Data Cleaning Guidelines

for 2nd Phase FATE-Surveys: Stata Syntax

Abridged version

Author: Patrick Illien[1]

Contents

[1] I would like to thank Dr. Christoph Bader and Dr. Maurice Tschopp for valuable inputs.

Introduction

These guidelines have been developed for the 2nd phase FATE surveys to ensure a minimum standard for comparability. The present docment has been inspired by the following sources, which contain useful tips and should be consulted for more information:

- ACAPS, 2016. *Data cleaning: technical brief*. Available here.
- Beaver, M., 2012. *Survey Data Cleaning Guidelines: (SPSS and Stata)*. Available here.
- Data Science Primer, 2018. *Chapter 3: Data Cleaning*. Available here.
- IFPRI (International Food Policy Research Institute), 2018. *A guide to data cleaning using Stata*. Available here.

Most of this guide is presented in tables. In each one those, the left columns describe the steps to be taken, whereas the right columns show the corresponding Stata syntax. All Stata codes were created and tested using version 15.1. The guidelines have a logical order and each step should be undertaken after the other because some syntax only works if the previous ones have already been applied successfully.

Of course, the better the survey is designed, the less errors will be made later. Investing time in survey design, coding and enumerator training at the start pays!

Unless otherwise stated, mark problematic observations, don't change values – it is up to the analysis how to deal with them!

Copying Dataset

- Perform all steps for each dataset (wide and all longs of part 1 & 2).
- **The first thing to do is to make a copy of the original data file (a simple copy without opening the raw data in a programme)!** ALWAYS keep the source files in a separate folder called "RAW DATA" and change its attribute to READ-ONLY, to avoid modification of any of the files. Then encrypt the raw-data files and don't forget to back them up as well.
- Never touch the original data and only work in the copied filed from now on, even if only importing the data into a programme. Discuss with your supervisors if the original data (which are not anonymised) shall be deleted after data analysis is finished.

Documenting Changes and Getting Started

- Perform all steps in each dataset (wide and all longs of part 1 & 2).
- Be sure to always document any changes you make directly in a R, Stata or SPSS when using any of these software packages. If you use any other software, document any changes in an excel sheet variable by variable.

What?	How? (STATA syntax)

• Make one change log per dataset and keep the change logs organised.
• Each change log contains all modifications to that dataset and will serve as an audit trail. It will allow a return to the original value if required. Within the change log, store the following information:
  ○ At the top of the file:
  ▪ Country and survey year (if necessary also survey type)
  ▪ Dataset
  ▪ Date of file status
  ▪ Author of file
  ▪ Purpose of file
  ▪ Comments
  ○ Throughout:
  ▪ Variable or observation concerned
  ▪ What the change was (e.g. old and new value)
  ▪ Comments/justification
	• Always document your comments and commands in do-files, the are the change log when working with Stata.
	• You can organise your do-files in Stata's project manager. Open in from within the do-file editor on Windows and Unix or from within the main menu on Mac (select File>New>Project....)
	• Create one master do-file per dataset and name it accordingly: type doedit into the command window to create a do-file where you write your commands. You can add comments after * at the start of a line. Save the do-file into the project manager.
	• Add all the relevant information at the top as described in the left column.
	• Apply the structure and cleaning process of this guide and add section titles accordingly. Take over all the relevant commands from each section in this guide. In each section, the datasets in which certain steps have to be performed (i.e. which have to be added to the do-file) are specified in colour.
	• Describe and justify each change you make to the data.
	• All the other information will be readable from the commands themselves but it is always good to add some descriptions.
	• Repeat the entire process for each dataset (all wide and longs of part 1 & 2).

• Increase the number of variables allowed by the software	• *clear* *set maxvar 10000*
• Increase the size of the results window display	• *set scrollbufsize 700000*

• Specify the directory/path to the copy of the raw data	• Use the cd command, for example: *cd "Z:\Data_Cleaning\Stata"*
• Import the copy of your raw data into your software package	• Use the import command with clear in order to start from a clean slate, make sure to specify the bindquotes option with strict as else it might mess up how it delimites doulbe quotes, for example: *import delimited Long_format_22.10/FATE_Rwanda_Part_1_221018_final.csv, bindquotes(strict) clear*

Deleting Cases

- Perform these steps only for the wide and long_final datasets of part 1 & 2.
- As a general rule, you should not delete any data!
- The only exception is if all or almost all entries (including key demographics) are blank for a household. This may be the case if the enumerators wrongly submitted a trial run or an aborted interview. We do this step at the start so that the faulty submissions don't add confusion when doing all subsequent cleaning steps (e.g. adding new identifiers, doing consistency checks, etc.).

What?	How? (STATA syntax)
• Delete line if there are no observations ("blank row")	• *egen nmiss = rowmiss(_all)* *display c(k)* *drop if nmiss==c(k)-1* • Alternatively you can install the "missings" package and type the following: *missings dropobs _all, force* *display r(n_dropped)*
• Delete line if almost all entries are blank	• *ds,* *return list* *local varlist=r(varlist)* *egen count_miss=rowmiss(`varlist')* *capture describe* *generate count_miss_prop=count_miss/r(k)*100* *list count_miss_prop* *list submissiondate code_id village today_date a2_nameprim key if count_miss_prop>99* *count if count_miss_prop>99* **check the list and double check (if necessary in the dataset) to see if you really want to delete those observations* *drop if count_miss_prop>99.5* *count* • You can also check the percentage of missing observations like this: *missings report, observations percent*

Tidying Dataset

- Perform all steps in each dataset (wide and all longs of part 1 & 2).
- Make sure that the dataset is tidied up as follows

What?	How? (STATA syntax)
• Fonts have been harmonised	• Automatically done in Stata when importing the csv file.
• Leading and trailing spaces of string variables have been deleted	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *replace `var'=stritrim(`var')* *}*
• Put all string variables into lower case	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *replace `var'=lower(`var')* *}*
• Put all variable names into lower case and delete spaces	• We don't need to delete spaces as Stata doesn't allow them in variable names anyways, only put in lower case: *rename *, lower*
• Replace all the NA and DK and convert only the variables with single numbers to numerics	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *replace `var'=".a" if `var'=="na" & length(`var')==2* *replace `var'=".b" if `var'=="dk" & length(`var')==2* *}*  *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *count if strmatch(`var',"* *")==1 | regexm(`var',"[a-zA-Z]")==1 & regexm(`var',"\.a")==0 & regexm(`var',"\.b")==0* *if r(N)==0 {* *destring `var', replace* *}* *}*
• Once, all the NA and DK have been changed, check for inconsistencies in alphabetic responses or categories by showing all string responses containing any alphabetical letters	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if `var'!="" & regexm(`var',"[a-zA-Z]")==1* *}*
• Go through the entire list and check for mislabeled categorical string labels, i.e. separate classes that should really be the same, e.g. If 'there is kgs' and 'kilo' and "kg", you should combine them. Modify this code to correct all mislabeled categorical variables	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *replace `var'="kg" if `var'=="kgs" | `var'=="kilo" | `var'=="kilos"* *}*
• Check for remaining inconsistencies (adjust relevant variable names according to dataset)	• You can exclude variables that you don't want to check as follows (this depends on the dataset): *preserve* *drop submissiondate a13_codeenumerator village starttime today_date a2_nameprim fam_name* end_time instanceid key*

	ds, has(type string) *foreach var of varlist `r(varlist)' {* *list `var' if `var'!="" & `var'!="kg" & regexm(`var',"[a-zA-Z]")==1* *}* *restore*
• List variables containing multiple answers	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *count if strmatch(`var',"* *")==1 & regexm(`var',"[a-zA-Z]")==0* *if r(N)>0 {* *list `var' if `var'!=""* *}* *}*
• List only the multiple answer observations	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if strmatch(`var',"* *")==1 & regexm(`var',"[a-zA-Z]")==0 & `var'!=""* *}*
• Show all string responses that contain any number	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if regexm(`var',"[0/9]")==1* *}*
• Make sure that there are no variables containing only single numbers that are characterised as strings. The following commands should yield no result	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *count if strmatch(`var',"* *")==1 | regexm(`var',"[a-zA-Z]")==1 & regexm(`var',"\.a")==0 & regexm(`var',"\.b")==0* *if r(N)==0 {* *list `var' if `var'!=""* *}* *}*
• Save everything you have done up to now under the name clean_draft	• For example: *save Wide_format_22.10/FATE_Rwanda_Part_1_221018_final_WIDE_clean_draft, replace* *clear*

Checking and Correcting Identifiers

- Perform all steps in the wide and long_final datasets of part 1 & 2).
- Each household will receive its unique identifier. We therefore have to first make sure that the code_id is attached to the correct household and that it is the same household for a specific code_id in part 1 and part 2. We will merge certain key characteristics of part 1 and 2 to check if the households match but we will continue working on the separate datasets after.

What?	**How? (STATA syntax)**
• First correct any wrong IDs of part 1, e.g. if they exist twice, manually in accordance with your sample frame. Part 1 code_id must be unique so that we can match part 1 and 2 of the survey correctly for each household. If you have discovered some manifest errors in the identity of the respondent, such as wrong code_ids or gender, correct them directly in the wide and long datasets of part 1 and 2 via the use command in your	• Run do-files provided by the project

do-file. Inspect households that were not available and have most entries missing (avail!=1) and mark them with "delete".	
• Then prepare part 2 for the merger. Inspect households that were not available and have most entries missing (avail!=1) and mark them with "delete". You don't need to check anything here, we will do that in the merged file.	
• Merge the key identifiers of part 1 and 2 in a new dataset to check if each household is correct • Now undertake as many cross-checks as possible in order to ensure that each household of part 1 corresponds to the correct household of part 2. Check any inconsistencies manually with the sampling frame and describe the problem. This is the most tedious cleaning step and might take some time. • If you have discovered some manifest errors in the identity of the respondent, such as wrong code_ids or gender, correct them directly in the relevant wide and long datasets of part 1 and 2 via the use command in your do-file. Other inconsistencies (such as different household types or agricultural production), do not have to be changed as it is up to the analyst how to deal with them (they might exclude them or make certain assumptions such as that every household has a vegetable garden). You should nevertheless describe these inconsistencies in a new variable called "remarks". They might in part be based on different interpretations of the question. • Make sure to save all the keys if the entire code_id will be deleted in a new dataset that you can later use for the long do-files of part 1 (other than long_final), see below.	
• Now merge the corrected datasets and run all checks again to see if they now yield the correct results.	

Adding new Identifiers

- Perform all steps in each dataset (wide and all longs of part 1 & 2).
- When you are sure that all your households are identified correctly and the data cleaning has been done, create new unique household identifiers in the following way (we will not create unique identifiers for each individual). This is to create simple, totally anonymised and unique identifiers across all datasets. We do this step before the consistency checks so that we have already corrected any possibly wrong household entries.

What?	How? (STATA syntax)
• Create new unique identifiers: ○ *Country code / last two digits of year in which survey took place/random consecutive three-digit household number (preceeded by zeros as*	• Run do-files provided by the project • This is the key step in it creating new unique identifiers: *set seed 2345* *gen random=runiform()* *sort code_id, stable* *by code_id: gen group_random=random[1]* *egen sequence=group(group_random), missing* *gen survey="418"* *tostring sequence, replace format(%03.0f)* *gen new_identifier=survey+sequence* *destring new_identifier, replace* *sort new_identifier*

placeholders if necessary)	
• Create an identifier template that you can use after to add the unique identifiers into each dataset.	• This step is included in the do-files provided by the project: *preserve* *keep code_id key_part1 dup_key_part1 new_identifier* *order new_identifier, first* *duplicates drop if dup_key_part1>0* *drop dup_key_part1* *rename key_part1 key* *save Identifiers/New_identifiers_part1_key, replace* *restore*
• Make sure to use the correct file	• For example: *use* *Wide_format_22.10/FATE_Rwanda_Part_1_221018_final_WIDE_idcheck, clear*
• Merge unique identifier variables into all datasets using the key variable	• Part 1, wide and long_final formats: *merge m:1 key using "Identifiers/New_identifiers_part1_key"* *order new_identifier, first* *list if new_identifier==. /*this list should be empty*/* *list if _merge!=3 /*this list should be empty*/* *save* *Wide_format_22.10/FATE_Rwanda_Part_1_221018_final_WIDE_idcheck, replace* • Part 1, all other long formats: *merge m:1 parent_key using "Identifiers/New_identifiers_part1_parent_key"* *order new_identifier, first* **check if _merge 1 observations are households that have been deleted* *preserve* *keep if _merge==1* *keep new_identifier parent_key _merge* *rename _merge _merge_master* *merge m:1 parent_key using "Identifiers/Deleted_households"* */*now go manually through list and identify all anomalies*/* *restore* *drop if _merge==1* *list if new_identifier==. /*this list should be empty*/* *list if parent_key=="" /*this list should be empty*/* *save Long_format_22.10/FATE_Rwanda_Part_1_221018_final-consent_given-hired_labour-repeat_cc5_idcheck, replace* • Part 2, wide and long formats: *merge m:1 key using "Identifiers/New_identifiers_part2_key"* *order new_identifier, first* *list if new_identifier==. /*this list should be empty*/* *list if _merge!=3 /*this list should be empty*/* *save* *Wide_format_22.10/FATE_Rwanda_Part_2_221018_final_WIDE_idcheck, replace*

Consistency Checks

- We undertake some key consistency checks at this stage. However, we don't check for all possible inconsistencies. First, many rules were already inserted in the ODK file in order to avoid logical problems from the start and we don't have to re-check for most of them (some were also inserted in order to facilitate the question presentation but are not relevant for analysis and can be ignored). Second, it is up to the analyst to detect and deal with outliers. We therefore went through all the survey questions and prioritised the following consistency checks that you should undertake now. The rest is up to the analyst.
- Mark all the inconsistencies you find but don't change any data. It is up to the analyst to effect those changes and to justify and document them.

What?	How? (STATA syntax)
• Undertake one check after the other	• Within a do-file, each check is a separate activity. The checks should be run, one at a time, where the list is checked and the problems identified and documented before the next check is run. DO NOT RUN the complete do-file at once! You will get garbage and you will not be able to figure out which listings to work on before working on others.
• Skipping: Certain variables should only have values if the answer to a previous filter question appropriate. However, as the skipping functions have been inserted in the original xls files, we will only double check module filter questions in part 1 as they would have the largest impact (see priority skipping checks below). Make sure that the original xls files contains all the necessary skipping functions, if that is not the case or if you want to add skipping checks, here is an example:	• Let's assume that if the filter question has a value of 1, there should be data in the subordinate questions and that if the filter question does not have a value of 1, there should be no data in the subordinate questions. We can then check if the skipping worked correctly with the following command (which can be adapted to match other skipping cases): *foreach var of varlist consent {* *gen s`var'=.* *replace s`var'=1 if `var'==1* *replace s`var'=0 if s`var'!=1* *list `var' if s`var'==0* *}*

Consistency Checks for Part 1

What?	How? (STATA syntax)
Checks in wide	
• We repeat several of the commands (e.g. forval) below on string variables with almost the same name (e.g. b1_name_1 to b1_name_11). We include as many numbers of the variable as there are string variables, e.g. if b1_name_12 is a numeric variable because there are only missing values (i.e. no household had 12 members), we cannot include it in the command as it would lead to a type mismatch. • The checks in this section are only possible in the wide format of part 1, however, you should also add all the checks of the long formats of part 1 (see below) to the do-file for the wide format so that you can check independently. The syntax will have to be adapted accordingly.	
• Priority skipping check module A: Check that there	• *list new_identifier if hh_people_nb==0*

are household members	
• Priority skipping check module B	• *list new_identifier if wage_employment!=1 & wage_employ!=""*
• Priority skipping check module C	• *list new_identifier if cd2_labour_exchange!=1 & howmany_exchange!=""*
• Priority skipping check module Da	• *list new_identifier if cd1_agr_production!=1 & cd1_how_many!=""*
• Priority skipping check module Db	• *list new_identifier if cd1_agr_production!=1 & cb1_crop!=.*
• No priority skipping check for module Dc necessary	
• No priority skipping check for module De necessary	
• Priority skipping check module E	• *list new_identifier if cc5!=1 & activities_hired_work!=""*
• Priority skipping check module F	• *list new_identifier if da1_ownland!=1 & daa_plots!=.* *list new_identifier if da1_ownland!=1 & da9_landcoop!=.* *list new_identifier if da4_rent_to!=1 & da4_rent_plot!=.* *list new_identifier if da5_otherland!=1 & da5_owner_other!=.*
• No priority skipping check for module G necessary	
• No priority skipping check for module Z necessary	
• Check household type with gender of head	• *forval i=1/45 {* *list new_identifier if (b02_hh_member_gender_ `i'==2 & b03_hhrelationship_ `i'==1 & a7_hhtype==3) | (b02_hh_member_gender_ `i'==1 & b03_hhrelationship_ `i'==1 & a7_hhtype==2)* *}*
• Check that each paid working household member is older than 14 years old	• *forval i=1/11 {* *forval j=1/4 {* *list b1_name_ `i' if b1_name_ `i'==name_display_ `j' & name_display_ `j'!="" & b04_hh_member_age_ `i'<14* *}* *}*
• Check if any numeric variables are negative • If any other variables other than GPS location have negative values, mark them	• *preserve* *drop gpslatitude* *ds, has(type numeric)* *foreach var of varlist `r(varlist)' {* *list `var' if `var'<0* *}* *restore*
• Check that each unpaid working household member is older than 14 years old	• *forval i=1/11 {* *forval j=1/2 {* *list b1_name_ `i' if b1_name_ `i'==name_display2_ `j' & name_display2_ `j'!="" & b04_hh_member_age_ `i'<14* *}* *}*
• List all the production units other than "kg": first check that everything that should be "kg" is	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if strmatch("`var'","*prod_unit*")==1 & `var'!="" & `var'!="kg"* *}*

spelled correctly, you then might have to convert the rest using a locally appropriate conversion factor or deal with them in some other way in the analysis.	
• Show all the area units that are not pre-coded. You might have to convert them using a locally appropriate conversion factor or deal with them in some other way in the analysis.	• *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if strmatch("`var'","*unit_other*")==1 & `var'!=""* *}*
• Check that if others were selected, that they are the same activities for across module E	• *preserve* *forval i=1/3 {* *forval j=1/5 {* *tostring cc2_2_activity_hired`i'_`j', replace* *replace cc2_2_activity_hired`i'_`j'="" if cc2_2_activity_hired`i'_`j'=="."* *gen test_`i'`j'=1 if activities_hired_work`i'==activities_most_work`i' & activities_hired_work`i'==cc2_2_activity_hired`i'_`j'* *list activities_hired_work`i' activities_most_work`i' cc2_2_activity_hired`i'_`j' if test_`i'`j'!=1* *}* *}* *restore*
• If there are child-headed households verify the age	*forval i=1/45 {* *list b1_name_`i' if b03_hhrelationship_`i'==1 & a7_hhtype==4 & b04_hh_member_age_`i'>18 | b03_hhrelationship_`i'==1 & a7_hhtype==4 & b04_hh_member_age_`i'<10 | b03_hhrelationship_`i'==1 & a7_hhtype==5 & b04_hh_member_age_`i'>18 | b03_hhrelationship_`i'==1 & a7_hhtype==5 & b04_hh_member_age_`i'<10* *}*
• Check that the household size is equal to the maximum of household members	• *preserve* *forval i=1/45 {* *tostring b1_name_`i', replace* *gen count_`i'=1 if b1_name_`i'!="" & b1_name_`i'!="."* *}* *egen total=rowtotal(count_1-count_45)* *list new_identifier if hh_people_nb!=total & hh_people_nb!=.* *restore*
• Check gender of primary respondent with gender in household list	• *forval i=1/11 {* *list new_identifier if a2_nameprim==b1_name_`i' & a2_nameprim!="" & a8_gender!=b02_hh_member_gender_`i'* *}*
• Check that primary	• *forval i=1/11 {*

respondent is over 18 years old	*list new_identifier if a2_nameprim==b1_name_`i' & a2_nameprim!="" & b04_hh_member_age_`i'<18* *}*
• Check that the primary respondent is a household member (mistakes could be due to spelling error!)	• *list new_identifier if a2_nameprim!=b1_name_1 & a2_nameprim!=b1_name_2 & a2_nameprim!=b1_name_3 & a2_nameprim!=b1_name_4 & a2_nameprim!=b1_name_5 & a2_nameprim!=b1_name_6 & a2_nameprim!=b1_name_7 & a2_nameprim!=b1_name_8 & a2_nameprim!=b1_name_9 & a2_nameprim!=b1_name_10 & a2_nameprim!=b1_name_11*
Checks in long _final	
• Households should only have been replaced if nobody was available in the household	• *list new_identifier if avail!=1 & replacement_id==.* *list new_identifier if avail==1 & replacement_id!=.*
• Check that only one id was given per household	• *list new_identifier if code_id!=. & replacement_id!=.* • Alternative ways of checking that could be employed throughout: *assert code_id!=. if replacement_id!=.* • Or: *count if code_id!=. & replacement_id!=.*
• Check if consent was given • If the consent is missing but questions were answered, keep the household. If the consent was no or if almost no questions have been answered, delete the household	• *list consent if consent!=1*
• Check that there are no duplicates of primary respondents • If there are duplicates you might check if they are from different households after unique identifiers have been developed (see below)	• *duplicates list a2_nameprim if a2_nameprim !=""*
• Check gender of primary respondent with status	• *list a2_nameprim if (a8_gender==2 & status==5) | (a8_gender==1 & status==4)*
• Check household type with primary respondent	• *list a2_nameprim if (a8_gender==2 & status==1 & a7_hhtype==3) | (a8_gender==2 & status==1 & a7_hhtype==4) | (a8_gender==1 &*

	status==1 & a7_hhtype==2) | (a8_gender==1 & status==1 & a7_hhtype==4)
• Check that all households have at least 1 member	• *list new_identifier if hh_people_nb<=0*
• Check that years are formatted correctly	• *foreach var of varlist da2g_surfaceprod da5_aquires1{ list `var' if (`var'!=. & `var'<1900) | (`var'!=. & `var'>2018) }*
• Check that percentages are correct	• list df9_interest if df9_interest!=. & df9_interest>100 & df9_interest<0
Checks in long _final-consent_given-repeat_hh	
• Check for duplicate household members	• *duplicates list b1_name if b1_name!=""*
• Check age difference between oldest child and head	• *by new_identifier, sort: egen head_age=min(b04_hh_member_age) if b03_hhrelationship==1 by new_identifier, sort: egen oldest_child=max(b04_hh_member_age) if b03_hhrelationship==3 by new_identifier, sort: egen min_head_age=min(head_age) by new_identifier, sort: egen max_oldest_child=max(oldest_child) by new_identifier, sort: gen difference=min_head_age-max_oldest_child list difference b1_name new_identifier if difference < 15*
• Check age of head	• *list b1_name if b04_hh_member_age<18 & b03_hhrelationship==1*
• Check age of spouse	• *list b1_name if b04_hh_member_age<18 & b03_hhrelationship==2*
• Check age and marital status	• *list b1_name if b6_marstat!=1 & b6_marstat!=.a & b6_marstat!=.b & b6_marstat!=. & b04_hh_member_age<18*
• Check age and education status	• *list b1_name if b04_hh_member_age<10 & b8_education>16 & b8_education!=96 & b8_education!=.*
Checks in long _final-consent_given-repeat_wage_employment	
• No consisteny checks specific to this dataset	
Checks in long _final-repeat_multiplicity	
• Check that there are no more than 31 working days per month	• *list bb7_work_day_permonth if bb7_work_day_permonth>31 & bb7_work_day_permonth!=.*
• Check that there are no more than 24 working hours per day	• *list bb8_work_hours_perday if bb8_work_hours_perday>24 & bb8_work_hours_perday!=.*
Checks in long _final-consent_given-repeat_cd	
• Check that there are no more than 365 working days per year	• *list cd6_work_days if cd6_work_days>365 & cd6_work_days!=.*
• Check that there are no more than 24 working hours per day	• *list cd7_hours if cd7_hours>24 & cd7_hours!=.*
Checks in long _final-consent_given-repeat_agrprod	

• No consisteny checks specific to this dataset	
Checks in long _final-consent_given-hired_labour-repeat_cc5	
• Check that there are no more than 365 working days per year	• *foreach var of varlist hire_onetime m1_b m1_d m1_f m1_h m1_j m1_l {* *list `var' if `var'>365 & `var'!=.* *}*
Checks in long _final-consent_given-repeat_cc2	
• Check that there are no more than 365 working days per year	• *foreach var of varlist cc4_work_day cc14_number {* *list `var' if `var'>365 & `var'!=.* *}*

Consistency Checks for Part 2

What?	How? (STATA syntax)
Checks in wide & long _final	
• Check if any numeric variables are negative • If any other variables have negative values, mark them	• *ds, has(type numeric)* *local varlist `r(varlist)'* *local toexclude r1_1 r1_2 r1_3 r1_4 r1_5 r1_6 gpslatitude* *local varlist: list varlist - toexclude* *foreach var of local varlist {* *list `var' if `var'<0* *}*
• Check if the person was available	• *list avail if avail!=1*
• Check if consent was given • If the consent is missing but questions were answered, keep the household. If the consent was no or if almost no questions have been answered, delete the household	• *list consent if consent!=1*
• Check that there are no duplicates of respondents • If there are duplicates you might check if they are from different households after unique identifiers have been developed (see below)	• *duplicates list a3_primary if a3_primary !=""*
• Check household type with respondent	• *list a3_primary if (a3b_gender==2 & a3c_relationship ==1 & a4_hhtype==3) | (a3b_gender==2 & a3c_relationship ==1 & a4_hhtype==4) | (a3b_gender==1 & a3c_relationship ==1 & a4_hhtype==2) | (a3b_gender==1 & a3c_relationship ==1 & a4_hhtype==5)*

• If there are child-headed households verify the age	• *list a3_primary if a4_hhtype==4 & a3b_age>18 | a4_hhtype==4 & a3b_age<10 | a4_hhtype==5 & a3b_age>18 | a4_hhtype==5 & a3b_age<10*
• Check age of head	• *list a3_primary if a3b_age<18 & a3c_relationship==1*
• Check age of spouse	• *list a3_primary if a3b_age<18 & a3c_relationship==2*
• Check if any houshold head or spouse was replaced	• *list a3_primary a3c_relationship a3b_age if a3c_relationship!=1 & a3c_relationship!=2 & a3c_relationship!=.*
• Check crop and coffee questions	• *foreach var of varlist i03_inputplot i04_tecno i05_fert i06_labor i07_harvest i08_selling i08_seed i08_income { list `var' if i01_crop1_1!=1 & i01_crop1_2!=1 & i01_crop1_3!=1 & i01_crop1_4!=1 & i01_crop1_5!=1 & `var'!=. }*
• Check finances and household type	• *list new_identifier if a4_hhtype!=1 & i11_finance1!=.*
• Check ownership and household type	• *foreach var of varlist i1_houseownership i2_hhgoods i3_productiveassets i4_livestock i6_whobuys_agr i8_whobuysmajor i9_whobuyspurchases { list `var' if (a4_hhtype!=1 & `var'==3) | (a4_hhtype!=1 & `var'==5) }*
• There is a large number of possible time-use inconsistencies depending on the context and they cannot be checked here. Random checks have been effected in the field which must suffice. The rest is up to the analyst.	
• Check that there are no more than 365 days per year	• *list excessive_hours if excessive_hours>365 & excessive_hours!=.*

The data verification will continue into the actual analysis. Some problems cannot be identified until analysis has begun.

Dealing with Missing Data

- Don't delete missing data, however, you cannot simply ignore missing values in your dataset either.
- In general, do not assume missing observations to mean 0 (for example, most "how many" questions only allowed an integer as answer so no answer might mean 0 but it is also possible that the person didn't know the answer but there was no choice for DK) unless this is heavily suggested by answers to previous questions or other variable within the housheold.
- Analysts should report how many observations are missing and inform the reader how missing data was handled.

What?	How? (STATA syntax)
• Do not drop observations that have missing values or impute the missing values based on other observations! You should always tell your algorithm that a value was missing because missingness is informative.	• Stata automatically assigns a "." for missigness to each missing numeric value upon import and a "" (blank) for missingness for each string value. • In addition to the default ".", which is called the "system missing value" or sysmiss, Stata has 26 other numeric missing values: ".a", ".b", ".c", ..., ".z", which are called the "extended missing

	values". This helps us to assign different reasons for missingness, such as NA (.a) or DK (.b), see below.
• Skipped questions: If certain questions did not appear in the question path, mark them as missing values.	• Stata automatically assigns a "." for answers in relations to questions that have been skipped.
• We do not distinguish here if a missing answer has been skipped or was left out despite the question having appeared. However, if needed, the analyst can always go back and test for each case if a variable was skipped or is missing despite the question having appeared. This might be the case if there is a very large number of missing values. The analyst can use descriptive statistics to see how many are missing and should look for meaning in non-random missing values. Maybe the respondents are indicating something important by not answering one of the questions or maybe an enumerator consistently left out questions despite them having appeared. There is a variety of statistical methods available for handling missing data. It is up to the analyst to use, document and justify the most appropriate one for his/her case.	

Dealing with Outliers

- It always helps to use descriptive statistics to get to know your data better and to check for problems and potential outliers. Here are some useful commands in that regard: codebook, describe, summarize, list, tabstat, tabulate (helpful explanations are provided in IFPRI, 2018). Plotting numeric variables might also be particularly helpful in identifying outliers.
- However, don't change any answers! Leave all outliers in the dataset. It is up to each researcher to deal with outliers in their variables. This is because different analysis goals require different outlier treatments and it is up to each analyst to choose the method most appropriate for his/her case.
- Exception: if you are very sure that an entry is wrong (e.g. because it is logically incoherent as for example shown by a consistency check), you can change it by the correct value or make it a missing variable, however, you MUST document this in the change log including the original value so that it is always possible to go back if necessary.

Labelling

In order to help you with the labelling process, we first prepare a new and separate dataset (i.e. not in your do-file) based on the choices sheet of the xls forms used in the survey. Proceed as follows:

- Copy the original xls form of part 1 used in the survey. Rename the copy to end in _replaced. Only work in the copy from now on. Open it with Excel.
- In the choices sheet, select the entire name column (column B), go to Home/Find & Select/Replace… under "Find what:" type "NA", under "Replace with:" type ".a". Click "Options >>" and make sure to tick the box that says "Match case" and "Match entire cell contents" and then click "Replace All".
- In the choices sheet, select the entire name column (column B), go to Home/Find & Select/Replace… under "Find what:" type "DK", under "Replace with:" type ".b". Click "Options >>" and make sure to tick the box that says "Match case" and "Match entire cell contents" and then click "Replace All".
- Save the xls file.
- Open a new dataset in your software package. Now copy the entire 4 first columns of the choices sheet of the xls form _replaced of part 1 used in the survey. Special paste them (selecting "Tab" as delimiter) into the data-editor (in the edit mode) of your software package. Save that dataset under the name "Value_labels_part1".
- Repeat the same procedure for part 2.

Perform all steps in the wide formats of part 1 & 2 and then copy the relevant labels into all corresponding long datasets of part 1 & 2.

How? (STATA syntax)

- Run the following commands in the "Value_labels_part1" dataset in order to create an answer list with the appropriate syntax to copy and paste the value labels rather than having to type them each time:

```
drop var3
replace var4=`""" + var4 + `"""
forvalues i=10/100 {
replace var4=regexr(var4,"`i'. ","")
}
forvalues i=1/9 {
replace var4=regexr(var4,"`i'. ","")
}
foreach var of varlist var4 {
replace `var'=stritrim(`var')
}
egen value_label = concat(var2 var4), p(" ")
save Z:\Data_Cleaning\Stata\Labels\Value_labels_part1, replace
```

- Export the dataset under the name "Value_labels_part1" to excel (export data to excel spreadsheet) selecting "Save variable names to first row in Excel file" and open the excel file. You can copy the value label list in the "value_label" column into the do-file as needed.
- Repeat the same procedure for part 2.

- The #delimit ; command is useful in do-files if you want to paste long lists (e.g. of variable names or value labels) from excel and you only have them in column. Stata will read everything before a semi-colon as one line and you don't have to type all the names in one row.
- However, once we change the line delimiter to semicolon, all lines, even short ones, must end in semicolons. Stata treats carriage returns as no different from blanks. We can change the delimiter back to carriage return by typing #delimit cr.
- Attach value labels to your variables in the following way, using var1 (list name) of "Value_labels_part1/2" as the name for the value labels :

```
#delimit;
label define gender
1 "Male"
2 "Female";
foreach var of varlist a8_gender a14_genderenumerator {;
label val `var' gender;
};
```

- The loop attaches the same label to multiple variables and can be expanded as needed.
- Use the survey question in the "label" column of the "survey" sheet of the original xls file to label the variables and label variable. You can also use a loop for multiple-choice dummies or variables in repeat sections of the wide format.
- SurveyCTO also created a Stata_do_template. While we cannot use that directly for our data cleaning here, you can copy and paste parts of the labelling section so that you don't have to copy each survey question from the original xls file.
- You might have to include syntax so that only numeric variables are labelled (otherwise, if string variables are included in the loop, e.g. because another variable with _other attached is included, you will encounter an error message).

```
ds, has(type string)
local strings `r(varlist)
#delimit;
```

```
        label define read
        1 "Cannot read and cannot write"
        2 "Can write only"
        3 "Can read only"
        4 "Can read and write "
        96 "Other, specify"
        .a "NA"
        .b "DK";
        #delimit cr
foreach rgvar of varlist b9_read_* {
        label variable `rgvar' "Can \${b1_name} read and write?"
}
        unab want: b9_read_*
        local numerics : list want - strings
        label values `numerics' read
```

- Multiple-choice variables are read as strings by Stata, however, string variables cannot be labelled in Stata. Fortunately, SurveyCTO automatically split all the select_multiple answers into additional dummy variables (both in the long and wide csv formats). Dummy variables have been created for each choice in the choice list, regardless if that choice was ever chosen or not, and whether that question was ever asked or not. You don't have to do anything else with multiple-choice variables other than labelling them. It is up to the analyst to decide how to deal with multiple-choice questions.
- Since multiple-choice variables have been split, you have to define a value label for each split variable and attach that value label to the correct split variable. Make sure that the value label always has the number 1 in front of it as a dummy variable by definition is only 1 or 0 (or missing). You might have to include syntax so that only numeric variables are labelled (otherwise, if string variables are included in the loop, e.g. the original multiple-choice variable, you will encounter an error message).

```
foreach rgvar of varlist cd4_return_* {
    label variable `rgvar' "Does \${name_display2} receive anything in return from this
    employer?"
    }
    label define return_1                    1 "No return"
    label define return_2                    1 "Cash"
    label define return_3                    1 "Goods"
    label define return_4                    1 "Labour"
    label define return_5                    1 "Food or drink"
    label define return_96                   1 "Other, specify"
        forval i=1/5 {
        foreach var of varlist cd4_return_`i'_* {
            capture confirm numeric variable `var'
            if !_rc {
                label values `var' return_`i'
            }
        }
    }
        foreach var of varlist cd4_return_96_* {
            capture confirm numeric variable `var'
            if !_rc {
                label values `var' return_96
            }
        }
```

• Save everything you have done up to now under the name clean_final	• For example: *save Wide_format_22.10/FATE_Rwanda_Part*

	_1_221018_final_WIDE_clean_final, replace

Anonymising Data

- Steps to be performed in each dataset indicated below.
- Once you have finised the data cleaning process, a uniquer identifier has been created for each household and you have made sure that it has been correctly added to each dataset, you can delete all the individual names and other identifying information from the dataset. Be aware that they are spread all over the datasets due to repeat functions. Proceed in the following way in order to find them:

What?	How? (STATA syntax)
• Randomise the village and enumerator codes	• Part 1 & 2, wide and long_final: Run do-files provided by the project
• Make sure to use the correct file	• Part 1 & 2, wide and long_final: *use Wide_format_22.10/ FATE_Rwanda_Part_1_221018_final_WIDE_clean_final_village_enu, clear*
• Find all the variables containing individual names by identifying all the variables with the word "name" in variable name (those are the ones containing people's names)	• Part 1 & 2, all wide and long formats: • *lookfor name* • Or: *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *list `var' if strmatch("`var'","*name*")==1 & `var'!=""* *}*
• Now drop all of those variables	• Part 1 & 2, all wide and long formats: *drop *name**
• Drop other variables containing identifying information and drop old code_ids as they add confusion (adjust relevant variable names according to dataset). Also drop variable avail as unavailable households have been deleted and the variable does not have meaning anymore.	• Part 1, wide and long_final: *drop avail submissiondate code_id today_date starttime end_time *gps* instanceid key setofrepeat** • Part 2, wide and long_final: *drop avail submissiondate code_id a12_starttime a13_datesecondvisit a3_primary end_time *gps* instanceid key*
• Drop automatically created variables with label "reserved_name_for_field_list_labels"	• Part 2, wide and long_final: *lookfor reserved_name_for_field_list_labels* *foreach var of varlist `r(varlist)' {* *drop `var'* *}*
• List all the string variables in order to check that no sensitive information remains (especially in the "other" answers), else you might have to anonymise manually.	• Part 1 & 2, all wide and long formats: *ds, has(type string)* *foreach var of varlist `r(varlist)' {* *count if strmatch(`var',"* *")==1 & regexm(`var',"[a-zA-Z]")==0 | strmatch(`var',"* *")==1 & regexm(`var',"\.a")==1 | strmatch(`var',"* *")==1 & regexm(`var',"\.b")==1* *if r(N)==0 {* *list `var' if `var'!="" & `var'!="kg"*

	} *}*
• List all the value labels to check that they contain no sensitive information (such as the village name which has already been replaced above), else you might have to anonymise manually.	• Part 1 & 2, all wide and long formats: *label list*
• Sort the table by the new identifiers and make sure that there are no duplicates in part 1 and no more than 1 duplicate in part 2	• Part 1 & 2, all wide and long_finals: *sort new_identifier* *duplicates report new_identifier*
• Save everything you have done up to now under the name clean_final_anonymised	• For example: *save Wide_format_22.10/FATE_Rwanda_Part_1_221018_final_WIDE_clean_final_anonymised, replace*

Backup

- Make sure to backup your cleaned datasets properly.
- Related information files (do-files for data cleaning, coding books, etc.) should be included together with the backups. Copies in cloud systems without adequate security (i.e. dropbox, google drive) should be avoided.
- Make sure you still have the encrypted raw-data files and discuss with your supervisor if they should be deleted after the analysis is finished.