

User Guide for microdata.no

Content list

Content list	2
About the user interface	5
1.1 The command window	7
1.2 Variables and variable definitions	9
1.3 The Command field	15
1.4 Useful Commands	16
1.5 The script window	17
1.5.1 Create a script	17
1.5.2 Save command window work sessions as a script	17
1.5.3 Run a script	18
1.5.4 Run parts of a script	19
1.5.5 Script editor advantages	23
1.5.6 Organization of scripts	24
1.5.7 Troubleshooting using scripts	25
Creating and changing datasets	26
2.1 Connect to data bank	26
2.2 Creating a dataset	27
2.3 Retrieving variables into a dataset	27
2.3.1 Datasets containing cross-sectional data	28
2.3.2 Datasets containing event information	30
2.3.3 Cross-sectional vs. event-based datasets	31
2.4 Datasets containing regular time measurements (panel data)	32
2.5 How to navigate between datasets	33
2.6 Population filtering	34
2.7 Removing variables from datasets	35
2.8 How to aggregate and link datasets	35
2.9 Examples: Creating and revising a dataset	37
2.10 Examples: How to link data with unit levels other than individual level	38
Variable adaptations	45
3.1 Creating new variables and recoding: generate/replace	45
3.2 Variable recoding: recode	48
3.3 Use of functions	48
3.4 How to generate time-aggregated values - collapse	50

3.5 Renaming variables	52
3.6 Using labels	52
3.7 Changing value format from alphanumerical (text) into numerical	53
3.8 Example	54
Descriptive variable statistics	55
4.1 Tabulate - frequency tables	56
4.1.1 One-way frequency tables	57
4.1.2 Multi-dimensional frequency tables	58
4.1.3 Frequency tables using percentages	59
4.1.4 Frequency tables and category labels	60
4.1.5 Frequency tables and missing values	61
4.1.6 Frequency table filtering	62
4.1.7 Volume tables	64
4.2 Summarize and boxplot - metrical statistics	65
4.3 Piecharts	68
4.4 Histogram - graphical frequency presentation	69
4.5 Barcharts	73
4.6 Hexbin - anonymized scatterplot	74
4.7 Sankey - transition diagrams	76
4.8 Examples	79
4.8.1 Tabulate	79
4.8.2 Summarize and boxplot	80
4.8.3 Histogram and barchart	81
4.8.4 Piechart and hexbin-plot	82
4.8.5 Sankey-diagram	83
Advanced analysis	85
5.1 Correlate - correlation measures	85
5.2 Anova	86
5.3 Normal test	87
5.4 Regress - ordinary least squares estimation	88
5.4.1 Factor variables	90
5.4.2 Model diagnostics	93
5.4.3 Cluster and robust estimation	95
5.4.4 Prediction and residual values	96
5.5 IV-regression - linear regression analysis with instrument variables	97
5.5.1 Factor variables	98
5.5.2 Model diagnostics	98
5.5.3 Cluster and robust estimation	98

5.5.4 Prediction and residual values	98
5.6 Logit and probit - logistic regression analysis	99
5.6.1 Factor variables	101
5.6.2 Marginal effects	101
5.6.3 Cluster and robust estimation	102
5.6.4 Prediction and residual values	102
5.7 Mlogit - multinomial logistic regression analysis	104
5.7.1 Factor variables	105
5.7.2 Marginal effects	105
5.7.3 Cluster and robust estimation	105
5.7.4 Prediction and residual values	105
5.8 Regress-panel - panel data regression analysis	106
5.8.1 Factor variables	111
5.8.2 Model diagnostics	111
5.8.3 Cluster and robust estimation	112
5.8.4 Prediction and residual values	112
5.9 Example	113
Appendix A: Command overview	116
Appendix B: Function overview	119
Misc. mathematical functions	119
Row calculations (based on two or more variables)	122
String functions	124
Sysmiss	126
Density functions	126
Date functions	135
Appendix C: Confidentiality in microdata.no	138

1. About the user interface

Microdata.no is a web-based analysis system that uses a command language similar to Stata¹. It is recommended to use browsers such as Chrome and Firefox for the best user experience. Internet Explorer may cause errors such as the screen becoming blank and / or inability to log in, and is therefore not recommended.

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

<https://microdata.no/en/>

The screenshot shows the microdata.no website. The header is dark blue with the logo and navigation links. The main content area has four large buttons: Variables, Analysis, Help, and Admin. The Analysis button has a 'Log in' link and a 'How to gain access?' link. A video player is shown on the right, displaying a graph. Below the video, there is text about data protection and a link to read more.

microdata.no

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

¹ The commands are implemented using Python and Pandas, and the syntax is similar to Stata in order to make it user friendly.

What you see after login is a website consisting of an analysis area, i.e. the command window, cf. chapter 1.1. 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 how to work in the script window.

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 command window through the use of single commands:

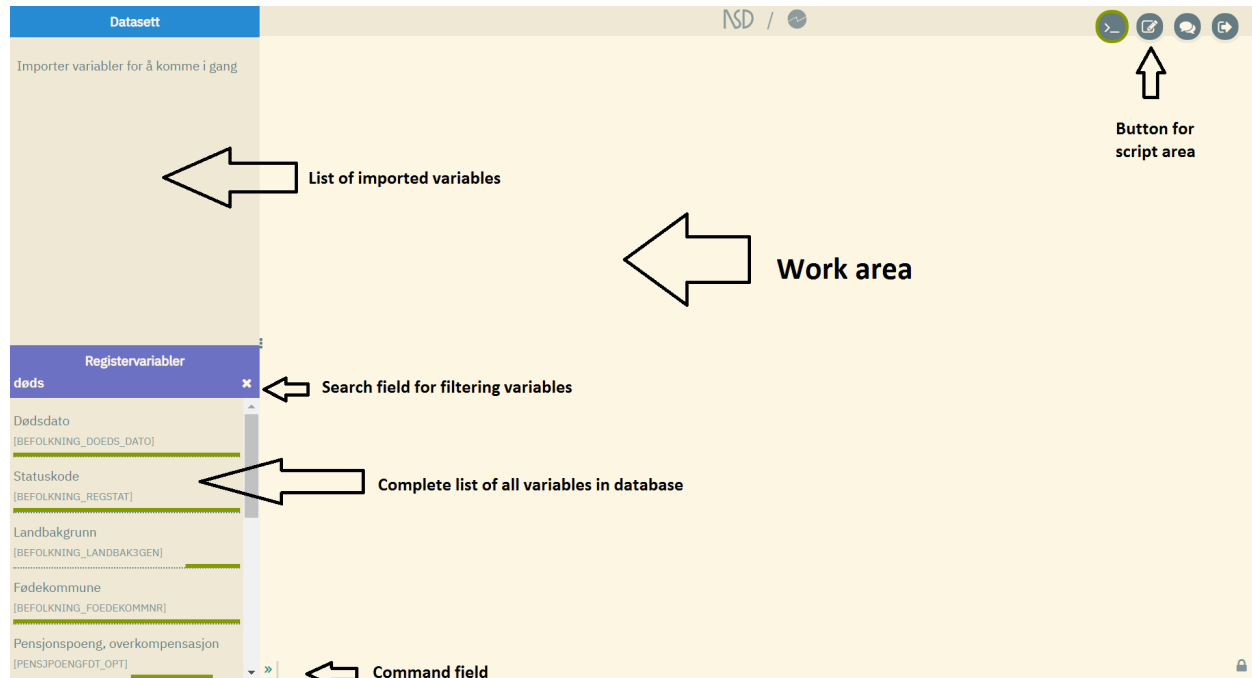
- a) One can construct command sequences to be run in one operation. The results of script executions are displayed continuously as commands are run and the execution is stopped if errors are detected (syntax or logical errors)
- b) Command sequences can be edited and rerun
- c) Much easier to identify 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 command window can easily be transferred into a script for further editing. This can be done in three 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"). Remember to enter a name for the script in the line above the script window. The script will then be saved with this name.
- Use the command `history` in the command window. This returns a chronological list of all commands run, which can be copied into the script window by clicking on the copy button that will appear when holding the mouse cursor over, and then using the <ctrl> + <c> keyboard combination. Remember to create a name for the script also in this case.
- In the command window, you can type the `save` command followed by an optional name that you use on the script. The name must have quotation marks around, for example `save 'Analysis of the unemployed'`. With this procedure, you do not need to name the script in the script window afterwards.

For more information on using scripts, see chapter 1.5.

1.1 The command window



The command window 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 by using the `require` command, ref. chapter 2.1)
- Overview of variables imported into own datasets (area at the top left)
- Command line (bottom of work area)

Custom datasets are built up by importing, processing and developing new variables based on variables from the data bank. Datasets are stored in the user's command window and do not disappear without the user deleting them. See Chapter 2 on how to create datasets and import variables.

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

NSD NORSK SENTER FOR FORSKNINGSDATA Statistisk sentralbyrå Statistics Norway 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

NSD NORSK SENTER FOR FORSKNINGSDATA Statistisk sentralbyrå Statistics Norway

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

microdata.no
NSD NORSK SENTER FOR FØRSKNINGSDATA
Statistisk sentralbyrå Statistics Norway
English

Databankoversikt / no.ssb.fdb

Søk i variabler

Data fra SSB no.ssb.fdb
Versjon 12 - Vis endringslogg
Registerdata som inngår i SSBs statistikkproduksjon

Variabler 323

Variabel	Beskrivelse	1967	2021	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
AFPP1992FDT_MOTTAK	Avtalefestet pensjon, privat sektor, mottaksperiode			Sosiale forhold og kriminalitet Trygd og stønad
AFPP2011FDT_GRAD	Avtalefestet pensjon, privat sektor, uttaksgrad			Sosiale forhold og kriminalitet Trygd og stønad
AFPP2011FDT_MOTTAK	Avtalefestet pensjon, privat sektor, mottaksperiode			Sosiale forhold og kriminalitet Trygd og stønad

The variable list shows an overview of all variables in the database (over 300 in Statistics Norway's database per 1/1 2022), and you can use search functionality to more easily find variables you are looking for.

By clicking on “Vis endringslogg” (“Show change log”) under the name of the database, you also get an overview of the different versions of the database, and what has been changed. By clicking on the version names, you will come to a new page that shows all the variables for this version.

By clicking on a variable in the variable list, you get definitions, code lists, change history and other key information related to the specific variable:

microdata.no
NSD NORSK SENTER FOR FORBARNINGSDATA
Statistisk sentralbyrå Statistics Norway
English

Databankoversikt / no.ssb.fdb / BEFOLKNING_KOMMNR_FAKTISK

Bostedskommune faktisk adresse

BEFOLKNING_KOMMNR_FAKTISK

Variabelen viser personens bostedskommune ifølge folkeregisteret (formell adresse). I folkeregisteret er bostedsadressen stedet der den enkelte person har sin overveiende døgnhvile. For ugifte studenter er det imidlertid frivillig å melde flytting fra hjemstedet selv om man reelt sett har et bosted på studiestedet. Ved bruk av andre datakilder finner man bostedsadresse på studiestedet.

Bostedskommune basert på faktisk adresse er relevant for husholdningsopplysninger. Før 2014 benyttes kun bostedskommune fra Folkeregisteret, KOMMNR_FORMELL. For bostedskommune basert på adresse i Folkeregisteret foreligger også en variabel BOSATTEFDT_BOSTED, denne angir bostedskommune som forløp med hendelsesdatoer.

Variabelen omfatter bosatte per 01.01.ÅÅÅÅ.

Befolkning
Husholdning

Databank	no.ssb.fdb
Enhetsstype	Person
Temporalitet	Tverrsnitt
Datatype	Alfanumerisk
Gyldighetsperiode	2014-01-01 - 2021-01-01
Format	N/A
Måleenhet	N/A

Tverrsnittsdatoer

2014-01-01
2015-01-01
2016-01-01
2017-01-01
2018-01-01
2019-01-01
2020-01-01
2021-01-01

359 kategorier

0301	Oslo
1101	Eigersund
1103	Stavanger
1106	Haugesund
1108	Fosnes

Vis 351 skjulte kategorier

3442	Unjarga - Ivesseby
5443	Båtsfjord
5444	Sør-Varanger
9999	Uoppgitt

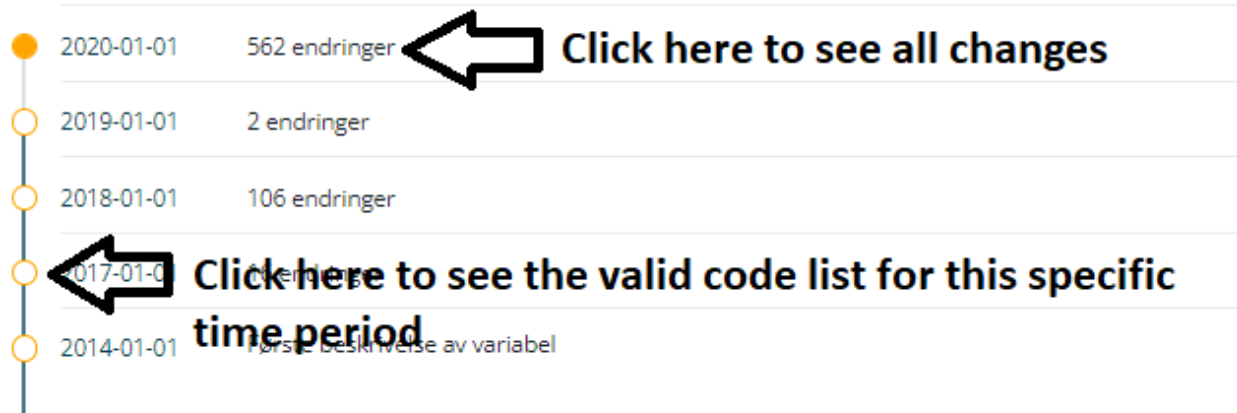
Historikk

2020-01-01	562 endringer
2019-01-01	2 endringer
2018-01-01	106 endringer
2017-01-01	16 endringer
2014-01-01	Første beskrivelse av variabel

Note that variable information is currently only available in Norwegian.

Be aware that the code list for a given variable changes over time. This must be taken into account if you work with longer time series that extend back in time. It is the current code list that applies for the specific time periods. Especially municipal codes, education codes and business codes have relatively frequent changes in the code lists. By clicking on the points in front of the times that indicate the start of a new code version, a current code list for the current time period is displayed. If you click on the indication of changes, you get an overview of which changes have been made for the same time period:

Historikk



As the illustration below shows, a complete variable overview is also found at the bottom left of the command window. Like the variable list found through the main login site, you may filter the variable list by entering parts of a variable name in the search field, and the filter works both against variable description and the name itself. In this way, it becomes easier to find the variable in question.

Dataseett

demografidata
10 variabler, 7 241 906 enheter

PERSONID_1
ABC Kjønn
123 faarmnd
ABC sivistand
123 formue
123 innt05
123 mann
123 gift
123 alder
123 formuehøy

Registervariabler

familie

Antall familier i husholdningen
[BEFOLKNING_ANTREGSTAT_FAMNR_1_HUSHN]

Familienummer
[BEFOLKNING_REGSTAT_FAMNR]

Familietype
[BEFOLKNING_REGSTAT_FAMTYP]

Pensjon til etterlatte familiepleiere, mottak
[ETLATFAM201FDT_MOTTAK]

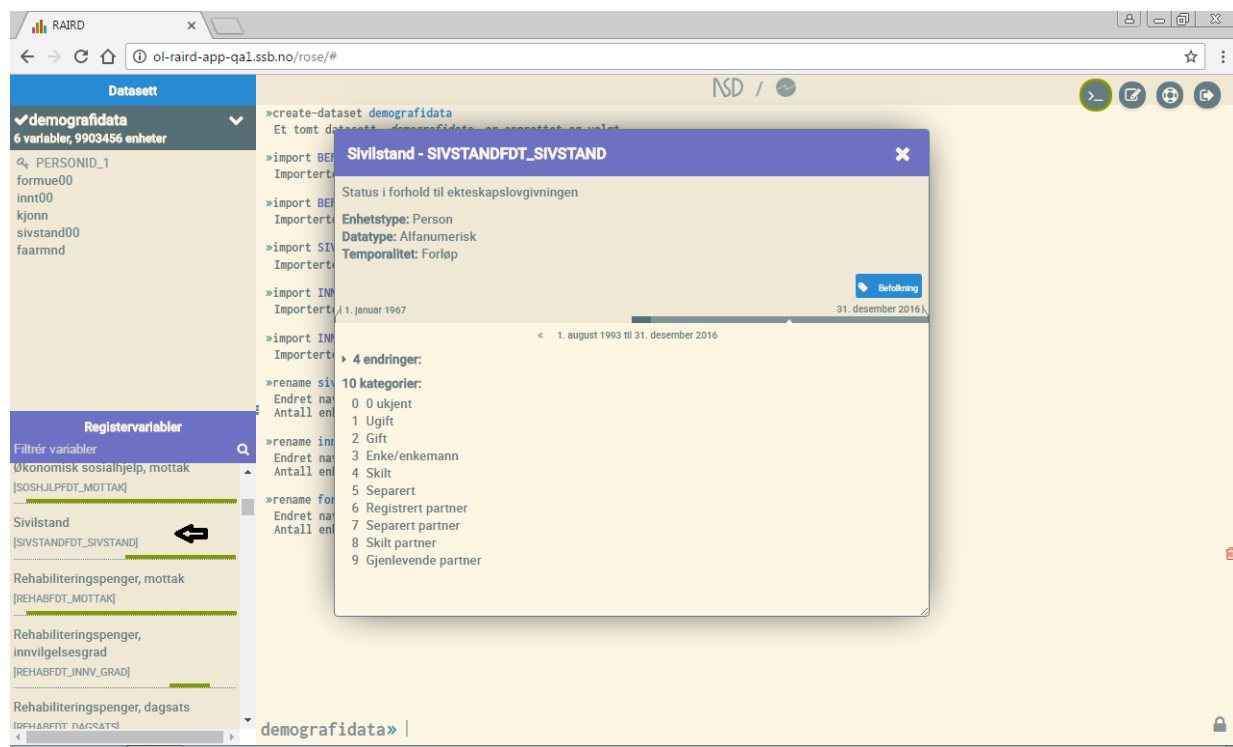
demografidata» history

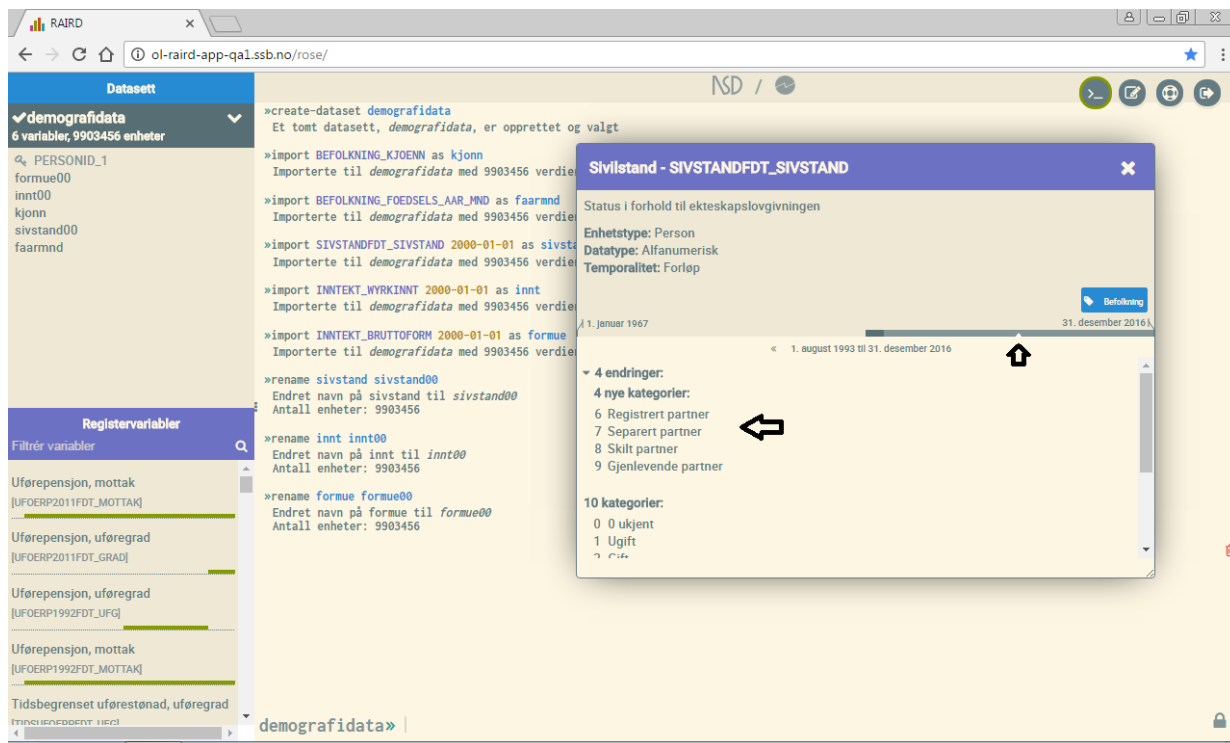
```
create-dataset demografidata
import BEFOLKNING_KJOENN as kjønn
import BEFOLKNING_FOEDELS_AAR_MND as faarmnd
import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivistand
import INNTEKT_BRUTTOFORM 2000-01-01 as formue
import INNTEKT_WYRKINNT 2005-01-01 as innt05
generate mann = 0
replace mann = 1 if kjønn == '1'
generate gift = 0
replace gift = 1 if sivistand == '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
```

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:

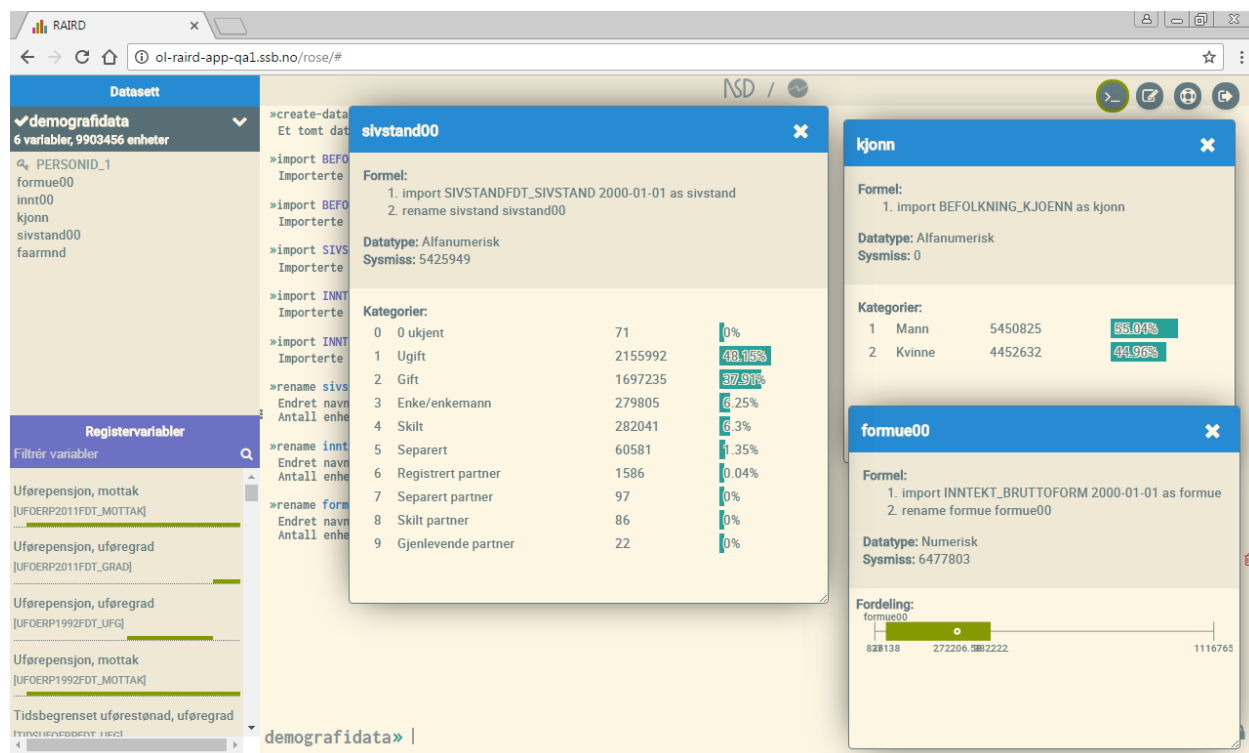
- Key information: Variable name, variable label, variable description, variable type
- Detailed interactive timeline that allows for studies of changes in coding over time: Changes in the coding are displayed through different colors illustrating the time periods to which they apply. Clicking on the 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





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.



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, 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 will list all commands that have been used in a session. This list can be copied into the system's script editor for an automatic execution of all the commands, cf. chapter 1.5, or into a private document.

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 window

The script editor is located to the left of the script window, allowing users to enter sequences of commands that can be run together as a script. The results of the run are shown in the right area of the script window when using the “Kjør” (“Run”) button.

The screenshot shows the microdata.no interface with the following components and annotations:

- Script editor (left):** Contains a menu for loading scripts, creating an empty script, or saving work sessions as scripts. It also shows a list of commands being executed, such as `import db[/SISTEMPROT_SIVSTAND 2018-01-01 as sivstand]`.
- Command window (right):** Displays the results of the script execution, including a table of results and a summary of the execution process.
- Annotations:**
 - Line for naming of script:** Points to the top of the script editor.
 - Return to command window:** Points to the top right of the command window.
 - Go to script window button:** Points to the top right of the script editor.
 - Result area:** Points to the right side of the command window.
 - Click here to show variable information like in command window:** Points to the bottom right of the command window.
 - Run script and show result to the right in the command script window:** Points to the bottom right of the command window.

1.5.1 Create a script

Scripts can be created in the following ways (can be combined):

- Enter manually, line by line
- Copy commands from the command window (use the `history` command and copy the commands you want by selecting them and using `ctrl + c`)
- Paste a finished script from different sources (remember to convert to plain text format first - do not paste scripts that come from formatted formats such as Word)
- Import / retrieve your entire work session from the command window (see chapter 1.5.2)

1.5.2 Save command window work sessions as a script

Work already performed in the command window can easily be transferred into a script for further editing. This can be done in three 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”).

Remember to enter a name for the script in the line above the script window. The script will then be saved with this name.

- Use the command `history` in the command window. This returns a chronological list of all commands run, which can be copied into the script window by clicking on the copy button that will appear when holding the mouse cursor over, and then using the `<ctrl> + <c>` keyboard combination. Remember to create a name for the script also in this case.
- In the command window, you can type the `save` command followed by an optional name that you use on the script. The name must have quotation marks around, for example `save 'Analysis of the unemployed'`. With this procedure, you do not need to name the script in the script window afterwards.

1.5.3 Run a script

You have two options when you are ready to run the script:

- At the bottom right there are two buttons, where one, "Kjør" ("Run"), runs through the script on the right side of the script area (result area)
- The second button "Send til kommandolinjen" ("Send to command line") sends all the commands to the command window and executes them there. When the run is complete, you are automatically sent to this window. Note that whatever content you have in the command window will be replaced by the result of the new script run, so remember to save your work in the command window in the form of a script before running the new one.

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.

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.

Load script (←) **Name of script (customizable)** **Return to analysis area** (→)

Eksempel: Lineære regresjonsanalyser #8

Script editor (←)

```

1 textblock
2 Lineær regresjonsanalyse
3 -----
4
5 Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
6 koeffisientverdier for et sett med forklaringsvariabler, der
7 utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
8 (ikke vise fastleddet, endre på signifikansnivået m.m.).
9 endblock
10
11 // Starter med å hente variablene en trenger
12 create-dataset demografidata
13 import BEFOLKNING_KJOENN as kjonn
14 import BEFOLKNING_FOEDELS_AAR_MND as faarmnd
15 import SIVSTANDFOT_SIVSTAND 2000-01-01 as sivstand
16 import INNTEKT_BRUTTOFORM 2000-01-01 as formue
17 import INNTEKT_WYRKINNT 2005-01-01 as innt05
18
19 // Tilrettelegger de uavhengige variablene slik at de passer med den statistiske
20 // modellen (inneberer at de fleste variabler gjøres som til dummy'er)
21 generate mann = 0
22 replace mann = 1 if kjonn == '1'
23
24 generate gift = 0
25 replace gift = 1 if sivstand == '2'
26
27 generate alder = 2000 - int(faarmnd / 100)
28 drop if alder < 16
29
30 generate formuehay = 0
31 replace formuehay = 1 if formue > 600000

```

Validation area (←)

Antall enheter: 9 903 456

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

demografidata> generate formuehay = 0
Genererte formuehay
Antall enheter: 7 241 906

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

demografidata> correlate alder formuehay

	formuehay
alder	0.0229

demografidata> regress innt05 mann gift alder formuehay

Kilde	SS	df	MS	Antall Obs:
Modell	1.0747154e+16	4	2.6867885e+15	2489943
Residual	8.6956061e+16	2.489938e+6	3.4922982e+10	F(4, 2489938): 76934.679305
Total	9.7703215e+16	2.489942e+6	3.9239153e+10	R ² : 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	99052.000 99052.000
gift	55131.8	276.388	199.472	0	55131.800 55131.800
alder	-1157.79	10.7419	-107.782	0	-1157.790 -1157.790
formuehay	88753.7	387.744	228.897	0	88753.700 88753.700
Konst	2.4550213e+5	393.996	623.107	0	2.4550213e+5 2.4550213e+5

Execute script in work area (↓) **Validate script** (↓)

Send til kommandolinjen Kjør

1.5.4 Run parts of a script

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



```

1 textblock
2 Lineær regresjonsanalyse
3 -----
4
5 Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
6 koeffisientverdier for et sett med forklaringsvariabler, der
7 utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
8 (ikke vise fastleddet, endre på signifikansnivået m.m.).
9 endblock
10
11 // Starter med å hente variablene en trenger
12 create-dataset demografidata
13 import BEFOLKNING_KJOENN as kjonn
14 import BEFOLKNING_FOEDELS_AAR_MND as faarmnd
15 import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
16 import INNTEKT_BRUTTOFORM 2000-01-01 as formue
17 import INNTEKT_WYRKINNT 2005-01-01 as innt05
18

```

- 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
- You may also automatically add "/" in front of several lines by marking them and using the hotkey combination alt + c. By repeating the procedure for the exact same lines, all "/" signs will be removed.

```

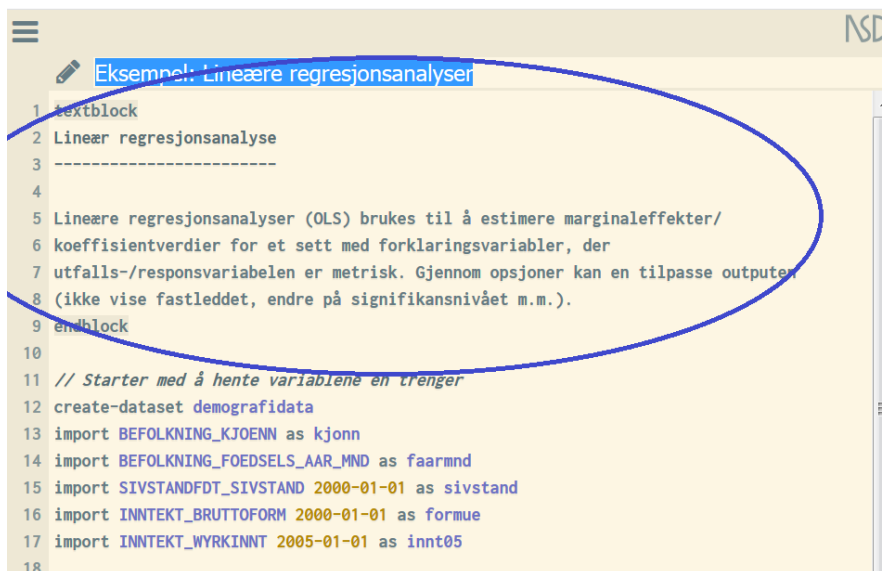
1 //Kobler til databank
2 require no.ssb.fdb:12 as db
3
4 //Starter med å hente variablene en trenger
5 create-dataset demografidata
6 import db/BEFOLKNING_KJOENN as kjonn
7 import db/BEFOLKNING_FOEDELS_AAR_MND as faarmnd
8 import db/BEFOLKNING_STATUSCODE 2018-01-01 as regstat
9 // import db/SIVSTANDFDT_SIVSTAND 2018-01-01 as sivstand
10 // import db/INNTEKT_BRUTTOFORM 2018-01-01 as formue
11 // import db/INNTEKT_WYRKINNT 2019-01-01 as innt19

```

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 command window. The command window 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 command window 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



```

1 textblock
2 Lineær regresjonsanalyse
3 -----
4
5 Lineære regresjonsanalyser (OLS) brukes til å estimere marginaleffekter/
6 koeffisientverdier for et sett med forklaringsvariabler, der
7 utfalls-/responsvariabelen er metrisk. Gjennom opsjoner kan en tilpasse outputen
8 (ikke vise fastleddet, endre på signifikansnivået m.m.).
9 endblock
10
11 // Starter med å hente variablene en trenger
12 create-dataset demografidata
13 import BEFOLKNING_KJOENN as kjonn
14 import BEFOLKNING_FOEDELS_AAR_MND as faarmnd
15 import SIVSTANDFDT_SIVSTAND 2000-01-01 as sivstand
16 import INNTEKT_BRUTTOFORM 2000-01-01 as formue
17 import INNTEKT_WYRKINNT 2005-01-01 as innt05
18

```

Datasett

demografidata
10 variabler, 7 241 906 enheter

PERSONID_1

ABC kjonn
123 faarmnd
ABC sivistand
123 formue
123 innt05
123 mann
123 gift
123 alder
123 formuehøy

Registreringsvariabler

famtyp

Familietype
(BEFOLKNING_REGSTAT_FAMTYP)

Lineær regresjonsanalyse

Lineære regresjonsanalyser (OLS) brukes til å estimere marginaeffekter/ 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_FOESELS_AAR_MND as **faarmnd**

Importerte **faarmnd** til **demografidata** med 9 903 456 verdier og 1 missingverdier

demografidata» import SIVSTANDFOT_SIVSTAND 2000-01-01 as **sivistand**

Importerte **sivistand** 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_MYRKINNT 2005-01-01 as **innt05**

Importerte **innt05** til **demografidata** med 9 903 456 verdier og 7 171 799 missingverdier

demografidata» generate **mann** = 0

C) Click on a line number in your script and then press “Kjør” or “Send til kommandolinjen”.

This will run through all the lines in your script until the line that was marked, and stop the execution there. You may use this procedure to run through gradually more and more of your script if you are not ready to run through it completely.

Example: Click on line number to run through all lines until this point (and not the complete script)

Kommandolinje skrevet 6. januar 2022 20:57:39

```
1 //Hobler til databank
2 require no.ssb.fdb:12 as db
3
4 //Starter med å hente variablene en trenger
5 create-dataset demografidata
6 import db/BEFOLKNING_KJOENN as kjonn
7 import db/BEFOLKNING_FOESELS_AAR_MND as faarmnd
8 import db/BEFOLKNING_STATUSKODE 2018-01-01 as regstat
9 import db/SIVSTANDFOT_SIVSTAND 2018-01-01 as sivistand
10 import db/INNTEKT_BRUTTOFORM 2018-01-01 as formue
11 import db/INNTEKT_MYRKINNT 2019-01-01 as innt19
12
13 //Begrenser populasjonen
14 generate alder = 2018 - int(faarmnd / 100)
15 replace regstat = '1' if kjonn == '1' & alder > 15 & alder < 67
16
17 //Tilrettelegger de uavhengige variablene slik at de passer med den statistiske modellen (inneholder at de fleste variabler
   gjøres som til dummy'er)
18 generate mann = 0
19 replace mann = 1 if kjonn == '1'
20
21 generate gift = 0
22 replace gift = 1 if sivistand == '2'
23
24 generate formuehøy = 0
25 replace formuehøy = 1 if formue > 1500000
26
27 //Tester for korrelasjon mellom to av de uavhengige variablene
28 correlate alder formuehøy
29
30 //Kjører regresjonsanalysen der den avhengige variabelen alltid listes først
31 regress innt19 mann gift alder formuehøy
```

» require no.ssb.fdb:12 as db

Opprettet en kobling fra no.ssb.fdb:12 til db

» create-dataset **demografidata**

Et tomt dataset, **demografidata** ble opprettet og valgt

demografidata» import db/BEFOLKNING_KJOENN as **kjonn**

Importerte **kjonn** til **demografidata** med 10 626 204 verdier

demografidata» import db/BEFOLKNING_FOESELS_AAR_MND as **faarmnd**

Importerte **faarmnd** til **demografidata** med 10 626 204 verdier

demografidata» import db/BEFOLKNING_STATUSKODE 2018-01-01 as **regstat**

Importerte **regstat** til **demografidata** med 10 626 204 verdier, hvorav 2 301 607 missingverdier

demografidata» import db/SIVSTANDFOT_SIVSTAND 2018-01-01 as **sivistand**

Importerte **sivistand** til **demografidata** med 10 626 204 verdier, hvorav 5 330 322 missingverdier

demografidata» import db/INNTEKT_BRUTTOFORM 2018-01-01 as **formue**

Importerte **formue** til **demografidata** med 10 626 204 verdier, hvorav 6 399 937 missingverdier

demografidata» import db/INNTEKT_MYRKINNT 2019-01-01 as **innt19**

Importerte **innt19** til **demografidata** med 10 626 204 verdier, hvorav 7 403 391 missingverdier

demografidata» generate **alder** = 2018 - int(faarmnd / 100)

Genererte **alder**

Antall enheter: 10 626 204

demografidata» keep if regstat == '1' & alder > 15 & alder < 67

7118667 enheter ble fjernet fra datasettet.

● Skriptkjøring stoppet ved stoppunkt

Send til kommandolinje Kjør

1.5.5 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 1.5.4) 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.

1.5.6 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).

The screenshot shows the RAIRD application interface. On the left, a menu titled "Kommandolinje" lists several programs and examples. The "Eksempler" (Examples) section includes tasks like "Varighet i en tilstand", "Bli kjent med variabler gjennom beskrivende statistikk", and "Lineær regresjonsanalyse". The main area displays a command script for creating a dataset and importing data. On the right, a table shows the results of a tabulate command, with columns for "Mann", "Kvinne", and "Sum".

	Mann	Kvinne	Sum
0 ukjent	6	--	6
Ugift	58531	52493	111036
Gift	35959	35257	71220
Enke/enkemann	2162	8785	10945
Skilt	6732	9499	16228
Separert	1214	1232	2447
Registrert partner	58	52	107
Sum	6	7	13

1.5.7 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 1.5.4 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.

2.1 Connect to data bank

When you log in to microdata.no for the first time, or create a completely new work session, the contents of the command window will be empty. 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 <atabank:version> as <alias>
```

Example:

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

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) Longitudinal variables with sequences of events (each observation represents a state change, i.e. the variable changes value, with varying start and stop dates)
- 2) Fixed variables with only one observation per unit (fixed information such as gender, date of birth, country of birth)
- 3) Cross-sectional variables measured at fixed times (this is mainly variables used for statistical purposes, where start date = stop date and only the value at this particular time in question is known)
- 4) Accumulated variables - mainly economical data on yearly income, wealth 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 (four temporality types):

- Fixed information (e.g. gender, birthdate, birth country)
- Longitudinal information (custom extractions from event based variables)
- Cross-sectional information on regularly predetermined measurement dates
- Accumulated information (mainly yearly economical measures like income, wealth etc)

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 cross-sectional variables, the regularly predetermined measurement dates must be used since the values in 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 longitudinal data on custom dates, the last used date is proposed. For variables with accumulated 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`, cf. 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 specific 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_STATUSKODE` measured at the corresponding measurement date, and then keeping all individuals who take the value '1' (= resident in Norway). Example:

```
require no.ssb.fdb:12 as db
create-dataset demographics
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_STATUSKODE 2015-01-01 as regstat15
keep if regstat15 == '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 occurred.

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 5.8.

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:12 as db

create-dataset population
import db/BOSATTEFDT_BOSTED 2010-01-01 as residence
import db/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 db/INNTEKT_WLONN db/SIVSTANDEFDT_SIVSTAND
db/BOSATTEFDT_BOSTED 2011-12-31 2012-12-31 2013-12-31 2014-12-31
```

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 aggregate and link datasets

Datasets are usually built up through the command `import`, which adds one and one variable measured at specified custom dates. Such variables must have the same unit type, usually persons represented by the unique keyidentifier `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, job level etc. Data with unit types other than persons cannot be imported directly into an individual level dataset (datasets with persons as unit type, given by the key identifier variable `PERSONID_1`). 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 level³, must be aggregated to person level using the command `collapse()` before merging the data into an individual level dataset using `merge`. The `collapse()`-command does two operations:

³ Course data is slightly different from other personal data as these data even after extraction at a given time contain personal data with several observations per individual. This reflects the fact that it is possible to participate in several different courses / studies at the same time. The same principle applies to job data where it is possible to have several jobs at the same time.

- Aggregates data into a higher unit level given by an identifier variable. In principle, all categorical variables may be used as identifier, e.g. by using municipality as identifier, you can aggregate data into municipality level data
- Calculates an aggregate value across all units, for each of the new aggregated units. Measurement type is specified in the parenthesis that follows the command, such as sum, mean, maximum, number of values etc.

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

Example on aggregating course data⁴ (data on ongoing studies) from course level to person level for merging into a person level dataset:

```
collapse(max) edulevel, by(personid)
rename edulevel highest_edulevel
merge highest_edulevel into persondataset
```

Example on aggregating from person level to family level (sums income across all family members within each family and calculate family income), and then merging family income into a person level dataset:

```
collapse(sum) income, by(familynumber)
rename income familyincome
merge familyincome into persondataset on familynumber
```

Note that in the example above, the link variable is specified through the expression `on familynumber`. This must always be done if you use other link variables than the `PERSONID_1` key identifier.

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.

⁴ Course data together with job data are a bit special in relation to other personal data. See footnote 3 on the previous page.

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:12 as db
```

```
create-dataset demographics
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
```

```
import db/SIVSTANFDT_SIVSTAND 2015-01-01 as maritalstate
```

```
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
```

```
// Rename variables by adding postfix referring to year
```

```
rename maritalstate maritalstate15
```

```
rename wealth wealth15
```

```
// Drop variable gender from dataset
```

```
drop gender
```

```
// Keep only married persons
```

```
keep if maritalstate15 == '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 key unit identification variable of the target dataset is used by default, unless customized through an "on"-option".

In this example, a separate dataset is 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 datastore
```

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

```
//Create a main dataset with links to fathers and mothers
```

```
create-dataset persondata
```

```
import db/INNTEKT_WYRKINNT 2019-01-01 as workincome
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/NUDB_BU 2019-01-01 as edu
```

```
import db/BEFOLKNING_FAR_FNR as idnr_father
```

```
import db/BEFOLKNING_MOR_FNR as idnr_mother
```

```
//Import data on parents and merge into main dataset
```

```
create-dataset parents
```

```
import db/INNTEKT_WYRKINNT 2019-01-01 as workincome_father
```

```
import db/NUDB_BU 2019-01-01 as edu_father
```

```
clone-variables workincome_father -> workincome_mother
```

```
clone-variables edu_father -> edu_mother
```

```
merge workincome_father edu_father into persondata on idnr_father
```

```
merge workincome_mother edu_mother into persondata on idnr_mother
```

```
//Perform basic linear regression analysis to test for covariation with parental income
```

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

destring edu
generate high_edu = 0
replace high_edu = 1 if edu >= 700000 & edu < 900000
replace high_edu = edu if sysmiss(edu)

destring edu_father
generate high_edu_father = 0
replace high_edu_father = 1 if edu_father >= 700000 & edu_father < 900000
replace high_edu_father = edu_father if sysmiss(edu_father)

destring edu_mother
generate high_edu_mother = 0
replace high_edu_mother = 1 if edu_mother >= 700000 & edu_mother < 900000
replace high_edu_mother = edu_mother if sysmiss(edu_mother)

summarize workincome workincome_father workincome_mother
histogram workincome_father, percent
histogram workincome_mother, percent
correlate workincome_father workincome_mother
tabulate high_edu_father high_edu_mother

regress workincome male workincome_father workincome_mother high_edu high_edu_father
high_edu_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 datastore
```

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

```
//Create an individual level dataset consisting of persons in families defined by married couples with small children
```

```
create-dataset persondata
```

```
import db/BEFOLKNING_REGSTAT_FAMTYP 2017-01-01 as famtype
```

```
tabulate famtype
```

```
keep if famtype == '2.1.1'
```

```
//Add demographical information
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/BEFOLKNING_FOEDSELS_AAR_MND as birthdate
```

```
generate age = 2017 - int(birthdate/100)
```

```
import db/BEFOLKNING_KOMMNR_FAKTISK 2017-01-01 as municipality
```

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

```
import db/BEFOLKNING_BARN_I_HUSH 2017-01-01 as children
```



```

//Create dataset for generating family level income (unit type = family)
create-dataset familydata
import db/BEFOLKNING_REGSTAT_FAMNR 2017-01-01 as famnr
import db/INNTEKT_WYRKINNT 2017-01-01 as workincome
collapse (sum) workincome, by(famnr)
rename workincome familyincome

//Merge family income into individual level dataset (unit type = individuals)
merge familyincome into persondata

//Generate family level statistics. The family number consists of the personal id of the eldest person in
the family, so by removing individuals with missing family level income, the dataset now has unit type
= family. All individual information will be associated with the eldest person in the family
use persondata
drop if sysmiss(familyincome)

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 og
Fjordane '15' Møre og Romsdal '16' Sør-Trøndelag '17' Nord-Trøndelag '18' Nordland '19' Troms
'20' Finnmark '21' Spitsbergen '25' Studying abroad '99' Unknown
assign-labels county countytxt

tabulate county

histogram age_oldest, discrete
histogram children, discrete percent

tabulate children
tabulate children, cellpct
tabulate children gender_oldest

summarize familyincome
barchart (mean) familyincome, by(county)
barchart (mean) familyincome, by(children)
histogram familyincome, freq
histogram familyincome, 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 2019-01-01. Thereafter, history of ongoing education is retrieved for the whole year of 2019, 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 year 2019, 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 datastore
```

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

```
//Create individual level dataset containing residents in Norway per 2019-01-01
```

```
create-dataset persondata
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/BEFOLKNING_STATUSKODE 2019-01-01 as regstatus
```

```
keep if regstatus == '1'
```

```
//Find individuals studying on higher education level during 2019
```

```
create-dataset coursedata
```

```
import-event db/NUDB_KURS_NUS 2019-01-01 to 2019-12-31 as coursetype
```

```
destring coursetype
```

```
keep if coursetype >= 700000 & coursetype < 900000
```

```
//Merge link between course-id and personal-id into course data
create-dataset link_course_person
import db/NUDB_KURS_FNR as idnr
merge idnr into coursedata

//Measure number of observations/records concerning higher educational studies per individual, and
merge into main individual level dataset
use coursedata
collapse (count) coursetype, by(idnr)
rename coursetype courses
merge courses into persondata

//Produce tabulation for higher level education studies (individual frequencies for 2019)
use persondata
generate edu_high = 0
replace edu_high = 1 if courses >= 1
tabulate edu_high gender
```

Example: Linking/extraction of course level data for a specific date

textblock
Retrieve information about ongoing education (courses) measured at a specific date

Data on ongoing education (studies) exist with course as unit level (through unique course-identifier numbers). Courses are defined by the combination person x course type, and each individual can be represented by more than one course types simultaneously.

As data on ongoing education do not have person as unit level, such data can not be imported directly into an individual level dataset, but instead need to be merged through the command merge.

First, one must add/import a variable containing a link between course-ids and corresponding person-ids onto the course data (ongoing education). Next, one must aggregate the data to individual level through the command collapse. Finally, the data need to be merged into the main individual level dataset.

In the example below, an individual level dataset containing persons resident in Norway (regstatus == '1') per 2019-01-01 is used as main dataset. Then ongoing education study events per 2019-11-01 are collected into a separate dataset. The command collapse (count) is used to count the number of

observations/events for ongoing education per individual on the specific date, and the result is finally merged into the main individual level dataset for further analysis.

Note: The values of the variable coursetype will after the collapse-transformation be replaced by numerical values referring to the statistical measure being used, in this case count (number of observations/events).

endblock

```
//Connect to datastore
```

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

```
//Create individual level dataset containing residents in Norway per 2019-01-01
```

```
create-dataset persondata
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/BEFOLKNING_STATUSKODE 2019-01-01 as regstatus
```

```
keep if regstatus == '1'
```

```
//Retrieve people who are studying as of 1st November 2019, and connect this onto the personal data set. Since course data can have several observations per individual, the collapse command must be used to aggregate up to person level. We use count as aggregation value (number of records)
```

```
create-dataset coursedata
```

```
import db/NUDB_KURS_NUS 2019-11-01 as coursetype
```

```
import db/NUDB_KURS_FNR as idnr
```

```
collapse (count) coursetype, by(idnr)
```

```
rename coursetype courses
```

```
merge courses into persondata
```

```
//Produce tabulation for individuals who are studying as of 1st November 2019
```

```
use persondata
```

```
generate student = 0
```

```
replace student = 1 if courses >= 1
```

```
tabulate student 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 needs 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
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)*

For more information about `recode`, use the `help recode` command. This also shows an overview of associated options.

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 alphanumeric values only.

3.4 How to generate time-aggregated values - collapse

In addition to aggregating data into a higher unit level⁵, e.g. from person level into family level (or municipality level), the command `collapse` may also be used as a tool for statistical measurements aggregated over a specified time span. In practice, this is the same as aggregating data from event/longitudinal level into person level. 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 | maximum value |
| - min | minimum value |
| - mean | mean value |
| - median | median value |
| - count | number of values |
| - sum | sum of values |
| - semean | standard error of mean value |
| - sebinomial | binomial standard error of mean value |
| - sd | standard deviation |
| - percent | percentage valid values |
| - iqr | interquartile range (range between 75th and 25th percentiles) |

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 variable `PERSONID_1`.

⁵ See chapter 2.8

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

```
require no.ssb.fdb:12 as db
create-dataset maritalevent
import-event db/SIVSTANDEFDT_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:12 as db
create-dataset maritalevent
import-event db/SIVSTANDEFDT_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 (alphanumeric) 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 quotation marks. The labels that are linked to the values do not need to have quotation marks around if they only contain letters. But labels that contain special characters such as spaces, hyphens, slashes, dots, etc., must have quotation marks around. If you are unsure, you can always use quotation marks around labels no matter what.

It is also allowed to use double quotation marks if preferred.

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.

For more information on the use of `destring`, e.g. overview of all available options, use the `help destring` command.

3.8 Example

```
// Connect to data bank
require no.ssb.fdb:12 as db

// Retrieve required variables
create-dataset demographics
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
import db/BOSATTEFDT_BOSTED 2015-01-01 as residence

// Generate age in 2015 from birthyear
generate age = 2015 - 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 countyststring
```

```
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>]
```

For more information about this command, use the `help tabulate` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

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 kjønn

kjønn	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				Sum
		bosatt	utvandret	død	uregistrert person	
kjonn	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

		regstat				Sum
		bosatt	utvandret	uregistrert person	død	
kjonn	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
	Kvinne	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													
							sivstand05						
		Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	Sum	
sivstand00	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
```

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

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

		sivstand05										
		Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	Sum
sivstand00	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
```

		Ugift	Gift	Skilt	Separert	0 ukjent	Enke/enkemann	Registrert partner	Separert partner	Skilt partner	Gjenlevende partner	Sum
sivstand00	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
```

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

	bosatt	utvandret	død	SYSMISS	ureregistrert person	Sum
fylke00	Østfold	144411	34	5	162	144597
	Akershus	291708	90	--	294	292087
	Oslo	309693	347	--	601	310657
	Hedmark	108642	17	--	78	108750
	Oppland	109810	5	--	71	109883
	Buskerud	143772	35	--	132	143931
	Vestfold	124417	33	5	125	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	66234
	Møre og Romsdal	146447	26	--	89	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
	Uppgitt	--	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				Sum
		bosatt	utvandret	død	uregistrert person	
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) |
| - sum | Sum |
| - std | Standard deviation |
| - p25 | 25%-quartile |
| - p50 | 50%-quartile (= median value) |
| - p75 | 75%-quartile |
| - gini | Gini coefficient value |
| - iqr | Interquartile value (range between 75th and 25th percentile) |

Example of table showing standard income value (mean) divided by gender:

```
tabulate gender, summarize(income)
```

Example of table that shows median value instead of mean value:

```
tabulate gender, summarize(income) p50
```

Example of table showing mean income divided by gender and marital status:

```
tabulate marital_status gender, summarize(income)
```


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 2019 and 2018 respectively, where the population is all residents between the ages 16-66.

The `summarize` command displays key statistics for the specified numeric variables:

- Average
- Standard deviation
- Number of units with valid value
- First percentage value (upper limit value)
- Internal quartile values (50% = median value)
- Last percentage value (lower limit value)

It is also possible to display gini coefficient values as well as interquartile values (range between 75th and 25th percentiles) by using resp. the options `gini` and `iqr`.

The command `boxplot` shows a graphical presentation using a standard boxplot (a box representing the two middle quartiles, plus mean, minimum, and maximum values).

```
demografidata» summarize innt formue
```

Variabel	Gj.snitt	Std.avvik	Antall	1%	25%	50%	75%	99%
innt	485031.3743	333780.7368	2958923	1253	241256	473966	644347.75	1729989
formue	935371.5972	1434013.7402	3389824	64	81067	528833	1094222	9289672

```
demografidata» summarize formue if alder > 50
```

Variabel	Gj.snitt	Std.avvik	Antall	1%	25%	50%	75%	99%
formue	1533890.3026	2227840.8478	1023086	264	401163.5	878830	1731050.5	15032181

```
demografidata» boxplot innt formue
```

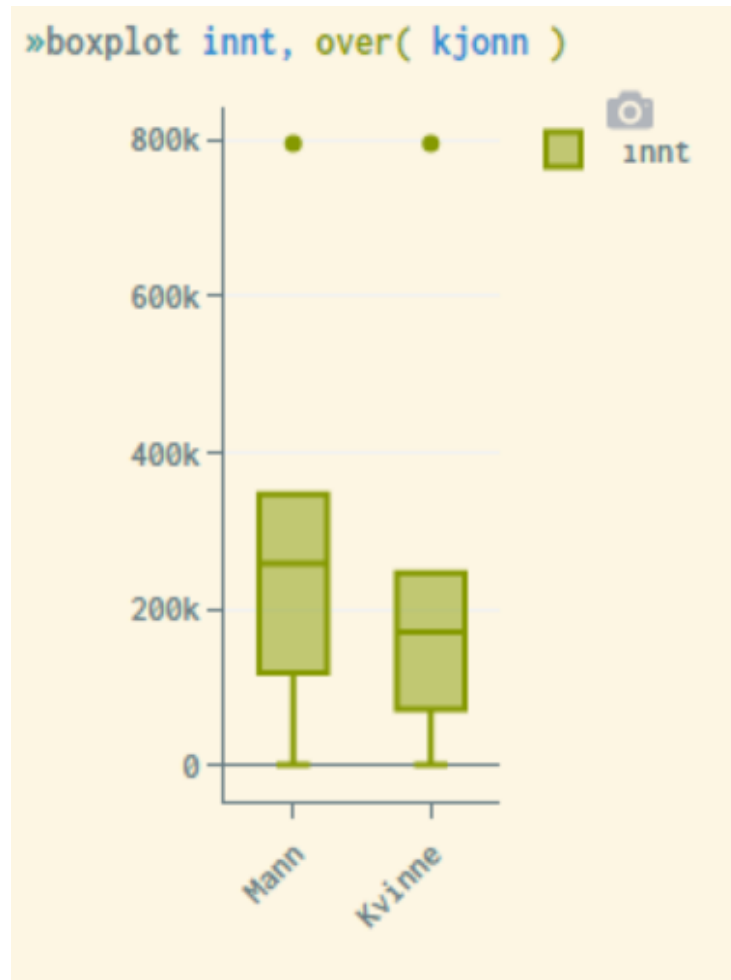


By holding the mouse cursor over the various boxplot areas, the corresponding values will be shown.

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:



For more information about these commands, use the `help summarize` or `help boxplot` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated. For example, the `gini` option can be used to display gini coefficient values in addition to the standard summarize result.

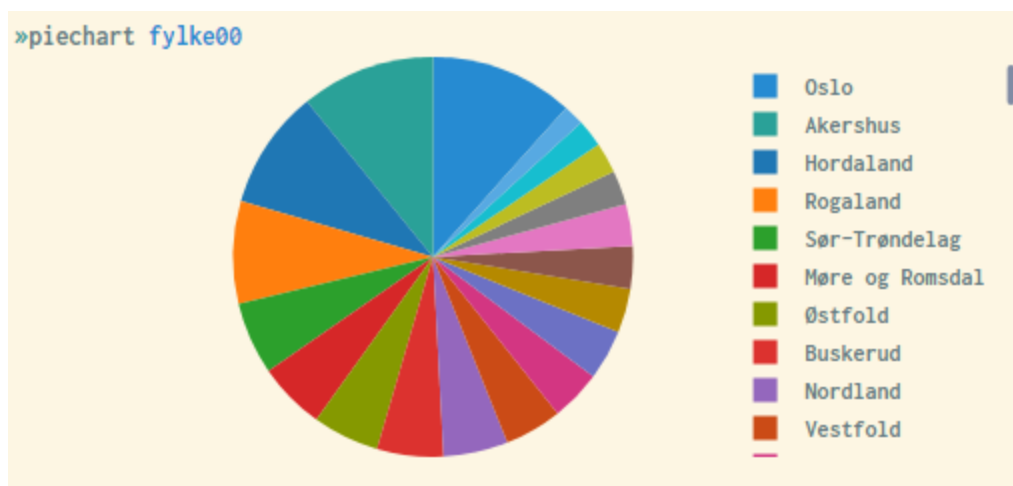
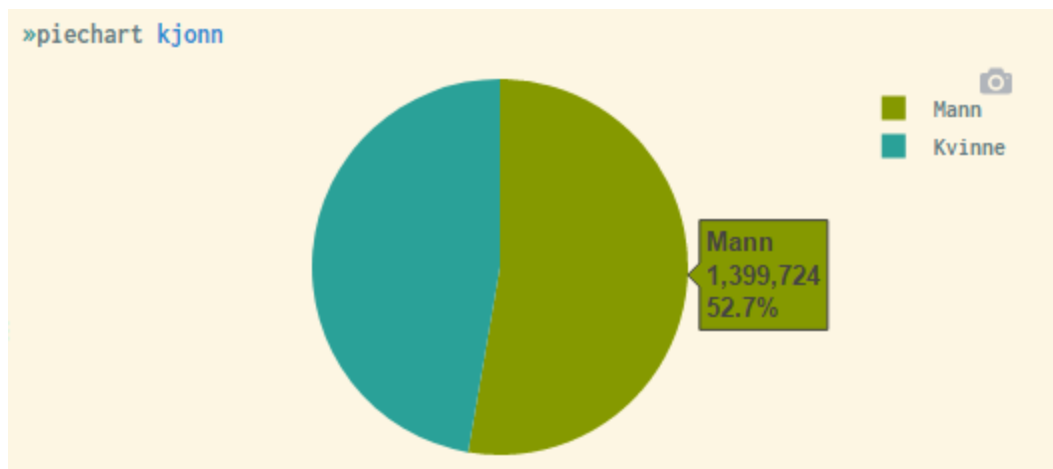
4.3 Piecharts

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

```
piechart <variable>
```

For more information about this command, use the `help piechart` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

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



4.4 Histogram - graphical frequency presentation

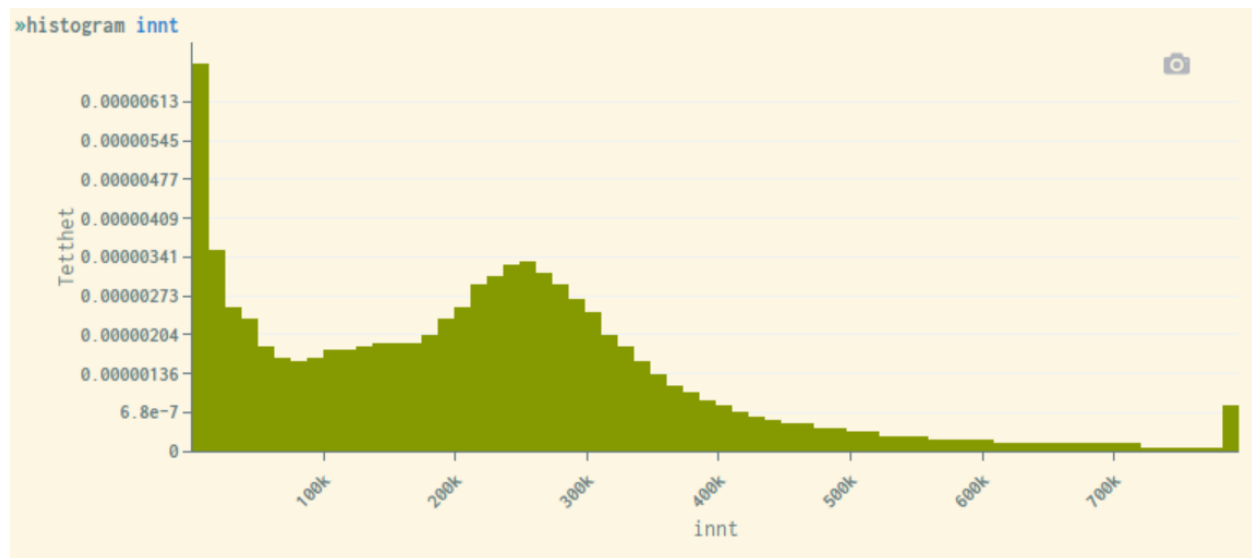
Histograms are graphical representations of univariate distributions for continuous variables (e.g., income). Each bar represents the frequency value for the corresponding predetermined variable interval. Through the options `bin()` and `width()`, it is possible to adjust and specify the number of bars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

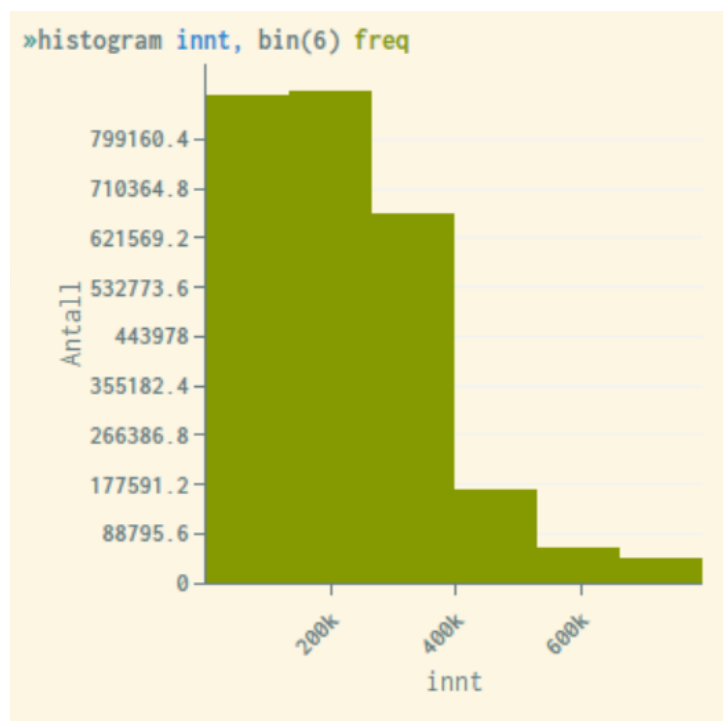
People with very high or very low income can easily be identified if the range of values becomes too narrow, which is problematic in terms of privacy. Therefore, the system performs a top/bottom coding where the 1% highest and 1% lowest values are replaced by the respective limit values. Thus, the first 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.

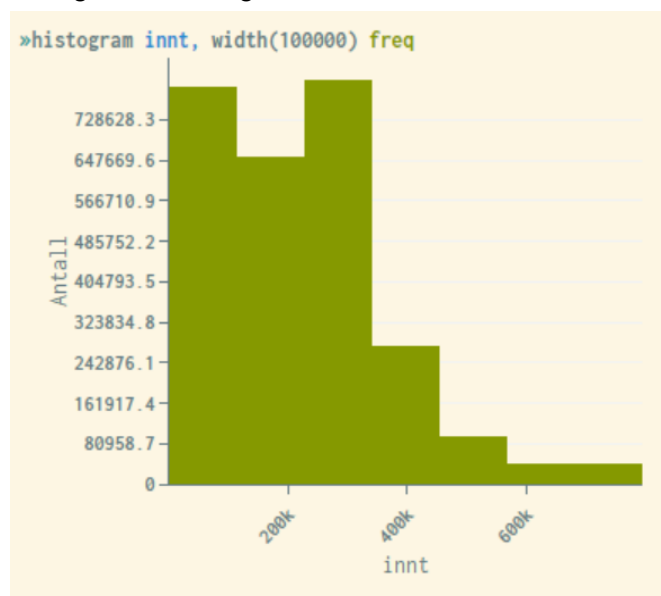
Example:



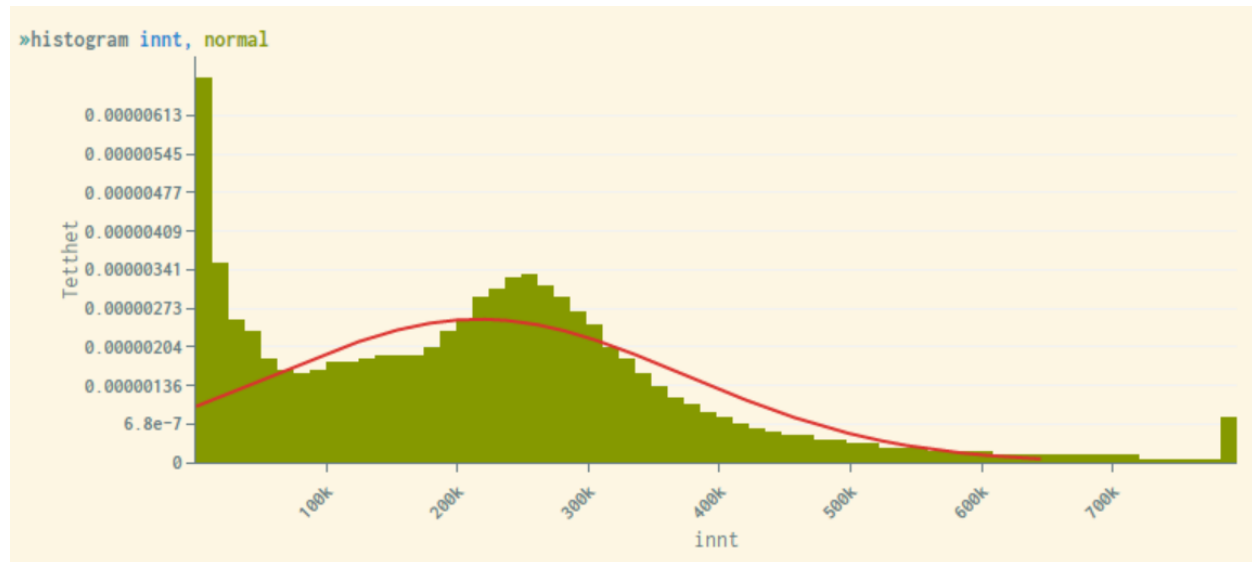
Histogram showing income distributed over 6 intervals, and frequency numbers on the y-axis (each bar has the same income interval width):



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

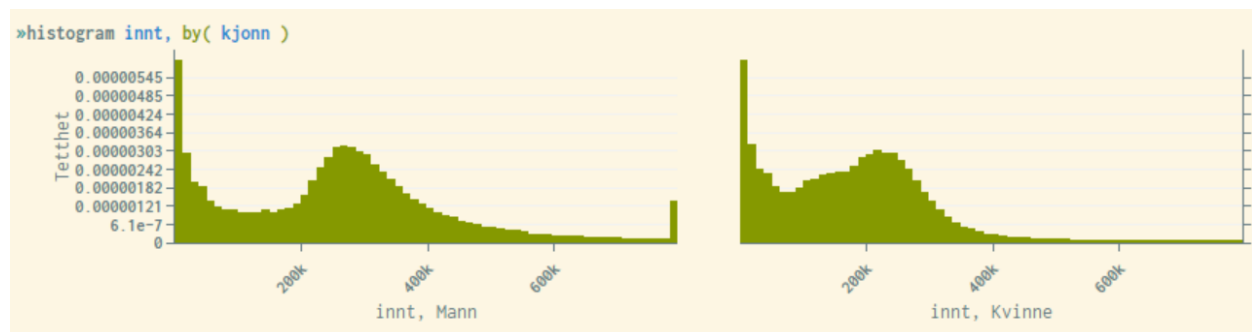


Through the option `normal`, a normal distributed curve is placed over the bars in the figure. This is helpful to study the degree of deviation from a normal distribution:



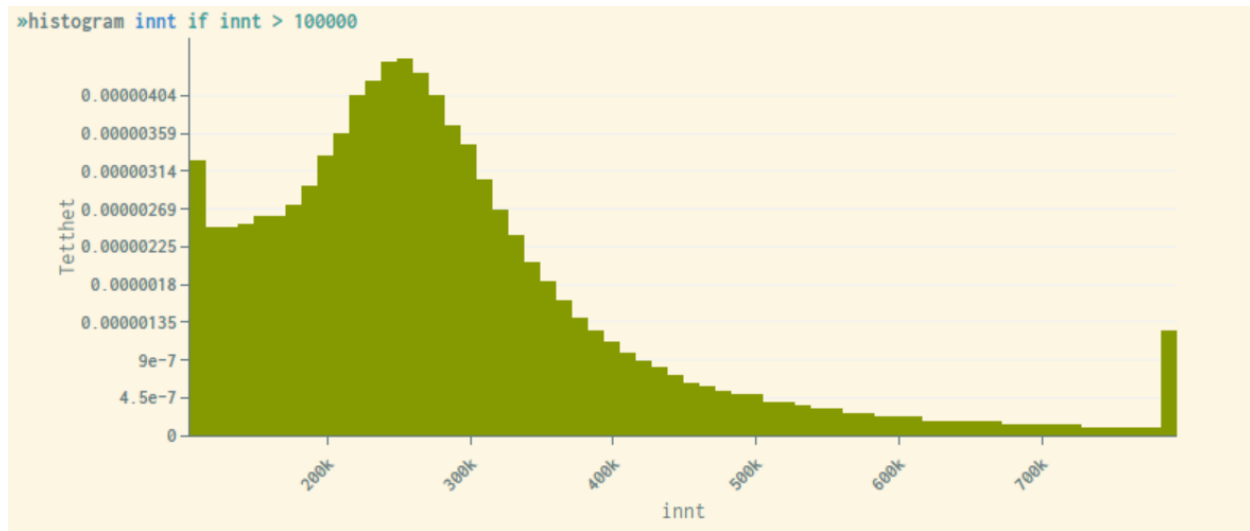
Histograms can be displayed over distributions for another variable that must be categorical, e.g. gender. This is done through the option `by(<variable>)`.

Example:



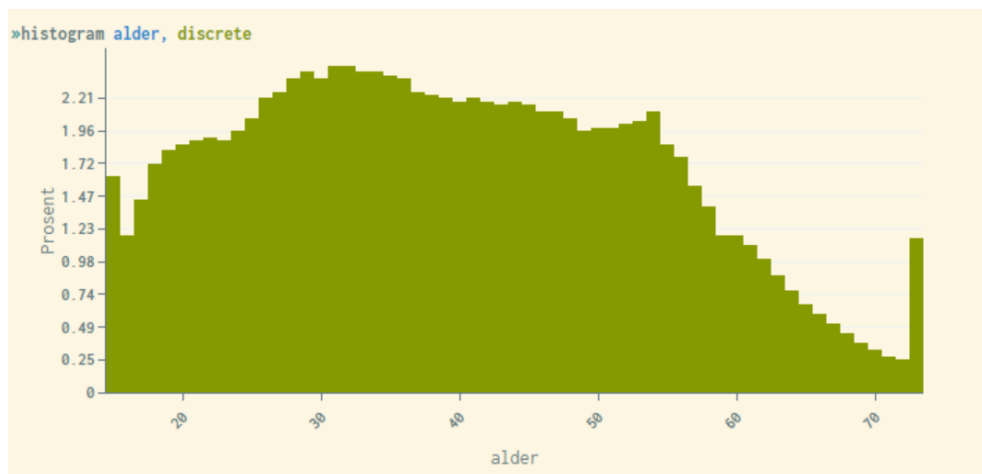
Like other statistical representations in microdata.no, filtering can be performed through if-conditions, where the histogram is shown only for a sub-population.

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



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):



For more information about this command, use the `help histogram` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

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.

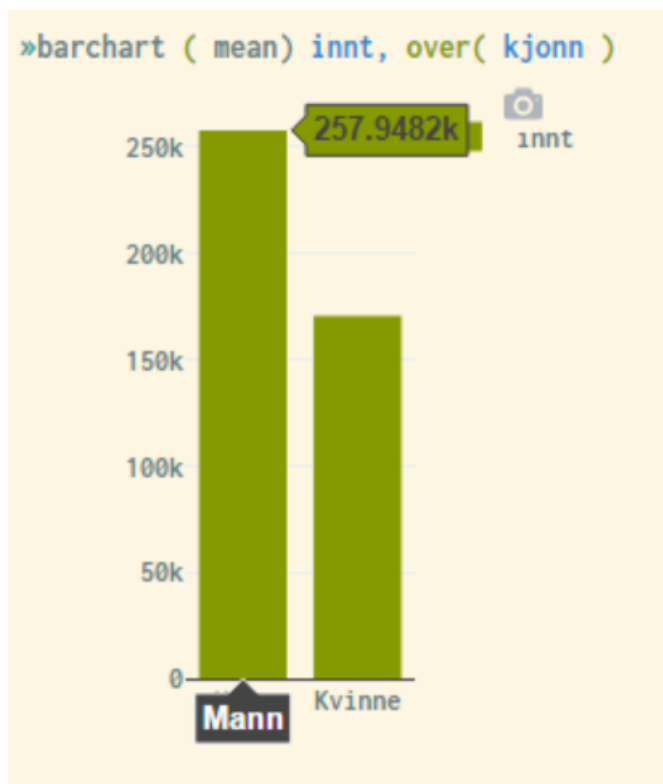
Syntax:

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

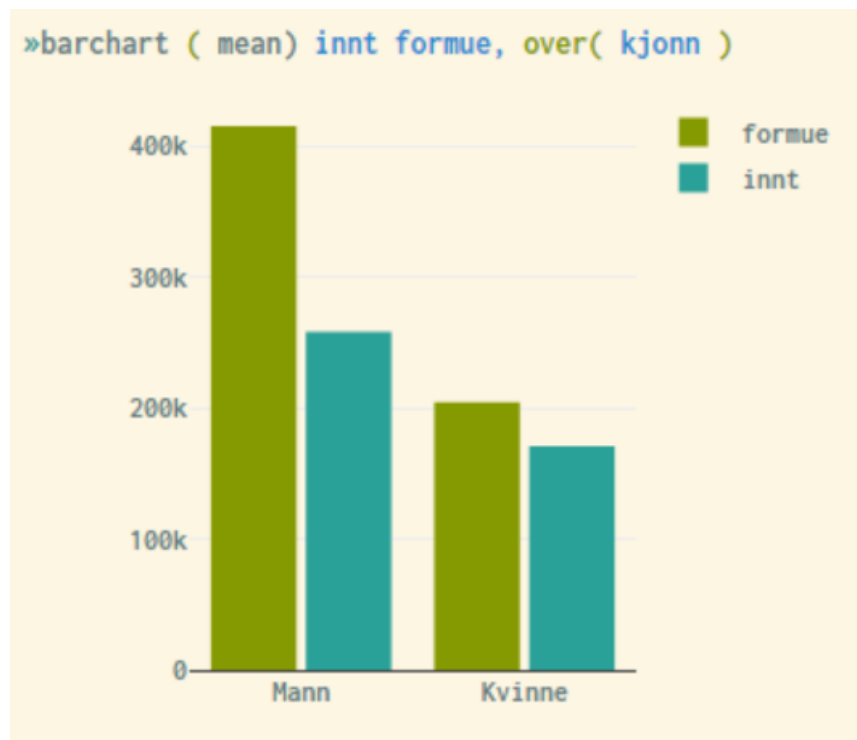
Stacked barcharts can be made through the option `stack`.

For more information about this command, use the `help barchart` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

Example of barcharts measuring mean income distributed over gender:



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



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.

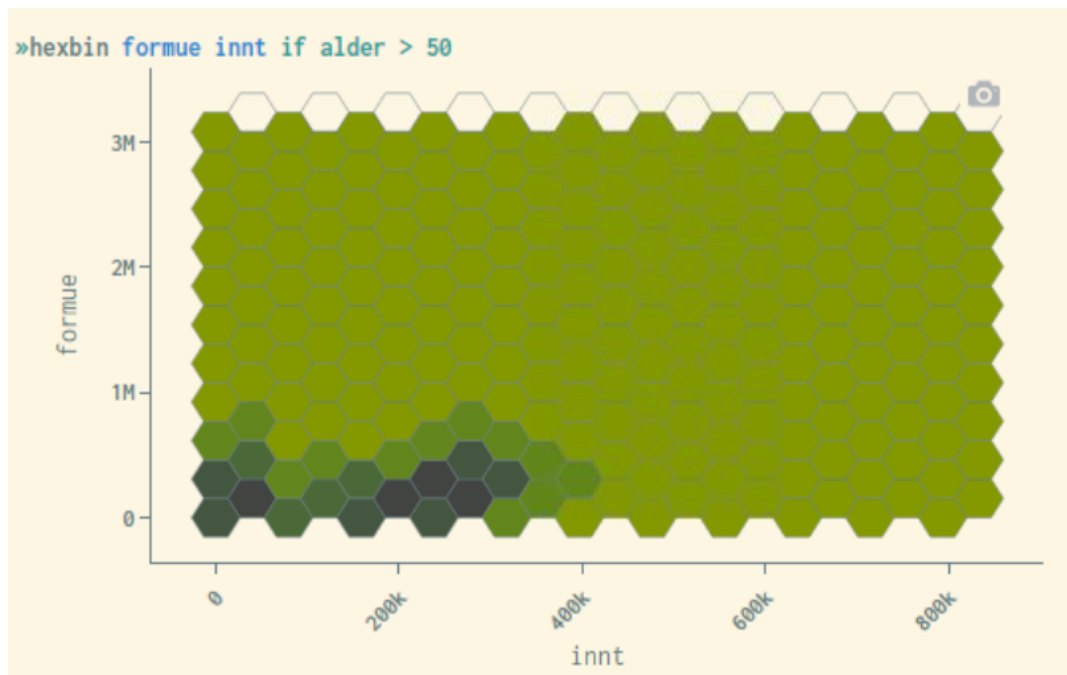
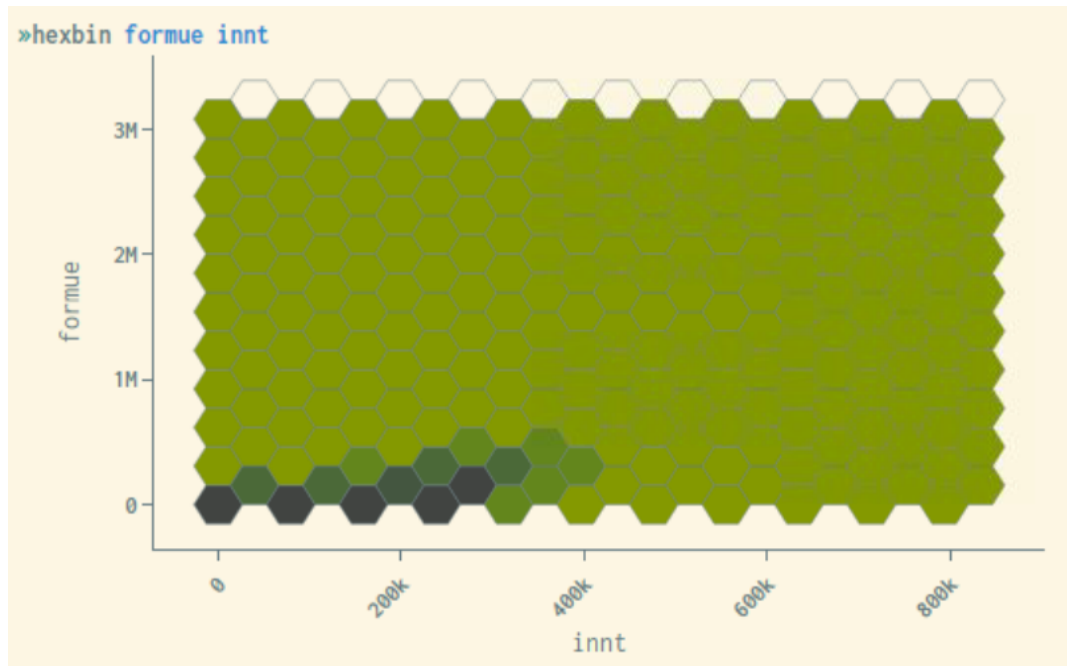
Hexbin diagrams produce a graphical presentation of the distribution of units between two continuous variables, and are not suitable for categorical variables.

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.

For more information about this command, use the `help hexbin` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

Examples:



4.7 Sankey - transition diagrams

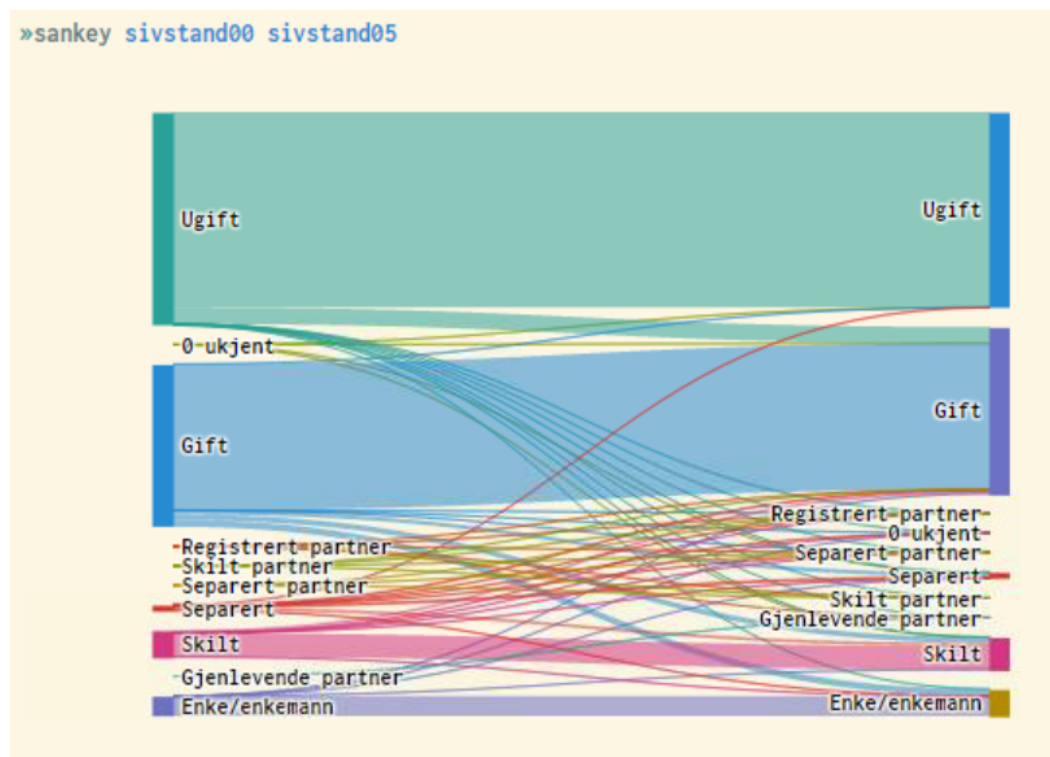
Sankey diagrams are a way to visualize transitions between statuses. In microdata.no, this can be used to get an overview of units' (individual's) movements between two time-measurements, either for the same variable or for different variables. Movements between different types of states (e.g. jobsearch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

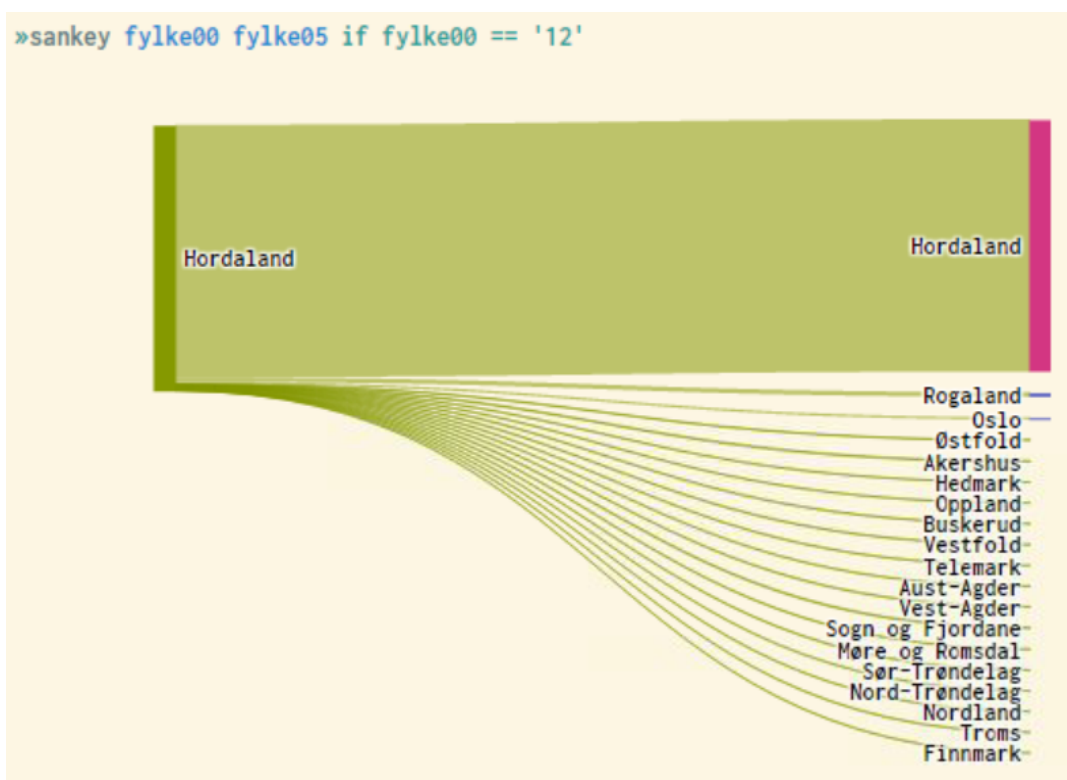
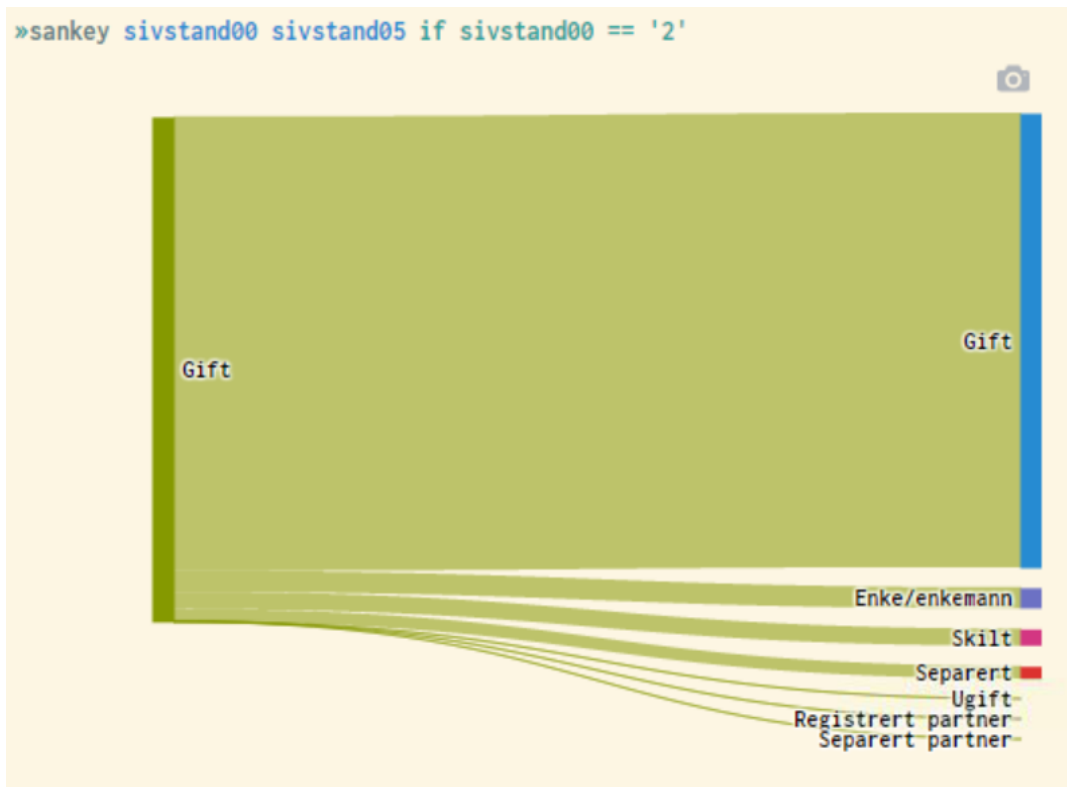
The transition visualization requires two categorical cross-sectional variables to be used. The number of categories should not be too large, as the chart can quickly become unreadable. This can be solved by converting into fewer categories or by using an if-condition that controls which transitions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

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

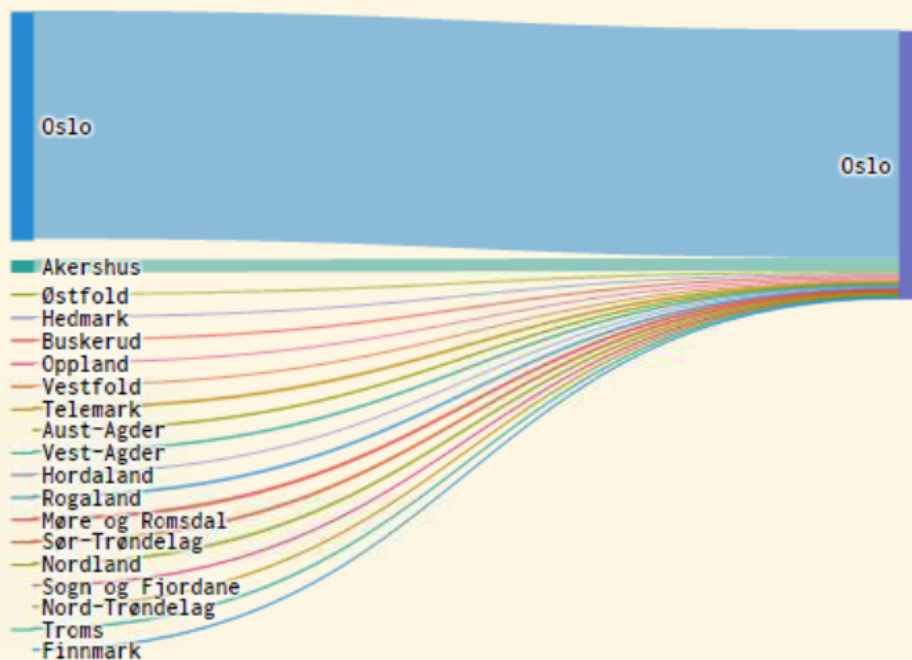
For more information about this command, use the `help sankey` command. This will display syntax examples and a complete list of available options that can be used to customize the appearance of the statistics generated.

Examples:

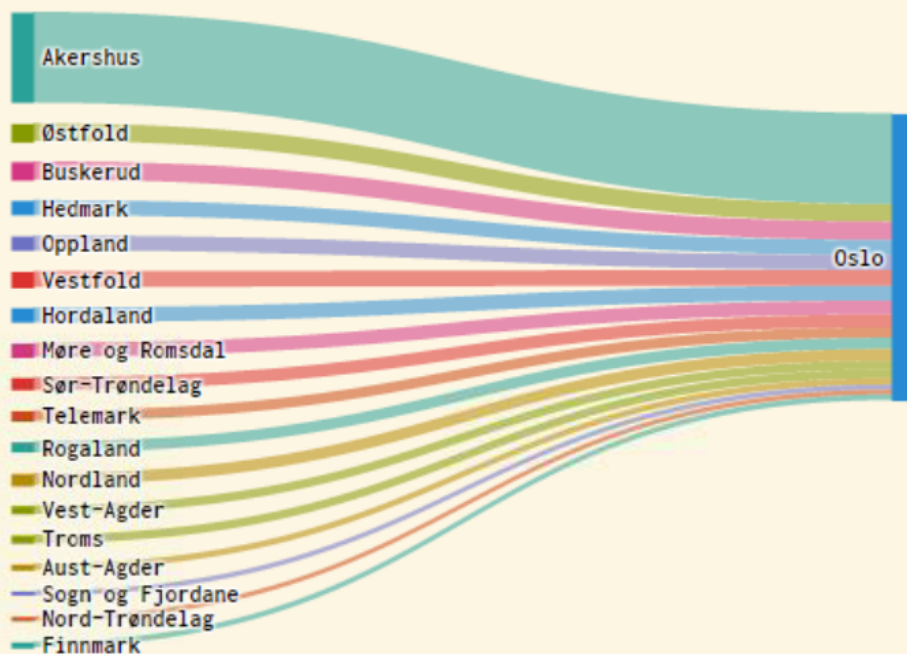




»sankey fylke00 fylke05 if fylke05 == '03'



»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:12 as db

create-dataset demographics
import db/INNTEKT_WYRKINNT 2015-01-01 as income
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import db/SIVSTANFDT_SIVSTAND 2015-01-01 as maritalstate15
import db/SIVSTANFDT_SIVSTAND 2019-01-01 as maritalstate19
import db/BOSATTEFDT_BOSTED 2015-01-01 as residence
import db/BEFOLKNING_STATUSKODE 2015-01-01 as regstat

// Recode from municipality to county level
generate county = substr(residence,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 county countystring

tabulate county
tabulate gender
```

```

tabulate gender regstat
tabulate gender county

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

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

// Cross-table only for persons over 30 years
generate age = 2015 - int(birth_year_month/100)
tabulate county regstat if age > 30

// Percentage tabulation
tabulate maritalstate15 maritalstate19, rowpct
tabulate maritalstate15 maritalstate19, colpct
tabulate maritalstate15 maritalstate19, cellpct
tabulate maritalstate15 maritalstate19, 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 county 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:12 as db

create-dataset demographics
import db/INNTEKT_WYRKINNT 2015-01-01 as income
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import db/BOSATTEFDT_BOSTED 2015-01-01 as residence

```



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

// Generate age per 2015
generate age = 2015 - int(birth_year_month/100)

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

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

4.8.3 Histogram and barchart

```
// Histogram and barchart

require no.ssb.fdb:12 as db

create-dataset demographics
import db/INNTEKT_WYRKINNT 2015-01-01 as income
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month

// Generate age per 2015
generate age = 2015 - 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(gender)
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:12 as db

create-dataset demographics
import db/INNTEKT_WYRKINNT 2015-01-01 as income
import db/INNTEKT_BRUTTOFORM 2015-01-01 as wealth
import db/BEFOLKNING_KJOENN as gender
import db/BEFOLKNING_FOEDSELS_AAR_MND as birth_year_month
import db/BOSATTEFDT_BOSTED 2015-01-01 as residence

// Recoding from municipality to county level
generate county = substr(residence,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 county countystring

```

```
// Generate age per 2015
generate age = 2015 - 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 county

// 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:12 as db

create-dataset demographics
import db/SIVSTANDFDT_SIVSTAND 2010-01-01 as maritalstate10
import db/SIVSTANDFDT_SIVSTAND 2015-01-01 as maritalstate15
import db/BOSATTEFDT_BOSTED 2010-01-01 as residence10
import db/BOSATTEFDT_BOSTED 2015-01-01 as residence15

// Recoding from municipality to county level
generate county10 = substr(residence10,1,2)
generate county15 = substr(residence15,1,2)

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 county10 countystring  
assign-labels county15 countystring
```

```
sankey county10 county15 if county10 == '12'
```

```
sankey county10 county15 if county15 == '03'
```

```
sankey county10 county15 if county15 == '03' & county10 != '03'
```

```
sankey maritalstate10 maritalstate15
```

```
sankey maritalstate10 maritalstate15 if maritalstate10 == '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`
- `normaltest`
- `regress`
- `ivregress`
- `logit / probit`
- `mlogit`
- `regress-panel`
- `predict` commands for retrieving prediction and residual values etc.

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 <condition>] [, <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 5.4).

Syntax:

```
anova <variable> <variable list> [if <condition>] [, <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: 1.905535e+5
 R²: 0.0746 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 Normal test

The Normal test command runs a selection of normal distribution tests for specified variables, or entire datasets if no variables are listed. For each test, the parameter value and p-value are stated. The tests that are run are skewness, kurtosis, s-k (not adjusted), Jarque-Bera and Shapiro-Wilk.

Syntax:

```
normaltest <variable list> [if <condition>]
```

Example of a test for normal distribution for the variable `innt19` (occupational income measured in 2019). P-values lower than 0.05 mean that the distribution is not normally distributed, and vice versa:

```
demografidata» normaltest innt19
```

		Test	P
innt19	Skjevhet	2300.9636637	0
	Kurtose	1295.6776859	0
	Shapiro-Wilk	0.6435465	0
	Jarque-Bera	1.0114248e+12	0
	Normaltest	6973214.4474433	0

5.4 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.

Syntax:

```
regress <variable> <variable list> [if <condition>] [,
<options>]
```

The dependent variable must be entered first, followed by the explanatory variables. Options can be used for various purposes, such as robust or cluster estimation, cf. the subchapters below. Like other statistical commands, regression commands can be combined with an if condition to run regressions on selected groups. For a full list of options, use the `help regress` command.

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 R²*” 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	Antall Obs: 2489943		
Modell	1.074715e+16	4	2.686788e+15	F(4, 2489938): 7693		
Residual	8.695606e+16	2.489938e+6	3.492298e+10	R ² : 0.1099		
Total	9.770321e+16	2.489942e+6	3.923915e+10	Justert R ² : 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 R² (*Adjusted R²*) is an overall measure of how much of the observed variance in the dependent variable is explained by the sum of the independent variables. 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 *R²* 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 statistically significant. *Justert R²* takes this into account and will only increase in value if the extra variables are significant.

If *Justert R²* 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), cf. 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.

5.4.1 Factor variables

Factor variables can be used to automate the recoding of multi-category variables so that they can be used in a regression expression. In practice, each category minus the reference category will be represented by separate dummy variables, where the estimates are interpreted relative to the reference category. The prefix `i.` is then used in front of the variable name in the regression expression, and the lowest value is automatically used as the reference value.

Factor variables can also be used to estimate the effects of combinations of values for categorical variables (in addition to the effect each individual explanatory variable has separately). The rationale behind this is that certain properties have different effects on the dependent variable when looking at different groups. For example, the effect of education on future income may be systematically different for men versus women. In such cases, factor variables can be useful.

In regression expressions, factor variables and combinations of these are specified in the same way as in Stata. Thus, the `i.` prefix is used to indicate that a variable is to be interpreted as categorical, while the `#` symbol is used to indicate that all categories except the reference groups are to be combined and estimated through the respective coefficient estimates. When using `##`, each individual category will also be estimated separately and included in the regression analysis.

Example:

Linear regression analysis with `income19` as the dependent variable. The independent variables are `man`, `edulevel`, and all subgroups of the two variables combined with each other except the reference group:

```
regress income19 i.man i.edulevel edulevel#man
```

Result:

demografidata» regress innt19 i.mann i.utdanningsnivå utdanningsnivå#mann

Kilde	SS	df	MS	Antall Obs: 2849451		
Modell	89493449031967680	17	5.2643205e+15	F(17, 2849433): 31688.092777		
Residual	473374297090160100	2.849433e+6	1.661293e+11	R²: 0.15899		
Total	562867746122127740	2.84945e+6	1.9753558e+11	Justert R²: 0.15899		
				Root MSE: 4.0758962e+5		

innt19	Coef.	Std.feil	t	P> t	[95% Konf. intervall]
mann 1	1.3303472e+5	13515.2	9.84328	0	1.0654525e+5 1.5952419e+5
utdanningsnivå 1 x mann	-62208.5	15641.7	-3.97709	0.00007	-92865.8 -31551.3
utdanningsnivå 2 x mann	-12423	13556.1	-0.91641	0.35945	-38992.6 14146.6
utdanningsnivå 3 x mann	16534.7	13702	1.20673	0.22753	-10320.8 43390.2
utdanningsnivå 4 x mann	54116.1	13543.4	3.99574	0.00006	27571.4 80660.7
utdanningsnivå 5 x mann	1.3340997e+5	13806.3	9.66294	0	1.0635002e+5 1.6046991e+5
utdanningsnivå 6 x mann	57306.9	13548.4	4.22979	0.00002	30752.5 83861.3
utdanningsnivå 7 x mann	1.3155888e+5	13598.2	9.67468	0	1.0490677e+5 1.5821099e+5
utdanningsnivå 8 x mann	85630.4	14292.6	5.99122	0	57617.3 1.1364354e+5
utdanningsnivå 1	12109.6	11206	1.08063	0.27985	-9853.81 34073.1
utdanningsnivå 2	-7298	9571.06	-0.7625	0.44575	-26056.9 11460.9
utdanningsnivå 3	1.1586814e+5	9669.74	11.9825	0	96915.7 1.348205e+5
utdanningsnivå 4	1.1891478e+5	9561.66	12.4366	0	1.0017425e+5 1.3765532e+5
utdanningsnivå 5	2.0559287e+5	9812.71	20.9516	0	1.8636029e+5 2.2482545e+5
utdanningsnivå 6	2.6033553e+5	9556.87	27.2406	0	2.4160439e+5 2.7906666e+5
utdanningsnivå 7	4.405264e+5	9596.47	45.905	0	4.2171765e+5 4.5933516e+5
utdanningsnivå 8	5.124071e+5	10152.7	50.4696	0	4.9250803e+5 5.3230618e+5
Konst	2.3898019e+5	9538.33	25.0546	0	2.2028538e+5 2.5767499e+5

This alternative expression will give the same result:

```
regress income19 edulevel##man
```

The `c.` prefix can be used to signal that a variable is to be regarded as a continuous variable (non-categorical). This may be relevant to use in cases where a variable can be interpreted as

continuous, e.g. level of education. The following expression runs a similar regression as above, but where education level is considered a continuous variable:

```
regress income19 i.mann c.edulevel edulevel#mann
```

Result:

demografidata» regress innt19 i.mann c.utdanningsnivå utdanningsnivå#mann						
Kilde	SS	df	MS	Antall Obs: 2849451		
Modell	8.9493449e+16	17	5.2643205e+15	F(17, 2849433): 31688.092777		
Residual	4.733743e+17	2.849433e+6	1.661293e+11	R²: 0.15899		
Total	5.6286775e+17	2.84945e+6	1.9753558e+11	Justert R²: 0.15899		
				Root MSE: 4.0758962e+5		
innt19	Coef.	Std.feil	t	P> t	[95% Konf.	intervall]
utdanningsnivå	58638.7	1194.24	49.101	0	56298	60979.4
utdanningsnivå x 0	-46529.1	10526.9	-4.41999	0.00001	-67161.5	-25896.6
utdanningsnivå x 1	-1.2457553e+5	8078.72	-15.4201	0	-1.4040955e+5	-1.0874151e+5
utdanningsnivå x 2	-60048.1	7627.63	-7.87244	0	-74998	-45098.2
utdanningsnivå x 3	-1.1564027e+5	7081.39	-16.3301	0	-1.2951956e+5	-1.0176098e+5
utdanningsnivå x 4	-87600.9	7156.52	-12.2407	0	-1.0162748e+5	-73574.4
utdanningsnivå x 5	-91497	6820.51	-13.4149	0	-1.0486502e+5	-78129
utdanningsnivå x 6	30055	7042.62	4.26759	0.00002	16251.7	43858.3
utdanningsnivå x 7	43297	7902.05	5.4792	0	27809.2	58784.7
utdanningsnivå x 8	1.3303472e+5	13515.2	9.84328	0	1.0654525e+5	1.5952419e+5
utdanningsnivå x 9	-1.0873769e+5	10209.8	-10.6503	0	-1.2874857e+5	-88726.8
utdanningsnivå x 10	-1.3699855e+5	8098.19	-16.9171	0	-1.5287072e+5	-1.2112638e+5
utdanningsnivå x 11	-43513.4	7651.56	-5.68686	0	-58510.2	-28516.6
utdanningsnivå x 12	-61524.1	7087.67	-8.68044	0	-75415.7	-47632.5
utdanningsnivå x 13	45809	6997.09	6.54686	0	32094.9	59523
utdanningsnivå x 14	-34190.1	6829.01	-5.00659	0	-47574.7	-20805.4
utdanningsnivå x 15	1.6161394e+5	7032.37	22.9814	0	1.4783073e+5	1.7539715e+5
utdanningsnivå x 16	1.2892746e+5	7770.6	16.5916	0	1.1369734e+5	1.4415758e+5
Konst	2.3898019e+5	9538.33	25.0546	0	2.2028538e+5	2.5767499e+5

5.4.2 Model diagnostics

It is possible to perform tests of the regression model against the data being analyzed, in order to check whether the selected model needs to be moderated. This is done by specifying options for the relevant test parameters to be displayed. The regression result will then include the parameter values below the main table.

The following options can be used for model testing:

- `ov`: Ramsey's RESET test for omitted variables. Displays a total F-value with associated P-value
- `vif`: Variance inflation factor test for multicollinearity. Displays variance inflation factor (VIF) values for the independent variables in a table
- `het_bp`: Breusch-Pagan test for heteroskedasticity. Displays a total chi-square value with the corresponding P-value
- `het_iid`: Studentized Breusch-Pagan test for heteroskedasticity. Displays a total chi-square value with the corresponding P-value (a newer version of the BP test that is more robust since it does not assume that the residuals are normally distributed)
- `het_fstat`: F-statistics from the Breusch-Pagan test for heteroskedasticity. Displays a total F-value with associated P-value. The number of degrees of freedom is not based on the number of variables in the regression model, but stems from an additional OLS model that compares residuals and predicted values. Therefore only one degree of freedom

Example of a test for omitted variables, multicollinearity and heteroskedasticity:

```
regress income2019 man married age highwealth, ov vif het_bp
```

Result (retrieves all test parametres):

demografidata» regress innt19 mann gift alder formuehøy, ov vif het_bp het_iid het_fstat

Kilde	SS	df	MS	Antall Obs: 2958926		
Modell	88819204460328320	4	22204801115082080	F(4, 2958921): 133699.007529		
Residual	491419147639550460	2.958921e+6	1.6608052e+11	R²: 0.15307		
Total	580238352099878800	2.958925e+6	1.9609769e+11	Justert R²: 0.15307		
				Root MSE: 4.0752978e+5		

innt19	Coef.	Std.feil	t	P> t	[95% Konf. intervall]
mann	1.3352883e+5	476.411	280.28	0	1.3259508e+5 1.3446258e+5
gift	87311.8	541.892	161.124	0	86249.7 88373.9
alder	4631.46	19.9379	232.293	0	4592.39 4670.54
formuehøy	2.7244037e+5	658.837	413.516	0	2.7114907e+5 2.7373167e+5
Konst	1.5790516e+5	787.586	200.492	0	1.5636152e+5 1.594488e+5

Breusch-Pagan

chi2(1): 2154648.283657
Prob > chi2: 0

Breusch-Pagan, studentisert

chi2(1): 1105.523026
Prob > chi2: 0

Breusch-Pagan, f-test

F(1, 2958924): 1105.935482
Prob > F: 0

Ramseys RESET test

F(3, 2958918): 4677.063546
Prob > F: 0

Variance inflation factor

	VIF	1/VIF
mann	1.009167	0.990917
gift	1.242664	0.804723
alder	1.346809	0.742496
formuehøy	1.114218	0.89749
Gj.snitt	1.178215	-

There are also other methods in microdata.no to test regression models:

- The `correlate` command provides pairwise measures of correlation between individual variables in a table matrix. This can reveal multicollinearity. Chapter 5.1 reviews this command.
- The `regress-predict` command provides the opportunity to study e.g. residual and prediction values. This can be combined with graphical tools such as `histogram` to show residual distributions and to check for e.g. normal distribution. The `hexbin` command can also display an anonymized 2-way plot. See section 5.4.4 for a review of this command.

5.4.3 Cluster and robust estimation

The options `robust` and `cluster()` are used separately to specify whether one wants resp. robust or cluster estimation, and will as a result present regression estimates with adjusted standard deviations for the estimated coefficients. Associated t-, z- and p-values are also affected. Other values are not affected compared to standard estimation.

Note that `robust` and `cluster` can not be used in combination (`cluster` implies robust estimation).

Robust estimation can be used where there is a suspicion of problematic outliers or heteroskedasticity.

Cluster estimation is used when it is suspected that there are systematic dependencies within groups of observations, e.g. within schools or municipalities. The groups are specified through a variable (cluster variable) which is included in the parentheses of the cluster option, e.g. `cluster(school)` or `cluster(municipality)`. The following conditions apply, otherwise the system will give an error message:

- The number of groups must be of a certain size
- The cluster variable must be numeric
- The cluster variable cannot be included as a variable in the regression expression.

Examples:

```
regress income man married high_education, robust
regress income man married high_education,
cluster(municipality)
```

Robust and cluster options can also be used on other regression types.

5.4.4 Prediction and residual values

All regression variants found in microdata.no have associated commands that generate, among other things, residual and prediction values. These are values that can be used to analyze the data spread and for testing regression models. Prediction values can also be used as input for further analyzes.

The commands have the same name as the associated regression command plus "-predict"

Syntax:

```
regress-predict <variable> <variabel list> [if <condition>] [,
<options>]
```

The variables are specified in the same way as for the corresponding regression model run with the `regress` command.

The following values can be retrieved: Prediction values, residuals and "Cook's distance"

You decide which values you want to generate through the use of options. The result of the runs is a set of variables that contain the different values. By default, the former value type is generated, but it is still recommended to specify value type through options as this makes you able to create names for the generated variables inside parentheses as shown in the syntax example below. If you run several `predict` commands, you have to create new names for the automatically generated variables.

Syntax example:

```
regress-predict wage age man wealth, residuals(res)
predicted(pred) cooks(cook)
```

The automatically generated variables can be used as input for further analyzes or to be displayed graphically. Current graphical commands are `hexbin` and `histogram`. By running a `histogram` on the residual variable, one can check whether the residuals are normally distributed. The `hexbin` command can also be used to create anonymized scatter plots where one combines two sets of values.

For more details, it is recommended to use the `help regress-predict` command.

5.5 IV-regression - linear regression analysis with instrument variables

If one suspects dependence between independent variables in a linear regression model, the `ivregress` command can be used to set up an expression that defines which variables this applies to.

Syntax:

```
ivregress <variable> <variable list> (<variable list> =
<variable list>) [if <condition>] [, <options>]
```

The dependent variable must be entered first, followed by the explanatory variables and the instrument expression which is indicated in parentheses. Options can be used for various purposes, such as robust or cluster estimation, cf. the subchapters below. Like other statistical commands, regression commands can be combined with an if condition to run regressions on selected groups. For a full list of possibilities, use the `help ivregress` command.

Example where one suspects that wealth is related to age and place of residence (= Oslo):

```
ivregress wage man (highwealth = age oslo)
```

The result of the run is a standard regression result where the instrument variable and instruments are listed below the table. In practice, all independent variables are treated as instruments, except for the variable which is defined as an instrument variable.

Example where `highwealth` (`formuehøy`) is instrument variable and `age` (`alder`) is instrument (`man` is also included as an instrument even if the variable is not specified in the parentheses expression):

```
ivtest» ivregress lønn mann (formuehøy = alder)
```

Kilde	SS	df				
Modell	-2.4773801e+17	3				
Residual	6.793658e+17	2.631922e+6				
Total	4.3162778e+17	2.631925e+6				

Antall Obs: 2631925						
chi2(2): 208368.33996						
Prob > chi2: 0						
R²: -0.57396						
Justert R²: -0.57396						
Root MSE: 5.0806009e+5						

lønn	Coef.	Std.feil	t	P> t	[95% Konf. intervall]
mann	69580.9	651.824	106.748	0	68303.4 70858.5
formuehøy	1.2590452e+6	3167.96	397.43	0	1.2528361e+6 1.2652543e+6
Konst	2.6450893e+5	599.623	441.124	0	2.6333369e+5 2.6568417e+5

Instrumenterte: formuehøy

Instrumenter: mann alder

5.5.1 Factor variables

See chapter 5.4.1 for information on how factor variables can be used. The procedure is the same as for ordinary linear regression.

5.5.2 Model diagnostics

Various diagnostics related to instrument variable modeling are under development and testing, and will soon be available. This includes standard test estimators for endogeneity, correlation and overidentification. Until then, the modeling can be tested in the following ways:

- Run regression with and without instrumentation, and compare the result: `regress` vs. `ivregress`
- Use the `correlate` command to check for correlation between selected variables
- Study residuals a.o. through the `ivregress-predict` command

5.5.3 Cluster and robust estimation

See chapter 5.4.3 for information on how to use cluster or robust estimation. The procedure is the same as for ordinary linear regression.

5.5.4 Prediction and residual values

All regression variants found in microdata.no have associated commands that generate, among other things, residual and prediction values. These are values that can be used to analyze the data spread and for testing regression models. Prediction values can also be used as input for further analyses.

The commands have the same name as the associated regression command plus "-predict"

Syntax:

```
ivregress-predict <variable> <variable list> (<variable list> =
<variable list>) [if <condition>] [, <options>]
```

The variables are set in the same way as for the corresponding regression model run with the `ivregress` command.

The following values can be retrieved: Prediction values, and residuals

You decide which values you want to generate through the use of options. The result of the runs is a set of variables that contain the different values. By default, the former value type is generated, but it is still recommended to specify value type through options as this makes you able to create names for the generated variables inside parentheses as shown in the syntax example below. If you run several `predict` commands, you have to create new names for the automatically generated variables.

Syntax example:

```
ivregress-predict wage man (wealth = age), residuals(res3)
predicted(pred3)
```

The automatically generated variables can be used as input for further analyzes or to be displayed graphically. Current graphical commands are `hexbin` and `histogram`. By running a `histogram` on the residual variable, one can check whether the residuals are normally distributed. The `hexbin` command can also be used to create anonymized scatter plots where one combines two sets of values.

For more details, it is recommended to use the `help ivregress-predict` command.

5.6 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.

Syntax:

```
logit <variable> <variable list> [if <condition>] [, <options>]
probit <variable> <variable list> [if <condition>] [,
<options>]
```

The dependent variable must be entered first, followed by the explanatory variables. Options can be used for various purposes, such as robust or cluster estimation, cf. the subchapters below. Like other statistical commands, regression commands can be combined with an if condition to run regressions on selected groups. For a full list of options, use the `help logit` or `help probit` command.

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 income05 > 400000
```

Like ordinary linear regression analysis (see chapter 5.4), 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 R²* (*Adjusted R²*) 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.

5.6.1 Factor variables

See chapter 5.4.1 for information on how factor variables can be used. The procedure is the same as for ordinary linear regression.

5.6.2 Marginal effects

The `mf()` option is used to specify that marginal effects are to be estimated in addition to the usual logistic coefficients. This is preferred by many since marginal effects are easier to interpret than standard estimates.

It is possible to choose between four different types of marginal effects:

- `dydx`: marginal effect = $d(y) / d(x)$

- e_{yex} : elasticity value = $d(\ln(y)) / d(\ln(x))$
- dy_{ex} : semi-elasticity = $d(y) / d(\ln(x))$
- e_{ydx} : semi-elasticity = $d(\ln(y)) / d(x)$

Example of a logit regression including estimated average marginal effects (most commonly used). Common logistical estimates are listed first, then the marginal effects:

demografidata» logit høyinnt mann gift alder innvandrere ettbarn flerebarn høyutd oslo ledig formuehøy, mfx(dydx)

Antall iter: 8 LR chi2(11): 33066.7
 Log sans: -85882.3 Prob > chi2: 0
 Antall obs: 667309 Pseudo R2: 0.16143

høyinnt	Coef.	Std.feil	z	P> z	[95% Konf. intervall]
mann	0.98596	0.01477	66.7287	0	0.957 1.01492
gift	0.57347	0.01664	34.4628	0	0.54086 0.60609
alder	0.02396	0.00063	37.6186	0	0.02271 0.02521
innvandrere	-0.42407	0.01834	-23.1177	0	-0.46003 -0.38812
ettbarn	0.41667	0.01973	21.1096	0	0.37798 0.45535
flerebarn	0.64685	0.01867	34.6332	0	0.61025 0.68346
høyutd	1.54034	0.01902	80.9701	0	1.50306 1.57763
oslo	0.14832	0.01858	7.97929	0	0.11189 0.18476
ledig	0.24728	0.02347	10.5358	0	0.20128 0.29328
formuehøy	1.42484	0.01573	90.5629	0	1.394 1.45567
Konst	-5.65992	0.02941	-192.447	0	-5.71756 -5.60227

dydx av høyinnt

	Margin	Std.feil	z	P> z	[% Konf. intervall]
mann	0.03052	0.00047	64.2524	0	0.02959 0.03145
gift	0.01775	0.00052	34.1367	0	0.01673 0.01877
alder	0.00074	0.00002	37.0107	0	0.0007 0.00078
innvandrere	-0.01313	0.00057	-23.0486	0	-0.01424 -0.01201
ettbarn	0.0129	0.00061	21.0301	0	0.01169 0.0141
flerebarn	0.02002	0.00058	34.2954	0	0.01888 0.02117
høyutd	0.04769	0.00061	78.0678	0	0.04649 0.04888
oslo	0.00459	0.00057	7.97561	0	0.00346 0.00572
ledig	0.00765	0.00072	10.5239	0	0.00623 0.00908
formuehøy	0.04411	0.00051	84.995	0	0.04309 0.04513

5.6.3 Cluster and robust estimation

See chapter 5.4.3 for information on how to use cluster or robust estimation. The procedure is the same as for ordinary linear regression.

5.6.4 Prediction and residual values

All regression variants found in microdata.no have associated commands that generate, among other things, residual and prediction values. These are values that can be used to analyze the

data spread and for testing regression models. Prediction values can also be used as input for further analyzes.

The commands have the same name as the associated regression command plus "-predict"

Syntax:

```
logit-predict <variable> <variable list> [if <condition>] [,
<options>]
probit-predict <variable> <variable list> [if <condition>] [,
<options>]
```

The variables are specified in the same way as for the associated regression model which is run with the command `logit` or `probit`.

The following values can be retrieved:

- `logit-predict`: Probability values, prediction values, and residuals
- `probit-predict`: Probability values and prediction values

You decide which values you want to generate through the use of options. The result of the runs is a set of variables that contain the different values. By default, the former value type is generated, but it is still recommended to specify value type through options as this makes you able to create names for the generated variables inside parentheses as shown in the syntax example below. If you run several `predict` commands, you have to create new names for the automatically generated variables.

Syntax example:

```
logit-predict highwage age man wealth, residuals(res4)
predicted(pred4) probabilities(prob4)
```

The automatically generated variables can be used as input for further analyzes or to be displayed graphically. Current graphical commands are `hexbin` and `histogram`. By running a `histogram` on the residual variable, one can check whether the residuals are normally distributed. The `hexbin` command can also be used to create anonymized scatter plots where one combines two sets of values.

For more details, it is recommended to use the `help logit-predict` or `help probit-predict` command.

5.7 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 <variable> <variable list> [if <condition>] [,
<options>]
```

The dependent variable must be entered first, followed by the explanatory variables. Options can be used for various purposes, such as robust or cluster estimation, cf. the subchapters below. Like other statistical commands, regression commands can be combined with an if condition to run regressions on selected groups. For a full list of options, use the `help mlogit` command.

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 5.6.

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 income05 > 200000
replace inntgr = 3 if income05 > 400000
```

5.7.1 Factor variables

See chapter 5.4.1 for information on how factor variables can be used. The procedure is the same as for ordinary linear regression.

5.7.2 Marginal effects

See chapter 5.6.2 for information on how marginal effects are estimated. The procedure is the same as for binary logistics models.

5.7.3 Cluster and robust estimation

See chapter 5.4.3 for information on how to use cluster or robust estimation. The procedure is the same as for ordinary linear regression.

5.7.4 Prediction and residual values

All regression variants found in microdata.no have associated commands that generate, among other things, residual and prediction values. These are values that can be used to analyze the data spread and for testing regression models. Prediction values can also be used as input for further analyzes.

The commands have the same name as the associated regression command plus `"-predict"`

Syntax:

```
mlogit-predict <variable> <variable list> [if <condition>] [,
<options>]
```

The variables are specified in the same way as for the corresponding regression model run with the `mlogit` command.

The following values can be retrieved: Probability values and prediction values

You decide which values you want to generate through the use of options. The result of the runs is a set of variables that contain the different values. By default, the former value type is generated, but it is still recommended to specify value type through options as this makes you able to create names for the generated variables inside parentheses as shown in the syntax example below. If you run several `predict` commands, you have to create new names for the automatically generated variables.

Syntax example:

```
mlogit-predict wagecat age man highwealth, predicted(pred6)
probabilities(prob6)
```

The automatically generated variables can be used as input for further analyzes or to be displayed graphically. Current graphical commands are `hexbin` and `histogram`. By running a `histogram` on the residual variable, one can check whether the residuals are normally distributed. The `hexbin` command can also be used to create anonymized scatter plots where one combines two sets of values.

For more details, it is recommended to use the `help mlogit-predict` command.

5.8 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.

Syntax:

```
regress-panel <variable> <variable list> [if <condition>] [,
<options>]
```

The dependent variable must be entered first, followed by the explanatory variables. Options can be used for various purposes, such as robust or cluster estimation, cf. the subchapters below. Like other statistical commands, regression commands can be combined with an if condition to run regressions on selected groups. For a full list of options, use the `help regress-panel` command.

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, cf. chapter 4.1, but shows instead 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, cf. 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	2011-12-31	408852.72 196058.35 4047 6066 1103085 303401 413573 497759.75
	2012-12-31	484347.39 198684.67 4065 13348 1202311 400655 474860 569977.25
	2013-12-31	527803.61 213332.71 4041 22857 1304797 431857.25 513947 617650.5
	2014-12-31	567728.92 235573.16 4043 21912 1462289 452964.25 546906 665830.75
	2015-12-31	596611.39 247937.55 4026 15000 1572028 474567.5 571551 701034.5
Total		516957.13 228922.05 20227 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.

5.8.1 Factor variables

See chapter 5.4.1 for information on how factor variables can be used. The procedure is the same as for ordinary linear regression.

5.8.2 Model diagnostics

It is possible to perform model testing to check whether fixed or random effects estimation should be used in connection with panel regressions. This is done by using the `hausman` command.

Syntax and input follow the same logic as the associated regression command (`regress-panel`): The dependent variable is used as the first input, and then the independent ones are listed.

Example:

```
regress-panel wage age highedu wealth oslo
hausman wage age highedu wealth oslo
```

The result of the `hausman` run is a standard regression result for resp. fixed and random effect estimation. In addition, the difference between the coefficients in the alternative estimates are also shown, as well as an aggregate measure that indicates which variant is best to use for the current data set: P-value based on chi-square diagnostics.

P-values < 0.05 indicate that there are systematic differences in the coefficient estimates and that fixed effect modeling fits the data best. P-values above this limit indicate the opposite (that random effect modeling should be used).

For more details, it is recommended to use the help command: `help hausman`

The `regress-panel-predict` command can also be used as a tool for model diagnostics, cf. chapter 5.8.4.

5.8.3 Cluster and robust estimation

See chapter 5.4.3 for information on how to use cluster or robust estimation. The procedure is the same as for ordinary linear regression.

5.8.4 Prediction and residual values

All regression variants found in microdata.no have associated commands that generate, among other things, residual and prediction values. These are values that can be used to analyze the data spread and for testing regression models. Prediction values can also be used as input for further analyzes.

The commands have the same name as the associated regression command plus "-predict"

Syntax:

```
regress-panel-predict <variable> <variable list> [if
<condition>] [, <options>]
```

The variables are specified in the same way as for the corresponding regression model run with the `regress-panel` command.

The following values can be retrieved: Prediction values, residuals, and unit effects

You decide which values you want to generate through the use of options. The result of the runs is a set of variables that contain the different values. By default, the former value type is generated, but it is still recommended to specify value type through options as this makes you able to create names for the generated variables inside parentheses as shown in the syntax example below. If you run several `predict` commands, you have to create new names for the automatically generated variables.

Syntax example:

```
regress-panel-predict wage man age wealth, predicted(ppred1)
residuals(pres1) effects(peff1)
```

The automatically generated variables can be used as input for further analyzes or to be displayed graphically. Current graphical commands are `hexbin` and `histogram`. By running a `histogram` on the residual variable, one can check whether the residuals are normally distributed. The `hexbin` command can also be used to create anonymized scatter plots where one combines two sets of values.

For more details, it is recommended to use the `help regress-panel-predict` command.

5.9 Example

```
textblock
```

```
Perform regression analysis, and retrieve prediction- and residual values
```

```
-----
```

This example demonstrates the use of the various regression commands, including how to extract prediction and residual values. Especially histogram is a very useful command that can be used to study visually the extent to which the residuals are normally distributed. But in principle, all available and relevant commands may be used for further analysis.

```
endblock
```

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

```
create-dataset regressiondata
```

```
import db/INNTEKT_WLONN 2019-12-31 as wage
```

```
import db/INNTEKT_BER_BRFORM 2019-12-31 as wealth
```

```
import db/BEFOLKNING_FOEDSELS_AAR_MND as birthdate
```

```
import db/BEFOLKNING_KJOENN as gender
```

```
import db/BEFOLKNING_STATUSKODE 2020-01-01 as residentstatus
```

```
keep if residentstatus == '1'
```

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

```
generate male = 0
```

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

```
//regress
```

```
regress wage age male wealth
```

```
regress-predict wage age male wealth
```

```
histogram predicted
```

```
hexbin predicted wage
```

```
regress-predict wage age male wealth, residuals(res) predicted(pred) cooks(cook)
```

```
regress-predict wage age male, residuals(res2) predicted(pred2) cooks(cook2)
```

```
histogram pred
```

```
histogram res
```

```
histogram cook
```

```
histogram res2
```

```

//ivregress
ivregress wage male (wealth = age)
ivregress-predict wage male (wealth = age), residuals(res3) predicted(pred3)
histogram pred3
histogram res3

//logit
summarize wage wealth
histogram wage
histogram wealth
generate highwage = 0
replace highwage = 1 if wage > 800000
generate highwealth = 0
replace highwealth = 1 if wealth > 4000000

logit highwage age male highwealth
logit-predict highwage age male highwealth, residuals(res4) predicted(pred4) probabilities(prob4)
histogram pred4
histogram res4
histogram prob4

//probit
probit highwage age male highwealth
probit-predict highwage age male highwealth, predicted(pred5) probabilities(prob5)
histogram pred5
histogram prob5

//mlogit
generate wagecat = 0
replace wagecat = 1 if wage > 0
replace wagecat = 2 if wage > 800000

mlogit wagecat age male highwealth
mlogit-predict wagecat age male highwealth, predicted(pred6) probabilities(prob6)
summarize pred6_1
histogram pred6_2
histogram prob6_1
histogram prob6_2

//regress-panel

```

```
sample 0.05 54321
```

```
clone-units regressiondata paneldata
```

```
use paneldata
```

```
import-panel db/INNTEKT_WLONN db/BEFOLKNING_FOEDSELS_AAR_MND db/BEFOLKNING_KJOENN  
db/INNTEKT_BER_BRFORM 2017-12-31 2018-12-31 2019-12-31
```

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

```
generate male = 0
```

```
replace male = 1 if BEFOLKNING_KJOENN == '1'
```

```
rename INNTEKT_WLONN wage
```

```
rename INNTEKT_BER_BRFORM wealth
```

```
regress-panel wage male age wealth
```

```
regress-panel wage male age wealth, re
```

```
regress-panel-predict wage male age wealth, predicted(ppred1) residuals(pres1) effects(peff1)
```

```
regress-panel-predict wage male age wealth, re predicted(ppred2) residuals(pres2) effects(peff2)
```

```
histogram ppred1
```

```
histogram pres1
```

```
histogram peff1
```

```
histogram ppred2
```

```
histogram pres2
```

```
histogram peff2
```

```
hausman wage male age wealth
```

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
hausman	Hausman test for panel data regression models	Analysis command
hexbin	Anonymized scatterplot (hexbin plot)	Analysis command
histogram	Histogram	Analysis command
ivregress	Linear regression with instrumental variabels	Analysis command
ivregress-predict	Generate variabels containing results from a linear regression with instrumental variabels	Analysis command
logit	Logistic regression analysis: Logit	Analysis command
logit-predict	Generate variabels containing results from a logit regression	Analysis command
mlogit	Multinomial logistic regression analysis	Analysis command
mlogit-predict	Generate variabels containing results from an mlogit regression	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
probit-predict	Generate variabels containing results from a probit regression	Analysis command
regress	Linear regression	Analysis command
regress-predict	Generate variabels containing results from an ordinary linear regression (OLS)	Analysis command
regress-panel	Linear regression for panel data	Analysis command

regress-panel-predict	Generate variabels containing results from a linear regression on 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: Positive values
 - ❑ Output: Values between -744 and 709
 - ❑ Examples:

- ❑ `log10(arg1)`
 - ❑ Description: The base 10-logarithm of arg1
 - ❑ arg1: Positive values
 - ❑ 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 ≥ 0
 - ❑ Examples:

- ❑ `sqrt(arg1)`
 - ❑ Description: The square root of arg1
 - ❑ arg1: Values ≥ 0
 - ❑ Output: Values ≥ 0
 - ❑ Examples:

- ❑ `abs(arg1)`
 - ❑ Description: The absolute value of arg1 (i.e. removes negative signs)
 - ❑ arg1: Positive or negative values
 - ❑ Output: Values ≥ 0
 - ❑ Examples:

- ❑ `sin(arg1)`
 - ❑ Description: Returns the sinus value of `arg1`
 - ❑ `arg1`: Positive or negative numbers
 - ❑ Output: Values between -1 and 1
 - ❑ Examples:

- ❑ `cos(arg1)`
 - ❑ Description: Returns the cosinus value of `arg1`
 - ❑ `arg1`: Positive or negative numbers
 - ❑ Output: Values between -1 and 1
 - ❑ Examples:

- ❑ `tan(arg1)`
 - ❑ Description: Returns the tangens value of `arg1`
 - ❑ `arg1`: Positive or negative numbers
 - ❑ Output: Positive or negative numbers or missing
 - ❑ Examples:

- ❑ `asin(arg1)`
 - ❑ Description: Returns the radian value of the arcsinus of `arg1`
 - ❑ `arg1`: Values between -1 and 1
 - ❑ Output: Values between $-\pi/2$ and $\pi/2$
 - ❑ 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`: Positive or negative values
 - ❑ Output: Values between $-\pi/2$ and $\pi/2$
 - ❑ Examples:

- ❑ `ceil(arg1)`
 - ❑ Description: Round upwards to nearest integer
 - ❑ `arg1`: Positive or negative values
 - ❑ Output: Positive or negative integer values
 - ❑ Examples: `ceil(5.2) = 6`
`ceil(-5.2) = -6`

- ❑ **floor(arg1)**
 - ❑ Description: Round downwards to nearest integer. Equal to the function `int(arg1)`
 - ❑ arg1: Positive or negative values
 - ❑ Output: Positive or negative integer values
 - ❑ 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: Positive or negative values
 - ❑ Output: Positive or negative integer values
 - ❑ 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: Positive or negative values or missing
 - ❑ Examples:

- ❑ **lnfactorial(arg1)**
 - ❑ Description: The natural logarithm of n-factor ($= \ln(n!)$)
 - ❑ n: Integer values ≥ 0
 - ❑ Output: Values ≥ 0
 - ❑ Examples:

- ❑ **comb(n,k)**
 - ❑ Description: Combinational function value ($= n!/\{k!(n-k)!\}$)
 - ❑ n: Integer values ≥ 1
 - ❑ k: Integer values between 0 and n
 - ❑ Output: Values ≥ 0 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: Positive or negative values
 - ❑ arg2: Positive or negative values (default = 1 if arg2 is dropped)
 - ❑ Output: Positive or negative values
 - ❑ 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
```
- ❑ `quantile(arg1, arg2)`
 - ❑ Beskrivelse: Returns value based on the ranking of a continuous value over a selected division with equal numbers of values in each group. Possible divisions: 2-100. If 100 is used as an argument, the values 0-99 are returned based on which percentile a value is in. If the value 10 is used, values are grouped in deciles (0-9)
 - ❑ arg1: Variable with continuous values
 - ❑ arg2: Integer values between 2 and 100
 - ❑ Output: Integer values between 0 and 99
 - ❑ Examples:


```
generate income_p100 = quantile(income,100) (generates percentiles)
generate income_p10 = quantile(income,10) (generates deciles)
generate income_p4 = quantile(income,4) (generates quartiles)
```

Row calculations (based on two or more variables)

- ❑ `rowmin(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the minimum value of arg1, arg2,... .., argn for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ arg1, arg2,... .., argn: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples:


```
generate min_value = rowmin(1,2,3,4)
generate min_income = rowmin(income18, income19, income20)
```

- ❑ `rowmax(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the maximum value of `arg1, arg2,... .., argn` for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ `arg1, arg2,... .., argn`: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples: `generate max_value = rowmax(1,2,3,4)`
`generate max_income = rowmax(income18, income19, income20)`

- ❑ `rowmean(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the average value of `arg1, arg2,... .., argn` for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ `arg1, arg2,... .., argn`: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples: `generate mean_value = rowmean(1,2,3,4)`
`generate mean_income = rowmean(income18, income19, income20)`

- ❑ `rowmedian(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the median value of `arg1, arg2,... .., argn` for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ `arg1, arg2,... .., argn`: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples: `generate median_value = rowmedian(1,2,3,4)`
`generate median_income = rowmedian(income18, income19, income20)`

- ❑ `rowstd(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the standard deviation of `arg1, arg2,... .., argn` for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ `arg1, arg2,... .., argn`: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples: `generate std_value = rowstd(1,2,3,4)`
`generate std_income = rowstd(income18, income19, income20)`

- ❑ `rowtotal(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the sum of `arg1, arg2,... .., argn` for the given unit. Returns missing value if at least one of the arguments contains missing.
 - ❑ `arg1, arg2,... .., argn`: Numeric values or variables with numeric values
 - ❑ Output: Numeric values or missing
 - ❑ Examples: `generate sum_value = rowtotal(1,2,3,4)`
`generate sum_income = rowtotal(income18, income19, income20)`

- ❑ `rowmissing(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the number of missing values among `arg1, arg2,... .., argn` for the given unit.
 - ❑ `arg1, arg2,... .., argn`: Numeric or alphanumeric values, or variables with numeric or alphanumeric values
 - ❑ Output: Numeric values
 - ❑ Examples: `generate miss_value = rowmissing(1,2,3,4)`
`generate miss_income = rowmissing(income18, income19, income20)`
- ❑ `rowvalid(arg1, arg2,... .., argn)`
 - ❑ Description: Retrieves the number of valid values among `arg1, arg2,... .., argn` for the given unit.
 - ❑ `arg1, arg2,... .., argn`: Numeric or alphanumeric values, or variables with numeric or alphanumeric values
 - ❑ Output: Numeric values
 - ❑ Examples: `generate valid_value = rowvalid(1,2,3,4)`
`generate valid_income = rowvalid(income18, income19, income20)`
- ❑ `rowconcat(arg1, arg2,... .., argn)`
 - ❑ Description: Merges `arg1, arg2,... .., argn` for the given unit.
 - ❑ `arg1, arg2,... .., argn`: Numeric or alphanumeric values, or variables with numeric or alphanumeric values
 - ❑ Output: Numeric or alphanumeric values
 - ❑ Examples: `generate concat_value = rowconcat(1,2,3,4)`
`generate full_name = rowconcat('Jim ','Smith')`
`generate full_name = rowconcat(var1, var2)`

String functions

- ❑ `string(arg1)`
 - ❑ Description: Converts `arg1` into string format
 - ❑ `arg1`: Positive or negative values 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 \Rightarrow 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 ($\text{ibeta}() = 0$ if $\text{arg3} < 0$, $\text{ibeta}() = 1$ if $\text{arg3} > 1$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Positive or negative values (relevant values: $0 \leq \text{arg3} \leq 1$)
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Positive or negative values (relevant values: $0 \leq \text{arg3} \leq 1$)
 - ❑ Output: Values ≥ 0
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Positive or negative values (relevant values: $0 \leq \text{arg3} \leq 1$)
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ 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 ≥ 0
 - ❑ arg2: Positive or negative values (relevant values: $0 \leq \arg2 < \arg1$)
 - ❑ 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 ≥ 0
 - ❑ arg2: Positive or negative values (relevant values: $0 \leq \arg2 < \arg1$)
 - ❑ 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: Positive values
 - ❑ arg2: Positive or negative values (relevant values: $\arg2 \geq 0$)
 - ❑ 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`: Positive values
 - ❑ `arg2`: Positive or negative values (relevant values: `arg2 >= 0`)
 - ❑ Output: Values `>= 0`
 - ❑ 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`: Positive values
 - ❑ `arg2`: Positive or negative values (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`: Positive values
 - ❑ `arg2`: Values between 0 and 1
 - ❑ Output: Values `>= 0`
 - ❑ 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`: Positive values
 - ❑ `arg2`: Values between 0 and 1
 - ❑ Output: Values `>= 0`
 - ❑ 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`: Positive or negative values (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`: Positive or negative values (relevant values: `arg3 >= 0`)
 - ❑ Output: Values ≥ 0
 - ❑ 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`: Positive or negative values (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`: Positive values
 - ❑ `arg2`: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Positive or negative values
 - ❑ 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 ($t(arg1,arg2) = p \Rightarrow invt(arg1,p) = arg2$)
 - ❑ `arg1`: Positive values
 - ❑ `arg2`: Values between 0 and 1
 - ❑ Output: Positive or negative values
 - ❑ Examples:

- ❑ `invttail(arg1,arg2)`
 - ❑ Description: Returns a value from the inverse opposite cumulative Student's t-distribution with `arg1` degrees of freedom ($ttail(arg1,arg2) = p \Rightarrow invttail(arg1,p) = arg2$)
 - ❑ `arg1`: Positive values
 - ❑ `arg2`: Values between 0 and 1
 - ❑ Output: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Values between -1000 and 1000
 - ❑ `arg3`: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Values between -1000 and 1000
 - ❑ `arg3`: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Values between -1000 and 1000
 - ❑ `arg3`: Positive or negative values
 - ❑ 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: Positive or negative values
 - ❑ 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`: Positive values
 - ❑ `arg2`: Positive values
 - ❑ `arg3`: Positive or negative values (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`: Positive values
 - ❑ `arg2`: Positive values
 - ❑ `arg3`: Positive or negative values (relevant values: `arg3 >= 0`)
 - ❑ Output: Values `>= 0`
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Positive or negative values (relevant values: $arg3 \geq 0$)
 - ❑ 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 ($F(arg1,arg2,arg3) = p \Rightarrow invF(arg1,arg2,p) = arg3$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 1
 - ❑ Output: Values ≥ 0
 - ❑ 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 ($Ftail(arg1,arg2,arg3) = p \Rightarrow invFtail(arg1,arg2,p) = arg3$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 1
 - ❑ Output: Values ≥ 0
 - ❑ 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: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 10000
 - ❑ arg4: Positive or negative values (relevant values: $arg4 \geq 0$)
 - ❑ 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 ($\text{nFden}(\text{arg1},\text{arg2},0,\text{arg4}) = \text{Fden}(\text{arg1},\text{arg2},\text{arg4})$, $\text{nFden}() = 0$ if $\text{arg4} < 0$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 1000
 - ❑ arg4: Positive or negative values (relevant values: $\text{arg4} \geq 0$)
 - ❑ Output: Values ≥ 0
 - ❑ 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 ($\text{nFtail}() = 1$ if $\text{arg4} < 0$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 1000
 - ❑ arg4: Positive or negative values (relevant values: $\text{arg4} \geq 0$)
 - ❑ 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 ($\text{nFtail}(\text{arg1},\text{arg2},\text{arg3},\text{arg4}) = p \Rightarrow \text{invnFtail}(\text{arg1},\text{arg2},\text{arg3},p) = \text{arg4}$)
 - ❑ arg1: Positive values
 - ❑ arg2: Positive values
 - ❑ arg3: Values between 0 and 1000
 - ❑ arg4: Values between 0 and 1
 - ❑ Output: Values ≥ 0
 - ❑ Examples:
- ❑ **normal(arg1)**
 - ❑ Description: Returns a value from the cumulative standardized normal distribution
 - ❑ arg1: Positive or negative values
 - ❑ 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: Positive or negative values
 - ❑ arg2: Positive or negative values
 - ❑ arg3: Positive values
 - ❑ Output: Values ≥ 0
 - ❑ 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-2019-06-21-32) § 14 (access to information for statistical results and analyzes) section (5) states that "The duty of confidentiality pursuant to § 8 applies correspondingly to the person who has access to information". Such data can only be provided to researchers in approved research institutions or to public authorities. 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 (lower limit value) and the 1% lowest values are set to the 1-percentile (upper limit value). This only happens when displaying statistical results, based on the current population from which the statistics are calculated.

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 to a certain degree. Both types of statistics will typically be estimated too low. On the other hand, medians, quartiles and other percentiles will not be affected.

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.

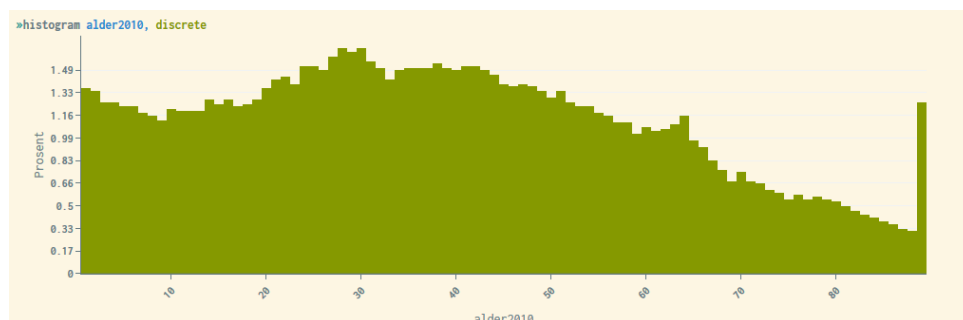
```
import BEFOLKNING_FOEDSELS_AAR_MND as faarmnd
generate faar = floor(faarmnd/100)
drop faarmnd
generate alder2010 = 2010 - faar
summarize alder2010
histogram alder2010, discrete
```

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

The result will look like this:

summarize alder2010

Variabel	mean	std	count	1%	25%	50%	75%	99%
alder2010	38.7467	22.681	255743	1	21	37	55	89



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, and graphical plots, and prevents the most extreme values from being visible.

Results produced through regression analyzes are not to be considered as personally identifiable information. Such analyzes therefore use the underlying non-winsorized data. Regression estimates will therefore not be affected by winsorization.

Measure 3: Randomized noise

All counts of the number of units in a dataset that are shown related to various operations, or statistical counts presented through commands such as **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

$$\mathcal{E}(p|n) = - \sum_{x=-5}^5 p(x|n) \log(p(x|n)) = -E(\log p(X|n))$$

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.)

$p(x)$	$n = 0$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$	$n = 8$	$n = 9$	$n \geq 10$
$x = -5$	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
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 $x \in \{-5, \dots, 5\}$ 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.*

$cf(n Y = y)$	$Y = 0$	$Y = 5$	$Y = 6$	$Y = 7$	$Y = 8$	$Y = 9$	$Y = 10$	$Y = 11$	$Y = 12$	$Y = 13$	$Y = 14$	$Y = 15$
$n = 0$	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

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

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:

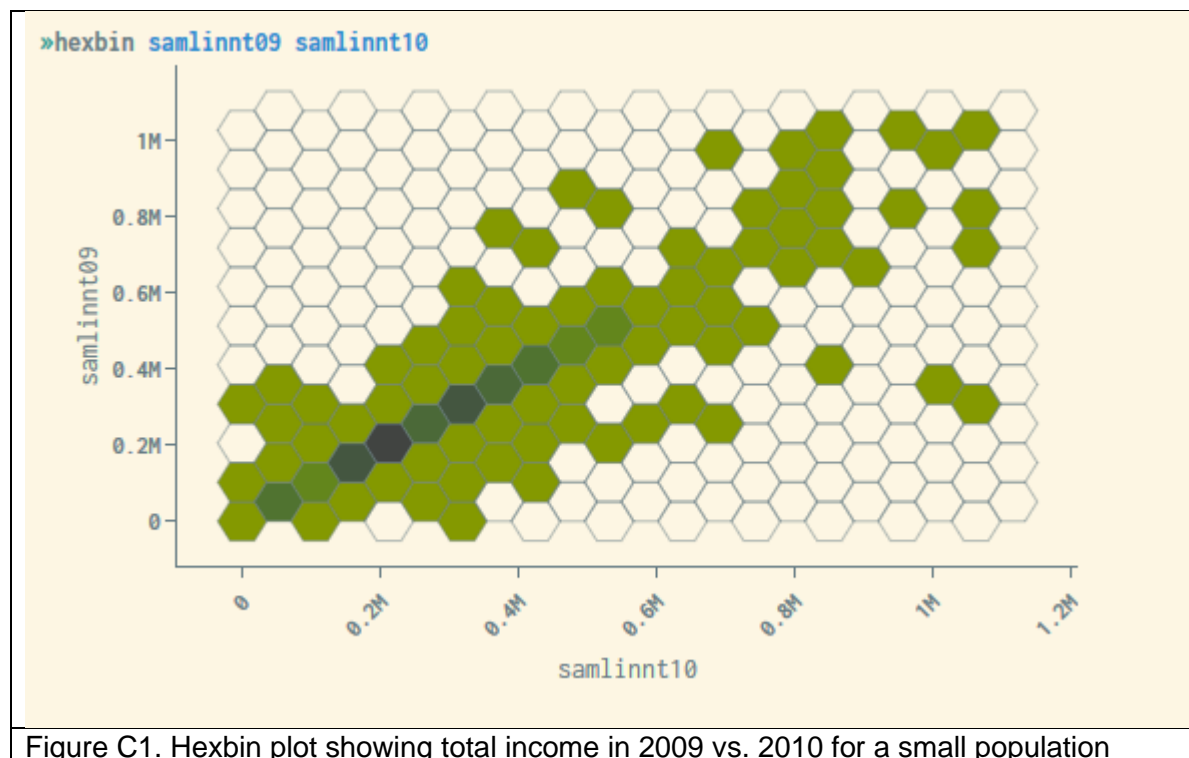


Figure C1. Hexbin plot showing total income in 2009 vs. 2010 for a small population

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.

Measure 5: Hiding tables with too many low values

Tables created by the [tabulate](#) command may in some cases contain many cells with low values for the number of units. This can be problematic as it makes it easier to indirectly identify individuals by studying combinations of values for the categorical variables that make up a table. Another problem with such tables is that the noise generation described under «Measure 3» gives a relatively high uncertainty for the relevant cell values (the percentage noise becomes relatively large with small numbers), so that the statistical usefulness of the table is low.

In microdata.no, a limit value of 50% is operated, i.e. tables where more than 50% of the cells contain frequency values lower than 5 will be stopped. In addition, an error message about this will be displayed.

It is possible to avoid the problem of tables being stopped due to many low cell values: By making coarser divisions for the categorical variables that make up the table, or by

increasing the size of the table population, you will be able to increase the number of units in each cell and thus fall below the 50% limit so that the table is approved and displayed.