A guide to CCDR Poverty Analysis code

Stata Code by Juan Manuel Monroy and Ercio Muñoz, interpreted by Renato Vargas

Table of contents

1 Master file

Filename: 01_Master.do

The master file explains the structured workflow:

1. **Setup and Preprocessing**: The initial segments define the working environment, set the necessary paths, and import necessary libraries or tools. The globals defined will help in driving the subsequent operations.
2. **Macro Inputs & Creation of Globals**: Fetches macro inputs from an Excel file and dynamically define some global variables. There are placeholder comments suggesting an alternate approach to fetch scenarios and years, which might have been used in earlier iterations or for debugging purposes.
3. **MFmod Microsimulation**: Run a microsimulation model to generate simulated weights and welfare aggregate measures.
4. **Indicator Generation**: Using the simulated data, compute various indicators. These are poverty rates, income measures, and other socio-economic indicators.
5. **Opening the Excel Scenario File**: Lastly, the main workflow ends by opening an Excel file to inspect or use the scenarios and results interactively.

2 Data Preparation file

Filename: 00_dataprep.do

2.1 Program setup

This sets the minimum required Stata version to run the code to 15.1. We also perform some housekeeping:

- `drop _all` removes all variables from the dataset in memory.
- `clear all` removes all data from memory.
- `set more off` turns off the 'more' feature, which pauses output at the screen every time it's filled.

```
version 15.1
drop _all
clear all
set more off
```

Checks if the current Stata user's name is `"wb527706"`. If it is, it sets the global macro `path` to the specified directory. This tailors the code to a specific user's directory structure.

```
if "`c(username)'"=="wb527706" {
    global path "C:\Users\WB527706\OneDrive - WBG\Madagascar"
}
```

The next lines set up a series of global macros with directory paths based on the earlier-defined `path`:

```
global HHsurvey    = "$path\data"
global inputs      = "$path/inputs"
global outputs     = "$path/outputs"
global MAGclimsim    "$path\MagClimSim.xlsx"
```

2.2 HH consumption aggregates and characteristics

This loads a dataset from the specified directory, which was previously defined in the global macro `HHsurvey`. The `clear` option ensures that any current data in memory is replaced. Note that we don't have access to that survey.

```
use "$HHsurvey\outputdata\ca_real_pl.dta", clear
```

Here, a new variable `rural` is created, which is equal to 1 if the area is "Rural" and 0 if the area is "Urbain".

```
gen     rural   = 1     if  area=="Rural"
replace rural   = 0     if  area=="Urbain"
```

The variable `pcer` is renamed to `pcc` (indicating per capita expenditure).

```
ren pcer pcc
```

A new variable `yhh` is created by multiplying `pcc` (per capita expenditure) by `hhsize` (household size).

```
gen yhh=pcc*hhsize
```

These lines rename the variables: `wgt` becomes `wgt0` and `wgt_adj` becomes `wgt`.

```
ren wgt wgt0
ren wgt_adj wgt
```

This line keeps only the listed variables in the dataset, dropping all others.

```
keep  hhid rural hhsize region pcc yhh wgt yhh //
      pline215 pline365 pline685 ubpl lbpl
```

A temporary file named `HHcha` is specified, and the current dataset is saved to that file.

```
tempfile HHcha
save `HHcha'
```

So far:

1. Sets up paths tailored to the user's directory structure.
2. Loads a specific dataset related to household consumption aggregates and characteristics.
3. Manipulates and keeps only select variables of interest.
4. Saves the modified data to a temporary file for later use.

2.3 Demographic characteristics

This line loads a dataset related to demographic characteristics from the directory specified by the global macro `HHsurvey`, replacing data in memory.

```
use "$HHsurvey\outputdata\S01_DEMO_01.dta", clear
```

Here, a new variable `hhgrap` is created, identical to the `cluster` variable. The dataset in memory is then merged with the `DéfinitionsQuatreMilieux.dta` dataset based on the `hhgrap` variable. The resulting `_merge` variable (indicating merge success for each observation) is dropped as customary.

```
gen hhgrap = cluster
merge n:1 hhgrap using "$HHsurvey\outputdata\DéfinitionsQuatreMilieux.dta"
drop _merge
```

The variable `MILIEU4` is renamed to `zone` and the variable `sex` is renamed to `gender`. A new binary variable `head` is created, which takes a value of 1 when `relhead` equals 1 (presumably indicating that the individual is the head of the household) and 0 otherwise. The variable `relhead` is renamed to `rel`.

```
ren MILIEU4 zone
ren sex gender
gen head = relhead == 1
ren relhead rel
```

Only the specified variables are retained, and all other variables are dropped.

```
keep hhid pid gender age head rel zone
```

The dataset in memory is saved to a temporary file named `demo`.

```
tempfile demo
save `demo'
```

2.4 Years of education

A new dataset related to education is loaded, replacing the current dataset in memory. Note that we don't have access to that one.

```
use "$HHsurvey\inputdata\EPM21\S02_EDUC.dta", clear
```

A new variable `yos` (probably standing for the availability of "years of schooling") is created based on the variable `q2_26`. However, its value is set to 0 wherever the variable `q2_01` is not equal to 1. Note that the consultant made a code annotation explicitly sayin that it is NOT "years of schooling". Then the variable `yos` is renamed to `educy`.

```
gen yos = q2_26
replace yos=0 if q2_01 != 1
rename yos educy
```

The data is sorted by `hhgrap` and `hhnum`. Then, a new variable `hhid` is created by concatenating `hhgrap` and `hhnum`, presumably to create a unique household ID. The variables `hhgrap` and `q2_0x` are then renamed to `cluster` and `pid`, respectively.

```
sort hhgrap hhnum
egen hhid = concat(hhgrap hhnum), format(%9.0g) punct(",")
rename hhgrap cluster
rename q2_0x pid
```

Only the specified variables are retained, and all others are dropped. The dataset in memory is saved to a temporary file named `edu`.

```
keep hhid pid educy
tempfile edu
save `edu'
```

To summarize, this code segment:

1. Loads a dataset related to demographic characteristics.
2. Merges it with another dataset to enhance the information.
3. Makes some modifications to the variable names and keeps relevant variables.
4. Saves the modified dataset to a temporary file.
5. Loads a new dataset related to years of education.
6. Makes transformations related to educational years and constructs a unique household ID.
7. Keeps the necessary variables and saves the modified dataset to another temporary file.

2.5 Non labor income

2.5.1 Pensions, rents, and dividends

This section of code is focused on calculating and aggregating non-labor income. First, we have pensions, rents, dividends, and lotteries. The script loads a dataset related to various revenues.

```
use "$HHsurvey\inputdata\EPM21\S05_REVE.dta", clear
```

The next series of lines, as before, creates a unique household ID by concatenating `hhgrap` and `hhnum`, and then renames some variables.

```
sort    hhgrap hhnum
egen    hhid =  concat(hhgrap hhnum), format(%9.0g) punct(",")
rename  hhgrap  cluster
rename  q5_0x   pid
```

For each type of income (pensions, rents, dividends, and occasional income like lottery winnings), the script creates a new variable only if another corresponding variable is set to 1 (probably indicating that the income source is relevant for that observation).

Survey: Pension annual basis: *retrate/veuvage/dinvalidité/alimentaire.*

```
gen nli_pen1=q5_02 if q5_01==1
gen nli_pen2=q5_04 if q5_03==1
gen nli_pen3=q5_06 if q5_05==1
gen nli_pen4=q5_08 if q5_07==1
```

Survey: Rents.

```
gen nli_rent=q5_10 if q5_09==1
```

Survey: Dividends.

```
gen nli_div=q5_12 if q5_11==1
```

Survey: Lottery winnings, inheritance, sale of property, etc.

```
gen nli_occ=q5_14 if q5_13==1
```

This line creates a new variable, `nli_prdo` (presumably non-labor income pensions, rents, dividents, other), that sums up the previously created income variables for each observation. The data is then collapsed (aggregated) by `hhid`, and saved into a temporary file.

```
egen nli_prdo=rsum(nli_pen1 nli_pen2 nli_pen3 nli_pen4 nli_rent nli_div nli_occ)
collapse (sum) nli_prdo ,by(hhid)
tempfile nli_prdo
save `nli_prdo'
```

2.5.2 Individual transfers

The next section focuses on individual transfers. It follows a similar structure: load data, create a unique household ID, create variables for different income sources (in this case, transfers), aggregate by household, and save to a temporary file.

The new variables (`nli_hhs1` and `nli_hhs2`) are generated based on conditions related to the frequency of the transfer (monthly, quarterly, etc.), and then aggregated into `nli_itrans`.

```
use "$HHsurvey\inputdata\EPM21\S13_TRAN_A.dta", clear

//ID at HH and IND definition

sort    hhgrap hhnum
egen    hhid =  concat(hhgrap hhnum), format(%9.0g) punct(",")
rename  hhgrap  cluster

gen      nli_hhs1=q13_18a*12     if q13_18b==1 & (q13_01==1 | q13_02==1)
replace  nli_hhs1=q13_18a*4      if q13_18b==2 & (q13_01==1 | q13_02==1)
replace  nli_hhs1=q13_18a*2      if q13_18b==3 & (q13_01==1 | q13_02==1)
replace  nli_hhs1=q13_18a        if q13_18b==4 & (q13_01==1 | q13_02==1)
replace  nli_hhs1=q13_18a        if q13_18b==5 & (q13_01==1 | q13_02==1)

gen     nli_hhs2=q13_21a*12      if q13_21b==1 & (q13_01==1 | q13_02==1)
replace  nli_hhs2=q13_21a*4      if q13_21b==2 & (q13_01==1 | q13_02==1)
replace  nli_hhs2=q13_21a*2      if q13_21b==3 & (q13_01==1 | q13_02==1)
replace  nli_hhs2=q13_21a        if q13_21b==4 & (q13_01==1 | q13_02==1)
replace  nli_hhs2=q13_21a        if q13_21b==5 & (q13_01==1 | q13_02==1)

egen nli_itrans=rsum(nli_hhs1 nli_hhs2)
collapse (sum) nli_itrans ,by(hhid)
```

Saved temporarily in:

```
tempfile nli_itrans
save `nli_itrans'
```

2.5.3 Public Transfers

This section deals with public transfers. The data handling is quite similar to the previous sections. The income from public transfers is calculated based on the frequency, aggregated by household, and saved into a temporary file `nli_gov`.

```
u "$HHsurvey\inputdata\EPM21\S15_FILE.dta", clear

//ID at HH and IND definition
sort    hhgrap hhnum
egen    hhid =  concat(hhgrap hhnum), format(%9.0g) punct(",")
rename  hhgrap  cluster

g        nli_gov=q15_10a*12     if q15_10b==1 & (q15_02==1 )
replace  nli_gov=q15_10a*4      if q15_10b==2 & (q15_02==1 )
replace  nli_gov=q15_10a*2      if q15_10b==3 & (q15_02==1 )
replace  nli_gov=q15_10a        if q15_10b==4 & (q15_02==1 )
replace  nli_gov=q15_10a        if q15_10b==5 & (q15_02==1 )

collapse (sum) nli_gov ,by(hhid)
tempfile nli_gov
save `nli_gov'
```

The last piece of code merges the aggregated data from the three previous sections

```
u  `nli_prdo', clear
merge 1:1 hhid using `nli_itrans' , nogen
merge 1:1 hhid using `nli_gov' , nogen
```

Here, the script loads the data related to pensions, rents, dividends, etc., and then merges it with the data from individual and public transfers using `hhid` as the key variable.

A new variable, `nli_hh`, is created, which is the sum of all non-labor income sources for each household.

```
egen nli_hh=rsum(nli_prdo nli_gov nli_itrans)
```

Finally, this aggregated data is saved into another temporary file.

```
tempfile nli
save `nli'
```

To summarize, this segment of the code:

1. Loads datasets related to various non-labor income sources: pensions, rents, dividends, individual transfers, and public transfers.
2. Processes each data source to calculate the total income for each type.
3. Merges the data from the various sources and aggregates the total non-labor income by household.
4. Saves the resulting dataset for future use.

2.6 Labor market variables

A dataset related to employment is loaded into Stata.

```
u "$HHsurvey\inputdata\EPM21\S04_EMPL_AI.dta", clear
```

The subsequent lines create a unique household ID by concatenating `hhgrap` and `hhnum`, and rename some variables. This pattern has been repeated from the previous chunks of code. The data is then merged with the non-labor income dataset (created in the previous section):

```
sort    hhgrap hhnum
egen    hhid =  concat(hhgrap hhnum), format(%9.0g) punct(",")
rename  hhgrap  cluster
rename  q4a_xx  pid

// Merge non-labor income part
merge n:1 hhid using `nli' , nogen
```

2.6.1 Labor Force Status:

The variable `lstatus` is created based on different responses to the employment-related questions. This variable categorizes individuals into employed, unemployed and seeking work, unemployed but not seeking work, or outside the labor force (OLF).

```
//employed
g lstatus = 1 if  q4a_01==1 | q4a_02==1 | q4a_03==1 | //
                  q4a_04==1 | q4a_09<4

// Unemployed - available and searching
replace lstatus = 2  if q4a_96==1

// Unemployed - available, but not searching
replace lstatus = 3  if q4a_96==2

//OLF
replace lstatus = 4  if missing(lstatus)

lab var lstatus "Labor force status"
lab def lstatus_lab 1 "employed" 2 "unemployed - seeking" //
                    3 "unemployed - not-seeking" 4 "OLF"
lab val lstatus lstatus_lab
```

2.6.2 Employed Population and Salaried Status:

The code identifies which individuals are employed (`employed` variable) and then aggregates this at the household level. It then determines if employed individuals are salaried or self-employed.

```
gen employed = lstatus==1
bysort hhid : egen employed_hh=max(employed)

gen salaried = .
replace salaried = 1 if q4a_24 ==1

// self-employed if employed and salary is missing.
replace salaried = 0 if mi(salaried) & employed==1
lab var salaried "Employment status"
lab def salaried 1 "paid employee" 0 "self-employed"
lab val salaried salaried
```

2.6.3 Labor Income Computations:

The subsequent part of the code calculates labor income (`lab_pri`) based on different frequencies of payment (daily, weekly, monthly, etc.). Similar calculations are done for bonuses (`lab_ikb`), other in-kind incomes (`lab_iko`, `lab_ikf`), and incomes from a second job (`lab_sec`, `lab_sik`, `lab_sit`, `lab_sif`). The total labor income is then aggregated into `lab_tot`.

Survey: *What was [NAME]'s salary for this job (for the time period considered)?*

```
g        lab_pri= q4a_46a*365               if q4a_46b==1
replace  lab_pri= q4a_46a*(365/7)*(1)       if q4a_46b==2
replace  lab_pri= q4a_46a*12                if q4a_46b==3
replace  lab_pri= q4a_46a                   if q4a_46b==4
```

Survey: In-kind bonus. *A combien évaluez-vous les primes (uniquement ceux qui ne sont pas inclus dans le salaire)?*

```
g        lab_ikb= q4a_48a*365               if q4a_48b==1
replace  lab_ikb= q4a_48a*(365/7)*(1)       if q4a_48b==2
replace  lab_ikb= q4a_48a*12                if q4a_48b==3
replace  lab_ikb= q4a_48a                   if q4a_48b==4
```

Survey: Other inkind. *A combien évaluez-vous ces avantages (uniquement ceux qui ne sont pas inclus dans le salaire)?*

```
g        lab_iko= q4a_50a*365                if q4a_50b==1
replace  lab_iko= q4a_50a*(365/7)*(1)        if q4a_50b==2
replace  lab_iko= q4a_50a*12                 if q4a_50b==3
replace  lab_iko= q4a_50a                    if q4a_50b==4
```

Survey: Food from the job.

```
g        lab_ikf= q4a_52a*365                if q4a_52b==1
replace  lab_ikf= q4a_52a*(365/7)*(1)        if q4a_52b==2
replace  lab_ikf= q4a_52a*12                       if q4a_52b==3
replace  lab_ikf= q4a_52a                    if q4a_52b==4
```

Survey: Second activity labor income.

```
g        lab_sec= q4a_62a*365                if q4a_62b==1
replace  lab_sec= q4a_62a*(365/7)*(1)        if q4a_62b==2
replace  lab_sec= q4a_62a*12                 if q4a_62b==3
replace  lab_sec= q4a_62a                    if q4a_62b==4
```

Survey: In-kind second bonus.

```
g        lab_sik= q4a_64a*365                if q4a_64b==1
replace  lab_sik= q4a_64a*(365/7)*(1)        if q4a_64b==2
replace  lab_sik= q4a_64a*12                 if q4a_64b==3
replace  lab_sik= q4a_64a                    if q4a_64b==4
```

Survey: Other in-kind, such as transport.

```
g        lab_sit= q4a_66a*365                if q4a_66b==1
replace  lab_sit= q4a_66a*(365/7)*(1)        if q4a_66b==2
replace  lab_sit= q4a_66a*12                 if q4a_66b==3
replace  lab_sit= q4a_66a                    if q4a_66b==4
```

Survey: In-kind food.

```
g        lab_sif= q4a_68a*365                if q4a_68b==1
replace  lab_sif= q4a_68a*(365/7)*(1)        if q4a_68b==2
replace  lab_sif= q4a_68a*12                 if q4a_68b==3
replace  lab_sif= q4a_68a                    if q4a_68b==4
```

All labor aggregated into `lab_tot`.

```
egen lab_tot=rsum(lab_pri lab_ikb lab_iko lab_ikf //
                  lab_sec lab_sik lab_sit lab_sif)

// Code turned off, presumably used with already processed surveys.
*egen lab_tot=rsum(lab_pri  lab_sec) * <1>

// All labor income expressed in thousands.
replace lab_tot=lab_tot*1000 * <2>
```

2.6.4 Treatment of Missing Data and Outliers:

The script examines missing labor income values among employed individuals and identifies outliers. Observations with labor incomes more than 5 standard deviations away from the mean are marked as outliers. There's an annotation that explains that missings represent 10% of those employed. Mainly primary sector and household workers.

```
//Outliers sd>5
sum lab_tot if employed==1 & lab_tot>0
gen d = lab_tot/r(sd)
gen outlier = 1 if d>5
replace outlier = 1 if employed==1 & lab_tot==0
replace outlier = 0 if outlier==.

// Missing
gen missings = 1     if employed==1 & lab_tot==0
replace missings = 0 if missings==.
```

2.6.5 Merge with Demographic Characteristics:

The dataset is merged with household characteristics (`HHcha`), demographics (`demo`), and education (`edu`) datasets.

```
merge n:1 hhid using `HHcha'
keep if _merge==3
merge 1:1 hhid pid using `demo' , nogen
merge 1:1 hhid pid using `edu' , gen(edu)
```

2.6.6 Sector Categorization:

The `sector` variable categorizes individuals into primary, secondary, or tertiary sectors based on their reported economic activity (`q4a_230`).

```
***Sector
gen sector=1 if q4a_230=="A" | q4a_230=="B" | q4a_230=="C" | q4a_230=="0" | //
                q4a_230=="1" | q4a_230=="2" | q4a_230=="3" | q4a_230=="4" | //
                q4a_230=="5" | q4a_230=="6" | q4a_230=="7" | q4a_230=="8" | //
                q4a_230=="9"

replace sector=2 if  q4a_230=="D"| q4a_230=="E"| q4a_230=="F"

replace sector=3 if  q4a_230=="G" | q4a_230=="H" | q4a_230=="I" |   //
                     q4a_230=="J" |q4a_230=="K"  | q4a_230=="L" |   //
                     q4a_230=="M" | q4a_230=="N" | q4a_230=="O" |   //
                     q4a_230=="P" | q4a_230=="Q" | q4a_230=="R" |   //
                     q4a_230=="S" | q4a_230=="T" | q4a_230=="U"
```

There's also imputation logic in place for missing sector information, where an employed individual without a specified sector is assigned the sector of the household head.

```
gen aux = sector if head==1
bysort hhid : egen sectorhh=max(aux)
replace sector=sectorhh if sector==. & employed==1

replace sector=2 if sector==. & employed==1 //to check
replace sector=. if lstatus==2 | lstatus==3
lab var sector "Economic activity sector"

lab def sector_lab 0 "unemployed" 1 "primary (Agr)" //
                   2 "secondary (Ind)" 3 "tertiary (Ser)"
lab val sector sector_lab
replace sector = . if lstatus==4 //No sector for OLF
```

2.7 Model for Predicting Labor Income:

The code calculates squared values for education (`educy2`) and age (`age2`) and creates a binary variable for males (`male`). A natural logarithm of total labor income is then calculated (`lnlab`).

```
clonevar industry = sector

gen educy2   = educy * educy
gen age2     = age*age
gen male     = 1 if gender==1
replace male = 0 if gender==2
```

Subsequently, a regression model (`reg`) predicts the logarithm of labor income (`lnlab`) based on various factors, but only for employed individuals who are neither outliers nor missing income values.

```
gen lnlab    = ln(lab_tot)
reg lnlab age gender i.educy age2 i.region i.q4a_24 i.sector [iw=wgt]  //
          if employed==1 & outlier==0 & missings==0
```

Using this model, predicted income values (`remp2` and `temp2`) are generated for employed outliers or those missing labor income data.

```
predict remp2 if employed==1 & outlier==1 | missings==1 , xb
predict temp2 if employed==1 & outlier==1 | missings==1
```

Finally, these predicted values are transformed to actual income scale (`simuli`) using the exponential function, and negative predictions are set to zero.

```
gen simuli = .
replace simuli = exp(temp2)
replace simuli = 0 if simuli<0
```

In summary, this section of the code:

1. Loads and processes labor market data.
2. Calculates labor force status and categorizes people based on their employment and income details.
3. Computes and aggregates labor income for different sources.
4. Treats missing data and outliers for labor income.
5. Merges the processed labor data with demographic and education datasets.
6. Categorizes individuals based on their sector of economic activity.
7. Predicts labor income for those missing data using regression, and adjusts the predictions as needed.

2.8 Shares and total income to the model

This section is focused on computing shares and total income in preparation for a model.

Here, for those individuals who are employed and are either considered outliers or have missing labor income data (`lab_tot`), their labor income is replaced with the simulated value (`simuli`) generated in the previous section.

```
replace lab_tot=simuli if  employed==1 & (outlier==1 | missings==1 )
```

This line calculates the total labor income at the household level, aggregating all individual incomes within each household.

```
bysort hhid : egen lab_hh=sum(lab_tot)
```

The total income for each household is computed by summing the total labor income (`lab_hh`) and non-labor income (`nli_hh`).

```
egen    income_hh=rsum(lab_hh nli_hh)
```

Then we calculate the share of labor and non-labor income out of the total income for each household.

```
gen       s_lab    = lab_hh/income_hh
gen       s_nli    = nli_hh/income_hh
```

And compute the natural logarithms of total income (`income_hh`) and household consumption (`yhh`).

```
gen     lny      =ln(income_hh)
gen     lnc      =ln(yhh)
```

The marginal propensity to consume (`mpc`) is calculated as the ratio of household consumption (`yhh`) to total income (`income_hh`).

```
g       mpc      = yhh/income_hh
```

The subsequent lines compute shares and derive other variables related to labor income and consumption:

- `share` captures the individual's share of labor income in the household's total labor income, but only for those employed.
- `ylb` gives the portion of household consumption funded by labor income, while `ynl` represents the portion of household consumption sourced from non-labor income.
- `ylbi` represents the individual's portion of household consumption funded by their labor income, again limited to the employed.

```
gen share = lab_tot/lab_hh  if employed==1

// HH Consumption from labor
gen ylb = yhh*s_lab
lab var ylb   //
    "Household consumption from labor income -nominal (Ariary/hh/year)"

// HH Consumption from non-labor
gen ynl = yhh*(1-s_lab)
lab var ynl  //
    "Household consumption from non-labor income - nominal (Ariary/hh/year)"

g ylbi=ylb*share if employed==1
```

The dataset is then pruned to keep only select variables, and saved to a temporary file.

```
keep hhid pid industry yhh ylb ynl ylbi salaried
tempfile labor
save `labor'
```

2.9 Final datasets

The main household characteristics dataset (`HHcha`) is loaded again. It then gets merged sequentially with the demographic (`demo`), education (`edu`), and labor (`labor`) datasets.

Once all necessary merges are done, the generation variables (`dem`, `edu`, `labo`) are dropped. The resulting final dataset, which combines household characteristics, demographics, education, and labor variables, is then saved as `MAG_assigned.dta`.

```
use `HHcha' , clear
merge 1:n hhid using `demo' , gen(dem)
keep if dem==3
merge 1:1 hhid pid using `edu' , gen(edu)
merge 1:1 hhid pid using `labor' , gen(labo)
```

```
drop dem edu labo
save  "$HHsurvey\MAG_assigned.dta", replace
```

In summary, this segment:

1. Adjusts individual labor income values based on simulations for outliers and missing data.
2. Computes various shares related to labor and non-labor incomes.
3. Creates the logarithm of incomes and computes the marginal propensity to consume.
4. Derives individual and household-level variables for consumption funded by labor and non-labor income.
5. Merges several datasets to compile a comprehensive dataset containing household characteristics, demographics, education, and labor information.
6. Saves the final dataset for subsequent analysis.

3 Creation of inputs for microsimulation

File: 02_inputs.do

3.1 Import population data to be used in the creation of globals

We first load a dataset specified by the global macro `popdata`, but only for the observations where the variable `country` matches the value specified by the global macro `country`. Here, we are aggregating the data by `Variant` and `country` using `gcollapse` (an enhanced version of Stata's `collapse` command). The `(sum)` function sums up all the variables starting with `yf` and `ym`. And we reshape the dataset from wide to long format based on the `yf` and `ym` variables. The combined identifiers are `country` and `Variant`, while the variable `year` is created to represent the time dimension.

```
use "$popdata" if country=="$country", clear
qui gcollapse (sum) yf* ym*, by(Variant country)
qui reshape long yf ym, i(country Variant) j(year)
```

Here, we generate a new variable named `pop`, which is the sum of the `yf` (likely female population) and `ym` (likely male population) variables, essentially getting the total population. The individual gender-based population variables (`yf` and `ym`) are then dropped, as we now have the combined `pop` variable. The variable `country` is also dropped from the dataset. The dataset is then saved into a temporary file named `pop`.

```
qui g pop = yf+ym
qui drop yf ym
drop country
tempfile pop
save `pop'
```

The `year` variable is summarized to get its basic statistics. Two local macros are defined: `minyear`, which is assigned the value of the global macro `surveyyear`, and `maxyear`, which is assigned the maximum value of the `year` variable.

```
qui sum year
local minyear = $surveyyear
local maxyear = r(max)
```

A local macro `v` is initialized to 1. The subsequent loop then runs over several population projection variants. For each variant and for every year between `minyear` and `maxyear`, the total population (`pop`) for that year (`pop_t`) and the base year (`pop_base`) is calculated. The population growth rate is then computed by dividing `pop_t` by `pop_base` and saved into a global macro with a naming convention based on the year and variant. The counter `v` increments for each variant.

```
local v = 1
foreach variant in "Medium" "Low" "High" "Constant-fertility" //
                   "Instant-replacement" "Momentum"  //
                   "Instant-replacement-zero-migration" //
                   "Constant-mortality" "No-change" {
  forvalues t = `minyear'/`maxyear' {
        /* Population growth */
        qui sum pop if year==`t' & Variant=="`variant'"
        local pop_t = r(sum)
        qui sum pop  if year==`minyear' & Variant=="`variant'"
        local pop_base = r(sum)
        global pop_growth_t`t'_v`v' = `pop_t'/`pop_base'
    }
    local v = `v'+1
}
```

Lastly, the code imports an Excel file specified by the macro `scenario_file` from a sheet named `elas_rep`, clearing the existing data in memory. These are three cells with the elasticities for Agriculture, Manufacturing (Industry), and Services. We put these in a matrix `E` from the dataset, variable `A` (first column).

```
import excel "${scenario_file}", sheet(elas_rep) clear
mkmat A  , mat(E)
```

In summary, this chunk of code:

1. Imports a dataset and reshapes it to long format based on population variables.
2. Aggregates the data to get total populations by year and variant.
3. Computes population growth rates for different scenarios across years.
4. Imports another dataset from Excel related to some scenarios.
5. Transforms part of the dataset into a matrix for further analysis.

3.2 Import data from the excel file received from macroeconomic modelers

A local macro `i` is initialized to 0. A loop is then started, iterating over a global macro called `scenarios`, which presumably contains a list of sheet names from the Excel file to import. For each sheet name in `scenarios`, the code imports data from the Excel file specified by the `scenario_file` macro. The data is imported from the cell range `A3:N63`. This command renames the variables in the dataset using the variable names stored in the global macro `$vars`. The counter `i` is incremented by 1, and a new variable `scenid` is created in the dataset, which is given the current value of `i`. This acts as an identifier for each scenario. The dataset is saved into a temporary file named based on the current value of `i` (like `scen1`, `scen2`, etc.). The loop then moves on to the next sheet name in `scenarios`.

```
foreach x of global scenarios {
    qui import excel "${scenario_file}", sheet("`x'") cellrange(A3:N63) clear
    rename (*) ($vars)
    local i = `i'+1
    g scenid = `i'
    tempfile scen`i'
    qui save `scen`i''
}
```

The dataset from the first scenario (`scen1`) is loaded into memory. A new local macro `num` is defined, which calculates the number of items in the `scenarios` global macro. Then, a loop starts from 2 up to the value of `num`. For each iteration, the dataset for the scenario corresponding to the current value of `i` is appended to the dataset in memory.

```
use `scen1', clear
local num : list sizeof global(scenarios)
forvalues i = 2/`num' {
    append using `scen`i''
}
```

Any observations with a missing value for the `year` variable are dropped. A new variable `rwage` is created, which represents the real wage by dividing the `wage` variable by the `cpi` (Consumer Price Index) variable. This adjusts the wage for inflation.

```
cap drop if year==.
qui g rwage = wage/cpi
```

The variable `Variant` is initialized with empty strings. Then, a loop iterates over another global macro named `variants`. For each iteration, the value of `Variant` is replaced with the current item from `variants` where the `scenid` matches the current value of `i`. This assigns the appropriate variant name to each observation based on its `scenid`.

```
g Variant = ""
local i = 1
    foreach variant of global variants {
        replace Variant = "`variant'" if scenid==`i'
        local i = `i' + 1
    }
```

The dataset is then merged with the one saved in the temporary file `pop` based on the `year` and `Variant` variables. Observations that didn't find a match in the `pop` dataset are dropped, and the `_merge` variable (created by the `merge` command) is also dropped. The dataset is sorted by `scenid` and then `year`.

```
merge m:1 year Variant using `pop'
drop if _merge==2
drop _merge
sort scenid year
```

Then a new variable `consumption_pc` is created, which calculates the per capita consumption. This is done by dividing the `consumption` variable by 100 times the `pop` variable.

```
g consumption_pc = consumption/(pop*10e2)
}
```

In summary:

1. We import various scenarios from an Excel file and save each as a temporary Stata dataset.
2. Combines all the scenarios into one dataset.
3. Processes the dataset by calculating real wages, assigning scenario names, merging with population data, and calculating per capita consumption values.

3.3 Creating globals by year/scenario to be used later in the microsimulation

These lines determine the range of years in the dataset. The minimum year is set to a predefined global macro (`$surveyyear`), and the maximum year is the maximum year observed in the dataset.

```
qui sum year
local minyear = $surveyyear
local maxyear = r(max)
```

This line identifies all the unique values of `scenid` in the dataset and stores them in a local macro called `scenarios`.

```
levelsof scenid, loc(scenarios)
```

The code then starts a loop, iterating over each unique scenario in the dataset. Inside the loop for each scenario, another loop starts which iterates over the years from the minimum to the maximum year. Within these nested loops, a series of commands calculate global growth rates and other indicators for various economic variables (GDP, consumption, employment, value added in different sectors, wages, etc.). The calculations are typically in the form:

```
qui sum [variable] if year==`t' & scenid==`sc'
local [variable]_t = r(sum)
qui sum [variable] if year==`minyear' & scenid==`sc'
local [variable]_base = r(sum)
global [variable]_growth_t`t'_sc`sc' = `variable_t'/`variable_base'
```

For each combination of year and scenario, these commands:

1. Sum up the variable's value for the current year (`t`) and scenario (`sc`).
2. Save this sum in a local macro.
3. Sum up the variable's value for the base year and the current scenario.
4. Save this sum in another local macro.
5. Calculate the growth rate by dividing the current year's sum by the base year's sum.
6. Save the growth rate in a global macro.

These steps are repeated for several economic indicators. At the end of the nested loops, the Stata dataset will contain a series of global macros capturing the growth rates and other economic indicators for each year and scenario.

```
foreach sc of numlist `scenarios' {
    forvalues t = `minyear'/`maxyear' {

        /* GDP */

        qui sum gdp if year==`t' & scenid==`sc'
        local gdp_t = r(sum)
        qui sum gdp  if year==`minyear' & scenid==`sc'
        local gdp_base = r(sum)
        global gdp_growth_t`t'_sc`sc' = `gdp_t'/`gdp_base'

        /* Consumption */

        qui sum consumption if year==`t' & scenid==`sc'
        local consumption_t = r(sum)
        qui sum consumption  if year==`minyear' & scenid==`sc'
        local consumption_base = r(sum)
        global c_growth_t`t'_sc`sc' = `consumption_t'/`consumption_base'

        /* Consumption per capita */

        qui sum consumption_pc if year==`t' & scenid==`sc'
        local consumption_t = r(sum)
        qui sum consumption_pc   if year==`minyear' & scenid==`sc'
        local consumption_base = r(sum)
        global pcc_growth_t`t'_sc`sc' = `consumption_t'/`consumption_base'

        /* Remittances */

        qui sum remittances if year==`t' & scenid==`sc'
        local remittances_t = r(sum)
        qui sum remittances  if year==`minyear' & scenid==`sc'
        local remittances_base = r(sum)
        global remitt_growth_t`t'_sc`sc' = `remittances_t'/`remittances_base'

        /* Employment */

        qui sum employment if year==`t' & scenid==`sc'
        local employment_t = r(sum)
        qui sum employment   if year==`minyear' & scenid==`sc'
        local employment_base = r(sum)
        global emp_growth_t`t'_sc`sc' = `employment_t'/`employment_base'
```

```
/* Employment rate */

qui sum employment    if year==`t' & scenid==`sc'
local employment_t_n = r(sum)
qui sum wa_population if year==`t' & scenid==`sc'
local employment_t_d = r(sum)
global emp_rate_t`t'_sc`sc' = `employment_t_n'/`employment_t_d'

/* Working age population */

qui sum wa_population if year==`t' & scenid==`sc'
local employment_t = r(sum)
qui sum wa_population    if year==`minyear' & scenid==`sc'
local employment_base = r(sum)
global wapop_growth_t`t'_sc`sc' = `employment_t'/`employment_base'

/* Value added */

qui sum agriculture_va if year==`t' & scenid==`sc'
local N_i = r(sum)
qui sum agriculture_va if year==`minyear' & scenid==`sc'
local N_base = r(sum)
global growth_agriculture_t`t'_sc`sc' = `N_i'/`N_base'

qui sum industry_va if year==`t' & scenid==`sc'
local N_i = r(sum)
qui sum industry_va if year==`minyear' & scenid==`sc'
local N_base = r(sum)
global growth_industry_t`t'_sc`sc' = `N_i'/`N_base'

qui sum services_va if year==`t' & scenid==`sc'
local N_i = r(sum)
qui sum services_va if year==`minyear' & scenid==`sc'
local N_base = r(sum)
global growth_services_t`t'_sc`sc' = `N_i'/`N_base'

/* Nominal Wages */

qui sum wage        if year==`t' & scenid==`sc'
local N_i = r(sum)
qui sum wage        if year==`minyear' & scenid==`sc'
local N_base = r(sum)
```

```
        global growth_wage_t`t'_sc`sc' = `N_i'/`N_base'

        /* Real Wages */

        qui sum rwage       if year==`t' & scenid==`sc'
        local N_i = r(sum)
        qui sum rwage       if year==`minyear' & scenid==`sc'
        local N_base = r(sum)
        global growth_rwage_t`t'_sc`sc' = `N_i'/`N_base'

    }
}
}
```

After finishing the calculations for all scenarios and years, the code then lists all global macros, giving you an overview of the generated variables. The block of Stata commands then ends.

In summary, this code processes the dataset to generate a comprehensive set of global macros containing values for various economic indicators, broken down by year and scenario. These global macros are then used in subsequent analyses or microsimulations.

```
global c:all globals
macro list c
```

4 Reweighting using education projections, UN age projections and sectoral shares from MFMOD

File: 03_MFmod.do

This code chunk pertains to a reweighting procedure using education projections, United Nations (UN) age projections, and sectoral shares from a macroeconomic model named "MFMOD".

These commands clear the first timer and then start it. Stata's `timer` functionality allows for tracking the time taken for specific operations, useful for benchmarking performance.

```
timer clear 1
timer on 1
```

The local macro `i` is initialized to 1. From the comments, it seems `i` will be used as a counter or reference number for the scenarios. Each scenario will likely be related to specific combinations of variants. The file is a massive loop.

```
local i = 1
```

Nested loops begin here. The outer loop iterates over each variant specified in a global macro called `variants`. The inner loop, for each variant, iterates over years specified in a global macro called `yearsto`.

```
foreach variant of global variants {
    foreach t of global yearsto {
```

This conditional checks if the current scenario (tracked by `i`) is the first one and the year is 2021, or if the year is after 2021. If either of these conditions is true, the code inside the block will execute.

```
if ( (`i'==1 & `t'==2021) | (`t'>2021) ) {
```

This line opens a dataset named `MAG_assigned.dta` from a specified path which we created previously. The `clear` option ensures that the current data in memory is replaced by the new dataset.

```
use "$path/data/MAG_assigned.dta", clear
```

This line renames the variable `ylbi` to `yflab`. A new variable, `pcnli`, is created. It is the result of dividing the `ynl` variable by `hhsize`, which represents the per capita non-labor income in the household.

```
ren (ylbi) (yflab)
g pcnli = ynl/hhsize
```

4.1 Define key variables and globals/rates

This command recodes the `educy` variable, which represents years of education. It bins the years into three categories: "0-3", "3-7", and "8+". The newly created categorical variable is named `calif`.

```
recode educy (0/3 = 1 "0-3") (3/7 = 2 "3-7") (7/11 = 3 "8+"), g(calif)
```

A matrix `industry_growth` is created, which is based on a combination of previously defined global variables and the matrix `E`, which you can recall we created before with the elasticities. The `industry_growth` matrix represents growth in different industry sectors.

```
mat industry_growth = ///
    1+(${growth_agriculture_t`t'_sc`i'}-1)*E[1,1] \ ///
    1+(   ${growth_industry_t`t'_sc`i'}-1)*E[2,1] \ ///
    1+(   ${growth_services_t`t'_sc`i'}-1)*E[3,1]
matlist industry_growth
```

A global variable `employment` is created based on employment rates for a particular year and scenario. This variable captures the growth in employment rate from the survey year to the target year.

```
global employment = (${emp_rate_t`t'_sc`i'})  / (${emp_rate_t${surveyyear}_sc`i'})
```

4.2 Main reweighting command

This is the most important part of the code. The reweighting of the dataset.

The command `mfmodms_reweight` appears to be a custom command (possibly from a user-written package or an in-house module) that performs the reweighting based on several constraints. The variables and parameters to be considered in the reweighting process are specified.

```
mfmodms_reweight, age(age) edu(calif) gender(gender) hhsize(hhsize)       //
                  id(hhid) iw(wgt) country("$country") iyear($surveyyear) //
                  tyear(`t') generate(wgtsim) industry(industry)          //
                  growth(industry_growth) match(HH) popdata("$popdata")   //
                  employment($employment) variant(`variant')
```

4.3 Wage bill at baseline

The command quietly calculates the weighted sum of the `yflab` variable, which represents the family labor income. The result is stored as a scalar named `Y0`.

```
qui sum yflab [aw=wgt]
scalar Y0 = r(mean)
```

4.4 Wage bill at simulated year/scenario

This portion of the code takes the user through calculating the simulated wage bill for different scenarios and years and then modifies the distribution of labor income accordingly.

The comments suggest adjusting the `yflab` (probably representing labor income or wage) in such a way that it grows consistently with Value Added (VA), ensuring the share of the wage bill on VA remains constant. This is done for three sectors: agriculture, industry, and services.

Here, the wage bill for the simulated year and scenario is computed and stored in a scalar `Y1`.

```
/* "wage" by sector grow in a way consistent with VA,
    such that the share of the wage bill on VA is constant */
qui replace yflab = yflab*(${growth_agriculture_t`t'_sc`i'} /    //
                    industry_growth[1,1]) if industry==1
qui replace yflab = yflab*(${growth_industry_t`t'_sc`i'}    /    //
                    industry_growth[2,1]) if industry==2
qui replace yflab = yflab*(${growth_services_t`t'_sc`i'}    /    //
                    industry_growth[3,1]) if industry==3

// Labor income (wage bill) at simulated year and scenario
    qui sum yflab [aw=wgtsim`t']
    scalar Y1 = r(mean)
```

4.5 Re-center and simulated welfare aggregate

This section aims to adjust the labor income, aggregate it to the household level, then adjust it further with non-labor income and consumption projections. The comment suggests re-centering the labor income distribution, ensuring that only the distribution of labor income is modified. Labor income is aggregated at the household level, producing a new variable named `ysim_t'_sci'`, which seems to represent simulated labor income for a specific year (`t`) and scenario (`i`). The simulated labor income is adjusted to become per capita values. To the simulated labor income per capita, non-labor income (denoted by `pcnli`) is added.

```
/* Re-center, so we only modify the distribution of "labor income" */
qui replace yflab = yflab * (Y0/Y1)

/* Aggregate to household again */
bys hhid: egen ysim_`t'_sc`i' = sum(yflab)
```

```
/* Divide by hhsize to make per capita again */
qui replace ysim_`t'_sc`i'=(ysim_`t'_sc`i')/hhsize

/* Adding back "non-labor" income */
qui replace ysim_`t'_sc`i'=ysim_`t'_sc`i' + pcnli
```

4.6 Adjusting the consumption according to the projection

This portion adjusts the simulated labor income based on the growth in per capita consumption from projections. The conditional statement checks if it's the base year (2021) and the first scenario. If it is, the dataset is saved anew. If not, the dataset is appended with new simulated values for the other years and scenarios.

```
qui sum pcc [aw=wgt]
scalar pccY0 = r(mean)
qui sum ysim_`t'_sc`i' [aw=wgtsim`t']
scalar pccrwY1 = r(mean)

qui replace ysim_`t'_sc`i' = ysim_`t'_sc`i' *   //
                             (pccY0*${pcc_growth_t`t'_sc`i'}) / pccrwY1
label var ysim_`t'_sc`i'  //
   "Simulated pcc - After reweighting and hh consumption growth, Year: `t'"

ren wgtsim`t' wgtsim`t'_sc`i'

if (`i'==1 & `t'==2021) save "$path/data/MAG_simulated.dta",replace
else {
    qui merge 1:1 hhid pid using  //
              "$path/data/MAG_simulated.dta", keepusing(ysim_*_sc* wgtsim*_sc* )
    drop _merge
    qui save "$path/data/MAG_simulated.dta", replace
} // This closes the loop.
```

After processing each scenario and year, a message is displayed in red, indicating the completion of the scenario-year pair.

```
dis in red "DONE WITH SCENARIO `i' AND YEAR `t'"
```

These closing brackets represent the end of loops and conditions. The code runs through various scenarios and years, updating the variable `i` (scenario counter) for each iteration.

```
} /* Conditional to skip scenarios for 2021 */
} /* Loop over years */
local i = `i'+1
} /* Loop over scenarios */
```

Finally, the timer that was started at the beginning of the code is turned off, and the time taken is displayed.

```
timer off 1
timer list 1
```

In summary: Overall, this code simulates labor income under different scenarios and years, adjusts it based on various factors, and then saves or appends the results to the file named `MAG_simulated.dta`.

5 Generating indicators

File: 04_indicators.do

5.1 Poverty statistics by regions, urban/rural, and gender

This code is primarily involved in generating poverty and inequality indicators for different scenarios and years. Load the simulated data created before for further processing. Merge the data with another dataset that likely contains poverty lines or thresholds (based on variable names).

```
use "$path/data/MAG_simulated",clear
merge 1:1 hhid pid using "$path/data/MAG_assigned", keepusing(pline* lbpl ubpl)
```

Define global macros for scenarios and years. Note that we only analyze every ten years.

```
global scenssim 1 2 3 4 5 6
global nscsteps 2030(10)2050
```

Create a binary indicator for urban areas based on binary "rural". If not present.

```
*gen urban = rural!=1
```

Keep only the specified variables in the dataset.

```
keep hhid pid wgt wgtsim* ysim* pcc region gender zone pline* lbpl ubpl
```

Define different poverty thresholds.

```
gen pov19 = 1.9
gen pov32 = 3.2
gen pov55 = 5.5
gen pov10 = 10
```

Create a variable called 'country' with value 1 for all observations. Also, define a new variable `Ysim_baseyear` equal to the `pcc` variable, which seems to represent per capita consumption.

```
g country=1
g Ysim_baseyear = pcc
```

The next set of commands uses the `sp_groupfunction` command multiple times. This command seems to be a user-defined or package-specific function that calculates some group-specific statistics, likely poverty rates, and Gini coefficients. For example, This block calculates statistics for the base year at the country level:

```
preserve
    sp_groupfunction [aw=wgt], poverty(Ysim_baseyear)  //
    povertyline(pline215 pline365 pline685 lbpl ubpl)  //
                gini(Ysim_baseyear) mean(Ysim_baseyear) by(country)
    g year=$surveyyear
    tempfile resultsbase
    qui save `resultsbase'
restore
```

Similar blocks are provided for calculating statistics at different levels: region, urban vs. rural (zone), and gender.

```
preserve
    sp_groupfunction [aw=wgt], poverty(Ysim_baseyear) //
    povertyline(pline215 pline365 pline685 lbpl ubpl) //
                mean(Ysim_baseyear) by(country region)
    g year=$surveyyear
    tempfile resultsbase_reg
    qui save `resultsbase_reg'
restore
```

```
preserve
    sp_groupfunction [aw=wgt], poverty(Ysim_baseyear)  //
    povertyline(pline215 pline365 pline685 lbpl ubpl)  //
                mean(Ysim_baseyear) by(country zone)
    g year=$surveyyear
    tempfile resultsbase_urb
    qui save `resultsbase_urb'
restore

preserve
    sp_groupfunction [aw=wgt], poverty(Ysim_baseyear)  //
    povertyline(pline215 pline365 pline685 lbpl ubpl)  //
                mean(Ysim_baseyear) by(country gender)
    g year=$surveyyear
    tempfile resultsbase_gender
    qui save `resultsbase_gender'
restore
```

And then we drop the previously created variable `Ysim_baseyear`.

```
drop Ysim_baseyear
```

The following loop iterates over scenarios and years, calculating the same set of statistics for each combination. For each scenario-year combination, the results are saved in a temporary file.

```
foreach sc in $scenssim {
    forvalues i=$nscsteps {
      preserve
      noisily sp_groupfunction [aw=wgtsim`i'_sc`sc'], poverty(ysim_`i'_sc`sc')..
      g year=`i'
      tempfile results`i'_`sc'
      qui save `results`i'_`sc''
      restore
    }
}
```

The following nested loops iterating over various scenarios (`$scenssim`) and years (`$nscsteps`) do the same. Within each loop, a function named `sp_groupfunction` is applied to the data, which likely computes poverty-related statistics. The results are then saved to separate temporary files for different combinations of scenarios, years, and groupings (region,

urban/rural zone, and gender). The results are saved in temporary files named with the pattern `results[year]_[grouping]_[scenario]`. In summary, the code generates poverty statistics based on simulated data, breaking down the results by different groupings (regions, urban/rural zones, and gender) for various scenarios and years.

Then the code focuses on aggregating results from various files, preparing data for output, and then performing additional calculations on income percentiles. Here's the step-by-step breakdown:

The script first loads the results from the base file (`resultsbase`) and then appends (or stacks) results from the different scenarios, regions, zones (urban/rural), and gender groupings created before.

```
use `resultsbase',clear
append using `resultsbase_reg'
append using `resultsbase_urb'
append using `resultsbase_gender'
foreach sc in $scenssim {
    forvalues i=$nscsteps {
        append using `results`i'_`sc''
        append using `results`i'_reg_`sc''
        append using `results`i'_urb_`sc''
        append using `results`i'_gender_`sc''
    }
}
```

We then create a Unique Identifier (`concat`) and Reorder.

```
gen concat = measure +"_"+ variable+"_" +reference+"_"+string(region)+ //
                      "_"+string(zone)+"_"+string(gender)
order concat, first
gen regionref = region
```

The aggregated data is then exported to an Excel file and then saved in a Stata format.

```
export excel using "$scenario_file", sheet("Indicators",modify) firstrow(varlabels)
drop concat
save "$path/outputs\MAG_indicators",replace
```

The script switches to another dataset (`MAG_simulated`) and prepares to compute income percentiles.

```
use "$path\data\MAG_simulated",clear
merge 1:1 hhid pid using "$path/data/MAG_assigned", keepusing(pline* lbpl ubpl)
keep hhid pid wgt wgtsim* ysim* pcc pline* lbpl ubpl
g Ysim_baseyear = pcc
g country = 1
drop pcc
```

It first calculates the percentiles for the base year income.

```
gquantiles Ysim_baseyear [aw=wgt], _pctile nq(100)
mat quantiles = r(quantiles_used)
mat colnames quantiles = Ysim_baseyear
```

In essence, this code is primarily focused on organizing and summarizing the simulated results from different scenarios, then performing statistical analyses on the income distribution by quantiles.

```
foreach sc in $scenssim {
    forval simyear = $nscsteps {
        gquantiles ysim_`simyear'_sc`sc' [aw=wgtsim`simyear'_sc`sc'], _pctile nq(100)
        mat mat0 = r(quantiles_used)
        mat colnames mat0 = ysim_`simyear'_sc`sc'
        mat quantiles = quantiles,mat0
    }
}
```

We export income by percentiles to our Excel file, modifying it (not replace). First, the data is cleared out of the current memory. The `quantiles` matrix is converted to a dataset format using `svmat`. Each column from the matrix becomes a variable in the dataset. A new variable, `percentil`, is created to indicate the row number (or the percentile). The dataset is exported to an Excel file under the sheet "Income_by_percentile".

```
clear
svmat quantiles, names(col)
g percentil = _n
export excel using "$scenario_file", sheet("Income_by_percentile",modify) //
                   firstrow(varlabels)
```

5.2 Compute deviations

The dataset `MAG_indicators` is loaded into memory. Several variables are modified to replace missing or specific values:

```
use "$path/outputs\MAG_indicators", clear
drop country
replace region = 99 if region==.
replace zone = 99 if zone==.
replace gender = 99 if gender==.
replace reference = "NA" if reference==""
replace _population=_population*10e-7
preserve
```

The dataset is then subsetted to keep observations where `measure` is "fgt0" (Foster-Greer-Thorbecke poverty measure with parameter 0). Some transformations are applied to the `value` variable based on the `measure` variable. A temporary file (`number_poor`) is saved. The original data is then restored and appended with this modified subset.

```
keep if inlist(measure,"fgt0")
replace value=value*_population
replace measure="np_fgt0" if measure=="fgt0"
tempfile number_poor
save `number_poor'
restore
append using `number_poor'
        ```

Further transformations are performed:

```stata
replace value=100*value if inlist(measure,"fgt0","fgt1","fgt2","gini")
replace variable = "ysim_base_sc0" if variable=="Ysim_baseyear"
```

The code then proceeds to extract `scenid` (scenario ID) and creates labels for it. It also creates a variable, `Ysim`, by extracting a part of the `variable` column. Some cleanup operations are also performed.

```
g scenid = real(substr(variable,13,2))
label define scenid 0 "base year" $scenid ,replace
label values scenid scenid
```

```
g Ysim = substr(variable,1,ustrpos(variable,"_sc")-5)
replace Ysim = "base" if year==$surveyyear
drop variable
```

The final version of the dataset is exported to an Excel file under the sheet "Results_long".

```
export excel using "$scenario_file", sheet("Results_long",modify) firstrow(varlabels)
```

Overall, this code focuses on data manipulation tasks, including reshaping the data, generating new variables, making adjustments to existing variables, and exporting the processed data to Excel for further use.

5.3 Generating Deviations from Baseline:

Remove observations where the variable `Ysim` is equal to "base". Keep only those observations where the gender is coded as 99. Drop the variable `_population` as it is not needed. Reshape the dataset from long format to wide format using the `value` variable, with new columns created for each `scenid`. This will generate variables like `value1`, `value2`, etc., for each scenario.

```
drop if Ysim=="base"
keep if gender==99
drop _population
reshape wide value, i(measure reference region zone year) j(scenid)
```

For each scenario, calculate the deviation as a percentage from the baseline (`value1`). This deviation indicates how much a value in a given scenario has changed relative to the baseline scenario. Similarly, compute the deviation in levels (absolute difference from the baseline).

```
local num : list sizeof global(scenarios)
forvalues i=2(1)`num' {
  g des_percent_`i' = 100*(value`i'-value1)/value1
}
forvalues i=2(1)`num' {
  g des_level_`i' = value`i'-value1
}
```

Drop all the `value*` columns. Reshape the dataset back to long format based on the deviation columns. Rename the columns for better clarity.

```
drop value*
reshape long des_percent_ des_level_, i(Ysim measure reference region zone year) j(scenid)
rename des_*_ des_*
```

Create a new variable, `concat`, that combines several variables into a single string. This is for uniquely identifying or categorizing each observation. Reorder the dataset so that `concat` appears first.

```
gen concat = measure +"_"+ Ysim+string(year)+"_" //
             +string(scenid)+"_"+reference+"_"+  //
             string(region)+"_"+string(zone)+"_"+ //
             string(gender)
order concat, first
```

Finally, export the current dataset to an Excel file under the sheet "Deviations_long".

```
export excel using "$scenario_file", sheet("Deviations_long",modify) //
                  firstrow(varlabels)
```

This code is mainly concerned with deriving deviations from a baseline across different scenarios and organizing the resulting dataset for further analysis and presentation. The computed deviations provide insights into how different scenarios compare to a reference or baseline scenario in the given dataset.