Day 1 : Data processing

Session 1: Introduction to Stata and Data Import

1.1. Basic Stata commands

```
// Display Stata's current working directory
pwd

// Change working directory (replace with your actual path)
cd "C:\Users\YourUsername\Documents\StataTraining"

// List files in the current directory
dir

// Open a log file to record commands and output
log using session1.log, replace

// Display help file for the 'summarize' command
help summarize

// Search for commands related to 'regression'
search regression

// Clear Stata's memory
clear

// Close the log file
log close
```

1.2. Importing and saving dataset

```
// Load a built-in Stata dataset
sysuse nlsw88.dta, clear
* Origin: It's an extract from the 1988 round of the National Longitudinal Survey
of Young Women (NLSW). This survey tracks the labor market experiences of a
specific group of women over time.
*Sample: The dataset contains observations for 2,246 women who were aged 34-41 in
1988.


// Save the current dataset
save my_auto_data.dta, replace

// Clear memory before importing new data
clear

// Import data from an Excel file (assuming 'mydata.xlsx' is in the working
directory)
// Assumes the first row contains variable names
import excel "mydata.xlsx", sheet("Sheet1") firstrow clear

// Save the imported data
save my_imported_excel_data.dta, replace

// Clear memory
clear

// Import data from a CSV file (assuming 'mydata.csv' is in the working
directory)
import delimited "mydata.csv", clear

// Save the imported data
save my_imported_csv_data.dta, replace
```

```
// Load a previously saved Stata dataset
use my_auto_data.dta, clear
```

Session 2: Data Inspection and Basic Cleaning

2.1. Initial exploration of dataset

- Let’s first generate a dummy dataset.

```
* Clear any existing data in memory
clear

* Set seed for reproducibility
set seed 12345

* Create a dataset with 1000 observations
set obs 1000

* Generate ID variable
gen id = _n

* Generate Age variable (between 18 and 70)
gen age = round(runiform() * 52 + 18)

* Generate Education variable (1: High School, 2: Bachelor's, 3: Master's, 4: PhD)
gen education = runiformint(1,4)

* Generate Income variable with a positive relationship with Age and Education
gen income = 20000 + 1000 * age + 5000 * education + rnormal(0, 5000)

* Introduce some missing values randomly
replace age = . if runiform() < 0.1 //randomly generates about 10% missing values
replace income = . if runiform() < 0.15 //randomly generates about 15% missing values
replace education = . if runiform() < 0.05 //randomly generates about 5% missing values
```

- Examples of initial data exploration commands.

```
* View the first ten rows
list in 1/10

* Browse the data
browse

* Describe the dataset
describe

* Summarize the data
summarize
* Detailed summary statistics
summarize, detail

* Check for missing values
misstable summarize

* details of each variables
codebook

* Frequency distribution for education
tabulate education

* Frequency distribution for education (including missing values)
tabulate education, missing
```

```
* Histogram for income
histogram income, normal

* Dot plot
dotplot income

* Density plot of continuous variable
kdensity income

* Box plot for income
graph box income

* Scatter plot for income vs. age
scatter income age
graph twoway (scatter income age) (lfit income age)

* Correlation matrix
corr age income education

* Label variables
label variable income "Annual Income"
label variable age "Age of Individuals"
label variable education "Education Level"
```

2.2. Identifying and handling missing values

- Identifying missing values.

```
* Summarize missing values in the dataset
misstable summarize

* Tabulate missing values for a specific variable
tabulate age, missing

* List observations with missing values for a specific variable
list id age income if missing(age) | missing(income)

* Browsing observations with missing values for a specific variable
browse id age income if missing(age) | missing(income)
```

- Handling missing values.

```
* Replace missing values in age with the mean age
summarize age
return list
replace age = r(mean) if missing(age)

* Drop observations with missing values in income
drop if missing(income)
```

Session 3: Data Types and Variable Management

3.1. Understanding variable types (numeric, string, etc.)

- Numeric variables

  Numeric variables store numbers and can be used for mathematical operations.

  Types:

  - **Integer:** Whole numbers without decimal points.
  - **Float:** Numbers with decimal points. These are approximate representations of real numbers.

```
* Clear any existing data in memory
clear
* Set seed for reproducibility
set seed 12345
```

```
* Create a dataset with 100 observations
set obs 100

* Generate an integer variable (age)
gen age = round(runiform() * 52 + 18)

* Generate a float variable (income)
gen income = runiform() * 80000 + 20000
```

- String Variables
  String variables store text and are used for non-numeric data.

```
* Generate a short string variable (name)
gen name = ""

* Assign values to the string variable
replace name = "John Doe" in 1
replace name = "Jane Smith" in 2
```

- Factor/Categorical variables
  Categorical variables take on a limited number of distinct values, representing different categories.

```
* Generate a categorical variable (education)
gen education = 1
replace education = 2 in 21/40
replace education = 3 in 41/60
replace education = 4 in 61/80

* Label the categorical variable
label define edu_labels 1 "High School" 2 "Bachelor's" 3 "Master's" 4
"PhD"

label list edu_labels

label values education edu_labels
```

Exercise:

Generate a factor/categorical variable named age_group based on the following rule.
age < 20 → Teen, 20 <= age < 65 → Adult, age >=65 → Senior.

```
* Generate a categorical Age Group variable
gen age_group = .
replace age_group = 1 if age < 20
replace age_group = 2 if age >= 20 & age < 65
replace age_group = 3 if age >= 65

* Label the Age Group variable
label define agegrp_labels 1 "Teen" 2 "Adult" 3 "Senior"
label values age_group agegrp_labels
```

- Date and Time variable
  Date and time variables store dates, times, and date-time combinations. They require special formats to perform calculations and manipulations.

```
* Generate a date variable
gen date = mdy(12, 25, 2024)

* Format the date variable
format date %td

* Format the date in YYYY-MM-DD format
format date %tdCCYY-NN-DD

* Format the date in MM/DD/YYYY format
```

```
format date %tdNN/DD/CCYY
```

3.2. Converting variable types using destring and tostring

Converting a string variable to numeric

```
* Clear any existing data in memory
clear

* Create a string variable with numeric values
gen str_var = "123"
replace str_var = "456" in 2
replace str_var = "789" in 3

* Convert the string variable to numeric
destring str_var, replace
```

Converting a numeric variable to string

```
* Clear any existing data in memory
clear

* Create a numeric variable
gen num_var = 123
replace num_var = 456 in 2
replace num_var = 789 in 3

* Convert the numeric variable to string
tostring num_var, replace
```

3.3. Encoding a string variable to a factor/categorical variable

```
* Clear any existing data in memory
clear

* set dataset size to 4 observations
set obs 4

* Create a string variable with categorical values
gen education_level = "PhD"
replace education_level = "Bachelor's" in 2
replace education_level = "High School" in 3
replace education_level = "Master's" in 4

* Define a label with a specific order
label define edu_labels 1 "High School" 2 "Bachelor's" 3 "Master's" 4
"PhD"

* Encode the string variable into a numeric variable using the defined
label
encode education_level, gen(education_encoded) label(edu_labels)

* Encode the string variable without defined labels
encode education_level, gen(education_encoded1)
```

3.4. Generating variables using egen command

```
* Clear any existing data
clear

* Set seed for reproducibility
set seed 12345

* Generate a dataset with 100 observations
```

```
set obs 100

* Generate income variable
gen income = round(runiform() * 80000 + 20000)

* Generate group variable
gen sex = round(runiform(0,1))

* Generate mean of income
egen mean_income = mean(income)

* Generate standard deviation of income
egen sd_income = sd(income)

* Generate mean income by group
egen group_mean_income = mean(income), by(sex)

* Generate maximum income
egen max_income = max(income)

* Generate maximum income by group
egen group_max_income = max(income), by(sex)

* Generate total income
egen total_income = total(income)
```

Exercise:

Generate two income groups (high income [income > 50,000], low income [income <= 50,000]) and calculate mean, median, and standard deviation by income group.

```
gen income_group = "Low Income"
replace income_group = "High Income" if income > 50000

egen income_group_mean = mean(income), by(income_group)
egen income_group_median = median(income), by(income_group)
egen income_group_sd = sd(income), by(income_group)
```

Session 4: Sorting and Filtering Data

4.1. Sorting data with gsort

```
* Clear any existing data
clear

* Set seed for reproducibility
set seed 12345

* Create a dataset with 10 observations
set obs 10

* Generate id, income, and age variables
gen id = _n
gen income = round(runiform() * 10000)
gen age = round(runiform() * 50 + 20)

* Sort data by income in descending order
gsort -income
list

* Sort data by age in ascending order and income in descending order
gsort age -income
list
```

4.2. Filtering data using keep and drop

Filtering variables using keep and drop

```
* Clear any existing data
clear

* Generate a sample dataset
set seed 12345
set obs 10
gen id = _n
gen age = round(runiform() * 50 + 20)
gen income = round(runiform() * 10000)
gen education = mod(_n, 4) + 1

* Display the original dataset
list

* Keep only the id and income variables
keep id income // drop age education [will produce same result]

* Display the filtered dataset
list
```

Filtering observations using keep and drop

```
* Clear any existing data
clear

* Generate a sample dataset
set seed 12345
set obs 10
gen id = _n
gen age = runiformint(20,70)
gen income = round(runiform() * 10000)
gen education = runiformint(1,4)

* Display the original dataset
list

* Keep only the id and income variables
keep if income > 2000
list

drop if age < 40
list
```

Sub-setting dataset

```
* Clear any existing data
clear

* Generate a sample dataset
set seed 12345
set obs 10
gen id = _n
gen age = runiformint(20,70)
gen income = round(runiform() * 10000)
gen education = runiformint(1,4)

* Display the original dataset
list

* Summarize income for individuals older than 30
```

```
summarize income if age > 30

* Drop observations where income is less than 5000
drop if income > 5000
list

* Keep observations where age is between 30 and 50
keep if age > 30 & age <50
list
```

Temporary dataset modification using ***preserve*** and ***restore*** command.

```
* Clear any existing data
clear

* Generate a sample dataset
set seed 12345
set obs 10
gen id = _n
gen age = runiformint(20,70)
gen income = round(runiform() * 10000)
gen education = runiformint(1,4)

* Display the original dataset
list

preserve
* Drop observations where income is less than 5000
drop if income > 5000
list
restore

preserve
* Keep observations where age is between 30 and 50
keep if age > 30 & age <50
list
restore
```

Day 2: Data Cleaning

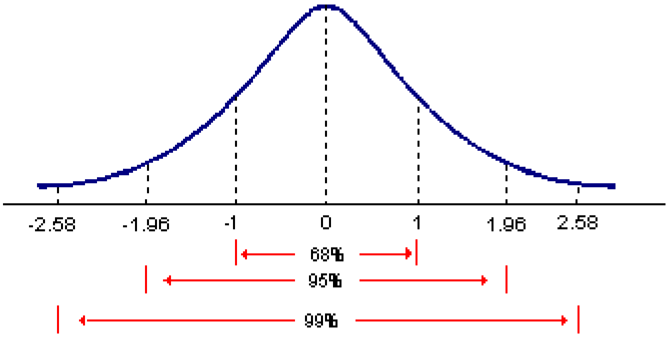
Session 5: Outliers, duplicates, recoding, dummy variable generation, and groupwise calculations

5.1. Handling Outliers and Duplicates

Handling Outliers

```
clear

set seed 12345
set obs 100
gen id = _n
gen income = round(runiform() * 100000)
replace income = income + 80000 if _n > 90

* Display summary statistics to identify outliers
summarize income

* Create a scatter plot to visually inspect outliers
scatter income id

* Apply a log transformation to reduce the impact of outliers
gen lincome = log(income)
scatter lincome id

* Caping the income at a certain threshold
gen capped_income = cond(income > 100000, 100000, income)
scatter capped_income id

* Dropping observations where income > 100000
drop if income > 100000
scatter income id
```

Identifying outliers using statistical methods (e.g., z-scores)



$$Z = \frac{X - \mu}{\sigma}$$

```
clear

set seed 12345
set obs 100
gen id = _n
gen income = round(runiform() * 100000)
replace income = income + 80000 if _n > 90

* Calculate mean and standard deviation
egen mean_income = mean(income)
egen sd_income = sd(income)

* Generate z-scores
gen zscore_income = (income - mean_income) / sd_income

* Identify outliers (z-scores beyond |1.96|) (signifinace level 10% -> 1.645, 5%
-> 1.96, 1% -> 2.58)
```

```
gen outlier = abs(zscore_income) > 1.96

scatter income id, name(with_outlier, replace)
scatter income id if outlier != 1, name(without_outlier, replace)
```

Handling duplicates

```
* Generate a sample dataset with duplicates
clear
input id income
1 50
3 50
2 70
4 80
3 50
1 50
4 80
3 50
end

* Show duplicates (total number count)
bysort id income: gen count = _N
list

* Show duplicates (incremental number count)
bysort id income: gen count_inc = _n
list

* Drop duplicates
drop if count_inc > 1
list

***************
* OR **********
***************

sort id income

* Show duplicates
duplicates examples
duplicates examples income
duplicates report
duplicates report income

* List duplicate observations
duplicates list

* duplicates tagging
duplicates tag, generate(flag1)
duplicates tag income, generate(flag2)

* Drop duplicates
duplicates drop
duplicates drop income, force
```

5.2. Recoding and dummy variable generation

```
clear
set seed 12345
set obs 50
gen id = _n
gen income = round(runiform() * 10000)

*Recode income variable into categories
```

```
recode income (0/3000 = 1 "Low") (3001/7000 = 2 "Medium") (7001/max = 3 "High"),
generate(income_cat)

*Generate dummy variable based on categorical variable
tabulate income_cat, gen(inc)
```

5.3. Using bysort and collapse for groupwise calculation and variable generation

```
clear
set seed 12345
set obs 20
gen id = _n
gen age = runiformint(20,70)
gen income = round(runiform() * 10000)
gen education = runiformint(1,4)

*Calculating mean income by education levels using bysort
bysort education: egen mean_income = mean(income)
list

*Calculating mean income by education levels using collapse
collapse (mean) income, by(education)
list
```

Exercise:

Using NMICS6's microdata,

a. Import ***hl.sav*** from NMICS6 dataset [import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hl.sav", clear]
b. Keep ***HH1, HH2, and HL1*** variables.
c. Use collapse to generate ***family_size*** variable by household.

```
import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hl.sav",
clear
keep HH1 HH2 HL1
collapse (count) family_size=HL1, by(HH1 HH2)
```

Session 6: Merging and Appending Datasets

6.1. Merging datasets (1:1, 1:m, m:1)

```
*Dataset 1
clear

input id str10 name
1 "Sita"
2 "Ram"
3 "Gita"
4 "Gokul"
end

save dataset1.dta, replace

*Dataset 2
clear
input id score
1 90
2 85
2 88
3 75
5 92
```

```
end

*Dataset 3
clear
input id str10 address
1 "Hetauda"
2 "Kathmandu"
3 "Biratnagar"
end

save dataset3.dta, replace

*1:1 merge
use dataset1.dta, clear
merge 1:1 id using dataset3.dta

*1:m merge
use dataset1.dta, clear
merge 1:m id using dataset2.dta

*m:1 merge
use dataset2.dta, clear
merge m:1 id using dataset1.dta
```

6.2. Appending datasets

```
* Clear existing data
clear

* Create dataset1
input id str10 name income
1 "John" 45000
2 "Jane" 52000
3 "Doe" 47000
end

* Save dataset1
save dataset1.dta, replace

* Clear existing data
clear

* Create dataset2
input id str10 name income
4 "Alice" 48000
5 "Bob" 51000
6 "Charlie" 53000
end

* Save dataset2
save dataset2.dta, replace

* Load dataset1
use dataset1.dta, clear

* Append dataset2 to dataset1
append using dataset2.dta

* List the combined dataset
list
```

Session 7: Reshaping dataset

```
clear
```

```
input id str10 name math2018 math2019 science2018 science2019
1 "Ram" 80 85 90 95
2 "Sita" 70 60 90 95
3 "Gita" 50 60 70 90
end
list

* Reshaping from wide to long
reshape long math science, i(id name) j(year)

* reshaping from long to wide
reshape wide math science, i(id name) j(year)
```

Session 8: Basics of Stata programming

8.1. Looping (foreach, forvalues, and while loop)

```
clear
set seed 12345
set obs 10
gen var1 = _n
gen var2 = _n * 2
gen var3 = _n * 3
gen group = ceil(_n/2) //similar to roundup function in excel
gen income = round(runiform() * 100)

* foreach loop
foreach var of varlist var1 var2 var3 {
    summarize `var'
}

* Loop through each observation to display income values
forvalues i = 1/10 {
    display "Observation `i' has income level " income[`i']
}

* Loop over observations to get total income
local N = _N
local x = 0
forvalues i = 1/`N' {
    local x = `x' + income[`i']
}
di "Total income : " `x'

* while loop
local i = 1
while `i' <= 15 {
    display "Value of i is `i'"
    local i = `i' + 1
}
```

8.2. If condition with looping.

```
* Create a dataset with 10 observations and a numeric variable 'income'
set obs 10
gen income = round(runiform() * 100)

* Initialize the 'income_category' variable
gen income_category = ""

* Loop through each observation to categorize income
forvalues i = 1/10 {
    if income[`i'] < 30 {
        replace income_category = "Low" in `i'
    }
    else if income[`i'] >= 30 & income[`i'] < 70 {
        replace income_category = "Medium" in `i'
    }
    else {
        replace income_category = "High" in `i'
    }
}

* List the dataset to see the results
list
```

8.3. Creating program in Stata and its use.

```
* Define a program to calculate mean and standard deviation
program define calc_stats
    args varname

    * Calculate mean and standard deviation
    quietly summarize `varname'
    local mean = r(mean)
    local sd = r(sd)

    * Display the results
    display "The mean of `varname' is " `mean'
    display "The standard deviation of `varname' is " `sd'
end

* Clear existing data
clear

* Create a dummy dataset with 20 observations
set obs 20
gen income = round(runiform() * 100000)
gen age = runiformint(20,60)

* List the dataset to check the values
list

* Run the program to calculate mean and standard deviation for 'income'
calc_stats income
calc_stats age

* Deleting the program
program drop calc_stats
```

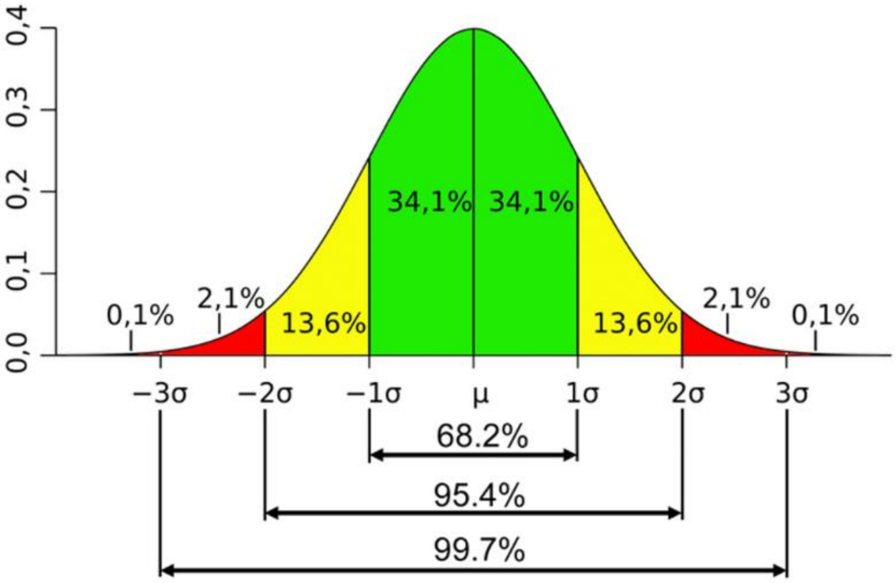
Day 3 : Data Analysis

Session 9: Hypothesis testing

9.1. The concept of normal distribution

a. What is a Normal Distribution?

- **Shape:** The normal distribution looks like a bell-shaped curve.
- **Symmetry:** It is perfectly symmetrical around the center.



b. Key Characteristics:

- **Mean (Average):** The center of the curve.
- **Standard Deviation:** Measures the spread of the data.
  - 68.2% of the data falls within 1 standard deviation of the mean.
  - 95.4% falls within 2 standard deviations.
  - 99.7% falls within 3 standard deviations.

c. Why is it Important?

- **Natural Occurrences:** Many natural phenomena follow this distribution (e.g., heights, test scores). For example, most students score around the average in a class, fewer scoring very high or very low.
- **Central Limit Theorem:** In large samples, the samples' mean tend to be normally distributed. (Video)
- **Statistical Inferences:** Helps in making predictions and decisions based on data.

9.2. Hypothesis testing

a. What is Hypothesis Testing?

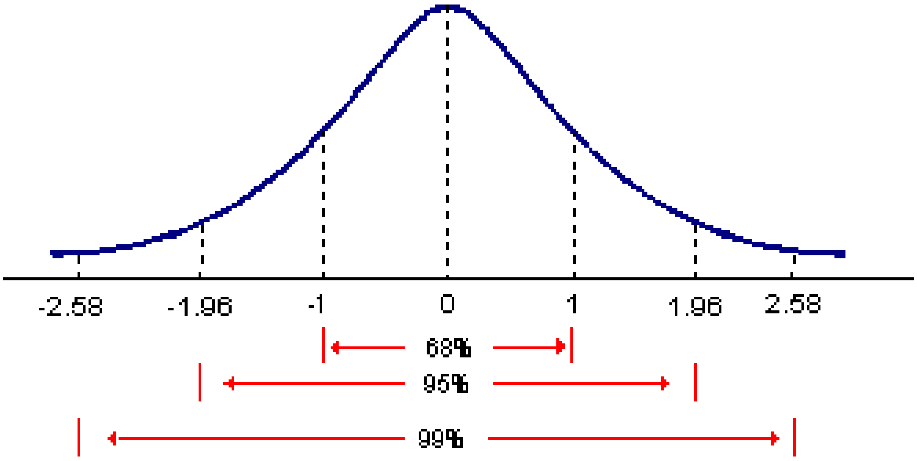
- Hypothesis testing is a method used to decide whether there is enough evidence to support a particular claim about a population based on a sample of data.
- **Null Hypothesis (H_0):** This is the default statement that there is no effect or no difference. It assumes that any observed differences are due to random chance.

Example: "The average age is equal to 20."

- **Alternative Hypothesis** (H$_1$): This is what you want to prove, stating there is an effect or a difference.
  Example: "The average age is not equal to 20."

b. Procedure of hypothesis testing

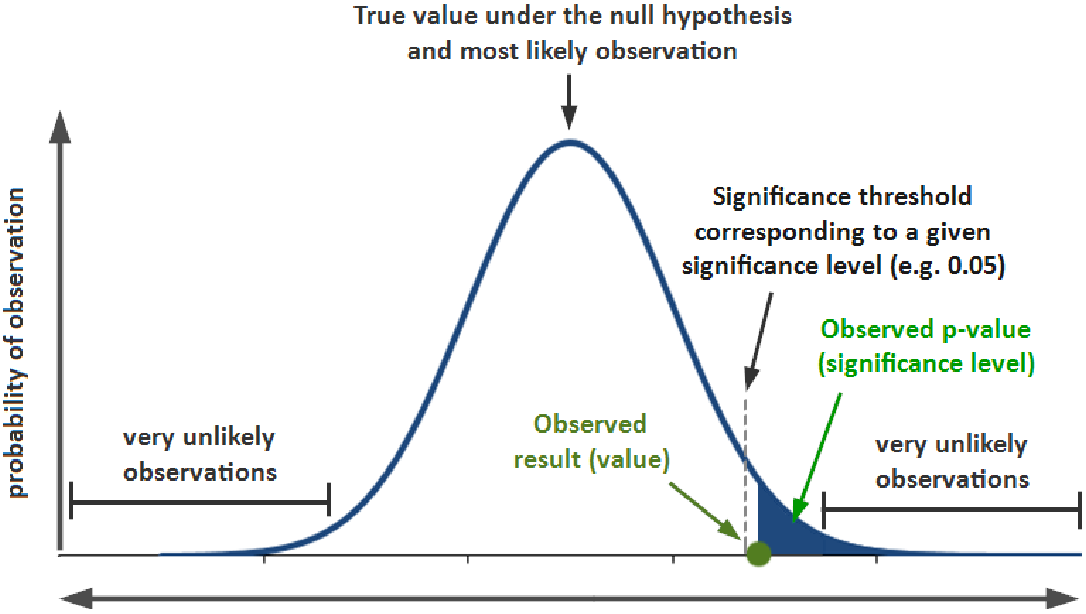
- State the null and alternative hypothesis. (e.g. $H_0: \mu = 0, H_1: \mu \neq 0$)
- Collect sample data.
- Calculate sample mean and stadard error ($\frac{s}{\sqrt{n}}$).
- Calculate t-statistics ($t = \frac{\bar{X}-\mu}{Standar\ Error}$).
- Compare absolute value of t-statistics |t| with critical values for given level of significance (α). [1.65 (10% significance level), 1.96 (5%), 2.58 (1%)]



- Decision: reject null hypothesis if |t| exceeds critical value, otherwise fail to reject null hypothesis.

c. Hypothesis testing with p-value

- p-value : probability (area under normal distribution) beyond |t|.


True value under the null hypothesis and most likely observation
probability of observation
Significance threshold corresponding to a given significance level (e.g. 0.05)
Observed p-value (significance level)
very unlikely observations
Observed result (value)
very unlikely observations

- **Decision :** reject null hypothesis if p-value is lower than the significance level, otherwise fail to reject null hypothesis.

- Easier to conduct hypothesis testing with p-value. No need to calculate t-statistics and remember different critical values.

9.3. Hypothesis testing in Stata

```
* Clear existing data
clear

* Create a dummy dataset
set seed 12345
set obs 100
gen group = mod(_n, 2)
gen score = 50 + group * 10 + rnormal(0, 10)

*conducting hypothesis testing
ttest score = 50 //H0: pop_mean = 50
ttest score = 55 //H0: pop_mean = 55
ttest score = 60 //H0: pop_mean = 60

* conducting two-sample t-test
ttest score, by(group) //H0: pop_mean_group1 = pop_mean_group2
                       //OR H0: pop_mean_group1 - pop_mean_group2 = 0

*Same answer can be obtained from regression
reg score group
```

Exercise:
Using NMICS6 data (hl.sav), conduct a hypothesis test whether average age between male and female is statistically different.

```
import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hl.sav",
clear

* HL6 -> Age, HL4 -> Sex
sum HL6 if HL4 == 1 //male : average age is 28.263
sum HL6 if HL4 == 2 //female : average age is 28.827

*Looks like the population means for male and female are not statistically
different.
*Let's conduct the hypothesis testing

ttest HL6, by(HL4)
*Alternatively

reg HL6 HL4
```

9.4. Hypothesis testing using non-parametric approach (bootstraping)

Bootstrap : generating distribution of statistics of interest by resampling the sample with replacement. Using Bootstrap, we can calculate standard errors, confidence intervals, and other statistical measures.

```
clear
set seed 1
set obs 100
gen score = round(runiform() * 100)

* Bootstrap the median and test against a specified value (e.g., 50)
bootstrap r(p50), reps(1000): summarize score, detail

* Testing whether median is equal to 50 or not
test _bs_1 = 50
```

Exercise:

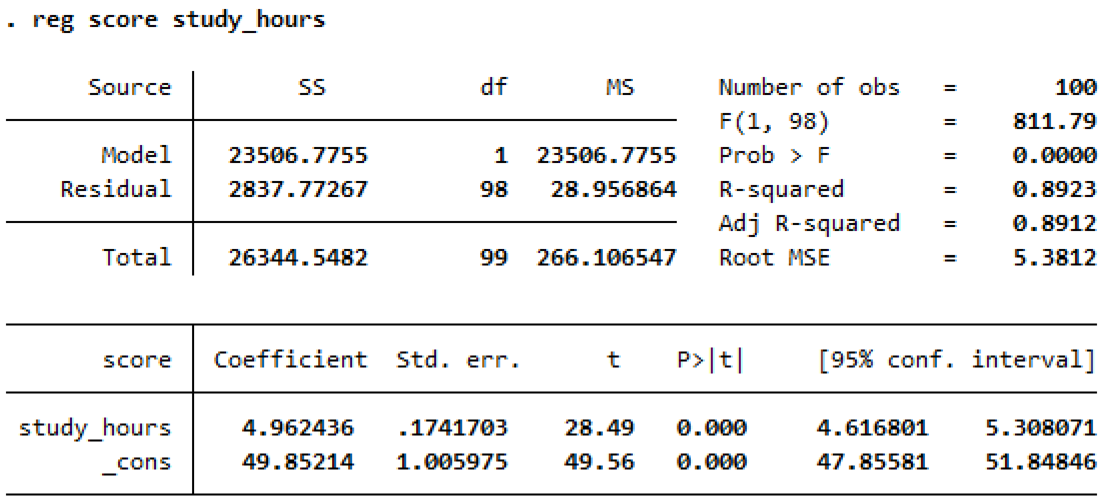
Using NMICS6 data (hl.sav), conduct a hypothesis test whether medeian age between male and female is statistically different.

```
import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hl.sav",
clear

set seed 12345
* Define a program to calculate the difference in medians
program define diff_medians, rclass
    summarize HL6 if HL4 == 1, detail
    local med0 = r(p50)
    summarize HL6 if HL4 == 2, detail
    local med1 = r(p50)
    return scalar diff = `med1' - `med0'
end

* Bootstrap the difference in medians
bootstrap r(diff), reps(100): diff_medians
```

Session 10: Regression analysis

10.1. Simple regression analysis

```
clear

set seed 12345
set obs 100
gen study_hours = round(runiform() * 10)
gen score = 50 + 5 * study_hours + rnormal(0, 5)

reg score study_hours
```



```
. reg score study_hours

      Source |       SS           df       MS      Number of obs   =       100
-------------+----------------------------------   F(1, 98)        =    811.79
       Model |  23506.7755         1  23506.7755   Prob > F        =    0.0000
    Residual |  2837.77267        98   28.956864   R-squared       =    0.8923
-------------+----------------------------------   Adj R-squared   =    0.8912
       Total |  26344.5482        99  266.106547   Root MSE        =    5.3812

------------------------------------------------------------------------------
       score | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
 study_hours |   4.962436   .1741703    28.49   0.000     4.616801    5.308071
       _cons |   49.85214   1.005975    49.56   0.000     47.85581    51.84846
------------------------------------------------------------------------------
```

10.2. Multiple regression and diagnostics

```
clear
set obs 200

gen age = mod(_n,52) + 18
gen educ_year = mod(_n,18)

* Generate Income variable with a positive relationship with Age and Education
gen income = 20000 + 800 * age + 3000 * educ_year + rnormal(0, 2000)

* Regression with omitted variable
reg income age

*******************************
```

```
* Residual diagnostics
*******************************
* Residual visual inspection
rvfplot

* Histogram plot for residual's distribution visualization
predict resid, residuals
hist resid

*Formal test of residuals normality
swilk resid
drop resid

* Multiple regression with correct specification
reg income age educ_year

*******************************
* residual diagnostics
*******************************
* Residual visual inspection
rvfplot

* Histogram plot for residual's distribution visualization
predict resid, residuals
hist resid

*Formal test of residuals normality
swilk resid
```

Session 11: Advance regression with binary dependent variables (logit/probit)

```
import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hh.sav",
clear

* dropping missing values
drop if missing(HHSEX)

* checking levels of HHSEX (Household Head Sex)
codebook HHSEX
label list labels410

gen hh_size = HH48 //HH member size variable
gen urb_rur = HH6 //1=Urban 2=Rural
gen province = HH7 //province number

* generating binary dependent variable separately
gen hhsex_male = 1
replace hhsex_male = 0 if HHSEX == 2 //1=Male 2=Female

*running logistic regression
logit hhsex_male hh_size ib1.urb_rur ib3.province
margins, dydx(hh_size urb_rur province)

* Similar results can be obtaine using probit
* Running probit regression
probit hhsex_male hh_size ib1.urb_rur ib3.province
margins, dydx(hh_size urb_rur province)
```

Session 12: Time series analysis

12.1. Stationarity concept

- Stationarity refers to a time series whose statistical properties, such as mean, variance, and autocorrelation, remain constant over time.
- Non-stationary series are prone to spurious relationships.

12.2. Spurious relationship

```
clear
set seed 1
set obs 100

gen year = 1900 + _n
tsset year
gen ice_cream_sales = year*10 + rnormal(0, 50)
gen shark_attacks = year*5 + rnormal(0, 20)

* visual inspection for stationarity
twoway line ice_cream_sales year, name(ice_cream_sales, replace)
twoway line shark_attacks year, name(shark_attacks, replace)

dfuller ice_cream_sales //H0 : Non-stationary
dfuller shark_attacks //H0 : Non-stationary

* Run the initial regression (spurious relationship)
reg shark_attacks ice_cream_sales
```

12.3. Making series stationary to avoid spurious relationship

```
*****************************
* Making Series Stationary
*****************************
* Differencing variable makes series stationary
* If a variable is stationary at first difference, then its called
* I(1). I(0) means the variable is stationary at level.
twoway line D.ice_cream_sales year, name(ice_cream_sales, replace)
twoway line D.shark_attacks year, name(shark_attacks, replace)

dfuller D.ice_cream_sales //H0 : Non-stationary
dfuller D.shark_attacks //H0 : Non-stationary

*no relationship observed after differencing
reg D.shark_attacks D.ice_cream_sales

** log difference is preferred over simple difference as
** interpretation of coefficient becomes easier.
gen lshark_attacks = log(shark_attacks)
gen lice_cream_sales = log(ice_cream_sales)

twoway line D.lice_cream_sales year, name(ice_cream_sales, replace)
twoway line D.lshark_attacks year, name(shark_attacks, replace)

dfuller D.lice_cream_sales //H0 : Non-stationary
dfuller D.lshark_attacks //H0 : Non-stationary

reg D.lshark_attacks D.lice_cream_sales
```

12.4. Example of non-stationary series with actual relationship

```
clear
set seed 1
set obs 100

gen year = 1900 + _n
tsset year
gen income = year*10 + rnormal(0, 50)
gen expenditure = income*0.5 + rnormal(0, 20)

* visual inspection for stationarity
twoway line income year, name(income, replace)
twoway line expenditure year, name(expenditure, replace)

dfuller income //H0 : Non-stationary
dfuller expenditure //H0 : Non-stationary

* Run the initial regression
reg expenditure income

*****************************
* Making Series Stationary
*****************************
gen lincome = log(income)
gen lexpenditure = log(expenditure)

* visual inspection for stationarity
twoway line D.lincome year, name(income, replace)
twoway line D.lexpenditure year, name(expenditure, replace)

dfuller D.lincome //H0 : Non-stationary
dfuller D.lexpenditure //H0 : Non-stationary

* Run the regression at first difference
reg D.lexpenditure D.lincome
```

Cross tab tabulation

```
use census_sim.dta, clear

tabulate sex urban
* Shows the percentage distribution across 'urban' for each 'sex' category.
tabulate sex urban, row

* Shows the percentage distribution across 'sex' for each 'urban' category.
tabulate sex urban, col

* Shows the percentage of each cell relative to the total number of observations.
tabulate sex urban, cell

table education ethnicity

* --- Advanced Cross-Tabulation Examples ---
** table (rowvar) (colvar) (tabvar)

table sex urban //two-way
table (sex urban) (education) //three-way
table (sex urban) (education) (province) //n-way


** Table of statistics
table (sex) (province) (urban), statistic(mean income) statistic(sd income)
statistic(median income) statistic(min income) statistic(max income)

collect export "my_table_output.xlsx", replace //exporting the recently prepared
table to excel
```

Survey design and analysis using STATA

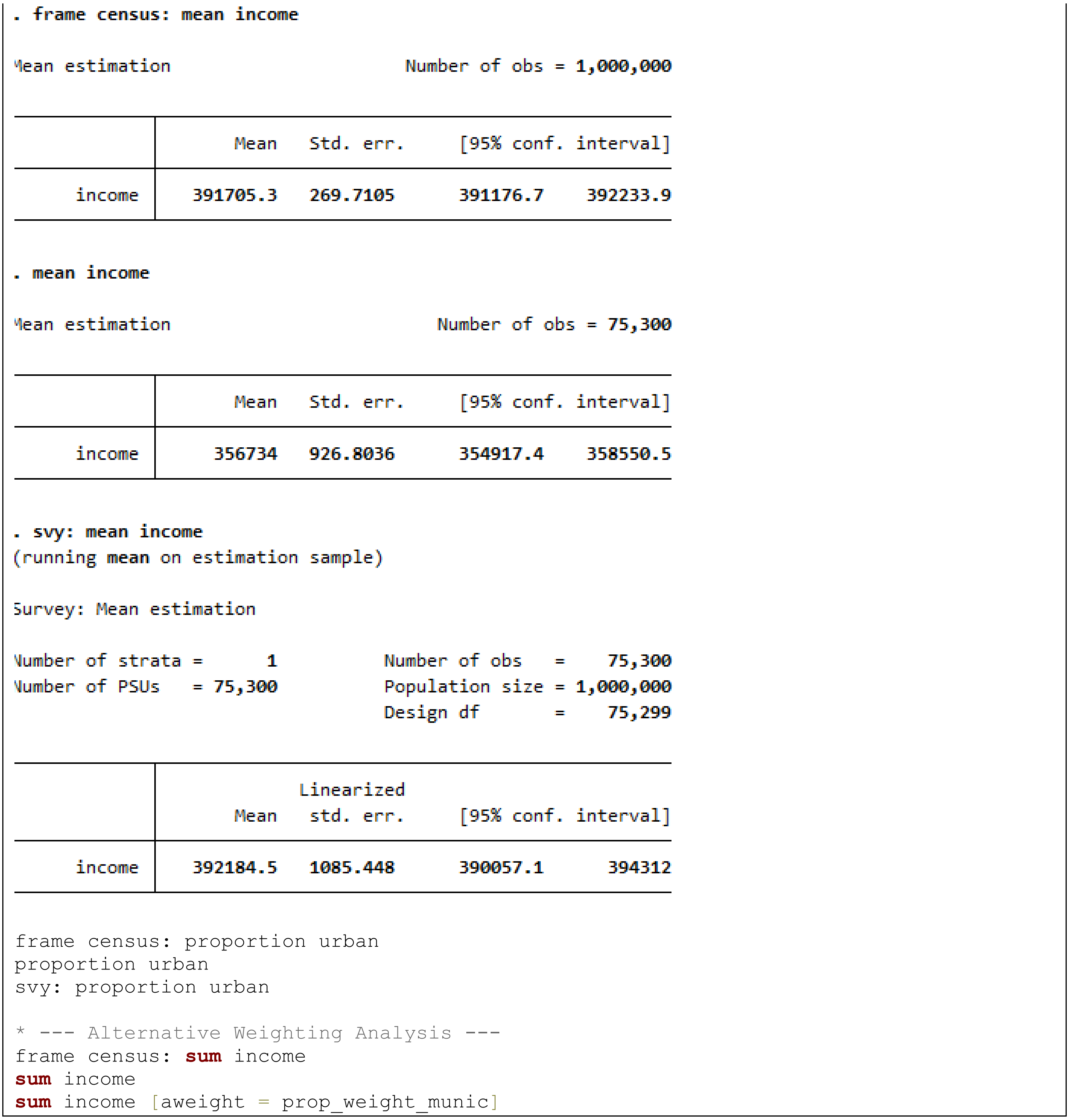
1. Simple Random Sampling & Weighting

```
************************************************************************
* Simple Random Sampling and weight calculation
************************************************************************

* Load the dataset, clearing any existing data in memory
use census_sim.dta, clear

* --- Data Preparation and Frame Management ---

capture {
    * Switch back to the default frame
    frame change default
    * Drop the 'census' frame if it exists
    frame drop census
}


frame put *, into(census)
set seed 123

* --- Calculate Population Proportions ---
gen pop = _N

* Calculate the population size for each province
bysort province: gen prov_pop = _N

* Calculate the population size for each municipality
bysort municipality_id: gen munic_pop = _N

* Calculate the proportion of the total population residing in each province
bysort province: gen prov_pop_propotion = prov_pop / pop

* Calculate the proportion of the total population residing in each municipality
bysort municipality_id: gen munic_pop_propotion = munic_pop / pop


* --- Create and Analyze a Sample ---
capture {
    frame change default
    frame drop smpl20
}

frame put *, into(smpl20)
frame change smpl20

sample 20 // Draw a random sample of 20% of the observations from the dataset

gen sample_size = _N

bysort province: gen prov_sample_size = _N
bysort municipality_id: gen munic_sample_size = _N

bysort province: gen prov_sample_propotion = prov_sample_size / sample_size
bysort municipality_id: gen munic_sample_propotion = munic_sample_size / sample_size


* --- Calculate Sampling Weights ---
* Calculate "absolute" or "design" weights for municipalities
* This weight represents how many population units each sample unit represents
gen abs_weight_munic = munic_pop / munic_sample_size

* Calculate "proportional" or "balancing" weights for municipalities
* This weight adjusts the sample proportions to match the population proportions
gen prop_weight_munic = munic_pop_propotion / munic_sample_propotion

* --- Survey Data Analysis ---

egen total_pop = total(abs_weight_munic)

svyset [pweight = abs_weight_munic]
frame census: mean income
mean income
svy: mean income

. frame census: mean income

Mean estimation                        Number of obs = 1,000,000

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   391705.3   269.7105      391176.7    392233.9
--------------------------------------------------------------

. mean income

Mean estimation                          Number of obs = 200,000

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   392632.9   604.0539        391449    393816.8
--------------------------------------------------------------

. svy: mean income
(running mean on estimation sample)

Survey: Mean estimation

Number of strata =       1        Number of obs   =    200,000
Number of PSUs   = 200,000        Population size = 1,000,000
                                  Design df       =    199,999

--------------------------------------------------------------
             |             Linearized
             |       Mean   std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   392505.2   604.8006      391319.8    393690.6
--------------------------------------------------------------

frame census: proportion urban
proportion urban
svy: proportion urban
* --- Alternative Weighting Analysis ---

frame census: sum income
sum income
sum income [aweight = prop_weight_munic]
```

```
+---------------------------------------------------+
| [Simple Random Sampling & Weighting] |
+---------------------------------------------------+
                         |
                       V
+-----------------------+
| 1. Initial Setup      |
+-----------------------+
         |
         +-- Load Data (`use census_sim.dta, clear`)
         |
         V
+-------------------------+
| 2. Frame Management     |
+-------------------------+
         |
         +-- Cleanup Old Frames (`capture frame drop`)
         |   (Attempts for 'census' & 'smpl20')
         |
         +-- Create Full Data Copy (`frame put *, into(census)`)
         |
         +-- Set Random Seed (`set seed 123`)
         |
         V
+--------------------------------------------------+
| 3. Population Calculations (in Default/Original) |
+--------------------------------------------------+
         |
         +-- Total Pop Size (`gen pop = _N`)
         |
         +-- Per Province:
         |   |
         |   +-- Pop Size (`bysort province: gen prov_pop = _N`)
         |   +-- Pop Proportion (`bysort province: gen prov_pop_propotion = ...`)
         |
         +-- Per Municipality:
             |
             +-- Pop Size (`bysort municipality_id: gen munic_pop = _N`)
             +-- Pop Proportion (`bysort municipality_id: gen munic_pop_propotion = ...`)
             |
             V
+-----------------------------------------+
| 4. Sample Creation & Analysis           |
+-----------------------------------------+
         |
         +-- Create Sample Frame Base (`frame put *, into(smpl20)`)
         |   (Copies data *with* population calculations)
         |
         +-- Switch to Sample Frame (`frame change smpl20`)
         |
         +-- Draw 20% Sample (`sample 20`)
         |   (Reduces observations *within* 'smpl20')
         |
         +-- Calculate Sample Sizes (in 'smpl20'):
         |   |
         |   +-- Total Sample Size (`gen sample_size = _N`)
         |   +-- Per Province (`bysort province: gen prov_sample_size = _N`)
         |   +-- Per Municipality (`bysort municipality_id: gen munic_sample_size = _N`)
         |
         +-- Calculate Sample Proportions (in 'smpl20'):
             |
             +-- Per Province (`bysort province: gen prov_sample_propotion = ...`)
             +-- Per Municipality (`bysort municipality_id: gen munic_sample_propotion = ...`)
             |
             V
+---------------------------------------------+
| 5. Calculate Sampling Weights (in 'smpl20') |
+---------------------------------------------+
         |
         +-- Absolute/Design Weight (Municipality)
         |   | (`gen abs_weight_munic = munic_pop / munic_sample_size`)
         |   | (Represents # of pop units per sample unit)
         |
         +-- Proportional/Balancing Weight (Municipality)
             | (`gen prop_weight_munic = munic_pop_propotion / munic_sample_propotion`)
             | (Adjusts sample distribution to match population)
             |
             V
+-----------------------------------------+
| 6. Survey Data Analysis                 |
+-----------------------------------------+
         |
         +-- Estimate Total Pop from Sample
         |    (`egen total_pop = total(abs_weight_munic)`)
         |
         +-- Declare Survey Design (in 'smpl20')
         |    (`svyset [pweight = abs_weight_munic]`)
         |
         +-- Compare Population vs. Raw Sample vs. Weighted Sample:
             |
             +-- Mean Income (`frame census: mean income` vs `mean income` vs `svy: mean income`)
             +-- Proportion Urban (`frame census: proportion urban` vs `proportion urban` vs `svy:
proportion urban`)
             |
             V
+-------------------------------------------+
| 7. Alternative Weighting Analysis        |
+-------------------------------------------+
         |
         +-- Compare Total Income:
             |
             +-- Population (`frame census: sum income`)
             +-- Raw Sample (`sum income` in 'smpl20')
             +-- Prop. Weighted Sample (`sum income [aweight=prop_weight_munic]` in 'smpl20')
```

2. Stratified Sampling (Fixed Count per Municipality)

```
************************************************************************
* Drawing equal samples from each municipality
************************************************************************

frame change census
capture {
    * Switch back to the default frame
    frame change default
    * Drop the 'smpl20' frame if it exists
    frame drop smpl_fixed
}
frame put *, into(smpl_fixed)



frame change smpl_fixed
set seed 123
bysort municipality_id: sample 100, count

gen sample_size = _N

bysort province: gen prov_sample_size = _N
bysort municipality_id: gen munic_sample_size = _N

bysort province: gen prov_sample_propotion = prov_sample_size / sample_size
bysort municipality_id: gen munic_sample_propotion = munic_sample_size / sample_size

gen abs_weight_munic = munic_pop / munic_sample_size
gen prop_weight_munic = munic_pop_propotion / munic_sample_propotion

* --- Survey Data Analysis ---
egen total_pop = total(abs_weight_munic)

svyset [pweight = abs_weight_munic]
frame census: mean income
mean income
svy: mean income
```



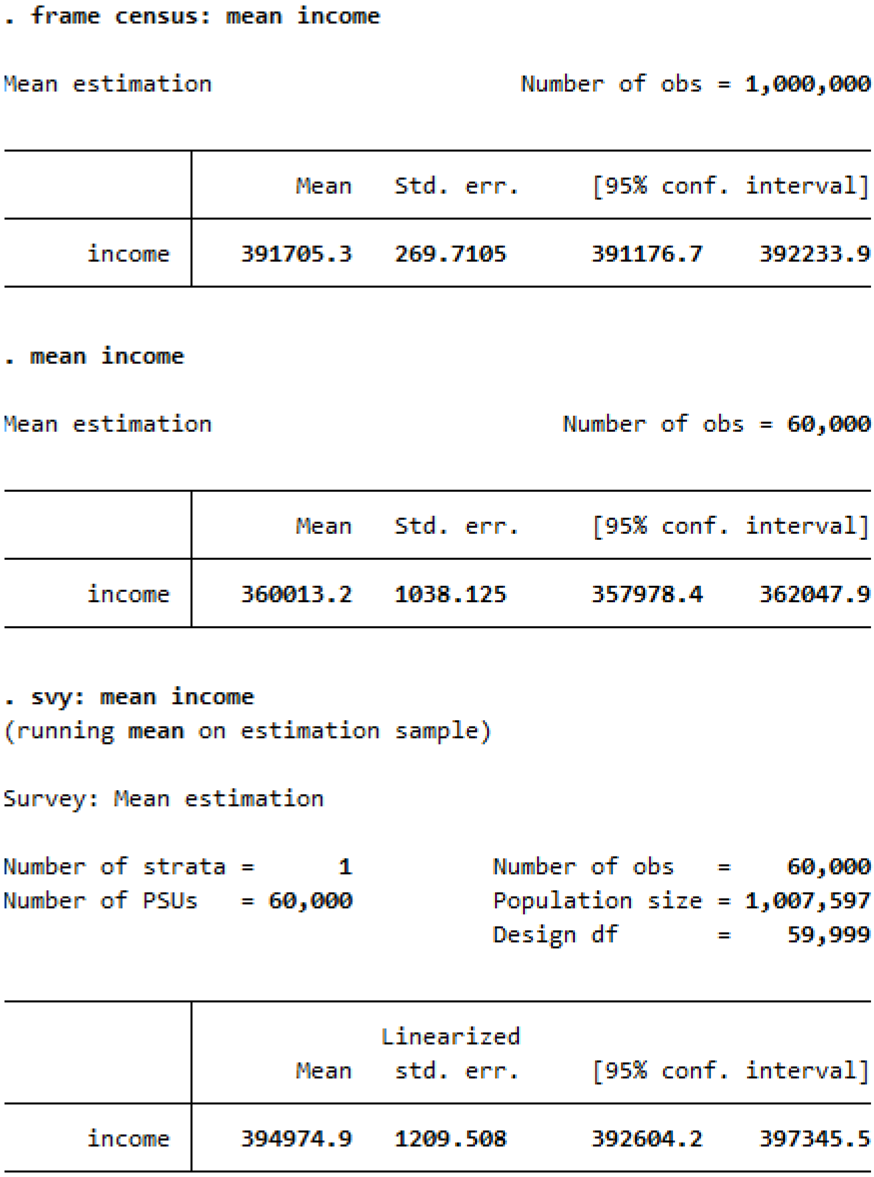
```
. frame census: mean income

Mean estimation                   Number of obs = 1,000,000

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   391705.3   269.7105      391176.7    392233.9
--------------------------------------------------------------

. mean income

Mean estimation                      Number of obs = 75,300

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |     356734   926.8036      354917.4    358550.5
--------------------------------------------------------------

. svy: mean income
(running mean on estimation sample)

Survey: Mean estimation

Number of strata =      1         Number of obs   =     75,300
Number of PSUs   = 75,300         Population size = 1,000,000
                                  Design df       =     75,299

--------------------------------------------------------------
             |             Linearized
             |       Mean   std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   392184.5   1085.448      390057.1      394312
--------------------------------------------------------------

frame census: proportion urban
proportion urban
svy: proportion urban

* --- Alternative Weighting Analysis ---
frame census: sum income
sum income
sum income [aweight = prop_weight_munic]
```

```
+----------------------------------------------------------------+
| [Stata Script: Stratified Sampling (Fixed Count per Municipality)] |
+----------------------------------------------------------------+
                              |
                              V
+--------------------------------------------------+
| 1. Frame Management (Start from 'census' frame) |
+--------------------------------------------------+
         |
         +-- Cleanup Old Frame (`capture frame drop smpl_fixed`)
         |    (In 'default' frame temporarily)
         |
         +-- Create New Frame Base (`frame put *, into(smpl_fixed)`)
         |    (Copies 'census' data, including pop variables)
         |
         +-- Switch to New Frame (`frame change smpl_fixed`)
         |
         +-- Set Random Seed (`set seed 123`)
         |
         V
+------------------------------------------------------+
| 2. Sampling (Fixed Count per Stratum/Municipality)   |
+------------------------------------------------------+
         |
         +-- Draw Fixed Sample per Municipality (`bysort municipality_id: sample 100, count`)
         |   (Takes 100 obs from each municipality, if available)
         |
         V
+------------------------------------------------------+
| 3. Sample Analysis (in 'smpl_fixed' frame)           |
+------------------------------------------------------+
         |
         +-- Calculate Sample Sizes:
         |   |
         |   +-- Total Sample Size (`gen sample_size = _N`)
         |   +-- Per Province (`bysort province: gen prov_sample_size = _N`)
         |   +-- Per Municipality (`bysort municipality_id: gen munic_sample_size = _N`)
         |       (Note: munic_sample_size will be <= 100)
         |
         +-- Calculate Sample Proportions:
             |
             +-- Per Province (`bysort province: gen prov_sample_propotion = ...`)
             +-- Per Municipality (`bysort municipality_id: gen munic_sample_propotion = ...`)
             |
             V
+------------------------------------------------------+
| 4. Calculate Sampling Weights (in 'smpl_fixed')      |
+------------------------------------------------------+
         |
         +-- Absolute/Design Weight (Municipality)
         |   | (`gen abs_weight_munic = munic_pop / munic_sample_size`)
         |   | (Uses fixed sample size per munic.)
         |
         +-- Proportional/Balancing Weight (Municipality)
             | (`gen prop_weight_munic = munic_pop_propotion / munic_sample_propotion`)
             | (Uses proportions based on fixed sample size)
             |
             V
+------------------------------------------------------+
| 5. Survey Data Analysis (Using Absolute Weights)     |
+------------------------------------------------------+
         |
         +-- Estimate Total Pop from Sample
         |   (`egen total_pop = total(abs_weight_munic)`)
         |
         +-- Declare Survey Design
         |   (`svyset [pweight = abs_weight_munic]`)
         |
         +-- Compare Population ('census') vs. Raw Sample ('smpl_fixed') vs. Weighted Sample ('svy:'):
             |
             +-- Mean Income (`frame census: mean income` vs `mean income` vs `svy: mean income`)
             +-- Proportion Urban (`frame census: proportion urban` vs `proportion urban` vs `svy:
proportion urban`)
             |
             V
+-----------------------------------------------------------+
| 6. Alternative Weighting Analysis (Using Proportional Wt) |
+-----------------------------------------------------------+
         |
         +-- Compare Total Income:
             |
             +-- Population (`frame census: sum income`)
             +-- Raw Sample (`sum income` in 'smpl_fixed')
             +-- Prop. Weighted Sample (`sum income [aweight=prop_weight_munic]` in 'smpl_fixed')
```

3. Multistage Sampling (SRS Municipality -> SRS Individual)

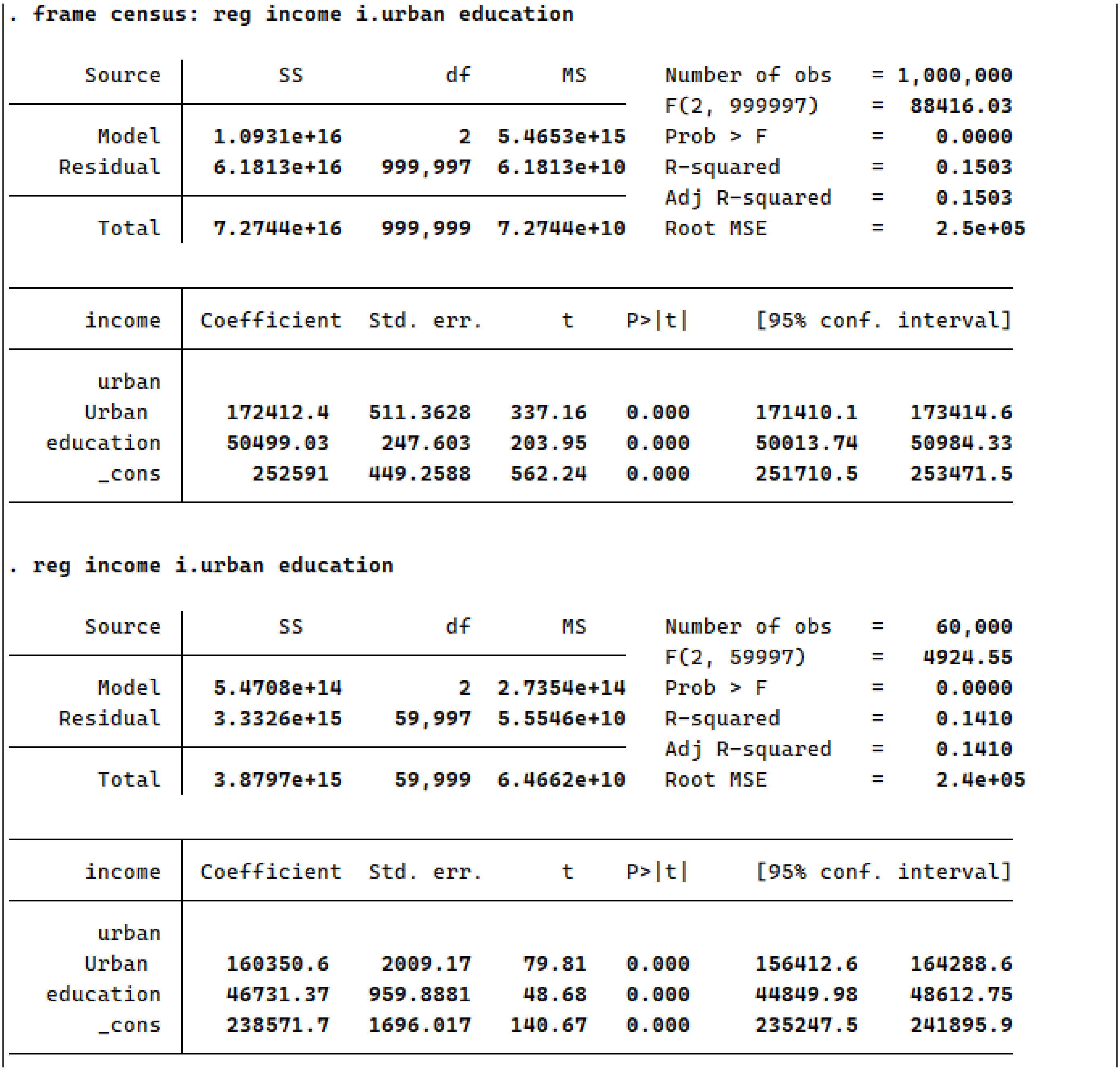
```
************************************************************************
* Multistage sampling
************************************************************************

frame change census
```

```
capture {
    * Switch back to the default frame
    frame change default
    * Drop the 'smpl20' frame if it exists
    frame drop smpl_mult_stg
}
frame put *, into(smpl_mult_stg)

capture {
    * Switch back to the default frame
    frame change default
    * Drop the 'smpl20' frame if it exists
    frame drop temp
}
frame put *, into(temp)

*Randomly selection 300 municipalities
frame change temp
collapse (count) province, by(municipality_id)
drop province
sample 300, count
gen smpld = 1

*merging and keeping randomly selected municipalities
frame change smpl_mult_stg

frlink m:1 municipality_id, frame(temp)
frget smpld, from(temp)
keep if smpld == 1
drop temp smpld


bysort municipality_id: sample 200, count

gen sample_size = _N

bysort province: gen prov_sample_size = _N
bysort municipality_id: gen munic_sample_size = _N

bysort province: gen prov_sample_propotion = prov_sample_size / sample_size
bysort municipality_id: gen munic_sample_propotion = munic_sample_size / sample_size

gen abs_weight_munic = (753/300) * (munic_pop/munic_sample_size)
gen prop_weight_munic = (753/300) * (munic_pop_propotion/munic_sample_propotion)

* --- Survey Data Analysis ---
egen total_pop = total(abs_weight_munic)

svyset [pweight = abs_weight_munic]
frame census: mean income
mean income
svy: mean income
```

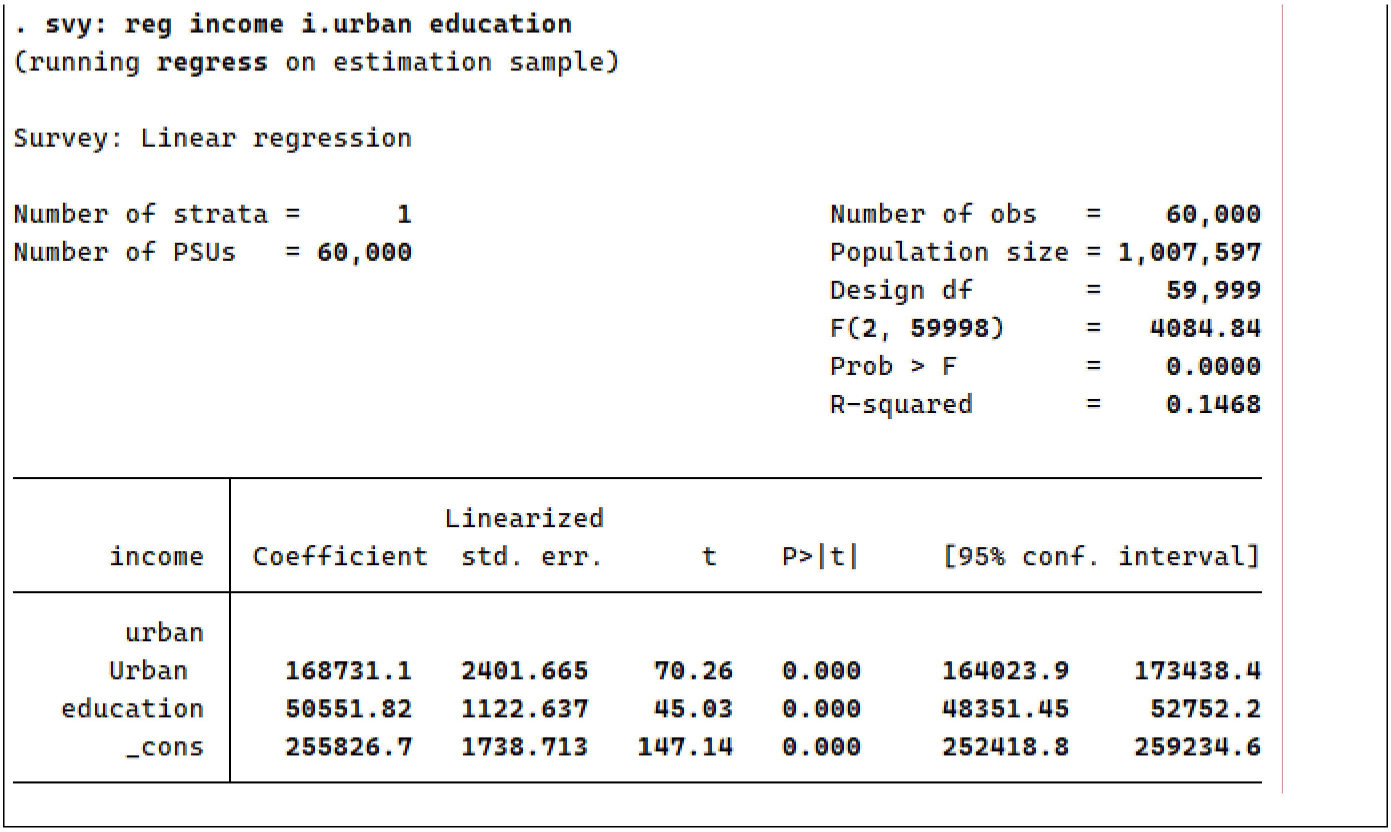


```
. frame census: mean income

Mean estimation                    Number of obs = 1,000,000

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   391705.3   269.7105      391176.7    392233.9
--------------------------------------------------------------

. mean income

Mean estimation                       Number of obs = 60,000

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   360013.2   1038.125      357978.4    362047.9
--------------------------------------------------------------

. svy: mean income
(running mean on estimation sample)

Survey: Mean estimation

Number of strata =      1          Number of obs   =    60,000
Number of PSUs   = 60,000          Population size = 1,007,597
                                   Design df       =    59,999

--------------------------------------------------------------
             |             Linearized
             |       Mean   std. err.     [95% conf. interval]
-------------+------------------------------------------------
      income |   394974.9   1209.508      392604.2    397345.5
--------------------------------------------------------------
```

```
frame census: proportion urban
proportion urban
svy: proportion urban

* --- using weight in regression ---
frame census: reg income i.urban education
reg income i.urban education
svy: reg income i.urban education
```



```
. frame census: reg income i.urban education

      Source |       SS           df       MS      Number of obs   = 1,000,000
-------------+----------------------------------   F(2, 999997)    =  88416.03
       Model |  1.0931e+16         2  5.4653e+15   Prob > F        =    0.0000
    Residual |  6.1813e+16   999,997  6.1813e+10   R-squared       =    0.1503
-------------+----------------------------------   Adj R-squared   =    0.1503
       Total |  7.2744e+16   999,999  7.2744e+10   Root MSE        =    2.5e+05

------------------------------------------------------------------------------
      income | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
       urban |
      Urban  |   172412.4   511.3628   337.16   0.000     171410.1    173414.6
   education |   50499.03    247.603   203.95   0.000     50013.74    50984.33
       _cons |     252591   449.2588   562.24   0.000     251710.5    253471.5
------------------------------------------------------------------------------

. reg income i.urban education

      Source |       SS           df       MS      Number of obs   =    60,000
-------------+----------------------------------   F(2, 59997)     =   4924.55
       Model |  5.4708e+14         2  2.7354e+14   Prob > F        =    0.0000
    Residual |  3.3326e+15    59,997  5.5546e+10   R-squared       =    0.1410
-------------+----------------------------------   Adj R-squared   =    0.1410
       Total |  3.8797e+15    59,999  6.4662e+10   Root MSE        =    2.4e+05

------------------------------------------------------------------------------
      income | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
       urban |
      Urban  |   160350.6    2009.17    79.81   0.000     156412.6    164288.6
   education |   46731.37   959.8881    48.68   0.000     44849.98    48612.75
       _cons |   238571.7   1696.017   140.67   0.000     235247.5    241895.9
------------------------------------------------------------------------------
```



```
. svy: reg income i.urban education
(running regress on estimation sample)

Survey: Linear regression

Number of strata =      1                    Number of obs   =     60,000
Number of PSUs   = 60,000                    Population size = 1,007,597
                                             Design df       =     59,999
                                             F(2, 59998)     =    4084.84
                                             Prob > F        =     0.0000
                                             R-squared       =     0.1468

------------------------------------------------------------------------------
             |             Linearized
      income | Coefficient  std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
       urban |
       Urban |   168731.1   2401.665    70.26   0.000     164023.9    173438.4
   education |   50551.82   1122.637    45.03   0.000     48351.45     52752.2
       _cons |   255826.7   1738.713   147.14   0.000     252418.8    259234.6
------------------------------------------------------------------------------
```

```
+--------------------------------------------------+
| [Stata Script: Multistage Sampling]              |
+--------------------------------------------------+
                        |
                        V
+--------------------------------------------------+
| 1. Frame Preparation                             |
+--------------------------------------------------+
         |
         +-- Clean up old 'smpl_mult_stg' frame
         |
         +-- Create 'smpl_mult_stg' frame from 'census' (will hold final sample)
         |
         +-- Clean up old 'temp' frame
         |
         +-- Create 'temp' frame from 'census' (for PSU selection)
         |
         V
+--------------------------------------------------+
| 2. Stage 1: Select Municipalities (PSUs)         |
+--------------------------------------------------+
         |
         +-- Switch to 'temp' frame
         |
         +-- Create dataset of unique municipalities (`collapse ... by(municipality_id)`)
         |
         +-- Sample 300 municipalities (`sample 300, count`)
         |
         +-- Mark selected municipalities (`gen smpld = 1`)
         |
         V
+--------------------------------------------------+
| 3. Merge & Filter Data by Selected PSUs          |
+--------------------------------------------------+
         |
         +-- Switch to 'smpl_mult_stg' frame
         |
         +-- Link to 'temp' frame (`frlink`)
         |
         +-- Get selection marker (`frget smpld`)
         |
         +-- Keep only observations in selected municipalities (`keep if smpld == 1`)
         |
         +-- Clean up (`drop temp smpld`)
         |
         V
+------------------------------------------------+
| 4. Stage 2: Select Individuals (SSUs)          |
+------------------------------------------------+
         |
         +-- Within each selected municipality:
         |   `bysort municipality_id: sample 200, count`
         |   (Sample 200 individuals)
         |
         V
+------------------------------------------------+
| 5. Sample Analysis (in 'smpl_mult_stg')        |
+------------------------------------------------+
         |
         +-- Calculate Sample Sizes (Total, Province, Municipality)
         |   (`gen sample_size`, `bysort ... gen ..._size`)
         |
         +-- Calculate Sample Proportions (Province, Municipality)
         |   (`bysort ... gen ..._propotion`)
         |
         V
+------------------------------------------------+
| 6. Weight Calculation (in 'smpl_mult_stg')     |
+------------------------------------------------+
         |
         +-- Absolute Weight (Reflects both stages)
         |   `gen abs_weight_munic = (753/300) * (munic_pop/munic_sample_size)`
         |   (Inverse probability: PSU selection * SSU selection)
         |
         +-- Proportional Weight (Adjusted for proportions)
         |   `gen prop_weight_munic = (753/300) * (munic_pop_propotion/munic_sample_propotion)`
         |
         V
+------------------------------------------------+
| 7. Survey Data Analysis (Means/Proportions)    |
+------------------------------------------------+
         |
         +-- Estimate Total Pop (`egen total_pop`)
         |
         +-- Declare Survey Design (`svyset [pweight = abs_weight_munic]`)
         |
         +-- Compare Population vs. Raw Sample vs. Weighted Sample:
         |   +-- Mean Income
         |   +-- Proportion Urban
         |
         V
+------------------------------------------------+
| 8. Regression Analysis                         |
+------------------------------------------------+
         |
         +-- Compare Regression (income ~ urban + education):
             |
             +-- Population (`frame census: reg ...`)
             +-- Raw Sample (`reg ...`)
             +-- Weighted Sample (`svy: reg ...`)
```

Absolute/Design Weight

- **Definition**: This weight is used to correct for unequal probabilities of selection in a survey sample.
- **Formula**: $\text{Design Weight} = \frac{1}{\text{Probability of Selection}} = \frac{\text{Population Size}}{\text{Sample Size}}$
- **Purpose**: Ensures that each unit in the sample represents the correct number of units in the population, compensating for any over- or under-sampling.
- **When to Use**: Use design weights when you have intentionally oversampled or undersampled certain groups within your population. For example, if you oversample

minority groups to ensure adequate representation, design weights will adjust for this oversampling[1].

Proportional/Balancing Weight

- **Definition**: This weight adjusts the sample to match the population proportions.
- **Formula**: $\text{Proportional Weight} = \frac{\text{Population Proportion}}{\text{Sample Proportion}}$
- **Purpose**: Balances the sample to reflect the true distribution of the population, correcting for any discrepancies between the sample and the population.
- **When to Use**: Use proportional weights when your sample does not accurately reflect the population's demographic proportions. For instance, if your sample has more females than males compared to the population, proportional weights will correct this imbalance[2].

Absolute/Design Weight Example

Suppose we have a population of 10,000 people and we select a sample of 500 people.

- **Population Size (N)**: 10,000
- **Sample Size (n)**: 500
- **Probability of Selection**: $\frac{n}{N} = \frac{500}{10{,}000} = 0.05$
- **Design Weight**: $\frac{1}{\text{Probability of Selection}} = \frac{1}{0.05} = 20$

Each person in the sample represents 20 people in the population.

Proportional/Balancing Weight Example

Suppose the population is 60% female and 40% male, but our sample is 50% female and 50% male.

- **Population Proportion (Female)**: 0.60
- **Sample Proportion (Female)**: 0.50
- **Proportional Weight (Female)**: $\frac{\text{Population Proportion (Female)}}{\text{Sample Proportion (Female)}} = \frac{0.60}{0.50} = 1.2$
- **Population Proportion (Male)}**: 0.40
- **Sample Proportion (Male)}**: 0.50
- **Proportional Weight (Male)}**: $\frac{\text{Population Proportion (Male)}}{\text{Sample Proportion (Male)}} = \frac{0.40}{0.50} = 0.8$

In this case, females in the sample are weighted by 1.2 to reflect their higher proportion in the population, while males are weighted by 0.8 to reflect their lower proportion.

Multistage Sampling Weight Formula

1. **First Stage Weight (e.g., selecting clusters like schools)**: $W1 = \frac{1}{P1}$ where ($P1$) is the probability of selecting the cluster.
2. **Second Stage Weight (e.g., selecting units within clusters like classrooms within schools)**: $W2 = \frac{1}{P2}$ where ($P2$) is the probability of selecting the unit within the cluster.
3. **Overall Weight**: $W \ = \ W1 \ \times W2$

Example

Suppose we have a two-stage sampling process:

1. **First Stage**: Select schools with a probability proportional to size (PPS).
2. **Second Stage**: Select students within each selected school with equal probability.

- **First Stage**: Probability of selecting a school (PPS): $P_1 = \frac{\text{Number of selected schools}}{\text{Total number of schools}}$
- **Second Stage**: Probability of selecting a student within a school: $P_2 = \frac{\text{Number of selected students}}{\text{Total number of students in the selected school}}$
- **First Stage Weight**: $W1 = \frac{1}{P1}$
- **Second Stage Weight**: $W2 = \frac{1}{P2}$
- **Overall Weight**: $W \ = \ W1 \ \times W2$

Numerical Example

1. **First Stage**: Select 10 schools out of 100. $P1 = \frac{10}{100} = 0.1W1 = \frac{1}{0.1} = 10$
2. **Second Stage**: Select 20 students out of 200 in each selected school. $P2 = \frac{20}{200} = 0.1W2 = \frac{1}{0.1} = 10$
3. **Overall Weight**: $W \ = \ W1 \ \times W2 \ = \ 10 \ \times 10 \ = \ 100$

Each selected student represents 100 students in the population.