Replication Project

Social protection amidst social upheaval: Examining the impact of a multi-faceted program for ultra-poor households in Yemen

Lasse Brune, Dean Karlan, Sikandra Kurdi, Christopher Udry

Written by:
H.Afifi; Y. Ben Hamida; R. Doss; C. Fayad; Z.A.Khan

Paris 1 Panthéon-Sorbonne University
Master 2 (EDD & ED)

2023-2024

Letter to our fellows

Authors
Paris, France

December 22, 2023

Dear Fellow Students,

We are excited to embark on this replication project together, where we will delve into the intricacies of Stata programming techniques. Our journey will be guided by the principles of econometric methods, with the goal of achieving meaningful results as a team. Our focus will be on the seminal work of Laura Brune, Dean Karlan, Salma Kurdi, and Christopher Udry, who explored the impact of social protection programs on ultra-poor households in Yemen. Their study provides an insightful framework for understanding real-world applications of econometrics.

In our project, we will methodically replicate a randomized control trial. This will involve comparing the outcomes of two distinct groups of households: one that received benefits from a social protection program (the treatment group) and another that did not (the control group).
Our aim is to provide you with comprehensive explanations and guidance throughout this educational endeavor. We hope that this experience not only enhances your understanding of econometric methods but also inspires a deeper appreciation for the practical applications of these techniques in addressing global challenges.

Looking forward to a productive and enlightening journey together.

Sincerely,
Y. Ben Hamida; R. Doss; C. Fayad; ZA.Khan ; H.Afifi

Contents

Check the Original Papper !

Before delving into our discussion, we encourage you to review the original paper for a comprehensive understanding. You can access it here.

Reference

Brune, L., Karlan, D., Kurdi, S., & Udry, C. (2022). Social protection amidst social upheaval: Examining the impact of a multi-faceted program for ultra-poor households in Yemen. Journal of Development Economics, 155, 102780.

BibTeX Entry

```
@article{brune2022social,
  title={Social protection amidst social upheaval: Examining the impact of a multi-faceted program for
  author={Brune, Lasse and Karlan, Dean and Kurdi, Sikandra and Udry, Christopher},
  journal={Journal of Development Economics},
  volume={155},
  pages={102780},
  year={2022},
  publisher={Elsevier}
}
```

1 Paper Presentation

Before starting the serious work, let's start by diving through the main outlines of our replication paper and a set a proper framework:

1.1 Summary of the paper

The paper offers a comprehensive analysis of a social protection program designed to assist ultra-poor households in Yemen, particularly during a period of significant social and political upheaval. The main focus of the research is on evaluating the program's effectiveness in enhancing the economic stability and resilience of these households. Key components of the program include enterprise development training, productive asset transfer, savings encouragement, and education in areas such as social awareness, health care, and financial management. This multifaceted approach aims to address the complex challenges of extreme poverty in Yemen. This study thus provides valuable insights into the effectiveness of comprehensive social protection programs in contexts similar to Yemen, contributing to broader poverty alleviation strategies.

1.2 Research question

What is the impact of a comprehensive social protection program on the economic stability and resilience of ultra-poor households in Yemen?

1.3 Methodology

The study employs a randomized control trial method, initially involving 1002 households which were surveyed and then assigned to either a treatment group, receiving program benefits, or a control group. The findings present a nuanced view of the program's impact: notable increases in total assets, especially productive assets like livestock and agricultural tools, and improved savings outcomes were observed in the treatment group. However, these positive changes did not lead to increases in consumption or income, indicating that while the program effectively boosted asset accumulation and savings, it did not immediately improve living standards in terms of higher consumption or income.

1.4 Main findings

The study revealed that households in the treatment group experienced a significant increase in total assets, particularly in productive assets like livestock and agricultural tools, and an improvement in savings outcomes. However, these positive changes did not translate into increased consumption or income. This indicates that while the program was effective in boosting asset accumulation and savings for ultra-poor households, it did not immediately lead to enhanced living standards in terms of higher consumption or income.

1.5 Final remarks about the paper

How this methodology belongs to a 'standard' of its kind ?

The methodology of the study on a social protection program in Yemen aligns with standard randomized control trial (RCT) approaches in several key aspects. Firstly, the study employs randomization at the household level, a fundamental characteristic of RCTs, ensuring comparability between treatment and control groups. This design allows any observed differences to be attributed directly to the intervention. Secondly, the study conducts orthogonality analysis to verify the balance between treatment and control groups, ensuring that baseline variables are evenly distributed. This is crucial to confirm that differences in outcomes are due to the intervention and not pre-existing disparities. Additionally, robustness analysis is performed to address potential biases, particularly focusing on selective attrition, which helps ensure the reliability of the results. The paper also acknowledges the possibility of spillover effects and examines sharing patterns and other mechanisms to understand these impacts, although a quantitative assessment of spillovers is not feasible within the study's design.

How this methodology departs from the 'standard' ?

In contrast, the methodology also diverges from standard RCT practices in various ways. The paper explores the robustness of results considering differential survey attrition rates between the treatment and control groups, an aspect not commonly addressed in standard RCTs. This is achieved through bounding and reweighting exercises to mitigate potential bias from attrition. The study focuses on intent-to-treat estimates rather than the typical per-protocol analysis, analyzing the average treatment effect on all households initially identified as eligible, irrespective of their participation in the program. This approach poses challenges in answering queries about heterogeneous treatment effects and lowers the power for detecting average treatment effects, particularly for outcomes like per-capita consumption or household income. Furthermore, the paper discusses the cost-benefit ratio of the program, highlighting the difficulties in conducting a comprehensive analysis without data covering the entire four-year period. This leads to acknowledged limitations in drawing conclusions for certain outcomes, such as borrowing or food security, reflecting a departure from more conclusive standard RCT methodologies.

2 Replication target

You may ask at this level what are we going to replicate now? In this section, we will set together our work agenda for this replication:

2.1 Table 1: Summary statistics

As a first step, we will need to dive into the data for a better understanding of the paper, so we will start by doing some summary statistics by replication the following table:

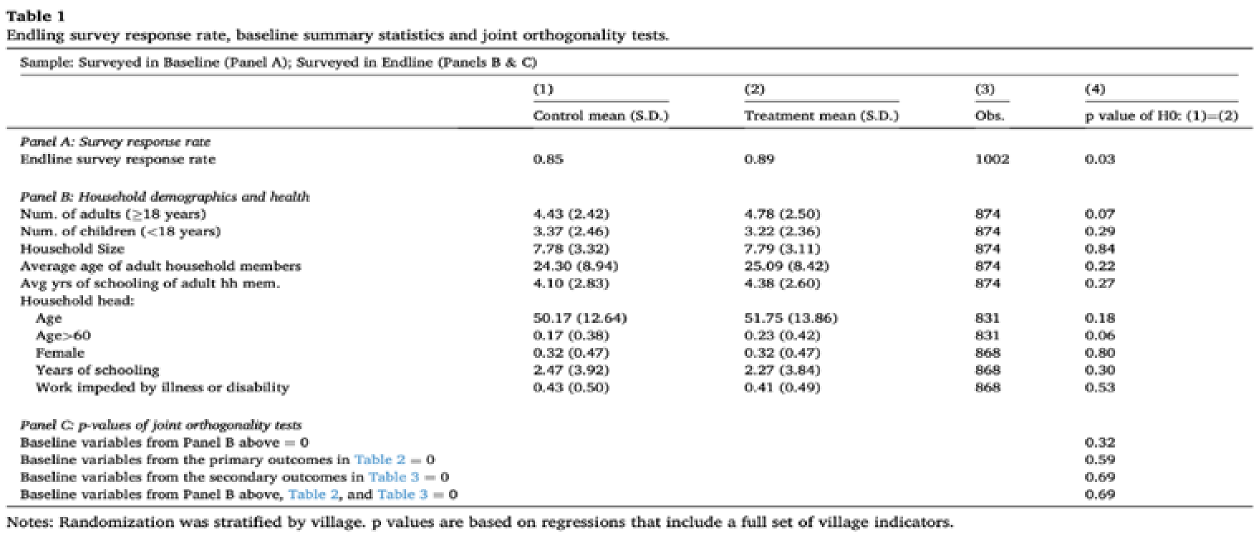

Table 1
Endling survey response rate, baseline summary statistics and joint orthogonality tests.

Sample: Surveyed in Baseline (Panel A); Surveyed in Endline (Panels B & C)

	(1)	(2)	(3)	(4)
	Control mean (S.D.)	Treatment mean (S.D.)	Obs.	p value of H0: (1)=(2)
Panel A: Survey response rate				
Endline survey response rate	0.85	0.89	1002	0.03
Panel B: Household demographics and health				
Num. of adults (≥18 years)	4.43 (2.42)	4.78 (2.50)	874	0.07
Num. of children (<18 years)	3.37 (2.46)	3.22 (2.36)	874	0.29
Household Size	7.78 (3.32)	7.79 (3.11)	874	0.84
Average age of adult household members	24.30 (8.94)	25.09 (8.42)	874	0.22
Avg yrs of schooling of adult hh mem.	4.10 (2.83)	4.38 (2.60)	874	0.27
Household head:				
Age	50.17 (12.64)	51.75 (13.86)	831	0.18
Age>60	0.17 (0.38)	0.23 (0.42)	831	0.06
Female	0.32 (0.47)	0.32 (0.47)	868	0.80
Years of schooling	2.47 (3.92)	2.27 (3.84)	868	0.30
Work impeded by illness or disability	0.43 (0.50)	0.41 (0.49)	868	0.53
Panel C: p-values of joint orthogonality tests				
Baseline variables from Panel B above = 0				0.32
Baseline variables from the primary outcomes in Table 2 = 0				0.59
Baseline variables from the secondary outcomes in Table 3 = 0				0.69
Baseline variables from Panel B above, Table 2, and Table 3 = 0				0.69

Notes: Randomization was stratified by village. p values are based on regressions that include a full set of village indicators.

This table provides important information about the survey response rate, baseline characteristics of the households, and the balance between the treatment and control groups in terms of these baseline variables.

- **Panel A** highlights the end line survey response rates, showing that the treatment group had a higher response rate (89 percentage point) compared to the control group (85 percentage point). This 4-percentage point difference is statistically significant.
- **Panel B** details baseline demographic and health variables of households, such as the number of adults and children, household size, average age and years of schooling of adult members, and characteristics of the household head (age, gender, schooling, and health). These statistics are presented with means and standard deviations for both treatment and control groups.
- **Panel C** shows the results of joint orthogonality tests, assessing if there are significant differences in baseline variables between the two groups. These tests cover various sets of variables, including those in Panel B, primary outcomes in Table 2, secondary outcomes in Table 3, and all combined. The p-values from these tests are all above 0.05, indicating no significant differences between the treatment and control groups in terms of baseline variables, suggesting a balanced distribution across these groups.

2.2 Table 2: treatment effects on key welfare outcomes

In the second step, we will try to get to the serious job by replicating the main treatment effect table:

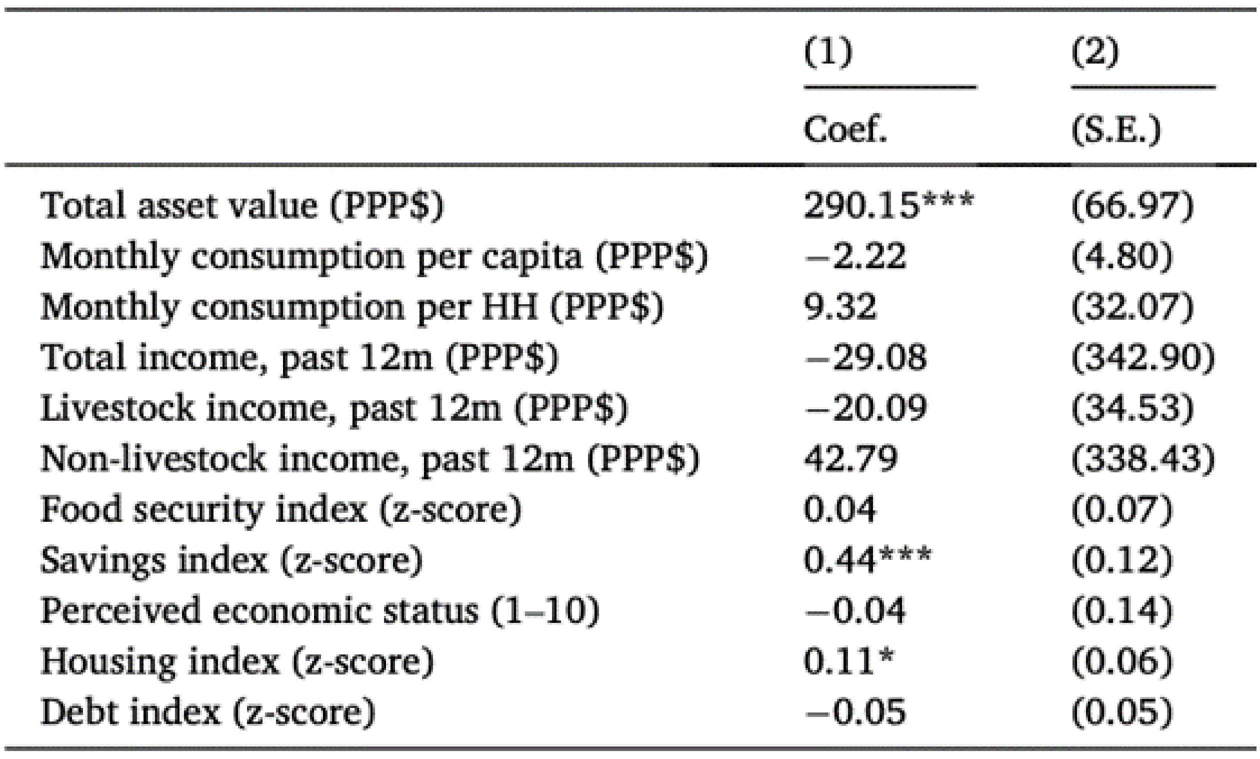

Table 2
Treatment effects on key welfare outcomes.

	(1)	(2)
	Coef.	(S.E.)
Total asset value (PPP$)	290.15***	(66.97)
Monthly consumption per capita (PPP$)	−2.22	(4.80)
Monthly consumption per HH (PPP$)	9.32	(32.07)
Total income, past 12m (PPP$)	−29.08	(342.90)
Livestock income, past 12m (PPP$)	−20.09	(34.53)
Non-livestock income, past 12m (PPP$)	42.79	(338.43)
Food security index (z-score)	0.04	(0.07)
Savings index (z-score)	0.44***	(0.12)
Perceived economic status (1–10)	−0.04	(0.14)
Housing index (z-score)	0.11*	(0.06)
Debt index (z-score)	−0.05	(0.05)

- Total Asset Value : Significant increase in total assets for the treatment group.
- Monthly Consumption Per Capita: No significant effect on individual monthly consumption.
- Monthly Consumption Per Household: No significant effect on household monthly consumption.
- Total Income, Past 12 Months: No significant effect on total income over the past year.
- Livestock Income, Past 12 Months: No significant effect on livestock income over the past year.
- Non-Livestock Income, Past 12 Months : Significant increase in non-livestock income over the past year.
- Food Security Index: No significant impact on food security.
- Saving Index: Improvement in savings habits, though not robust to multiple hypothesis testing adjustments.
- Perceived Economic Status: No significant impact on perceived economic status.
- Housing Index: Information on statistical significance not provided.
- Debt Index: Negative impact on the debt index, but not statistically significant.

The observed treatment effects in the study, specifically on primary outcomes like total assets and the savings index, are reliably linked to the program's objectives and remain robust even after adjustments for multiple hypothesis testing. This robustness indicates that the statistical significance of these effects is not just a result of random variation or the influence of conducting multiple comparisons. By making these adjustments, the researchers have enhanced the credibility of their findings, ensuring that the positive impacts on these key welfare outcomes are indeed attributable to the program and not chance occurrences.

3 Replication codes and explanation

In this section, we will try to go through the replication codes of Stata to be able to get the same results as the framework paper. The work is getting serious now, but don't worry, we will go through this together!

3.1 Table 1: Endline survey response rate and baseline summary statistics

3.1.1 Program definition

```
prog def yemenbalancetable
```

This line defines a Stata program named "yemenbalancetable" that will generate endline survey response rate and baseline summary statistics for demographic variables, comparing control and treatment groups means

3.1.2 syntax definition

```
syntax varlist [if/], /// Variables to be included in the table [filename(string)] /// File name (default is TableX) [foldername(string)] /// Output folder name (default is Replication/Output) [title(string)] /// Table title [footnote(string)] /// Footnote text [winsorize] /// Winsorize variables [balancevar(varlist)] /// Balance variable (default is treatment)
```

This section defines the syntax for the program, specifying the input parameters such as variables, file name, output folder name, table title, footnote, whether to winsorize variables, and the balance variable.

3.1.3 Prepare table inputs

```
// FOLDER NAME
// Set default folder to Replication/Output if not specified
if "‘foldername’"=="" {
loc foldername "${rep_output}" }
```

Checks if a folder name is specified; if not, sets the default folder to "Replication/Output."

```
// FILE NAME
// Set a default filename
if "‘filename’"=="" {
loc filename "Table1"}
if "‘winsorize’"=="winsorize" {
loc filename "‘filename’_win"}
if "‘winsorize’"=="winsorize" {
loc filename "‘filename’_win"}
```

This line checks if a filename is specified; if not, sets the default filename to "Table1" or appends "win" if winsorization is selected.

```
// WINSORIZE
// Edit varlist to include winsorized variable names if "winsorize" selected
if "‘winsorize’"=="winsorize" {
    loc templist ‘varlist’
loc varlist // Clear varlist
foreach var in ‘templist’ {
loc currvar = subinstr("‘var’","_bsl","_win1_bsl",.) // Add "_win1" suffix
loc varlist ‘varlist’ ‘currvar’ // Add to varlist if "winsorize" is selected
}
}
if "‘winsorize’"=="winsorize" {
loc filename "‘filename’_win"}
```

If winsorization is selected, this code line modifies the variable list to include winsorized variables.

3.1.4 Create Table Framework

```
clear all
eststo clear
estimates drop _all
```

This line clears existing data, stored estimates, and drops all existing estimates

```
loc columns = 5 //Change the number of columns
set obs 10
gen x = 1
gen y = 1
```

Sets the number of columns in the table, creates a dataset with 10 observations, and generates dummy variables x and y.

```
forval i = 1/‘columns’ {
eststo col‘i’: qui reg x y
}
```

Runs a loop to estimate regression models for x and y and stores the results in separate matrices (”col1”, ”col2”, ..., ”col5”).

```
loc count = 1
loc countamt = ‘count’ + 1
```

Initializes counters for loop control.

```
loc stats "" // Added scalars to be filled
loc varlabels "" // Labels for row vars to be filled
```

Initializes empty locals to store statistical values and variable labels

```
use ${dirdata}hh_yemen_analysis, clear
```

Imports the dataset for analysis.

Fill Table Cells

```
loc i = 0 // Start a counter
foreach var in ‘varlist’ {
    // ...
}
```

The first line initiates a loop to iterate over each variable in the variable list.

The following code within the loop calculates statistics for different categories (full sample, balance variable, control, p-values, observations) and updates locals.

3.1.5 Export table

```
cd "‘foldername’" // Call the output folder directory
```

Changes the current directory to the specified output folder.

```
esttab col* using "‘filename’.csv", title("‘title’") cells(none) ///
mtitle( "Full Sample" "‘label1’" "‘label2’" ///
"‘label3’" ///
"Total N") stats(‘stats’, labels(‘varlabels’)) ///
note("‘footnote’")  ///
compress wrap lines nonum replace plain
```

Exports the table to a CSV file with the specified title, column titles, and statistical values.

```
eststo clear
end
```

These two lines clear stored estimates and end the program definition.

[breaklines=true] yemenbalancetable num$_adults_bslnb_children_bslhhsize_bslavg_age_bslavg_edu_years_bslage_head_bslhh_head_over6$
$1, ///filename(”Table1”)title(”Table1 : BalanceTestofTreatmentAssignmentforEndlineHouseholdsOnly”)///winsor$

Calls the defined program ”yemenbalancetable” with specific variable list, conditions, and options:

- yemenbalancetable: Calls the program named ”yemenbalancetable” that was defined earlier in the code.
- `Variables`: Lists the variables to be included in the analysis. These variables are num_adults_bsl, nb_children_bsl, hhsize_bsl, avg_age_bsl, avg_edu_years_bsl, age_head_bsl, hh_head_over60_bsl, gender_head_bsl, hh_head_educ_bsl, and disabl_head_bsl.
- `if endline_survey==1`: Specifies a condition for including only observations where the variable endline_survey is equal to 1. This condition filters the data for endline households only.
- `filename("Table1")`: Specifies the output filename as ”Table1.”
- `title("Table 1:  Balance Test of Treatment Assignment for Endline Households Only")`: Sets the title for the table as ”Table 1: Balance Test of Treatment Assignment for Endline Households Only.”
- `winsorize`: Activates the winsorization option, indicating that the program should winsorize the specified variables.
- `foldername("rep_output")`: Sets the output folder name as ”rep_output,” where rep_output is the macro containing the directory path for the output.

3.2 Replicate Table2: Treatment effect on key welfare outcomes

3.2.1 Prepare table inputs

Folder name

Comment on preparing table inputs:

```
// PREPARE TABLE INPUTS
```

This is a comment for readability, indicating that the following lines of code are used for preparing inputs for the table generation process.

Comment on Folder Name:

```
// FOLDER NAME
//  Set default folder to Replication/Output if not specified
```

Another comment explaining that the next line of code will set a default folder path for saving the output, specifically to a folder named ”Replication/Output” if no other folder name is specified.

Conditional Folder Name Setting:

```
if "‘foldername’"=="" {
loc foldername "${rep_output}"
}
```

- This line checks if the macro ‘foldername’ is empty (i.e., if no folder name has been specified earlier in the script).
- `If ‘foldername’ is indeed empty, the line sets ‘foldername’ to the value stored in another macro called ‘${rep_output}’.`
- `The ${rep_output} macro presumably contains the default path where the output should be saved, which in this context is likely to be something like "E:/.../Replication/Output".`
- `The ‘loc’ command is short for local, which in Stata is used to define a local macro. A local macro is a sort of variable that can store text, which in this case, is being used to store a file path.`

In summary, this segment ensures that there is always a specified folder path to save the output of the script. If the user does not specify this path, the script defaults to a predefined path stored in the repoutput macro. This approach helps in automating the process and makes the script more user-friendly by reducing the need for manual input of file paths.

File name

This section of the code is designed to set up the filename for the output table in a statistical analysis. It ensures that the table is saved with an appropriate and descriptive name, reflecting various data processing steps like winsorization, baseline value adjustments, and additional controls. Comment on File Name:

```
// FILE NAME
//  Set a default filename
```

These lines are comments for readability, indicating that the following code will be used for setting a default filename for the output file.

Setting Default Filename:

```
if "‘filename’"=="" {
loc filename "TableX"
}
```

- This line checks if the filename macro is empty (meaning no filename has been previously specified).
- If it is empty, it sets the filename macro to a default value ”TableX”. This is done using the local (abbreviated as loc) command, which is used in Stata to define a temporary macro.

Adjusting Filename for Winsorization:

```
if "‘winsorize’"=="winsorize" {
loc filename "‘filename’_win"
}
```

- This line checks if the data needs to be winsorized (a process to limit extreme values in the data for more robust statistical analysis).
- If winsorization is specified (”winsorize”), it modifies the filename macro by appending win to its current value. This helps to indicate that the output file contains results from winsorized data.

Appending Baseline Values Indicator to Filename:

```
if "‘baselinevals’"=="baselinevals" {
loc filename "‘filename’_blv"
}
```

- This line checks if baseline values are included in the analysis.
- If so, it appends blv to the current filename macro value to indicate that the file includes baseline value adjustments.

Including Additional Controls in Filename:

```
if "‘addlcontrols’"!="" {
loc filename "‘filename’_ctl"
}
```

- This line checks if there are any additional controls specified in the analysis.
- If additional controls are included (indicated by `addlcontrols` not being empty), it appends `_ctl` to the filename to reflect that the output file includes these additional controls.

In summary, these lines of code systematically adjust the filename for the output file based on specific data processing steps taken in the analysis. This naming convention makes it easier to understand what processing has been applied just by looking at the filename, thereby enhancing the clarity and manageability of the output files.

WINSORIZE

This code segment is part of a statistical analysis script in Stata, specifically designed to modify variable names for winsorization.

Reminder: What is Winsorization ?: Winsorization is a technique used to reduce the effect of extreme outliers in data by replacing these extreme values with less extreme ones. This segment of the code adjusts the list of variables (varlist) to reflect whether they have been winsorized.

Comment on Winsorization:

```
// WINSORIZE
// Edit varlist to include winsorized variable names if "winsorize" selected
```

These comments explain that the following lines of code deal with editing the variable list for winsorization, indicating the purpose of the code block.

Temporary List Creation:

```
loc templist ‘varlist’
```

Creates a temporary list (templist) that duplicates the original varlist. This is a common practice to preserve the original list while making modifications to the copy.

Clearing the Original Variable List:

```
loc varlist // Clear varlist
```

Clears the original varlist. This is done to rebuild this list with modified variable names as needed.

And now will detail the following command:

```
foreach var in ‘templist’ {
loc currvar = subinstr("‘var’","_end","",.) // Cut out "_end" suffix
if "‘winsorize’"=="winsorize" {
loc varlist ‘varlist’ ‘currvar’_win1 // Add "_win1" suffix if "winsorize" is selected
}
if "‘winsorize’"=="" {
loc varlist ‘varlist’ ‘currvar’
}
}
```

Loop Through Each Variable:

```
foreach var in ‘templist’ {
    ...
}
```

Begins a loop to iterate through each variable in the temporary list (templist). This loop allows the script to process each variable one by one.

Modifying Variable Names for Winsorization:

```
loc currvar = subinstr("‘var’","_end","",.) // Cut out "_end" suffix
```

For each variable (var) in templist, this line removes any ”end” suffix from its name. This is done using the subinstr function, which substitutes part of a string with another string (in this case, replacing ”end” with nothing).

Appending Winsorization Suffix:

```
if "‘winsorize’"=="winsorize" {
    loc varlist ‘varlist’ ‘currvar’_win1 // Add "_win1" suffix if "winsorize" is selected
}
```

If winsorization is selected (indicated by the macro winsorize being equal to ”winsorize”), this line appends ”win1” to the current variable name (currvar). This indicates that the variable has been winsorized.

Handling Non-Winsorized Variables:

```
if "‘winsorize’"=="" {
    loc varlist ‘varlist’ ‘currvar’
}
```

If winsorization is not selected, this line adds the current variable name (currvar) to the varlist without any modification.

In summary, this code segment is responsible for updating the list of variables in the dataset to reflect whether they have been subjected to winsorization. This is crucial for accurate data analysis, as it ensures that any statistical procedures applied later in the script are aware of which variables have been modified to limit the impact of outliers.

ADDITIONAL CONTROLS

This code segment from a Stata script is designed to handle additional control variables in a dataset, particularly in the context of winsorization. Winsorization is a statistical technique used to reduce the influence of extreme values. The script modifies the names of these additional control variables to reflect whether they have been winsorized.

Comment on Additional Controls:

```
// ADDITIONAL CONTROLS
```

This comment indicates that the following lines of code will manage additional control variables in the analysis.

Creation of Temporary List for Additional Controls:

```
loc templist ‘addlcontrols’
```

Creates a temporary list named templist that contains the variables specified in addlcontrols. This list will be used to process each additional control variable.

Clearing the Original Additional Controls List:

```
loc addlcontrols
```

Clears the existing addlcontrols list. Why is this command important?: In order to repopulate it later with modified variable names.

And now will detail the following command:

```
foreach var in ‘templist’ {
loc currvar = subinstr("‘var’","_end","",.) // Cut out "_end" suffix
if "‘winsorize’"=="winsorize" {
loc varlist ‘varlist’ ‘currvar’_win1 // Add "_win1" suffix if "winsorize" is selected
}
if "‘winsorize’"=="" {
loc varlist ‘varlist’ ‘currvar’
}
}
```

Looping Through Each Variable in the Temporary List:

```
foreach var in ‘templist’ {
    ...
}
```

Starts a loop that will iterate through each variable in the templist. This allows for individual processing of each additional control variable.

Modifying Variable Names for Winsorization:

```
loc currvar = subinstr("‘var’","_bsl","_win1_bsl",.) // Cut out "_end" suffix
```

For each variable in templist, this line replaces the ”bsl” suffix with ”win1bsl”. The bsl likely indicates a baseline measurement, and the win1bsl indicates a winsorized baseline variable. The subinstr function is used for this string substitution.

Appending Winsorization Suffix for Additional Controls:

```
if "‘winsorize’"=="winsorize" {
    loc addlcontrols ‘addlcontrols’ ‘currvar’
    m_‘currvar’ // Add "_win1" suffix if "winsorize" is selected
}
```

If winsorization is selected, this line appends the modified variable name (currvar) and its ’missing’ indicator version (mcurrvar) to the addlcontrols‘ list. This step indicates that the variable and its associated missing indicator have been winsorized.

Handling Non-Winsorized Additional Controls:

```
if "‘winsorize’"=="" {
    loc varlist ‘addlcontrols’ ‘var’ m_‘var’
}
```

If winsorization is not applied, this line adds both the original variable name (var) and its ’missing’ indicator (mvar) to the addlcontrols‘ list without any modification.

All in all, this script segment deals with preparing additional control variables, specifically adjusting their names to reflect whether they have been winsorized. This is important for accurate data handling in later statistical analyses, ensuring that the analysis accurately reflects the treatment of these variables.

Fixed effects

This snippet of Stata code is designed to set up fixed effects for a statistical model.

What are fixed effects used for? : Fixed effects are used in statistical analyses to control for variables that could affect the results but are not the primary focus of the research.

In this case, the script sets the village as the default fixed effect, provided that no other fixed effects have been specified.

Comment on Fixed Effects:

```
// FIXED EFFECTS
//  Set fixed effects to village, if not specified
```

These comments explain the purpose of the following line of code.

What is their purpose?: To set a default fixed effect for the analysis if none is already specified. The default fixed effect is implied to be based on the ’village’ variable.

Setting Default Fixed Effects:

```
if "‘fixedeffects’"=="" {
    loc fixedeffects village
}
```

- This line checks whether the fixedeffects macro is empty. The fixedeffects macro is presumably intended to store the names of variables that should be used as fixed effects in the analysis.
- If fixedeffects is indeed empty (i.e., no fixed effects have been specified previously), the script sets fixedeffects to village using the local command (loc). This means that the variable village will be used as the default fixed effect in subsequent analyses.
- Using the village as a fixed effect would control for any unobserved characteristics specific to each village that might influence the outcome being studied.

NB: This is a common approach in analyses where data is collected from different geographic locations or groups and where these locations or groups might have unique characteristics that could affect the results.

In summary, this code ensures that there is a default fixed effect (village) in the statistical model, which is crucial for controlling for location-specific variables that could otherwise bias the analysis. This is particularly relevant in studies where geographic or group-specific factors are significant but are not the main focus of the research.

STANDARD ERRORS

This segment of Stata code configures the default setting for calculating standard errors in a statistical analysis. Standard errors are a crucial part of statistical analyses, as they measure the variability or uncertainty in the estimated parameters. The script specifically sets the standard errors to be 'robust,' provided that no other type of standard errors have been specified.

Comment on Standard Errors:

```
// STANDARD ERRORS
// Default standard errors to robust
```

These comments explain that the following line of code sets a default method for calculating standard errors in the statistical analysis. The default method mentioned here is 'robust' standard errors.

Setting Default Standard Errors:

```
if "‘stderrors’"=="" {
    loc stderrors robust
}
```

- This line checks if the stderrors macro is empty, meaning no specific method for calculating standard errors has been defined earlier in the script.
- If stderrors is indeed empty, it sets stderrors to 'robust' using the local command (loc). This implies that, in the absence of a specified method, the script will use 'robust' standard errors for its statistical analyses.
- Robust standard errors, often used in regression analyses, are designed to provide more accurate standard error estimates when the standard assumptions (like homoscedasticity, where the variance of the error term is constant across observations) do not hold. They help to maintain the validity of hypothesis tests even when there are violations in these assumptions.

In summary, this code ensures that if no specific method for calculating standard errors is defined, the script will default to using 'robust' standard errors. This is a prudent choice in many empirical analyses, as it provides an extra layer of protection against certain types of model misspecification.

IF-STATEMENT

This portion of the Stata code is designed to manage conditional statements within the script. Conditional statements in statistical programming are used to apply certain operations or analyses only to specific subsets of the data, based on defined criteria. In this case, the code handles an if condition provided by the user.

Comment on IF-Statement:

```
// IF-STATEMENT
```

This comment indicates that the following lines are related to setting up an if statement, a common programming construct used for conditional execution of code.

Setting Up Conditional Statement

```
if "‘if’"!="" {
    loc ifif "if ‘if’"
    loc andif "& ‘if’"
}
```

- This block checks whether the macro if is not empty (”!=””). The if macro is used here to store a condition that determines which subset of the data should be analyzed or manipulated. For example, it could be a condition like age ¿ 30 to include only observations where age is greater than 30.
- If the if macro is not empty, meaning a condition has been specified, two local macros are created:
  - ifif: This macro is set to ”if if’”‘, effectively creating a conditional statement that can be used directly in Stata commands. It allows for applying operations only to observations that meet the specified condition.
  - `andif`: Similar to `ifif`, but prefixed with ”& if”. This is useful for adding the condition as an additional clause in commands where other conditions might already be present. The ampersand (&) is a logical AND operator in Stata, used for combining multiple conditions.

In summary, this code segment prepares the script to handle user-defined conditions, enabling selective analysis of data based on specific criteria. It ensures that subsequent operations or analyses in the script are applied only to the subset of data that meets the defined condition, enhancing the script’s flexibility and functionality.

BALANCE TESTING This segment of the Stata code is designed to configure settings for balance testing in a statistical analysis. Balance testing is often used in experimental or quasi-experimental designs to ensure that different groups (like treatment and control groups) are comparable on various characteristics. This code sets up the necessary parameters for conducting such balance tests, contingent on specific user-defined conditions.

Comment on Balance Testing

```
// BALANCE TESTING
```

This comment indicates that the ensuing lines of code are related to setting up balance testing in the analysis. Balance testing is a crucial step in many statistical analyses, particularly in observational studies or experiments, to check for equivalency between groups.

Conditional Setup for Balance Testing:

```
if "‘balance’"!="" {
    loc balancevar ‘balance’
    loc balance balance
}
```

- This block checks if the macro balance is not empty. The balance macro is expected to contain a condition or a variable name that is relevant for conducting balance testing. For example, it could specify a treatment indicator variable.
- If the balance macro is not empty (i.e., a balance condition or variable is specified):
  - The line loc balancevar balance’creates a local macrobalancevarand assigns it the value stored in thebalance‘ macro. This step essentially stores the specified balance testing variable or condition for use in subsequent analyses.
  - The line loc balance balance seems redundant since it creates a local macro balance with the same value it already holds. This might be intended for clarity or to ensure consistency in macro naming conventions across different parts of the script.

In summary, this code segment prepares the statistical analysis for balance testing by setting up the necessary parameters based on user-defined inputs. It ensures that the script can perform appropriate balance tests, which are essential for validating the comparability of different groups in the study.

FOOTNOTES

This section of Stata code is designed to generate footnotes for a statistical output, typically a table or a report.

What do these footnotes provide? These footnotes provide important information regarding specific data processing steps applied during the analysis, such as winsorization, inclusion of baseline values, or additional control variables. Accurate and informative footnotes are crucial for the transparency and clarity of statistical results.

Comment on Footnotes:

```
// FOOTNOTES
// Add footnotes about baseline values and additional controls
```

This comment explains that the upcoming lines of code will add footnotes to a document or table. The purpose of these footnotes is to clarify certain aspects of the data processing, specifically regarding baseline values and additional controls.

Adding Footnote for Winsorization:

```
if "‘winsorize’"=="winsorize" {
    loc footnote "‘footnote’ Variables winsorized at 1%."
}
```

- This line checks if the winsorize macro is set to ”winsorize”, indicating that winsorization has been applied to the variables in the analysis.
- If winsorization is applied, it appends a statement to the existing footnote macro (or creates it if it doesn’t exist), specifying that the variables have been winsorized at 1 percentage point. Winsorization is a method of limiting extreme values to reduce their impact, and specifying the level (1 percentage point in this case) is important for interpreting the results.

Adding Footnote for Baseline Values:

```
if "‘baselinevals’"=="baselinevals" {
    loc footnote "‘footnote’ Controls for baseline value of dependent variable."
}
```

- This line checks if the baselinevals macro is set to ”baselinevals”, indicating that the analysis includes controls for baseline values of the dependent variable.
- If baseline values are included, it appends a note to the footnote macro stating this fact. This is important for readers to understand the adjustments or controls applied in the analysis

Adding Footnote for Additional Controls:

```
if "‘addlcontrols’"!="" {
    loc footnote "‘footnote’ Controls for additional variables."
}
```

- This line checks if the addlcontrols macro is not empty, meaning additional control variables have been specified for the analysis.
- If there are additional controls, it appends a note to the footnote macro to indicate that the analysis controls for these additional variables.

In summary, these lines of code efficiently handle the addition of explanatory footnotes to statistical outputs. They ensure that crucial information about the data processing steps, like winsorization, baseline adjustments, and additional controls, is clearly communicated to anyone reviewing the results, enhancing the transparency and understanding of the analysis.

3.2.2 CREATE TABLE FRAMEWORK

This segment of Stata code is focused on creating the framework for a table intended for statistical analysis. It involves clearing previous results, setting up a new data structure, and initializing various elements necessary for filling the table with appropriate statistical measures.

Comment on Creating Table Framework:

```
// CREATE TABLE FRAMEWORK
```

This comment indicates that the following lines of code are dedicated to setting up the structure for a statistical table.

Clear estimates

Clearing Previous Estimates:

```
clear
eststo clear
estimates drop _all
```

- **clear**: Clears the current dataset from memory.
- **eststo clear**: Clears any stored estimation results.
- estimates drop all: Drops all stored estimates.

Create blank table

Creating a Blank Table

```
set obs 10
gen x = 1
gen y = 1
```

- set obs 10: Sets the number of observations in the dataset to 10.
- gen x = 1 and gen y = 1: Generates two new variables, x and y, both filled with the value 1. This is a preparatory step for creating placeholder regressions.

Count outcomes and create placeholder estimates

Counting Outcomes and Creating Placeholder Estimates

```
loc columns: word count ‘varlist’

forval i = 1/‘columns’ {
    eststo col‘i’: qui reg x y
}
```

- loc columns: word count varlist’: Counts the number of variables in varlistand stores this number in the local macrocolumns‘.
- The forval loop creates placeholder regression models for each variable in varlist. This is a preparatory step for later analyses.

And now will detail the following command:

```
loc j = 1
loc treatcoef = ‘j’// Treatment starred coefficient
loc ++j
loc treatse = ‘j’// Treatment standard error
loc ++j
loc contmean = ‘j’// Control mean
loc ++j
loc contsd = ‘j’// Control standard deviation
loc ++j
loc obs = ‘j’// Observations
loc ++j
loc blcontrols  = ‘j’// Binary for baseline controls
loc ++j
loc numzero = ‘j’// Count with outcome == 0
```

Set table cell locals

Setting Table Cell Locals:

```
loc j = 1
loc treatcoef = ‘j’    // Treatment starred coefficient
...
loc numzero = ‘j’      // Count with outcome == 0
```

These lines initialize a series of local macros (treatcoef, treatse, contmean, etc.) to keep track of various statistical measures (like treatment effect, standard errors, means, standard deviations) for later use in the table.

Balance testing Balance Testing Setup:

```
if "‘balance’"=="balance" {
    loc ++j
    loc baltest = ‘j’    // Balance test
    loc ++j
    loc balproxy = ‘j’   // Proxy Flag
    loc balword1 "Balance Test p-Value"
    loc balword2 "Proxy for Balance"
}
```

If balance testing is required (indicated by the balance macro), additional locals (baltest, balproxy) are created for storing balance test results.

Initializing Scalars and Column Titles:

```
loc stats ""            // Added scalars to be filled
loc col_titles ""       // Labels for columns vars to be filled
```

Two locals, stats and coltitles, are initialized as empty. They will later be used to store statistical results and column labels for the table.

Loading the Dataset:

```
use ${dirdata}hh_yemen_analysis, clear
```

Loads a specified dataset (hhyemenanalysis) from a directory path stored in the macro dirdata, clearing any existing data in memory.

In summary, this code segment efficiently sets up the necessary infrastructure for creating a detailed statistical table. It involves initializing the dataset, setting up placeholder variables, and preparing locals for various statistical measures that will be calculated and displayed in the table. This setup is crucial for the subsequent steps in the analysis, where actual data will be processed and results populated into this framework.

3.2.3 REPLACE MISSINGS WITH ZEROS

This section of the Stata code focuses on handling missing data within the dataset, specifically for additional control variables.

Why is it important to manage missing data?: In statistical analyses, managing missing data is crucial as it can impact the results and interpretations. This code replaces missing values in the specified additional control variables with zeros.

Comment on Replacing Missings with Zeros:

```
// REPLACE MISSINGS WITH ZEROS
```

This comment indicates that the following lines of code are designed to handle missing data within the dataset.

ADDITIONAL CONTROLS Comment on Additional Controls:

```
// ADDITIONAL CONTROLS
//  Replace missings with zeroes
```

This comment further specifies that the operation of replacing missing values with zeros is particularly focused on the additional control variables. These are variables that might be used in the analysis to control for other factors, ensuring that the main effects studied are not confounded by these additional variables.

Looping Through Additional Controls:

```
foreach var in ‘addlcontrols’ {
    replace ‘var’ = 0 if missing(‘var’) // Replace to 0 if missing
}
```

- The foreach loop iterates through each variable listed in the addlcontrols macro. This macro is expected to contain the names of the additional control variables.
- Within the loop, the replace command is used for each variable (var). It sets the value of the variable to 0 wherever it is missing (missing(var')‘). This step is crucial as it ensures that any analysis involving these variables does not get skewed or halted due to missing data.

In summary, this code segment efficiently handles missing data in additional control variables by replacing missing values with zeros. This approach is a common method in data preparation, particularly useful when the absence of data can be reasonably substituted with a neutral value like zero, ensuring continuity and completeness of the dataset for further analysis.

FILL TABLE CELLS This section of the Stata code is aimed at populating a table with statistical measures, such as coefficients, standard errors, and p-values. This is a crucial part of the data analysis process where results are organized into a structured format for interpretation and presentation.

Comment on Filling Table Cells:

```
// FILL TABLE CELLS
// Table cells include coefficients, standard errors, p-values
```

These comments introduce the objective of the following code: to fill the cells of a table with important statistical measures. Specifically, the table will include coefficients (indicative of the magnitude and direction of relationships in the data), standard errors (measuring the precision of the coefficients), and p-values (indicating the statistical significance of the results).

Initializing a Counter:

```
loc i = 1 // Start a counter
```

This line initializes a local macro i with a value of 1. This macro serves as a counter that will be used to iterate through the variables and to keep track of the position or index within the table where results will be placed.

Loop Through Table Variables:

```
foreach y_var in ‘varlist’ { // Loop through table variables
```

- • Begins a foreach loop that iterates over each variable in varlist. The varlist macro is expected to contain a list of variables for which the statistical analysis is being conducted.

- yvar is a placeholder within the loop, representing each variable in varlist as the loop progresses. For each iteration of the loop, yvar will take on the value of the next variable in varlist, and the statistical measures for that variable will be calculated and added to the table.

To conclude, this code segment sets up a loop to systematically process a list of variables, calculating and recording key statistical measures for each one.

The loop, guided by the counter i, ensures that all relevant variables in varlist are included in the analysis and that their corresponding results are accurately placed in the table

3.2.4 BASELINE CONTROLS

This section of the Stata code deals with the handling of baseline control variables in a statistical analysis.

What are Baseline controls ?: Baseline controls are variables measured at the beginning of a study and are crucial for ensuring accurate and meaningful comparisons over time or between groups.

This code manages missing data in baseline variables, creates placeholders for missing baseline data, and notes whether baseline controls are utilized in the analysis.

Comment on Baseline Controls:

```
// BASELINE CONTROLS
//  Replace missings with zeroes
//  Add "missing" dummies to controls
```

These comments explain that the following lines of code will handle baseline control variables, specifically focusing on replacing missing values and adding dummy variables for missing data.
Check whether baseline value exists Checking for Baseline Variable:

```
cap confirm variable ‘y_var’_bsl
```

This line uses the ‘confirm’ command (prefixed with ‘cap’ to capture any errors without stopping the script) to check if a baseline version of the current variable (denoted as ‘yvar'bsl’) exists in the dataset.
NB: The bsl suffix likely indicates a baseline measurement.
If baseline doesn’t exist Handling Non-Existent Baseline Variable:

```
if _rc {
    loc proxyflag = 1
    loc created_var = 1
    gen ‘y_var’_bsl = 1          // Create stand-in baseline var
    gen m_‘y_var’_bsl = 1        // Create stand-in baseline dummy var
    estadd loc stat‘blcontrols’ = "No" : col‘i’ // Note that baseline controls not used
}
```

- • This block is executed if the baseline variable does not exist (indicated by a non-zero return code rc from the previous confirm command).
- It creates a new baseline variable (yvar’bsl) and a corresponding dummy variable (myvar’bsl‘) filled with ones. These are placeholders indicating the absence of actual baseline data.
- The script also notes that baseline controls are not used in this case by adding a note (”No”) to a local macro (statblcontrols’).

If baseline exists Handling Existing Baseline Variable:

```
else if !_rc {
    loc proxyflag = 0
    loc created_var = 0
    replace ‘y_var’_bsl = 0 if missing(‘y_var’_bsl)
    // Replace baseline to 0 if missing
    if ("‘baselinevals’"=="baselinevals")
        estadd loc stat‘blcontrols’ = "Yes" : col‘i’
    // Note baseline controls used
    else
        estadd loc stat‘blcontrols’ = "No" : col‘i’
    // Note baseline controls not used
}
```

- If the baseline variable exists (rc equals zero), this block replaces missing values in the baseline variable with zeros.
- It also updates a local macro to indicate whether baseline controls are used in the analysis, based on whether baselinevals is set.

Setting Local Macro for Baseline Variables:

```
if "‘baselinevals’"=="baselinevals" local blvals ‘y_var’_bsl m_‘y_var’_bsl
else local blvals // Empty
```

This part sets the local macro blvals to include the baseline variable and its dummy counterpart if baseline values are being used in the analysis (”baselinevals” is set). If not, blvals is left empty.

In summary, this code segment meticulously handles baseline control variables in the analysis, addressing missing data and creating appropriate placeholders. It ensures that the analysis correctly accounts for the presence or absence of baseline data, which is vital for the integrity and accuracy of the statistical results.

3.2.5 RUN REGRESSION

This section of the Stata code is focused on running various regression models as part of the statistical analysis. The script includes different types of regression analyses based on specific conditions, such as :

1. Treatment on Treated (ToT)
2. ”Simple” inverse probability weighting (simple IPW)
3. ”Fancy” inverse probability weighting (fancy IPW)
4. Main / default regression

Let’s develop each one of them, starting first by:

```
// RUN REGRESSION

// TREATMENT ON TREATED (ToT)
if "‘tot’"=="tot" {
xi: ivreg ‘y_var’_end ‘blvals’ ‘addlcontrols’ i.‘fixedeffects’ (treat_received = treatment), r
loc coef "treat_received"
loc main "false"
}
```

The lines of codes indicate the following:

```
// RUN REGRESSION
```

This comment indicates the beginning of the regression analysis section of the code.

TREATMENT ON TREATED (ToT)

Treatment on Treated (ToT) Regression:

```
if "‘tot’"=="tot" {
    xi: ivreg ‘y_var’_end ‘blvals’ ‘addlcontrols’
    i.‘fixedeffects’ (treat_received = treatment), r
    loc coef "treat_received"
    loc main "false"
}
```

- • if ”tot’”==”tot”: This line checks if the tot‘ macro is set to ”tot”. If yes, it proceeds to run the Treatment on Treated regression. This conditional approach allows the code to selectively execute the ToT regression based on the user’s specification.
- xi: ivreg: The xi: prefix is used for handling categorical variables, which might be present in fixedeffects or other parts of the model. ivreg (instrumental variables regression) is then executed. This command is used to address potential endogeneity issues by using instruments.
- yvar’end ‘blvals’ ‘addlcontrols’ i.‘fixedeffects’ (treatreceived = treatment), r: This part of the command specifies the regression model. It regresses the endline outcome variable (yvar’end) on the baseline values (blvals), additional control variables (addlcontrols), and includes fixed effects (fixedeffects). The part (treatreceived = treatment) indicates that treatreceived is an instrumental variable for treatment. The r option at the end specifies that robust standard errors should be used.
- loc coef ”treatreceived”: Creates a local macro coef and sets it to ”treatreceived”. This is likely used later in the script to reference the coefficient of interest in the model.
- loc main ”false”: Sets a local macro main to ”false”. This might be a flag used later in the script to control the flow of the code or to determine which sections of the code to execute.

In summary, this code segment is essential for conducting a Treatment on Treated analysis within a broader statistical study.

It carefully sets up an instrumental variables regression to estimate the effect of treatment on a specific subset of the population, while controlling for various other factors and potential confounders. The use of local macros (coef and main) indicates that these results feed into subsequent steps or conditional operations in the analysis.

”SIMPLE” INVERSE PROBABILITY WEIGHTING (Simple IPW)

This section of the Stata code focuses on implementing a ”Simple” Inverse Probability Weighting (Simple IPW) regression. Inverse Probability Weighting is a statistical technique used to adjust for potential sample bias, particularly in observational studies. This method aims to create a pseudo-population where the treatment assignment is independent of the measured covariates.

Simple IPW Conditional Check:

```
if "‘simpleipw’"=="simpleipw"{
```

Checks if the macro simpleipw is set to ”simpleipw”, indicating that a Simple IPW regression should be executed.

1. Drop existing inverse probability weighting variable

   Drop Existing IPW Variable:

   ```
   cap drop ‘ipw_simple3’
   ```

Attempts to drop any existing variable named ipwsimple3 (which is presumably used for IPW calculations) from the dataset. The cap command is used to capture any errors (for instance, if the variable doesn't exist) without stopping the script.

Create Temporary IPW Variable:

```
tempvar ipw_simple3
gen ‘ipw_simple3’ = .x
```

Creates a new temporary variable ipwsimple3 and initializes it with missing values (.x). This variable will be used to store the inverse probability weights.

2. For each group, grab counts
   After this, we will provide explanation of the following:

```
count if treatment==0
local N_c = r(N)
count if !mi(‘y_var’_end) & treatment==0
local N_c_found = r(N)

count if treatment==1 & eligible==1
local N_t_elig = r(N)
count if !mi(‘y_var’_end) & treatment==1 & eligible==1
local N_t_elig_found = r(N)

count if treatment==1 & eligible==0
local N_t_notElig = r(N)
count if !mi(‘y_var’_end) & treatment==1 & eligible==0
local N_t_notElig_found = r(N)
```

   Counting and Calculating Weights:

```
count if treatment==0
local N_c = r(N)
...
```

   - This series of count commands calculates the number of observations in various subgroups of the data (such as treated, untreated, eligible, not eligible, etc.).
   - It stores these counts in local macros (like `N_c`, `N_c_found`, etc.) for later use in calculating the IPW weights.

3. Use counts to calculate IPW
   The next step consists of the following:

```
replace ‘ipw_simple3’ = ‘N_c’/‘N_c_found’ if treatment==0
replace ‘ipw_simple3’ = ‘N_t_elig’/‘N_t_elig_found’
    if treatment==1 & eligible==1
replace ‘ipw_simple3’ = ‘N_t_notElig’/‘N_t_notElig_found’
    if treatment==1 & eligible==0
```

   Here, we are calculating IPW:

```
replace ‘ipw_simple3’ = ‘N_c’/‘N_c_found’ if treatment==0
...
```

   Calculates the inverse probability weights based on the counts obtained in the previous steps and assigns these weights to the ipwsimple3 variable.

4. IPW regression
   Running the IPW Regression:

```
    areg ‘y_var’_end treatment ‘blvals’ ‘addlcontrols’
    ‘ifif’ [pw=‘ipw_simple3’], absorb(‘fixedeffects’)
    vce(‘stderrors’)
```

- Conducts a regression (areg) of the endline outcome variable (`y_var’_end`) on the treatment, baseline variables (`blvals`), and additional controls, while accounting for fixed effects and using the calculated IPW weights.
- The `vce(stderrors’)` part indicates that the type of standard errors to be used is specified in the `stderrors` macro.

```
loc coef "treatment"
loc main "false"
```

- Sets the local macro coef to ”treatment”, likely indicating the variable of interest for coefficient extraction in subsequent analyses.
- Sets main to ”false”, which might be used to control the flow of the script or as a flag for other conditional operations.

We can conclude this code segment by saying that it efficiently implements the Simple IPW method to adjust for potential biases in the treatment assignment.

By creating a pseudo-population where the treatment assignment is more balanced with respect to observed covariates, this approach aims to provide more reliable and unbiased estimates of the treatment effect.

”FANCY” INVERSE PROBABILITY WEIGHTING (Fancy IPW)

This segment of the Stata code is dedicated to implementing a ”Fancy” Inverse Probability Weighting (Fancy IPW) regression analysis. Fancy IPW is a more sophisticated version of inverse probability weighting, often used in observational studies to adjust for potential selection bias.

Fancy IPW Conditional Check:

```
if "‘fancyipw’"=="fancyipw" {
```

This line checks whether the fancyipw macro is set to ”fancyipw”, indicating that a Fancy IPW regression is to be executed.

1. Drop existing inverse probability weighting variable
   Drop Existing IPW Variables:

   ```
   cap drop ipw_temp
   cap drop pr_temp
   ```

   Attempts to drop any existing variables named ipwtemp and prtemp from the dataset. The cap (capture) command is used to prevent the script from stopping if these variables do not exist.

2. Predict weight
   Predicting Weights Using Logistic Regression:

   ```
   logit surveyed ‘addlcontrols’
   // <--- here can sub in other ML but i suspect it won’t matter
   ```

   Runs a logistic regression (logit) with the variable surveyed as the dependent variable and additional control variables (addlcontrols) as independent variables.

   NB: The comment suggests that other machine learning methods could potentially be substituted, but logistic regression is expected to be sufficient.

   Calculating Inverse Probability Weights:

```
predict pr_temp, pr
generate double ipw_temp = surveyed/pr_temp
```

- `predict pr_temp, pr`: Generates predicted probabilities (propensity scores) from the logistic regression and stores them in the variable `pr_temp`.
- `generate double ipw_temp = surveyed/pr_temp`: Creates a new variable `ipw_temp`, which contains the inverse of the predicted probabilities (`pr_temp`). This is part of calculating the inverse probability weights.

3. IPW regression
   Running the IPW Regression:

```
areg ‘y_var’_end treatment ‘blvals’ ‘addlcontrols’ ‘ifif’
[pw=ipw_temp], absorb(‘fixedeffects’) vce(‘stderrors’)
```

Conducts a regression (areg) of the endline outcome variable (`y_var’_end`) on the treatment, baseline variables (`blvals`), and additional controls. This regression is weighted by the inverse probability weights (`[pw=ipw_temp]`) and includes fixed effects. The type of standard errors is specified by the `stderrors` macro.

Setting Local Macros:

```
loc coef "treatment"
loc main "false"
```

- Sets the local macro coef to ”treatment”, likely indicating the variable of interest for extracting the coefficient in subsequent analyses.
- Sets main to ”false”, which might be used to control the flow of the script or as a flag for other conditional operations.

To swap up, this code segment implements the Fancy IPW method, which is a more advanced approach to adjust for potential biases in treatment assignment.

By using predicted probabilities from a logistic regression to create inverse probability weights, the Fancy IPW aims to provide a more balanced representation of the population, thereby enabling more accurate estimation of treatment effects.

MAIN / DEFAULT REGRESSION

This section outlines the process for running a main or default regression analysis. This type of regression is typically executed when specific alternative methods (like Treatment on Treated, Simple IPW, or Fancy IPW) are not indicated. It serves as the standard analysis approach in the absence of other specified conditions.
Comment on Main/Default Regression:

```
// MAIN / DEFAULT REGRESSION
```

This comment signifies that the upcoming code block is for running the main or default regression model in the analysis.

Conditional Execution of Default Regression:

```
else if "‘main’"!="false" {
```

This line checks if the macro main is not set to ”false”. If main is anything other than ”false”, it implies that the default regression should be executed. This allows for conditional execution based on the analysis requirements or the data structure.

Running the Default Regression:

```
qui areg ‘y_var’_end treatment ‘blvals’ ‘addlcontrols’
‘ifif’, absorb(‘fixedeffects’) vce(‘stderrors’)
```

- `qui areg`: Runs a regression using the `areg` command quietly (`qui`), meaning it suppresses output to the results window. This is useful for keeping the script's output clean and focused.
- `y_var’_end treatment ‘blvals’ ‘addlcontrols’ ‘ifif’`: Specifies the regression model. It regresses the endline outcome variable (`y_var’_end`) on the treatment variable, baseline variables (`blvals`), additional control variables (`addlcontrols`), and includes any conditions specified in `ifif`.
- `absorb(‘fixedeffects’)`: Includes fixed effects in the model, specified in the macro `fixedeffects`.
- `vce(‘stderrors’)`: Specifies the type of standard errors to use, as defined in the macro `stderrors`.

Setting Local Macro for Coefficient Reference:

```
loc coef "treatment"
```

Sets a local macro coef to ”treatment”. This is likely used later in the script for referencing or extracting the coefficient associated with the treatment variable in the regression model.

In summary, this code segment related to the main or default regression analysis is crucial for conducting the standard regression analysis in situations where specialized methods are not required or specified.

What does this regression ensure?
It ensures that a comprehensive regression analysis is performed, including relevant variables, fixed effects, and specified conditions, thereby forming the backbone of the analytical process in the absence of more complex methods.

3.3 Replicate Table13A: Robustness Inverse probability weighting of key outcomes

In The following Stata code we will replicate Table 13A titled Robustness Inverse probability weighting of key outcomes, that focuses on robustness checks through inverse probability weighting (IPW) of key outcomes. The code is structured to generate results for two different types of IPW, ’simple’ and ’fancy’, and includes detailed specifications for the variables and controls used in the analysis.
Here's a breakdown of the code:

Setting Controls:

```
loc ipwcontrols num_adults_bsl nb_children_bsl hhsize_bsl avg_age_bsl
avg_edu_years_bsl gender_head_bsl age_head_bsl hh_head_over60_bsl
hh_head_educ_bsl disabl_head_bsl
```

- `loc` defines a local macro named `ipwcontrols` containing a list of baseline control variables.
- These control variables (e.g., `num_adults_bsl`, `nb_children_bsl`) are likely demographic or household characteristics measured at baseline.

Looping through IPW Types:

```
foreach type in simple fancy {
```

This loop iterates over both ’simple’ and ’fancy’ IPW methods, indicating an advanced approach to robustness checks. The inclusion of multiple IPW methods allows for comparison and validation of results under different weighting schemes.

Setting Label for Output Files

```
loc typelabel = substr(proper("‘type’"), 1, 4)
```

Here, the code dynamically creates labels (typelabel) based on the IPW method being used. This not only automates the file-naming process for output tables but also ensures clarity in distinguishing the results from different IPW methods.

Generating the Table:

```
yemenstandardtable asset_tot_value_end ctotalpc_end ctotalhh_end
inc_total_end inc_LS_end inc_nonLS_end fs_index_end savings_index_end
percep_econ_end hous_sindex_end debt_index_end, ///
filename("ATable13_‘typelabel’IPW")
title("Appendix Table 13: Treatment Effects on Key Welfare Outcomes
with Inverse Probability Weighting") ///
baselinevals winsorize ‘type’ipw ///
addlcontrols(‘ipwcontrols’) foldername("$rep_output")
```

- The `yemenstandardtable` command is used to generate the results table. This command is a custom function defined elsewhere in the do-file.
- The list of variables (e.g., `asset_tot_value_end`, `ctotalpc_end`, etc.) represents different welfare outcomes being analyzed.
- The options such as `baselinevals`, `winsorize`, and `addlcontrols` tailor the regression to include baseline values, apply winsorization for outliers, and add additional controls (defined in `ipwcontrols`) respectively.
- `ṫype’ipw` dynamically selects the weighting method based on the loop iteration, showcasing the code's flexibility in adapting to different methodologies.

Drafting a Codebook for Variables: A codebook provides descriptive information about variables, which is crucial for understanding the data's structure and characteristics.

```
codebook num_adults_bsl nb_children_bsl hhsize_bsl avg_age_bsl
avg_edu_years_bsl age_head_bsl hh_head_over60_bsl gender_head_bsl
hh_head_educ_bsl disabl_head_bsl asset_tot_value_end ctotalpc_end
ctotalhh_end inc_total_end inc_LS_end inc_nonLS_end fs_index_end
savings_index_end percep_econ_end hous_sindex_end debt_index_end
endline_survey, compact
```

This line of code is used to generate a codebook for a set of baseline control variables such as `num_adults_bsl`, `nb_children_bsl`, which are demographic or household characteristics at baseline. These are crucial in the IPW process to adjust for potential confounders.
The term "`compact`" in the codebook command produces a more concise summary for each variable, focusing on key information without extensive details. This is useful for getting a quick overview of the variables.