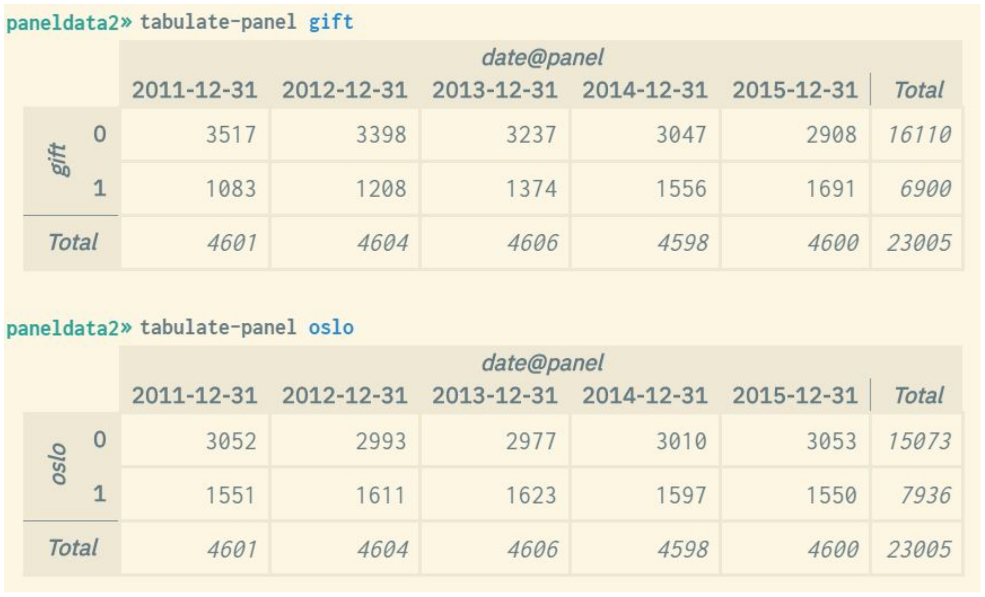
In addition to regression analyzes, it is possible to map out panel data through various descriptive tools:

- `tabulate-panel` corresponds to the command `tabulate` used for regular datasets, c.f. chapter 4.1, but shows in stead values for all measurement dates. Like `tabulate`, percentage options can be used. If multiple variables are specified, multi-dimensional cross tables are displayed for the relevant variables

- `summarize-panel` corresponds to the command `summarize` used for regular datasets, c.f. chapter 4.2, but shows instead values for all measurement dates. Values are displayed vertically and not horizontally, and the mouse cursor need to be held over the respective values to show their meaning

- `transitions-panel` shows a two-way matrix containing frequencies/probabilities of transitions between all combinations of categorical values over time (transition probabilities), for a given variable. The leading column represents the base values, while the table header represents the transition values. If multiple variables are specified, two-way transition tables are displayed for each variable. Transitions are by default represented by frequencies and percentages (row percentage). Transitions either from or to missing values (sysmiss) are kept out of the tabulation.

Example 3: Tabulate-panel for "married" and "Oslo" respectively (same dataset as example 1 and 2)



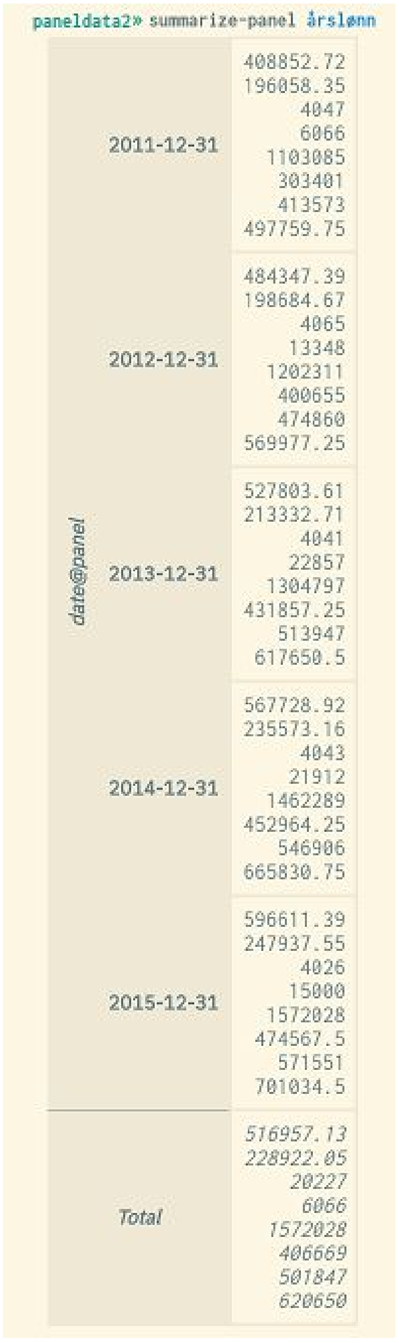
```
paneldata2» tabulate-panel gift
```

		date@panel					
		2011-12-31	2012-12-31	2013-12-31	2014-12-31	2015-12-31	*Total*
gift	0	3517	3398	3237	3047	2908	*16110*
	1	1083	1208	1374	1556	1691	*6900*
Total		*4601*	*4604*	*4606*	*4598*	*4600*	*23005*

```
paneldata2» tabulate-panel oslo
```

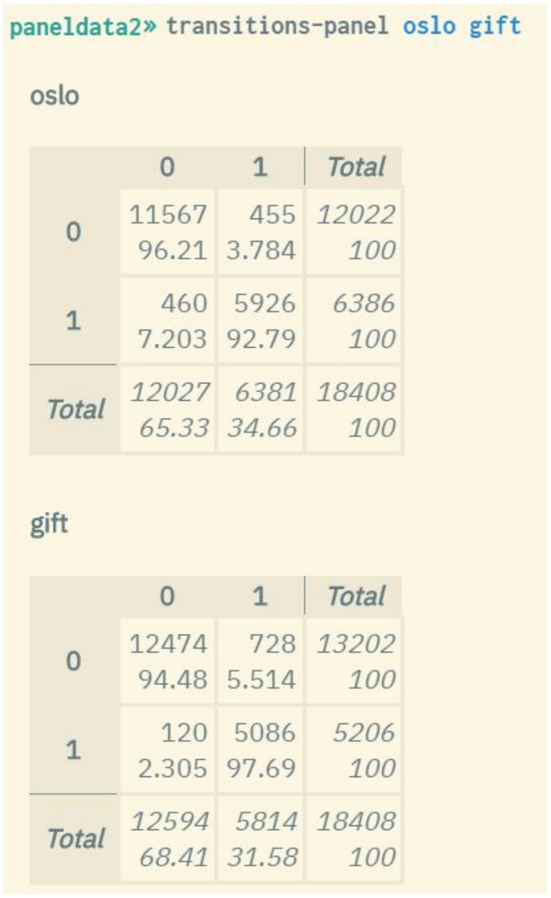
		date@panel					
		2011-12-31	2012-12-31	2013-12-31	2014-12-31	2015-12-31	*Total*
oslo	0	3052	2993	2977	3010	3053	*15073*
	1	1551	1611	1623	1597	1550	*7936*
Total		*4601*	*4604*	*4606*	*4598*	*4600*	*23005*

Example 4: Summarize-panel for the dependent variable “yearly salary” (same dataset)



```
paneldata2» summarize-panel årslønn

date@panel
             408852.72
             196058.35
                  4047
2011-12-31        6066
               1103085
                303401
                413573
             497759.75

             484347.39
             198684.67
                  4065
2012-12-31       13348
               1202311
                400655
                474860
             569977.25

             527803.61
             213332.71
                  4041
2013-12-31       22857
               1304797
             431857.25
                513947
              617650.5

             567728.92
             235573.16
                  4043
2014-12-31       21912
               1462289
             452964.25
                546906
             665830.75

             596611.39
             247937.55
                  4026
2015-12-31       15000
               1572028
              474567.5
                571551
              701034.5

             516957.13
             228922.05
                 20227
Total             6066
               1572028
                406669
                501847
                620650
```

Example 5: Transitions-panel (transition rates for combinations of categorical values) for the variables "Oslo" and "married" respectively (same dataset)



```
paneldata2» transitions-panel oslo gift
```

oslo

	0	1	*Total*
0	11567 96.21	455 3.784	*12022* *100*
1	460 7.203	5926 92.79	*6386* *100*
Total	*12027* *65.33*	*6381* *34.66*	*18408* *100*

gift

	0	1	*Total*
0	12474 94.48	728 5.514	*13202* *100*
1	120 2.305	5086 97.69	*5206* *100*
Total	*12594* *68.41*	*5814* *31.58*	*18408* *100*

Comment on table in example 5:
In 96.21% of the cases, persons not resident in Oslo will have the same condition the following year (next measurement). The rest, 3.78%, will move to Oslo. Among those in the population who live in Oslo at a given time, 7.2% will move out of Oslo while 92.8% will remain the following year (next measurement).

The same principle applies to the variable *"gift" (married)*: Here we see that 5.5% changes the status from non-married to married from one year to another (next measurement) over the total measurement period. 2.3% changes status from married to non-married.

Appendix A: Command overview

Command	**Formål**	**Type funksjonalitet**
`clear`	Clear command session (reset)	Support command
`help`	Help	Support command
`help-function`	Help - functions	Support command
`history`	Command history	Support command
`load`	Load script and execute script	Support command
`save`	Save command session as a script	Support command
`variables`	Show metadata for registry variables	Support command
`clone-dataset`	Duplicate dataset	Dataset command
`clone-units`	Duplicate units in a dataset	Dataset command
`create-dataset`	Create new and empty dataset	Dataset command
`delete-dataset`	Delete dataset	Dataset command
`rename-dataset`	Rename dataset	Dataset command
`require`	Connect to data bank	Dataset command
`use`	Use a specific dataset	Dataset command
`assign-labels`	Put labels on variables and values	Adaptation command
`clone-variables`	Duplicate variables	Adaptation command
`collapse`	Collapse/aggregate dataset	Adaptation command
`define-labels`	Define list of value-labels	Adaptation command
`destring`	Convert from alphanumeric to numeric values	Adaptation command
`drop`	Drop variables or records (drop if)	Adaptation command
`drop-labels`	Drop value-labels	Adaptation command
`generate`	Create a new variable through expression	Adaptation command
`import`	Import variable into dataset	Adaptation command
`import-event`	Import event variable into dataset (need to be empty)	Adaptation command
`import-panel`	Import panel data into dataset (need to be empty)	Adaptation command
`keep`	Keep variables or records (drop the rest)	Adaptation command
`list-labels`	Show list of custom value-labels sets	Adaptation command

`merge`	Merge variables from a dataset into another dataset. Default link key is the unit-identification of the source dataset, which can be customized via an “on”-option that make it possible to specify a custom link key	Adaptation command
`recode`	Recode numeric variable	Adaptation command
`rename`	Rename variable	Adaptation command
`replace`	Recode existing variable through expression	Adaptation command
`sample`	Create a random sample from total population	Adaptation command
`split`	Split string variables into sub parts	Adaptation command
`anova`	Anova/ancova variance analysis	Analysis command
`barchart`	Barchart diagram (categorical variables)	Analysis command
`boxplot`	Boxplot diagram (numeric variables)	Analysis command
`ci`	Confidence interval and standard errors	Analysis command
`correlate`	Correlation matrix	Analysis command
`hexbin`	Anonymized scatterplot (hexbin plot)	Analysis command
`histogram`	Histogram	Analysis command
`logit`	Logistic regression analysis: Logit	Analysis command
`mlogit`	Multinomial logistic regression analysis	Analysis command
`normaltest`	Selection of normal distribution tests for variables specified	Analysis command
`piechart`	Piechart diagram	Analysis command
`probit`	Logistic regression analysis: Probit	Analysis command
`regress`	Linear regression	Analysis command
`regress-panel`	Linear regression for panel data	Analysis command
`sankey`	Sankey diagram (transitions diagram)	Analysis command
`summarize`	Summary statistics (numeric variables)	Analysis command
`summarize-panel`	Summary statistics for panel data	Analysis command
`tabulate`	Frequency- and volume tables (categorical variables)	Analysis command
`tabulate-panel`	Frequency tables for panel data	Analysis command
`transitions-panel`	Transition-probabilities for panel data	Analysis command

Appendix B: Function overview

Misc. mathematical functions

- ln(arg1)
  - Description: The natural logarithm of arg1 (the inverse of exp(arg1))
  - arg1: Values between 1e–323 and 8e+307
  - Output: Values between -744 and 709
  - Examples:

- log10(arg1)
  - Description: The base 10-logarithm of arg1
  - arg1: Values between 1e–323 and 8e+307
  - Output: Values between −323 and 308
  - Examples:

- exp(arg1)
  - Description: The exponential function e^{arg1} (the inverse of ln(arg1))
  - arg1: Values between −8e+307 and 709
  - Output: Values between 0 and 8e+307
  - Examples:

- sqrt(arg1)
  - Description: The square root of arg1
  - arg1: Values between 0 and 8e+307
  - Output: Values between 0 and 1e+154
  - Examples:

- abs(arg1)
  - Description: The absolute value of arg1 (i.e. removes negative signs)
  - arg1: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 8e+307
  - Examples:

- sin(arg1)
  - Description: Returns the sinus value of arg1
  - arg1: Values between −1e+18 and 1e+18
  - Output: Values between -1 and 1
  - Examples:

- cos(arg1)
  - Description: Returns the cosinus value of arg1
  - arg1: Values between −1e+18 and 1e+18
  - Output: Values between -1 and 1
  - Examples:

- tan(arg1)
  - Description: Returns the tangens value of arg1
  - arg1: Values between −1e+18 and 1e+18
  - Output: Values between −1e+17 and 1e+17 eller missing
  - Examples:

- asin(arg1)
  - Description: Returns the radian value of the arcsinus of arg1
  - arg1: Values between -1 and 1
  - Output: Values between −π/2 and π/2
  - Examples:

- acos(arg1)
  - Description: Returns the radian value of the arccosinus of arg1
  - arg1: Values between -1 and 1
  - Output: Values between 0 and π
  - Examples:

- atan(arg1)
  - Description: Returns the radian value of the arctangens of arg1
  - arg1: Values between −8e+307 and 8e+307
  - Output: Values between −π/2 and π/2
  - Examples:

- ceil(arg1)
  - Description: Round upwards to nearest integer
  - arg1: Values between −8e+307 and 8e+307

- Output: Integer between −8e+307 and 8e+307
- Examples: `ceil(5.2) = 6`
  `ceil(-5.2) = -6`

- floor(arg1)
  - Description: Round downwards to nearest integer. Equal to the function int(arg1)
  - arg1: Values between −8e+307 and 8e+307
  - Output: Integer between −8e+307 and 8e+307
  - Examples: `floor(5.8) = 5`
    `floor(-5.8) = -5`

- int(arg1)
  - Description: Integer value of arg1 (i.e. drops decimals). Equal to the function floor(arg1)
  - arg1: Values between −8e+307 and 8e+307
  - Output: Integer between −8e+307 and 8e+307
  - Examples: `int(5.8) = 5`
    `int(-5.8) = -5`

- logit(arg1)
  - Description: Log value of the odds ratio of arg1 (= ln {arg1/(1-arg1)})
  - arg1: Values between 0 and 1 (not included)
  - Output: Values between −8e+307 and 8e+307 or missing
  - Examples:

- lnfactorial(arg1)
  - Description: The natural logarithm of n-factor (= ln(n!))
  - n: Integer values between 0 and 1e+305
  - Output: Values between 0 and 8e+307
  - Examples:

- comb(n,k)
  - Description: Combinational function value (= n!/{k!(nk)!})
  - n: Integer values between 1 and 1e+305
  - k: Integer values between 0 and n
  - Output: Values between 0 and 8e+307 or missing
  - Examples:

- round(arg1,arg2)
  - Description: Rounds to nearest integer if arg2 is not specified or set to 1. arg2 decides on which level to round arg1
  - arg1: Values between −8e+307 and 8e+307
  - arg2: Values between −8e+307 and 8e+307 (default = 1 if arg2 is dropped)
  - Output: Values between −8e+307 and 8e+307
  - Examples:

```
round(5.2) = 5
round(5.8) = 6
round(5.8,1) = 6
round(5.8,5) = 5
round(5.8,10) = 10
round(5.8621,0.01) = 5.86
```

- max(arg1, arg2, ….., argn)
  - Description: Returns the max value of the variables arg1, arg2, ….., argn for the given unit. Missing values are ignored (count as 0-values), given that not all values = missing
  - arg1, arg2, ….., argn: Variables with values between −8e+307 and 8e+307 or missing
  - Output: Values between −8e+307 and 8e+307 or missing
  - Examples:

- min(arg1, arg2, ….., argn)
  - Description: Returns minimum value of the variables arg1, arg2, ….., argn for the given unit. Missing values are ignored (count as 0-values), given that not all values = missing
  - arg1, arg2, ….., argn: Variables with values between −8e+307 and 8e+307 or missing
  - Output: Values between −8e+307 and 8e+307 or missing
  - Examples:

String functions

- string(arg1)
  - Description: Converts arg1 into string format
  - arg1: Values between −8e+307 and 8e+307 or missing
  - Output: arg1 converted into string format
  - Examples: `string(1234567) = '1234567'`

- upper(arg1)
  - Description: Converts text/string into uppercase (ASCII) (unicode characters outside the ASCII range are ignored)
  - arg1: String values
  - Output: String values converted into uppercase
  - Examples: `upper('abcde') = 'ABCDE'`
    `upper('abcdé') = 'ABCDé'`

- lower(arg1)
  - Description: Converts text/string into lowercase (ASCII) (unicode characters outside the ASCII range are ignored)
  - arg1: String values
  - Output: String values converted into lowercase
  - Examples: `lower('ABCDE') = 'abcde'`
    `lower('ABCDÉ') = 'abcdÉ'`

- ltrim(arg1)
  - Description: Removes leading blank characters (space) from text
  - arg1: String values
  - Output: String values without leading blanks
  - Examples: `trim(' this') = 'this'`

- rtrim(arg1)
  - Description: Removes blank characters (space) from the end of the text
  - arg1: String values
  - Output: String values without blank characters at the end
  - Examples: `trim('this ') = 'this'`

- trim(arg1)
  - Description: Removes blank characters (space) from the start and end of text value
  - arg1: String values
  - Output: String values without leading blanks or blank characters at the end
  - Examples: `trim(' this ') = 'this'`

- length(arg1)
  - Description: Returns the number of characters in a text value (ASCII) (note: for unicode characters outside the ASCII range the number of bytes is returned instead)
  - arg1: String values
  - Output: Integers >= 0
  - Examples: `length('ab') = 2`

- substr(arg1,arg2,arg3)
  - Description: Returns subpart of text starting with position arg2 and with length arg3
  - arg1: String values
  - arg2: Integer >=1 and <= -1 (negative values => the position is relative to the position of the last character)
  - arg3: Integers >= 1
  - Output: Subpart of arg1
  - Examples:
    ```
    substr('y32ssx',2,3) = '32s'
    substr('y32ssx',-3,2) = 'ss'
    substr('y32ssx',1,1) = 'y'
    ```

Sysmiss

- sysmiss(arg1)
  - Description: Logical function set to “true” if the variable arg1 takes the value “missing” (= no valid observations in the dataset)
  - arg1: Variable (all types)
  - Output: “true” or “false”
  - Examples: `generate variable1 = 0 if sysmiss(variable2)`

Density functions

- ibeta(arg1,arg2,arg3)
  - Description: Returns a value from the cumulative beta-distribution with shape parameters arg1 and arg2, also called the regularized incomplete beta function or the incomplete beta function ratio (ibeta() = 0 if arg3 < 0, ibeta() = 1 if arg3 > 1)
  - arg1: Values between 1e-10 and 1e+17
  - arg2: Values between 1e-10 and 1e+17
  - arg3: Values between -8e+307 and 8e+307 (relevant values: 0 <= arg3 <= 1)
  - Output: Values between 0 and 1
  - Examples:

- betaden(arg1,arg2,arg3)
  - Description: Returns a value from the probability density of the beta-distribution with shape parameters arg1 and arg2 (betaden() = 0 if arg3 < 0 eller arg3 > 1)
  - arg1: Values between 1e-323 and 8e+307
  - arg2: Values between 1e-323 and 8e+307
  - arg3: Values between -8e+307 and 8e+307 (relevant values: 0 <= arg3 <= 1)
  - Output: Values between 0 and 8e+307
  - Examples:

- ibetatail(arg1,arg2,arg3)
  - Description: Returns a value from the opposite cumulative beta-distribution with shape parameters arg1 and arg2, also called the complementary incomplete beta function (ibetatail() = 1 if arg3 < 0, ibetatail() = 0 if arg3 > 1)
  - arg1: Values between 1e-10 and 1e+17
  - arg2: Values between 1e-10 and 1e+17
  - arg3: Values between -8e+307 and 8e+307 (relevant values: 0 <= arg3 <= 1)
  - Output: Values between 0 and 1
  - Examples:

- invibeta(arg1,arg2,arg3)
  - Description: Returns a value from the inverse cumulative beta-distribution with shape parameters arg1 and arg2
  - arg1: Values between 1e-10 and 1e+17
  - arg2: Values between 1e-10 and 1e+17
  - arg3: Values between 0 and 1
  - Output: Values between 0 and 1
  - Examples:

- invibetatail(arg1,arg2,arg3)
  - Description: Returns a value from the inverse opposite cumulative beta-distribution with shape parameters arg1 and arg2 (ibetatail(a,b,x) = p => invibetatail(a,b,p) = x)
  - arg1: Values between 1e-10 and 1e+17
  - arg2: Values between 1e-10 and 1e+17
  - arg3: Values between 0 and 1
  - Output: Values between 0 and 1
  - Examples:

- binomial(arg1,arg2,arg3)
  - Description: Returns the probability of observing floor(arg2) or less successes in floor(arg1) tries with the probability of success in one try set to arg3. binomial() = 0 if arg2 < 0. binomial() = 1 if arg2 > arg1
  - arg1: Values between 0 and 1e+17
  - arg2: Values between −8e+307 and 8e+307 (relevant values: 0 <= arg2 < arg1)
  - arg3: Values between 0 and 1
  - Output: Values between 0 and 1
  - Examples:

- binomialp(arg1,arg2,arg3)
  - Description: Returns the probability of observing floor(arg2) successes in floor(arg1) tries with the probability of success in one try set to arg3
  - arg1: Values between 1 and 1e+6
  - arg2: Values between 0 and arg1
  - arg3: Values between 0 and 1
  - Output: Values between 0 and 1
  - Examples:

- binomialtail(arg1,arg2,arg3)
  - Description: Returns the probability of observing floor(arg2) or more successes in floor(arg1) tries with the probability of success in one try set to arg3. binomialtail() = 1 if arg2 < 0. binomialtail() = 0 if arg2 > arg1
  - arg1: Values between 0 and 1e+17
  - arg2: Values between −8e+307 and 8e+307 (relevant values: 0 <= arg2 < arg1)
  - arg3: Values between 0 and 1
  - Output: Values between 0 and 1
  - Examples:

- chi2(arg1,arg2)
  - Description: Returns a value from the cumulative chisquare-distribution with arg1 degrees of freedom (chi2() = 0 if arg2 < 0)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between −8e+307 and 8e+307 (relevant values: arg2 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- chi2den(arg1,arg2)
  - Description: Returns a value from probability density of the chisquare-distribution with arg1 degrees of freedom (chi2den() = 0 if arg2 < 0)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between −8e+307 and 8e+307 (relevant values: arg2 >= 0)
  - Output: Values between 0 and 8e+307
  - Examples:

- chi2tail(arg1,arg2)
  - Description: Returns a value from the opposite cumulative chisquare-distribution with arg1 degrees of freedom (chi2tail() = 1 if arg2 < 0). chi2tail() = 1 − chi2()
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between −8e+307 and 8e+307 (relevant values: arg2 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- invchi2(arg1,arg2)
  - Description: Returns a value from the inverse of the cumulative chisquare-distribution with arg1 degrees of freedom (chi2(arg1,arg2) = p => invchi2(arg1,p) = arg2)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between 0 and 1
  - Output: Values between 0 and 8e+307
  - Examples:

- invchi2tail(arg1,arg2)
  - Description: Returns a value from the inverse of the opposite cumulative chisquare-distribution with arg1 degrees of freedom (chi2tail(arg1,arg2) = p => invchi2tail(arg1,p) = arg2)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between 0 and 1
  - Output: Values between 0 and 8e+307
  - Examples:

- nchi2(arg1,arg2,arg3)
  - Description: Returns a value from the cumulative non-centered chisquare-distribution with arg1 degrees of freedom and center parameter arg2 (noncentral parameter), where arg3 is chisquare value (nchi2() = 0 if arg3 < 0)
  - arg1: Values between 2e–10 and 1e+6
  - arg2: Values between 0 and 10000
  - arg3: Values between −8e+307 and 8e+307 (relevant values: arg3 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- nchi2den(arg1,arg2,arg3)
  - Description: Returns a value from the probability density of the non-centered chisquare-distribution with arg1 degrees of freedom and center parameter arg2 (noncentral parameter), where arg3 is chisquare value (nchi2den() = 0 if arg3 < 0)
  - arg1: Values between 2e–10 and 1e+6
  - arg2: Values between 0 and 10000
  - arg3: Values between −8e+307 and 8e+307 (relevant values: arg3 >= 0)
  - Output: Values between 0 and 8e+307
  - Examples:

- nchi2tail(arg1,arg2,arg3)
  - Description: Returns a value from the opposite cumulative non-centered chisquare-distribution with arg1 degrees of freedom and center parameter arg2 (noncentral parameter), where arg3 is chisquare value (nchi2tail() = 1 if arg3 < 0)
  - arg1: Values between 2e–10 and 1e+6
  - arg2: Values between 0 and 10000
  - arg3: Values between −8e+307 and 8e+307 (relevant values: arg3 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- t(arg1,arg2)
  - Description: Returns a value from the cumulative Student's t-distribution with arg1 degrees of freedom
  - arg1: Values between 2e-10 and 2e+17
  - arg2: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 1
  - Examples:

- tden(arg1,arg2)
  - Description: Returns a value from probability density of Student's t-distribution with arg1 degrees of freedom
  - arg1: Values between 1e–323 and 8e+307
  - arg2: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 0.39894 . . .
  - Examples:

- ttail(arg1,arg2)
  - Description: Returns a value from the opposite cumulative Student's t-distribution with arg1 degrees of freedom
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 1
  - Examples:

- invt(arg1,arg2)
  - Description: Returns a value from the inverse cumulative Student's t-distribution with arg1 degrees of freedom (t(arg1,arg2) = p => invt(arg1,p) = arg2)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between 0 and 1
  - Output: Values between −8e+307 and 8e+307
  - Examples:

- invttail(arg1,arg2)
  - Description: Returns a value from the inverse opposite cumulative Student's t-distribution with arg1 degrees of freedom (ttail(arg1,arg2) = p => invttail(arg1,p) = arg2)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between 0 and 1
  - Output: Values between −8e+307 and 8e+307
  - Examples:

- nt(arg1,arg2,arg3)
  - Description: Returns a value from the cumulative non-centered Student's t-distribution with arg1 degrees of freedom and center parameter arg2 (nt(arg1,0,arg3) = t(arg1,arg3))
  - arg1: Values between 1e–100 and 1e+10
  - arg2: Values between -1000 and 1000
  - arg3: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 1
  - Examples:

- ntden(arg1,arg2,arg3)
  - Description: Returns a value from the probability density of the non-centered Student's t-distribution with arg1 degrees of freedom and center parameter arg2
  - arg1: Values between 1e–100 and 1e+10
  - arg2: Values between -1000 and 1000
  - arg3: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 0.39894 . . .
  - Examples:

- nttail(arg1,arg2,arg3)
  - Description: Returns a value from the opposite cumulative non-centered Student's t-distribution with arg1 degrees of freedom and center parameter arg2
  - arg1: Values between 1e–100 and 1e+10
  - arg2: Values between -1000 and 1000
  - arg3: Values between −8e+307 and 8e+307
  - Output: Values between 0 and 1
  - Examples:

- invnttail(arg1,arg2,arg3)
  - Description: Returns a value from the inverse opposite cumulative non-centered Student's t-distribution with arg1 degrees of freedom and center parameter arg2 (nttail(arg1,arg2,arg3) = p => invnttail(arg1,arg2,p) = arg3)
  - arg1: Values between 1 and 1e+6
  - arg2: Values between -1000 and 1000
  - arg3: Values between 0 and 1
  - Output: Values between -8e+10 and 8e+10
  - Examples:

- F(arg1,arg2,arg3)
  - Description: Returns a value from the cumulative F-distribution with arg1 and arg2 degrees of freedom in the numerator and denominator respectively (F() = 0 if arg3 < 0)
  - arg1: Values between 2e–10 and 2e+17
  - arg2: Values between 2e–10 and 2e+17
  - arg3: Values between −8e+307 and 8e+307 (relevant values: arg3 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- Fden(arg1,arg2,arg3)
    - Description: Returns a value from the probability density of the F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively (Fden() = 0 if arg3 < 0)
    - arg1: Values between 1e–323 and 8e+307
    - arg2: Values between 1e–323 and 8e+307
    - arg3: Values between -8e+307 and 8e+307 (relevant values: arg3 >= 0)
    - Output: Values between 0 and 8e+307
    - Examples:

- Ftail(arg1,arg2,arg3)
    - Description: Returns a value from the opposite cumulative F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively (Ftail() = 1 - F(), Ftail() = 1 if arg3 < 0)
    - arg1: Values between 2e–10 and 2e+17
    - arg2: Values between 2e–10 and 2e+17
    - arg3: Values between −8e+307 and 8e+307 (relevant values: arg3 >= 0)
    - Output: Values between 0 and 1
    - Examples:

- invF(arg1,arg2,arg3)
    - Description: Returns a value from the inverse cumulative F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively (F(arg1,arg2,arg3) = p => invF(arg1,arg2,p) = arg3)
    - arg1: Values between 2e–10 and 2e+17
    - arg2: Values between 2e–10 and 2e+17
    - arg3: Values between 0 and 1
    - Output: Values between 0 and 8e+307
    - Examples:

- invFtail(arg1,arg2,arg3)
    - Description: Returns a value from the inverse opposite cumulative F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively (Ftail(arg1,arg2,arg3) = p => invFtail(arg1,arg2,p) = arg3)
    - arg1: Values between 2e–10 and 2e+17
    - arg2: Values between 2e–10 and 2e+17
    - arg3: Values between 0 and 1
    - Output: Values between 0 and 8e+307
    - Examples:

- nF(arg1,arg2,arg3,arg4)
  - Description: Returns a value from the cumulative non-centered F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively, and center parameter arg3 (nF(arg1,arg2,0,arg4) = F(arg1,arg2,arg4), nF() = 0 if arg4 < 0)
  - arg1: Values between 2e–10 and 1e+8
  - arg2: Values between 2e–10 and 1e+8
  - arg3: Values between 0 and 10000
  - arg4: Values between -8e+307 and 8e+307 (relevant values: arg4 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- nFden(arg1,arg2,arg3,arg4)
  - Description: Returns a value from the probability density of the non-centered F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively, and center parameter arg3 (nFden(arg1,arg2,0,arg4) = Fden(arg1,arg2,arg4), nFden() = 0 if arg4 < 0)
  - arg1: Values between 1e–323 and 8e+307
  - arg2: Values between 1e–323 and 8e+307
  - arg3: Values between 0 and 1000
  - arg4: Values between -8e+307 and 8e+307 (relevant values: arg4 >= 0)
  - Output: Values between 0 and 8e+307
  - Examples:

- nFtail(arg1,arg2,arg3,arg4)
  - Description: Returns a value from the opposite cumulative non-centered F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively, and center parameter arg3 (nFtail() = 1 if arg4 < 0)
  - arg1: Values between 1e–323 and 8e+307
  - arg2: Values between 1e–323 and 8e+307
  - arg3: Values between 0 and 1000
  - arg4: Values between -8e+307 and 8e+307 (relevant values: arg4 >= 0)
  - Output: Values between 0 and 1
  - Examples:

- invnFtail(arg1,arg2,arg3,arg4)
  - Description: Returns a value from the inverse opposite cumulative non-centered F-distribution with arg1 and arg2 degrees of freedom in numerator and denominator respectively, and center parameter arg3 (nFtail(arg1,arg2,arg3,arg4) = p => invnFtail(arg1,arg2,arg3,p) = arg4)
  - arg1: Values between 1e–323 and 8e+307
  - arg2: Values between 1e–323 and 8e+307
  - arg3: Values between 0 and 1000
  - arg4: Values between 0 and 1
  - Output: Values between 0 and 8e+307
  - Examples:

- normal(arg1)
  - Description: Returns a value from the cumulative standardized normal distribution
  - arg1: Values between -8e+307 and 8e+307
  - Output: Values between 0 and 1
  - Examples:

- normalden(arg1,arg2,arg3)
  - Description: Returns a value from the normal distribution with mean value arg2 and standard deviation arg3
  - arg1: Values between -8e+307 and 8e+307
  - arg2: Values between -8e+307 and 8e+307
  - arg3: Values between 1e-308 and 8e+307
  - Output: Values between 0 and 8e+307
  - Examples:

Date functions

Dates in microdata.no utilize a date format that indicates the number of days measured from 01.01.1970. This makes it trivial to measure the number of days between two dates, and to calculate duration in a state (duration = stop date - start date).

The date functions listed below can be used to convert from built-in date format to more intuitive values, such as year, month, day of the week, etc.

- date(arg1, arg2, arg3)
  - Description: Converts from set date to built-in date format (number of days from 01.01.1970)
  - arg1: Year (4 digits)
  - arg2: Month (1-12)
  - arg3: Day (1-31)
  - Output: Built-in date format (number of days from 01.01.1970)
  - Examples:
    - date(2015,12,31) = 16800
    - date(1970,1,1) = 0
    - date(1967,5,27) = -950

- year(arg1)
  - Description: Retrieves year from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
  - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
  - Output: Year corresponding to date value
  - Examples:
    - year(16800) = 2015
    - year(0) = 1970
    - year(-950) = 1967

- month(arg1)
  - Description: Retrieves month from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
  - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
  - Output: Month corresponding to date value (1-12)
  - Examples:
    - month(16800) = 12
    - month(0) = 1
    - month(-950) = 5

- day(arg1)
  - Description: Retrieves day in month from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
  - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
  - Output: Day in month corresponding to date value (1-31)
  - Examples:
    - day(16800) = 31
    - day(0) = 1

        - day(-950) = 27

- dow(arg1)
    - Description: Retrieves weekday from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
    - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
    - Output: Weekday corresponding to date value (1-7) (1 = monday, 2 = tuesday etc)
    - Examples:
        - dow(16800) = 4
        - dow(0) = 4
        - dow(-950) = 6

- doy(arg1)
    - Description: Retrieves day of year from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
    - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
    - Output: Day of year corresponding to date value (1-366)
    - Examples:
        - doy(16800) = 365
        - doy(0) = 1
        - doy(-950) = 147

- week(arg1)
    - Description: Retrieves week number from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
    - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
    - Output: Week number corresponding to date value (1-53)
    - Examples:
        - week(16800) = 53
        - week(0) = 1
        - week(-950) = 21

- quarter(arg1)
    - Description: Retrieves quarter from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
    - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
    - Output: Quarter corresponding to date value (1-4)
    - Examples:
        - quarter(16800) = 4
        - quarter(0) = 1
        - quarter(-950) = 2

- halfyear(arg1)
  - Description: Retrieves value for first or second half of year from date value. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
  - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
  - Output: First or second half of year corresponding to date value (1-2)
  - Examples:
    - halfyear(16800) = 2
    - halfyear(0) = 1
    - halfyear(-950) = 1

- isoformatdate(arg1)
  - Description: Converts from date value to the YYYY-MM-DD format. Can be used on start and stop variables to convert from built-in date value format (1970-01-01 = 0)
  - arg1: Date value variable (START@<*variable name*> or STOP@<*variable name*>)
  - Output: Date on the YYYY-MM-DD format (string value)
  - Examples:
    - isoformatdate(16800) = '2015-12-31'
    - isoformatdate(0) = '1970-01-01'
    - isoformatdate(-950) = '1967-05-27'

Appendix C: Confidentiality in microdata.no

Background

The Act on Official Statistics and Statistics Norway (LOV-1989-06-16-54) § 2-5 (use of information) section (1) states that "If information is made available, confidentiality pursuant to § 2-4 shall also apply to the recipient of information." Such data can only be provided to researchers in approved research institutions or researchers who have funding from the Research Council of Norway. Therefore, strict requirements are imposed on the supply of data for research and an application for access to microdata for research is a long process. You can find the criteria for applying for access to research data for research at Statistics Norway's pages on data for research.

The microdata.no analysis system is designed to make it possible to access microdata from registers without having to go through the lengthy application process for obtaining data. But it is a condition for such a simplification that the security and confidentiality of the microdata are as well taken care of as when delivered, preferably better. It has therefore been an explicit requirement from the outset that users should not be able to view microdata or otherwise be able to disclose information about individuals. When Statistics Norway publishes official statistics, this is aggregated data. Nevertheless, Statistics Norway must make sure through various types of measures that it is not possible to disclose information about individuals or other types of statistical units to which the statistics relate.

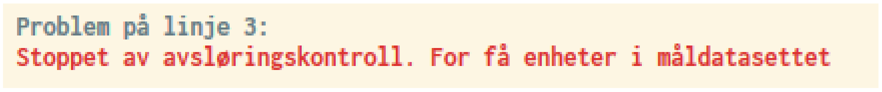
The results/output that microdata.no produces for its users (tables or analyzes) are, like Statistics Norway's statistics, aggregated data. But without limitations, a user of microdata.no could easily produce tables and other types of statistical results that Statistics Norway would not be able to publish. To prevent this from happening, several types of measures have been introduced that will limit the possibilities of being able to disclose information that should be confidential.

This appendix will describe the measures implemented to safeguard confidentiality in microdata.no. The measures are based on scenarios on how the confidentiality of microdata.no can be attacked or accidentally compromised. These scenarios will not be described. Emphasis will however be placed on what is necessary, in order for the user to understand the measures implemented and properly relate to the statistical results.

The measures described below are those that have been implemented so far. There will be more measures added over time or even adjustments to the measures described below. This appendix will be updated when changes occur.

Measure 1: Minimum population size

It is not allowed to define populations with fewer than 1000 people. Attempts to define such will be met with an error message of the type



```
Problem på linje 3:
Stoppet av avsløringskontroll. For få enheter i måldatasettet
```

Measure 2: Winsorisation

Winsorisation is a technique often used in analyzes to prevent extreme observations from having too much influence on the analysis results. The technique is applied to all numerical variables and consists of cutting the distribution at both ends by specific percentiles. We use 2% winsorisation which means that the 1% highest values are set to the 99-percentile and the 1% lowest values are set to the 1-percentile. This occurs when importing variables and in relation to the distribution in the population that the user has filtered out as their study population.

The distributions for many numerical statistical variables will be skewed, typically with long tails at the upper end (e.g. income or wealth). Therefore, winsorization will affect average and standard deviations. Both types of statistics will typically be estimated too low. On the other hand, medians, quartiles and other percentiles will not be affected.

Winsorization will automatically take place when using the commands

```
import

keep if

drop if
```

and will always be related to the distribution in the population defined by these commands regardless of the order in which they are executed.

Example: Consider the following script where the goal is to create descriptive statistics and histograms for the age distribution of the population as of 2010.

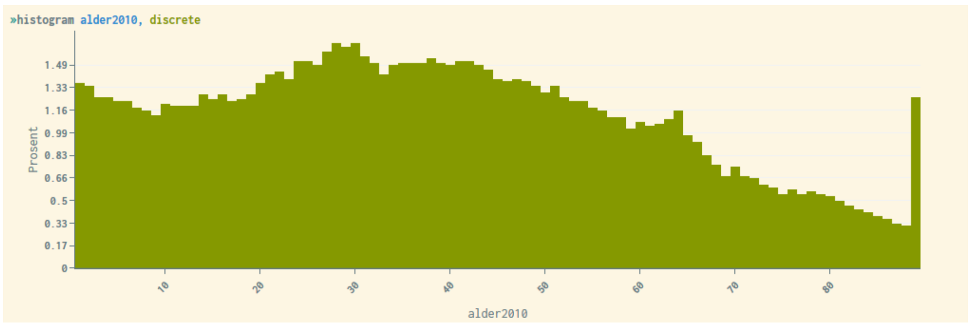
```
import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
generate faar = floor(faarmnd/100)
drop faarmnd
generate alder2010 = 2010 - faar
summarize alder2010
histogram alder2010, discrete
```

Note: faarmnd = birth year and month (*YYYYMM*), and alder2010 = age per 2010

The result will look like this:

summarize alder2010

Variabel	**mean**	**std**	**count**	**min**	**max**	**25%**	**50%**	**75%**
alder2010	38.7467	22.681	255743	1	89	21	37	55



```
»histogram alder2010, discrete
```

Anyone older than 89 years will be set to 89, and 0 year olds will be set to 1 year. This is the cause of the large bar to the right of the histogram. We are aware that the winsorisation can create problems when studying the elders. The same applies to studies of other groups that are defined by belonging to the tail in the distribution for a numerical variable.

Winsorisation affects all statistics, analyzes, and graphical plots, and prevents the most extreme values from being visible or influencing the analysis.

Measure 3: Randomized noise

All counts of the number of units in a population or subpopulation through the commands **tabulate** or **summarize** are noise-inflicted. Summations of numerical statistics variables associated with the units in a table cell, such as income, will be adjusted proportionally to the noise factor so that average numbers are unaffected. Where the random noise results in the number of units behind the sum being 0, the sum is set to 0 and the average, which then becomes 0/0 is set to NaN.

Let n be the original number without noise (for example in a table cell), and X is the noise (stochastic, integer). Then microdata.no will show the noise-inflicted number

$$Y = X + n$$

The random noise is defined by statistical distributions with the following requirements:

1. The smallest positive number to be displayed in counts, Y, should be 5. Numbers 1-4 should not be shown in the tables. But $Y = 0$ may occur
2. No counts (numbers) should be noise-inflicted by more than ± 5, i.e. $-5 \leq X \leq 5$.
3. It should not be possible to repeat the same count several times within the framework of the same population and get different results. In this sense, the random noise is *constant*.
4. It must not be possible to distinguish between true zeros and zeros resulting from noise infliction.
5. The noise is stochastic with expectation 0, $E(X) = 0$.
6. Under conditions 1-3 and 5, the noise distribution that generates X should be a maximum entropy distribution, i.e. if $p(x) = P(X = x), -5 \leq x \leq 5$, then $p(x)$ should maximize
$$\mathcal{E}(p|n) = -\sum_{x=-5}^{5} p(x|n)\log\big(p(x|n)\big) = -E(\log p(X|n))$$

The maximum entropy distribution is, to some extent, the noise distribution that removes the most information about the original value n under the given conditions.

In order to support the user's interpretation of the uncertainty that the noise infliction entails, we present the exact noise distributions that conditions 1-3 and 5-6 generate for different values of n in table C1. Based on table C1, we derive *confidence distributions* for n given Y in tables C2 and C3. A confidence distribution is in itself a stochastic size that depends on the observed value of Y and not a probability distribution for n. (And here the *confidence* has nothing to do with confidentiality.)

$p(x)$	$n=0$	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n>=10$
$x=-5$	0,0000	0,0000	0,0000	0,0000	0,0000	0,2987	0,0000	0,0000	0,0000	0,0000	0,0909
$x=-4$	0,0000	0,0000	0,0000	0,0000	0,3988	0,0000	0,0000	0,0000	0,0000	0,1296	0,0909
$x=-3$	0,0000	0,0000	0,0000	0,5175	0,0000	0,0000	0,0000	0,0000	0,1908	0,1219	0,0909
$x=-2$	0,0000	0,0000	0,6555	0,0000	0,0000	0,0000	0,0000	0,2940	0,1634	0,1147	0,0909
$x=-1$	0,0000	0,8149	0,0000	0,0000	0,0000	0,0000	0,4880	0,2145	0,1400	0,1079	0,0909
$x=0$	1,0000	0,0000	0,0000	0,0000	0,0000	0,1573	0,2522	0,1565	0,1199	0,1015	0,0909
$x=1$	0,0000	0,0000	0,0000	0,0000	0,1657	0,1383	0,1303	0,1142	0,1027	0,0955	0,0909
$x=2$	0,0000	0,0000	0,0000	0,1646	0,1390	0,1217	0,0674	0,0833	0,0880	0,0899	0,0909
$x=3$	0,0000	0,0000	0,1499	0,1309	0,1166	0,1070	0,0348	0,0608	0,0754	0,0846	0,0909
$x=4$	0,0000	0,1108	0,1116	0,1041	0,0978	0,0941	0,0180	0,0444	0,0645	0,0796	0,0909
$x=5$	0,0000	0,0743	0,0830	0,0828	0,0821	0,0828	0,0093	0,0324	0,0553	0,0748	0,0909
$\mathrm{Sum}(p(x))$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
$E(X\|n)$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$Var(X\|n)$	0,0000	4,4460	7,8320	10,2306	11,7688	12,6325	1,7215	3,9038	6,0585	8,0973	10,0000
$\mathcal{E}(p\|n)$	0,0000	0,6038	1,0126	1,3459	1,6219	1,8497	1,3775	1,8547	2,1210	2,2874	2,3979

Table C1 Noise probability distribution for different values of n.

🟦	Combinations where $Y = n + X < 0$.
🟥	Combinations where $1 \leq Y = n + X \leq 5$

By summing the numbers in Table C1 that give the same value for y= $x + n$ and dividing by the sum (standardize to sum equal to 1), table C2 is derived. Table C2 indicates what we can call *confidence levels* for each value of n given the value y that microdata.no has returned for Y. Since we are looking at n as a fixed number and not a stochastic variable, the confidence levels $cf(n|y)$ are *not* probabilities in the normal sense, even if they sum up to 1.

For $n > 10$ the noise distribution will be flat with $p(x) = \frac{1}{11} \approx 0{,}0909$ for any $x \epsilon \{-5, \dots, 5\}$ as for $n = 10$.

Note that in tables, the inner and the marginal cells will be noise inflicted independent of each other. ***Noise inflicted tables will therefore not be additive****. The noise variance of the marginal cells becomes the same as for inner cells, and less than the variance summed over the inner cells.*

$cf(n \mid Y=y)$	$Y=0$	$Y=5$	$Y=6$	$Y=7$	$Y=8$	$Y=9$	$Y=10$	$Y=11$	$Y=12$	$Y=13$	$Y=14$	$Y=15$
$n=0$	**0,2713**	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n=1$	**0,2211**	**0,0571**	**0,0486**	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n=2$	**0,1779**	**0,0772**	**0,0730**	**0,0670**	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n=3$	**0,1404**	**0,0848**	**0,0857**	**0,0840**	**0,0781**	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n=4$	**0,1082**	**0,0853**	**0,0910**	**0,0941**	**0,0922**	**0,0861**	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n=5$	**0,0810**	**0,0810**	**0,0905**	**0,0982**	**0,1009**	**0,0988**	**0,0930**	0,0000	0,0000	0,0000	0,0000	0,0000
$n=6$	0,0000	**0,2513**	**0,1650**	**0,1051**	**0,0635**	**0,0365**	**0,0202**	**0,0109**	0,0000	0,0000	0,0000	0,0000
$n=7$	0,0000	**0,1514**	**0,1404**	**0,1262**	**0,1077**	**0,0874**	**0,0683**	**0,0519**	**0,0356**	0,0000	0,0000	0,0000
$n=8$	0,0000	**0,0983**	**0,1070**	**0,1129**	**0,1130**	**0,1078**	**0,0988**	**0,0881**	**0,0710**	**0,0580**	0,0000	0,0000
$n=9$	0,0000	**0,0667**	**0,0798**	**0,0925**	**0,1017**	**0,1065**	**0,1073**	**0,1051**	**0,0930**	**0,0835**	**0,0761**	0,0000
$n=10$	0,0000	**0,0468**	**0,0595**	**0,0733**	**0,0857**	**0,0954**	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=11$	0,0000	0,0000	**0,0595**	**0,0733**	**0,0857**	**0,0954**	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=12$	0,0000	0,0000	0,0000	**0,0733**	**0,0857**	**0,0954**	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=13$	0,0000	0,0000	0,0000	0,0000	**0,0857**	**0,0954**	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=14$	0,0000	0,0000	0,0000	0,0000	0,0000	**0,0954**	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=15$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,1021**	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=16$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,1063**	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=17$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,1000**	**0,0954**	**0,0924**	**0,0909**
$n=18$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,0954**	**0,0924**	**0,0909**
$n=19$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,0924**	**0,0909**
$n=20$	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	**0,0909**
Sum	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

Table C2 Confidence distribution of n for different values of y. Positive confidence levels are represented by bold font.

$CD(n|y)$ in table C3 are cumulative aggregations of the confidence levels from table C2 defined as

$$CD(n|y) = \sum_{j=0}^{n} cd(j|y)$$

$CD(n\|y)$	$Y = 0$	$Y = 5$	$Y = 6$	$Y = 7$	$Y = 8$	$Y = 9$	$Y = 10$	$Y = 11$	$Y = 12$	$Y = 13$	$Y = 14$	$Y = 15$
$n = 0$	0,2713	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 1$	0,4924	0,0571	0,0486	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 2$	0,6703	0,1343	0,1217	0,0670	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 3$	0,8107	0,2190	0,2073	0,1510	0,0781	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 4$	0,9190	0,3044	0,2983	0,2450	0,1703	0,0861	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 5$	1,0000	0,3854	0,3888	0,3432	0,2712	0,1849	0,0930	0,0000	0,0000	0,0000	0,0000	0,0000
$n = 6$	1,0000	0,6367	0,5539	0,4483	0,3347	0,2214	0,1132	0,0109	0,0000	0,0000	0,0000	0,0000
$n = 7$	1,0000	0,7882	0,6943	0,5746	0,4424	0,3088	0,1815	0,0627	0,0356	0,0000	0,0000	0,0000
$n = 8$	1,0000	0,8864	0,8012	0,6875	0,5554	0,4166	0,2802	0,1508	0,1066	0,0580	0,0000	0,0000
$n = 9$	1,0000	0,9532	0,8810	0,7800	0,6572	0,5231	0,3875	0,2559	0,1997	0,1415	0,0761	0,0000
$n = 10$	1,0000	1,0000	0,9405	0,8533	0,7429	0,6185	0,4896	0,3622	0,2997	0,2369	0,1685	0,0909
$n = 11$	1,0000	1,0000	1,0000	0,9267	0,8286	0,7139	0,5917	0,4685	0,3998	0,3323	0,2609	0,1818
$n = 12$	1,0000	1,0000	1,0000	1,0000	0,9143	0,8092	0,6938	0,5748	0,4998	0,4277	0,3532	0,2727
$n = 13$	1,0000	1,0000	1,0000	1,0000	1,0000	0,9046	0,7958	0,6811	0,5998	0,5230	0,4456	0,3636
$n = 14$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,8979	0,7874	0,6999	0,6184	0,5380	0,4545
$n = 15$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,8937	0,7999	0,7138	0,6304	0,5455
$n = 16$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,9000	0,8092	0,7228	0,6364
$n = 17$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,9046	0,8152	0,7273
$n = 18$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,9076	0,8182
$n = 19$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	0,9091
$n = 20$	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000

Table C3 Cumulative confidence distribution for different values of y.

Let $\{5-9\}$ denote the set of values $\{5,6,7,8,9\}$. The confidence level of the values $\{5-9\}$ if observing for example $Y = 7$ then becomes

$$cd(\{5-9\}|Y = 7) = CD(9|Y = 7) - CD(4|Y = 7) = 0{,}7800 - 0{,}2450 = 0{,}5350$$

This corresponds to a confidence interval. Note again that it is not the same as probabilities since n is not stochastic.

If $Y > 15$, the confidence distribution $cd(y|n)$ in table C2 will be flat and equal to $\frac{1}{11} \approx 0{,}0909$ for all values of n from $Y - 5$ to $Y + 5$, as for $Y = 15$. Let a and b be integer numbers where $b > a$. Suppose that $Y = y \geq 15$. Then,

$$cd(\{a - b\}|y) = CD(b|y) - CD(a|y) = (\min(b, y + 5) - \max(a, y - 5))/11$$

Example: Let $a = 37, b = 44$ and $y = 39$. Then,

$$cd(\{37 - 44\}) = \frac{\min(44{,}44) - \max(37{,}34)}{11} = \frac{44 - 37}{11} = \frac{7}{11} \approx 0{,}6364.$$

When aggregating numerical sizes, for example in table cells, the sums will be adjusted in relation to the noise added.

Let Z_i be the value of a numerical variable (such as an income variable) for person number i, and T_c the original sum of this variable in a cell c of n_c persons, that is

$$T_c = \sum_{i \epsilon c} Z_i,$$

Suppose n_c is noise-added into Y_c. Then T_c will be adjusted to

$$T_c^* = \frac{Y_c}{n_c} T_c$$

This adjustment can be dramatic for cells with few observations but will have less importance in cells with many observations. This is however the intention. Also note that

$$\bar{Z}_c = \frac{T_c^*}{Y_c} = \frac{T_c}{n_c}$$

So the average is not affected, except if $Y_c = 0$. Then $\bar{Z}_c = NaN$.

If $Y_c \leq 9$, also standard deviation, median and quartiles, which can be generated by the **summarize**-option in **tabulate**, will be set to NaN.

Measure 4: Graphic plots - Hexbin plots

It is common to use scatterplot diagrams to establish a visual image of data or to show the relationship between numerical variables. Such plots can be very revealing, especially if there are few observations in relation to the graphical area or in areas outside the main mass of points. If, for a given unit/person in the population, the value of one of the variables that span out the plot is known, it will often be possible to read the value of the second variable with too much accuracy.

To prevent this from happening, we have in microdata.no chosen to *smooth* such plots with a smoothing technique. For this purpose, we have attempted to focus on a technique called *hexbin plot*. In a hexbin plot, the graphic area is divided into regular hexagons.

Example of hexbin plot made in microdata.no:

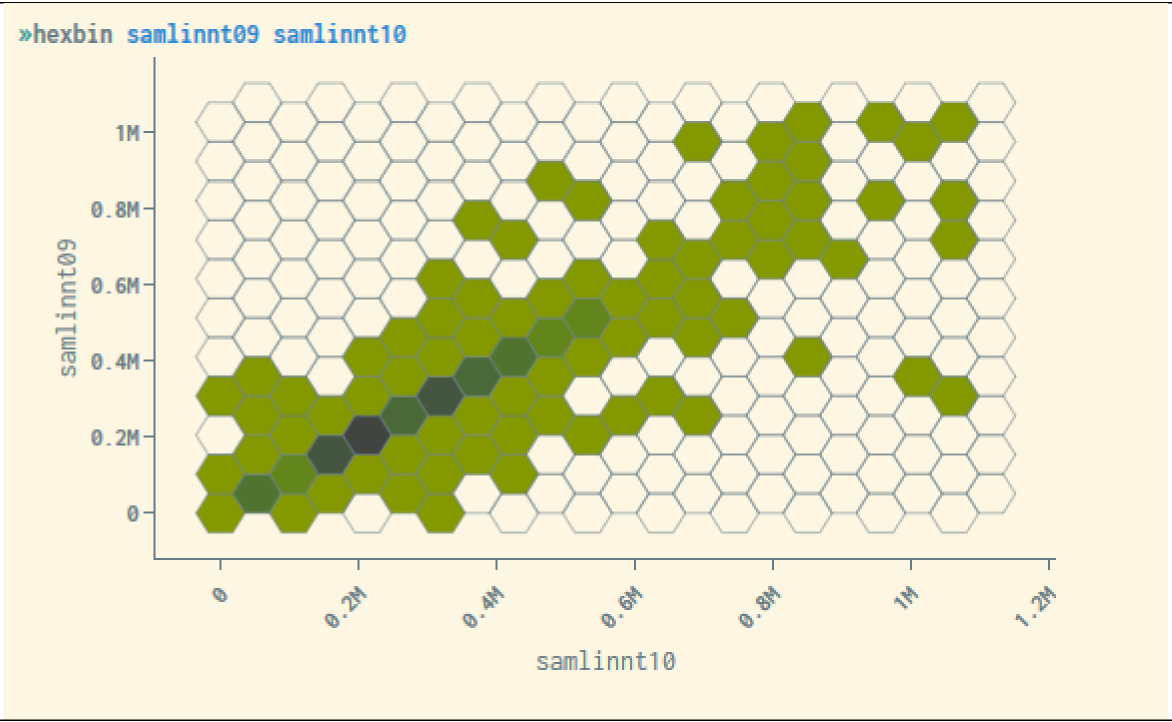

```
»hexbin samlinnt09 samlinnt10
```

Figure C1. Hexbin plot showing total income in 2009 vs. 2010 for a small population

In a hexbin plot, the graphical area is scaled based on the largest and smallest values that occur for the variables being plotted. The largest and smallest values are influenced by the winsorisation referred to in measure 2. The hexagons are given a color or hue indicating an interval for how many units there are in them, for example 30-59, 60-89, etc. The range of units/ persons each hue represent are equally long and are automatically adjusted according to the distribution in the data.

Hexbin plot is under trial. In the current version, all hexagons where the number of people is less than 20% of the most populated hexagon form are blanked. This criterion will be adjusted as soon as it is possible to give priority. Note that the winsorisation of the numerical variables that span the plot will affect the plot.