Day 1 : Data processing

Session 1: Introduction to Stata and Data Import

1.1. Importing dataset

```
* Import CSV file
import delimited "path_to_your_file.csv", clear

* Import Excel file (First Sheet)
import excel "path_to_your_file.xlsx", sheet("Sheet1") firstrow clear

* Import SPSS file
import spss "path_to_your_file.sav", clear

* Import Stata .dta file
use "path_to_your_file.dta", clear
```

These examples cover the basic commands needed to import different file formats into Stata. Make sure to replace "path_to_your_file" with the actual path to your dataset.

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

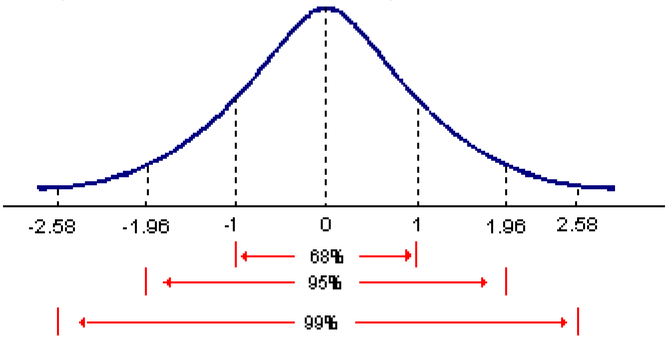
Session 5: Outliers, duplicates, recoding, dummy variable generation, and groupwise calculations

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 **********
***************

* Show duplicates
duplicates examples

* List duplicate observations
duplicates list

* Drop duplicates
duplicates drop
```

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
```

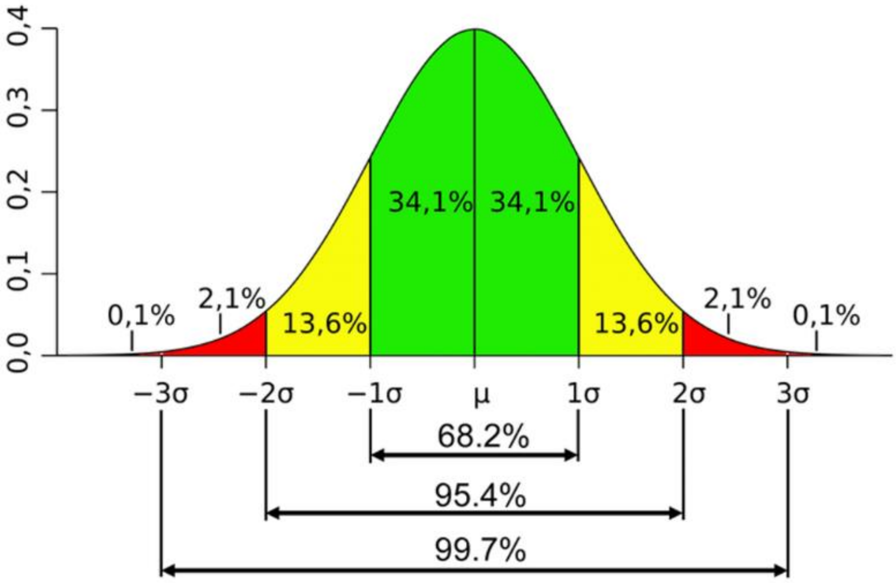
Day 3 : Data Analysis

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?

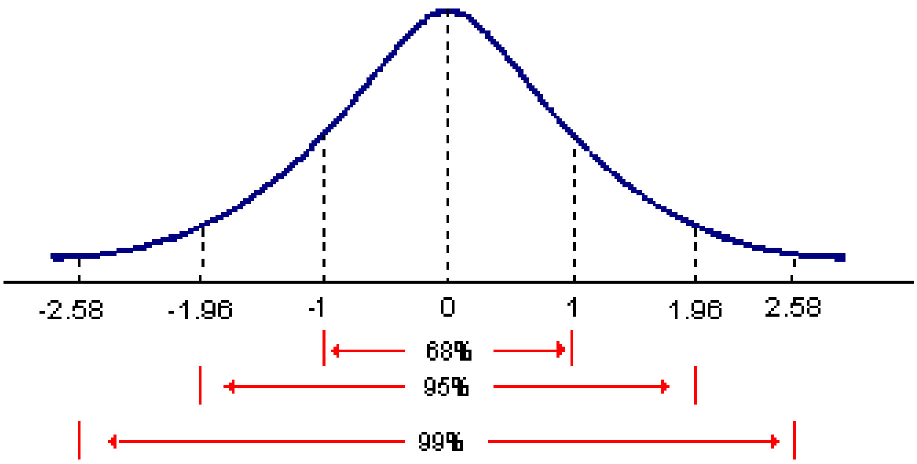
- Hypothesis testing is a method used to decide whether there is enough evidence to support a particular claim about a population based on a sample of data.
- **Null Hypothesis (H_0):** This is the default statement that there is no effect or no difference. It assumes that any observed differences are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

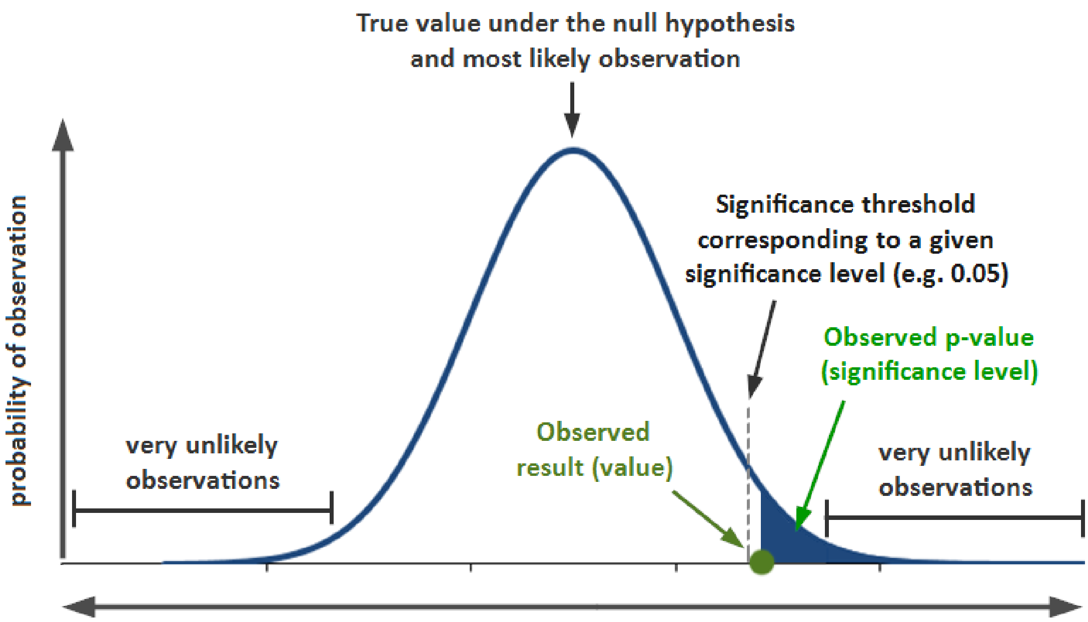
- State the null and alternative hypothesis. (e.g. $H_0: \mu = 0, H_1: \mu \neq 0$)
- Collect sample data.
- Calculate sample mean and stadard error ($\frac{s}{\sqrt{n}}$).
- Calculate t-statistics ($t = \frac{\bar{X}-\mu}{Standar\ Error}$).
- Compare absolute value of t-statistics |t| with critical 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|.



- **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
```