**Day 1 : Data processing**

**Session 1: Introduction to Stata and Data Import**

* 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. **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**

* 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"**

* 1. **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**

* 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**

* 1. **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**

* 1. **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)

* 1. **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**

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**

1. **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**

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)

A diagram of a function

Description automatically generated 

**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 \*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Show duplicates

duplicates examples

\* List duplicate observations

duplicates **list**

\* Drop duplicates

duplicates **drop**

1. **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)

1. **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,

1. Import ***hl.sav*** from NMICS6 dataset [import spss "https://gitlab.com/misc.a/referenced/-/raw/main/NMICS6/hl.sav", clear]
2. Keep ***HH1, HH2, and HL1*** variables.
3. 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**

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

1. **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**

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

}

1. **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**

1. **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**

1. **The concept of normal distribution**
2. **What is a Normal Distribution?**

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

A diagram of a normal distribution

Description automatically generated

1. **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.

1. **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](https://www.youtube.com/shorts/TwctT3Ncm1w))
* **Statistical Inferences:** Helps in making predictions and decisions based on data.

1. **Hypothesis testing**
2. **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 (**H0**)**: 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 (**H1**)**: This is what you want to prove, stating there is an effect or a difference.

Example: "The average age is not equal to 20."

1. **Procedure of hypothesis testing**

* State the null and alternative hypothesis. (e.g. , )
* Collect sample data.
* Calculate sample mean and stadard error ().
* Calculate t-statistics ( ).
* Compare absolute value of t-statistics |t| with critical values for given level of significance (). [1.65 (10% significance level), 1.96 (5%), 2.58 (1%)]

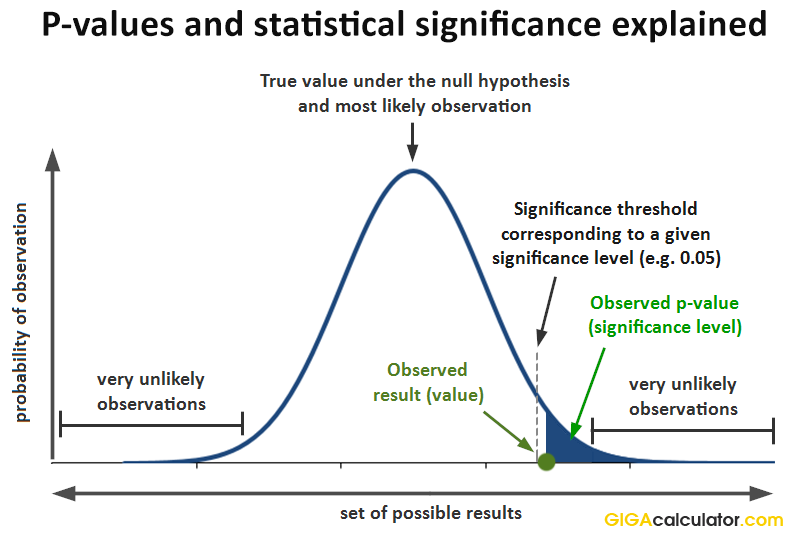
A diagram of a function

Description automatically generated

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

1. **Hypothesis testing with p-value**

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



* **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.

1. **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

1. **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**

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

**A screenshot of a computer

Description automatically generated**

1. **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**

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.

1. **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

1. **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

1. **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

A screenshot of a computer screen

AI-generated content may be incorrect.

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')

* 1. 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

A screenshot of a document

AI-generated content may be incorrect.

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')

* 1. 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

A screenshot of a document

AI-generated content may be incorrect.

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

***A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.***

+-------------------------------------------------+

| [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**:
* **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]](https://pages.nyu.edu/jackson/design.of.social.research/Readings/Johnson%20-%20Introduction%20to%20survey%20weights%20%28PRI%20version%29.pdf).

**Proportional/Balancing Weight**

* **Definition**: This weight adjusts the sample to match the population proportions.
* **Formula**:
* **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]](https://www.decisionanalyst.com/blog/dataweighting/).

**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**:
* **Design Weight**:

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)**:
* **Population Proportion (Male)}**: 0.40
* **Sample Proportion (Male)}**: 0.50
* **Proportional Weight (Male)}**:

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)**: where ( ) is the probability of selecting the cluster.
2. **Second Stage Weight (e.g., selecting units within clusters like classrooms within schools)**: where ( ) is the probability of selecting the unit within the cluster.
3. **Overall Weight**:

**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):
* **Second Stage**: Probability of selecting a student within a school:
* **First Stage Weight**:
* **Second Stage Weight**:
* **Overall Weight**:

**Numerical Example**

1. **First Stage**: Select 10 schools out of 100.
2. **Second Stage**: Select 20 students out of 200 in each selected school.
3. **Overall Weight**:

Each selected student represents 100 students in the population.