STATA Basics

Session D1S2: STATA Interface and Windows

1. Steps to Set the Working Directory in Stata

Method 1: Using the Command Window

1. Identify the path to the folder you want to set as the working directory.
   Example (Windows): `C:\Users\YourName\Documents\StataWork`
   Example (Mac): `/Users/YourName/Documents/StataWork`
2. Use the `cd` command to set the working directory.

```
cd " D:\Local Disk D\A_Training Section\Training Materials\STATA_2081"
```

   Or for Mac/Linux:

```
cd "/Users/YourName/Documents/StataWork"
```

4. Verify the working directory using the `pwd` (print working directory) command:

```
pwd
```

Method 2: Using the File Menu

1. Go to **File > Change Working Directory**.
2. Browse to the folder you want to set as the working directory.
3. Click **OK** to confirm.

Tips:

- Always set the working directory at the beginning of your session.
- Add the `cd` command to the top of your **do-file** to ensure consistency:
- `cd "C:\Users\YourName\Documents\StataWork"`
- `help (h) command`

2. System used data: to use example dataset installed with Stata

```
sysuse
sysuse dir
```

3. Basic STATA Interface and Windows

This session is designed to introduce participants to the Stata interface, its windows, and basic navigation. The goal is to familiarize users with the environment to build confidence in using Stata.

3.1 Navigating the Stata Windows

- **Command Window**:
  - Inputting simple commands:

```
sysuse auto
summarize
```

  - Using the arrow keys to recall past commands.
- **Results Window**:
  - Understanding outputs and messages.
  - Using the scroll bar to review past outputs.
- **Variables Window**:
  - Sorting variables by name, type, or label.
  - Clicking variables to use them in commands.
- **Data Editor**:
  - Switching between Browse (`Ctrl+B`) and Edit (`Ctrl+E`) modes.
  - Searching for specific observations or values.

3.2 Accessing and Using the Do-File Editor

- Purpose of a Do-File (reproducibility of analysis).
- Writing and running basic commands in the Do-File Editor:

```
sysuse auto
summarize
list make price if price > 10000
```

3.3 Managing Files

- **Opening a Dataset**:
  - Using the File Menu:
    File > Open
  - Using a command:

```
use "filename.dta", clear
```

- **Saving a Dataset**:
  - Using the File Menu:
    File > Save As
  - Using a command:

```
save "newfile.dta", replace
```

3.4 Basic Stata Commands to Explore the Interface

- **To clear window output**
  `cls`
- **To edit**
  `Edit.`
- **Load Example Dataset**:

```
sysuse auto
import excel "filename.xlsx", firstrow
```

- **Describe/check Variables in the Dataset**:
  `describe`
- **Browse Data**:
  `browse`
- **Generate a New Variable**:
  `generate price_in_thousands = price / 1000`
- **Save a Dataset:**
  `save "mydata.dta", replace`
- **View the First Few Observations:**
  `list 1/10`
- **Summarize Variables:**
  `Summarize var`
  `Summarize var, detail`

3.5 Keyboard Shortcuts and Tips

- **Basic Shortcuts**:
  - `Ctrl+D`: Open the Do-File Editor.
  - `Ctrl+9`: Open the Data Editor (Browse).
  - `Ctrl+8`: Open the Data Editor (Edit).

3.6 Interactive Demo and Hands-On Practice

- **Live Demo**:
  - Open the built-in dataset (`sysuse auto`).
  - Explore the Variables and Data Editor windows.
  - Execute basic commands and view results.
- **Practice Exercise**:
  1. Load a dataset and browse through it.
  2. Use the Do-File Editor to run:

     ```
     summarize mpg
     list make price if mpg > 25
     ```

  3. Save the dataset with a new name.

---

4. Coding styles generally refer to the approaches and practices used to write clear, efficient, and reproducible code.

---

1. Interactive vs. Do-File-Based Coding

Interactive Coding

- Commands are entered directly in the **Command Window** and executed immediately.
- Suitable for quick exploration or testing small tasks.
- **Drawbacks**: Hard to replicate or document the workflow.

Do-File-Based Coding

- Commands are written in the **Do-File Editor** and executed as a batch.
- Promotes reproducibility and documentation.
- Example:

```
* Load data
sysuse auto

* Summarize variables
summarize

* Generate new variable
generate price_in_thousands = price / 1000
```

---

2. Imperative vs. Modular Coding

Imperative Coding

- Code is written as a sequence of commands, often without abstraction or modularity.
- Quick for small projects but harder to debug in larger ones.
- Example:

```
sysuse auto
generate mpg_efficiency = mpg > 25
summarize mpg
```

Modular Coding

- Code is divided into reusable sections or scripts for better organization.
- Often involves the use of **programs**, **macros**, or **ado-files**.
- Example:

```
* Define a program
program define summarize_auto
    sysuse auto, clear
    summarize
end

* Call the program
summarize_auto
```

---

3. Inline vs. Annotated Coding

Inline Coding

- Code is written without comments or annotations.
- Example:

```
sysuse auto
summarize
```

Annotated Coding

- Code includes comments for clarity and documentation.
- Recommended for teamwork and reproducibility.
- Example:

```
* Load the built-in auto dataset
sysuse auto

* Summarize key variables
summarize mpg price
```

---

4. Long Command Style vs. Short Command Style

Long Command Style

- Uses detailed and explicit syntax for clarity.
- Example:

```
summarize mpg, detail
```

Short Command Style

- Uses minimal syntax for brevity.
- Example:

```
sum mpg
```

---

5. Script-Oriented vs. GUI-Based Coding

Script-Oriented Coding

- All tasks are performed using commands in the **Do-File Editor** or Command Window.
- Example:

```
regress price mpg weight
```

GUI-Based Coding

- Utilizes Stata's graphical user interface (e.g., menus and dialog boxes) to execute commands.
- Suitable for beginners but less reproducible.

---

6. Efficient vs. Verbose Coding

Efficient Coding

- Focuses on writing concise and optimized code.
- Example:

```
gen high_mpg = (mpg > 25)
```

Verbose Coding

- Uses longer code for clarity or step-by-step execution.
- Example:

```
generate high_mpg = .
replace high_mpg = 1 if mpg > 25
replace high_mpg = 0 if mpg <= 25
```

---

7. Descriptive vs. Cryptic Variable Naming

Descriptive Naming

- Uses meaningful variable names for clarity.
- Example:

```
generate fuel_efficiency = mpg / 10
```

Cryptic Naming

- Uses short or unclear variable names.
- Example:

```
gen fe = mpg / 10
```

---

8. Dynamic vs. Hard-Coded Styles

Dynamic Coding

- Uses macros and loops to make the code adaptable to different datasets.
- Example:

```
foreach var in mpg weight price {
    summarize `var'
}
```

Hard-Coded Coding

- Variables and values are explicitly specified.
- Example:

```
summarize mpg
summarize weight
summarize price
```

Session D2S1: Generating Data Files, Logical Operators (Conceptual), and Basic Command/Syntax

---

1. Generating Data Files

1. **Introduction to Datasets in Stata**
   - Types of data files (.dta, .csv, .xls, .txt).
   - Structure of Stata datasets: variables, observations, labels.
2. **Creating Data Files in Stata**
   - Using the `input` command:

```
clear
input age income
25 50000
30 60000
35 70000
End
```

   - Generating variables using `generate` and `replace`:

```
generate Bonus = income * 0.1
replace Salary = income + 5000 if Age > 30
```

   - Copy Data from Other Formats/using data editor

3. **Importing External Data**
   - Importing a CSV file:

```
import delimited "datafile.csv"
```

   - Importing an Excel file:

```
import excel "datafile.xlsx", firstrow
```

4. **Exporting Data Files**
   - Save data in Stata format:

```
save "mydata.dta", replace
```

   - Exporting to CSV:

```
export delimited using "outputfile.csv"
```

```
*******Descriptive Statistics/Analysis******
// Basic summary
use data3.dta,clear
summarize age
summarize age, detail
sum age if gender==1
sum age if ethnicity==3
egen mean_age = mean(age), by(ethnicity)
tab mean_age ethnicity
mean age
table ethnicity, statistic(mean age)
tabstat age, by(ethnicity) stat(mean sd min max)
// Percentile
centile age, centile(25 40 50 60 70 75 90)
// Exploratory analysis
```

```
// histogram
histogram age
histogram age, normal
histogram age, percent
// Boxplot
graph box age
graph box age, over(ethnicity)

// correlation
corr age edutype married yearsedu

pwcorr age edutype married yearsedu, sig

reg yearsedu edutype

**********BarGraph****
sysuse auto,clear
graph bar (count), over (rep78)

graph bar (count), over (rep78) title("Frequency of Repair Records")

graph bar (mean) price, over (rep78)
codebook foreign
graph bar (mean) price, over(rep78) title("My Chart") by(foreign)  //multiple
graph

graph bar (mean) price, over(rep78) title("My Chart") by(foreign, total)

graph bar (mean) price, over(rep78) title("My Chart") by(foreign)

graph bar (mean) price, over(rep78) title("My Chart") by(foreign, total)
blabel(bar)

graph pie, over(foreign)

graph pie, over(foreign) plabel(_all percent)

graph pie, by(rep78) over(foreign) plabel(_all percent)
*******************************
sysuse uslifeexp, clear
graph twoway (line le year)

graph twoway (line le year, lcolor(red))
```

---

2. Logical Operators (Conceptual)

1. **Overview of Logical Operators in Stata**
    - **Equality:** == (equal to)
    - **Inequality:** != (not equal to)
    - **Relational operators:** <, <=, >, >=
    - **Logical AND:** &
    - **Logical OR:** |
    - **Logical NOT:** ~
    - **Basic Operators:**

Arithmetic	Relational Logical	(numeric and string)
+ addition	& and	> greater than
- subtraction	| or	< less than
* multiplication	! not	>= greater than or equal
/ division	~ not	<= lessthan or equal
^ power		== equal
- negation		!= not equal
+ string concatenation		~= not equal

A double equal sign (==) is used for equality testing.

The order of evaluation (from first to last) of all operators is ! (or ~), ^, - (negation), /, *, - (subtraction), +, != (or ~=), >, <, <=, >=, ==, &, and |.

2. **Using Logical Operators in Stata**
   - Filtering data using `if`:

```
list Name Age if Age > 25
```

   - Combining conditions:

```
list Name Age if Age > 25 & Salary < 40000
```

3. **Generating Logical Variables**
   - Creating a binary variable:

```
generate Adult = Age >= 18
```

4. **Common Logical Expressions**
   - Missing values check:

```
list Name if Age == .
```

5. Data storage types

A. Numeric Data Types

Stata stores numeric variables in six formats, depending on the level of precision and size of the data.

Overview of Numeric Data Types

Type	Bytes	Minimum Value	Maximum Value	Description
byte	1	-127	100	For small integers, uses minimal storage.
int	2	-32,767	32,740	For integers with a moderate range.
long	4	-2,147,483,647	2,147,483,620	For large integers.
float	4	~ -1.701e38	~ 1.701e38	Approx. 7 digits of precision for decimals
double	8	~ -8.988e307	~ 8.988e307	Approx. 16 digits of precision for decimal

Key Notes:

- Use **byte**, **int**, or **long** for integer values to save memory.
- Use **float** or **double** for variables requiring decimal precision.
- If exact decimal precision is critical, prefer **double**.

Examples:

- Generate an integer variable:
```
generate age = 25
```
- Create a float variable with decimals:
```
generate height = 5.7
```

B. String Data Types

String variables store text or alphanumeric characters.

Key Features:

- **String length**: Can range from 1 to 2,045 characters.
- A string variable's type is defined by its maximum length (e.g., `str10` for a string with up to 10 characters).

Examples:

- **Short string variables**:
```
generate name = "John"        // Stata assigns a string type like str4
```
- **Long string variables**:
```
generate long_text = "This is a longer description"
```

Use Case:

- Use strings for categorical data (e.g., names, IDs, and descriptive text).

C. Labelled Data

Although not a distinct storage type, variables in Stata can have **value labels** attached to make categorical data more descriptive.

Example:

- Define labels:
```
label define gender 1 "Male" 2 "Female"
```
- Assign labels to a variable:
```
label values sex gender
```

Storage Note:

- Categorical variables should ideally be stored as numeric types with labels for efficient memory use.

D. Missing Data

- **Numeric Missing**: Represented as a period (`.`) or extended missing (`.a`, `.b`, etc.). These are stored as very large numbers and do not affect data storage type.
- **String Missing**: Represented as an empty string (`""`).

Choosing the Right Data Type

Numeric or String?

- Numeric variables are faster to process and require less memory. Use them for variables like age, income, and scores.
- Use string variables for data such as names, IDs, or descriptions.

Choosing Numeric Types:

- Small integers? Use `byte` or `int`.
- Large integers? Use `long`.
- Decimal precision? Use `float` (approx. 7 digits) or `double` (16 digits).

Practical Example:

Variable	Suggested Type
Age	`byte`
Income	`long`
Weight (kg)	`float`
Product Name	`str20`
Categorical Gender	`byte` with labels

---

5. Some function in Stata

Function	Meaning	Example
abs(x)	computes the absolute value of x	abs(-2) = 2
exp(x)	calculates e to the x power.	exp(1) = 2.7182818
ln(x)	computes the natural logarithm of x	ln(10) = 2.3025851
log(x)	is a synonym for ln(x), the natural logarithm.	log(10) = 2.3025851
log10(x)	computes the log base 10 of x.	log10(1000) = 3
sqrt(x)	computes the square root of x.	sqrt(36) = 6
int(x)	gives the integer obtained by truncating x.	int(3.8532) = 3
round(x,y)	gives x rounded into units of y.	round(3.8532) = 4
norm(z)	provides the cumulative standard normal.	norm(0) = 0.5
chi2(n,x)	returns the cumulative chi-squared distribution with n degrees of freedom	chi2(100,80) =.07033507
reverse(str)	reverse the order of letters in a string (text data)	reverse(mood) = doom
uniform()	generates a random number between 0 and 1	random() = 0.739538

Examples:

```
sysuse auto, clear

* Math Function: Create log price
generate log_price = ln(price)

* Logical Function: Identify high mpg cars
generate high_mpg = (mpg > 25)

* String Function: Extract first three characters of make
generate first3_make = substr(make, 1, 3)

* Statistical Function: Calculate mean price
egen mean_price = mean(price)
```

```
* Time Function: Calculate car age
generate car_age = 2025 - year

* Random Number: Add random uniform variable
generate rand_uniform = runiform()
```

3. Basic Command Syntax

1. **Understanding Command Syntax**
   - Basic operations (as a calculator):

```
display command
dis 3+4
dis 3*4
dis 20/4
dis 45-5
display sqrt(16)
```

   - General format:

```
command varlist [if] [in] [, options]
```

   - Examples:

```
List var1 var2
list Name Age if Age < 25
summarize var, detail
```

2. **Basic Commands for Data Exploration**
   - Viewing data:

```
browse
list Name Age
describe (varlist)  *displays the characteristics of variables
```

   - Descriptive statistics:

```
summarize
summarize var, detail
```

3. **Sorting and Filtering Data**
   - Sorting:

```
sort Age
```

   - Filtering with conditions:

```
keep if Age >= 18
drop if Salary < 30000
```

4. **Renaming and Labeling Variables**
   - Renaming variables:

```
rename Salary MonthlySalary
```

   - Adding variable labels:

```
label variable Age "Age of the individual"
```

5. **Change the type of variable**
(numeric to types and vs string to types)

- Stata handles two different types of variables:
  - numeric variables – are variables whose values are only real numbers,
  - string variables – are variables whose values are string (or combinations of alphabetic and/or numeric) characters.

- `recast` *changes the type of variables

```
recast type varlist
```

- `type`: the new storage type for the variable (e.g., `byte`, `int`, `long`, `float`, `double`, `str#`).
- `varlist`: The variables to be recast.

- Convert numeric to string and vice versa:

```
tostring converts numeric variables to string
example: tostring sex, replace | generate (newvarlist)
destring converts string variables to numeric
example: destring sex, replace | generate (newvarlist)
```

In Stata, the **`tostring`** and **`destring`** commands are used to convert variables between string and numeric formats.

---

1. `tostring` Command

The **`tostring`** command converts a numeric variable into a string variable.

Syntax:

```
tostring varlist [, generate(newvar) format(%fmt) force replace]
```

Options:

- **`generate(newvar)`** : Creates a new variable instead of overwriting the original.
- **`format(%fmt)`** : Specifies the format for the conversion (e.g., `%9.0f` for integers).
- **`force`**: Forces the conversion even if some values cannot be converted to strings.
- **`replace`**: Replaces the original variable with the string version.

Examples:

a. Convert a Numeric Variable to String:

```
tostring id, generate(id_str)
```

This converts the numeric variable `id` into a string variable `id_str`.

b. Specify a Format:

```
tostring salary, generate(salary_str) format(%10.2f)
```

This converts the numeric variable `salary` into a string variable `salary_str` with two decimal places.

c. Overwrite the Original Variable:

```
tostring age, replace
```

This converts the numeric variable `age` to string, replacing the original variable.

---

2. `destring` Command

The **`destring`** command converts a string variable into a numeric variable.

Syntax:

```
destring varlist [, generate(newvar) replace ignore("chars") force]
```

Options:

- **`generate(newvar)`**: Creates a new variable instead of overwriting the original.
- **`replace`**: Replaces the original variable with the numeric version.
- **`ignore("chars")`**: Ignores specified characters (e.g., `$`, commas).
- **`force`**: Forces conversion for variables with some non-numeric values, converting unconvertible strings to missing values.

Examples:

a. Convert a String Variable to Numeric:

```
destring price, generate(price_num)
```

This converts the string variable `price` to a numeric variable `price_num`.

b. Ignore Non-Numeric Characters:

```
destring income, generate(income_num) ignore(",")
```

If the string variable `income` contains commas (e.g., `"1,000"`), this option removes the commas before conversion.

c. Overwrite the Original Variable:

```
destring score, replace
```

This converts the string variable `score` to numeric, replacing the original variable.

d. Handle Mixed Data (Numeric and Non-Numeric):

```
destring profit, replace force
```

This forces the conversion even if some values cannot be converted to numbers. Non-convertible values are replaced with missing values (.).

---

6. Formatting variable properties: the **`format`** command in Stata is used to change the display format of variables, allowing you to control how numbers, dates, and strings are displayed in your dataset and output. It does not change the actual values of the variables, only how they are presented.

Syntax:

```
format varlist %fmt
```

- **`varlist`**: List of variables whose display format you want to change.
- **`%fmt`**: The desired format (e.g., `%9.2f` for numbers with two decimal places).

Types of Formats:

1. **Numeric Formats**:
   - `%w.d` : General numeric format.
     - `w`: Total width of the display, including the decimal point.
     - `d`: Number of digits displayed after the decimal point.
   - Examples:
     - `%9.2f`: Displays a number with 9 characters wide and 2 decimal places.
     - `%5.0f`: Displays integers with 5 characters wide and no decimals.
   - **Exponential notation**:
     - `%9.2e`: Displays numbers in scientific notation.
2. **Date/Time Formats**:
   - `%td`: For daily dates (e.g., `01jan2025`).
   - `%tc`: For datetime variables.
   - Examples:
     - `%tdDD-Mon-YY`: Displays dates as `01-Jan-25`.
     - `%tdYY`: Displays the year only (e.g., `2025`).
3. **String Formats**:
   - `%ws`: String format where `w` is the width.
   - Example:
     - `%15s`: Displays up to 15 characters of a string variable.

Examples:

1. Change Numeric Display Formats:

Display with Two Decimal Places:

```
format income %9.2f
```

If `income` was originally displayed as `1234.56789`, it will now show as `1234.57`.

Display as Integers:

```
format age %3.0f
```

If `age` was `25.6`, it will now display as `26`.

2. Format Dates:

Display Dates in `DD-Mon-YY` Format:

```
format date %tdDD-Mon-YY
```

If `date` is stored as an integer (Stata's internal date storage), this will display it as `11-Jan-25`.

Display Year Only:

```
format birthdate %tdCCYY
```

This will display the year (e.g., `2025`).

3. Format Strings:

Truncate String Variables:

```
format name %10s
```

This will display up to 10 characters of the string variable `name`.

View Current Formats:

To see the current display formats of variables, use:

```
describe
```

Reverting to Default Format:

To revert to the default format, use the **`format`** command with `.`:

```
format income .
```

---

7. Sorting data

- `sort` arrange the variable either in ascending or descending order
- example: `sort age [, stable]`
- `gsort` is used to sort either in ascending or descending order.
- example:
  - `gsort age`

- `gsort age, generate(order) mfirst`
- `gsort -age, generate(order) mfirst`

Note: order = just looks like order number and mfirst = missing first

---

8. Keeping and removing variables

- `keep` stores only the specified variables or observations in the memory
- example:
  - `keep name age sex`
  - `keep if sex==2`
- `drop` eliminates variables or observations from the data in memory
- example:
  - `drop name age sex`
  - `drop if sex==2`

---

9. Creating new variable

- `generate` *and* `egen` creates new variables.
- example:
  - `gen agegrp = 1 if age>=0&age<=14`
  - `replace agegrp  = 2 if age>=15 & age<=59`
  - `replace agegrp = 3 if age>=60 & age<=98`

`egen` creates new variable using some types of functions such as mean, total etc.

example:

```
sort xhnum
egen dailyinc=total(cash_per_day), by (xhnum)
```

Other Basic Commands

- ***label variable***
- ***label define***
- ***label values***
- ***rename***
- ***recode***
- ***decode***
- ***encode***
- **order**
- **move**
- **aorder**
- **merge**
- **append**
- **Codebook**
- **Count**
- **Describe**
- **List**
- **Tabulate**
- **Table**
- **tabstat**
- **preserve**
- **restore**

List of **basic Stata commands**, categorized for ease of reference. These commands are essential for data management, analysis, and visualization in Stata.

---

1. Data Management Commands

`use`: Load a dataset.
```
use filename.dta, clear
```
`save`: Save the dataset.
```
save filename.dta, replace
```
`clear`: Clear the current dataset from memory.
```
clear
```
`describe`: Display variable names, types, and labels.
```
describe
```
`list`: Display data in the Results window.
```
list var1 var2 if condition
```
`browse`: View the data in the Data Browser.
```
browse
```
`edit`: Open the Data Editor for editing.
```
edit
```
`rename`: Rename a variable.
```
rename oldname newname
```
`drop`: Remove variables or observations.
```
drop var1
drop if condition
```
`keep`: Keep only specified variables or observations.
```
keep var1 var2
keep if condition
```
`generate`: Create a new variable.
```
generate newvar = expression
```
`replace`: Modify the values of an existing variable.
```
replace var = expression if condition
```
`recode`: Recode the values of a variable.
```
recode var1 (1=10) (2=20)
```
`sort`: Sort the dataset by one or more variables.
```
sort var1 var2
```
`order`: Reorder variables in the dataset.
```
order var1 var2
```

---

2. Data Inspection and Summarization Commands

`summarize`: Get summary statistics for variables.
```
summarize var1 var2
```
`inspect`: Examine the distribution of values.
```
inspect var1
```
`codebook`: Display detailed information about variables.
```
codebook var1
```
`tabulate`: Create frequency tables.
```
tabulate var1
tabulate var1 var2
```

mean: Compute means of variables.

```
mean var1
```

pctile: Calculate percentiles.

```
pctile newvar = var1, p(25 50 75)
summarize var1, detail
```

```
sysuse auto, clear

* Calculate median using summarize
summarize price, detail

* Generate median using egen
egen median_price = median(price)

* Calculate the 90th percentile using pctile
pctile p90_price = price, p(90)

* Calculate the 25th percentile using egen
egen p25_price = pctile(price), p(25)

* Display results
list median_price p90_price p25_price if _n == 1
```

---

3. Data Transformation Commands

egen: Generate new variables with advanced functions.

```
egen newvar = mean(var1)
```

reshape: Reshape data between wide and long formats.

```
reshape long var, i(id) j(time)
reshape wide var, i(id) j(time)
```

encode: Convert string variables to numeric variables.

```
encode strvar, gen(numvar)
```

decode: Convert numeric variables to string variables.

```
decode numvar, gen(strvar)
```

tostring: Convert numeric variables to strings.

```
tostring var, gen(strvar)
```

destring: Convert string variables to numeric variables.

```
destring strvar, replace
```

---

4. Statistical Analysis Commands

regress: Run linear regression.

```
regress y x1 x2
```

logit: Run logistic regression.

```
logit y x1 x2
```

probit: Run probit regression.

```
probit y x1 x2
```

ttest: Perform a t-test.

```
ttest var1, by(groupvar)
```

anova: Conduct an analysis of variance.

```
anova y x1 x2
```

corr: Calculate correlation coefficients.

```
corr var1 var2
```

pwcorr: Pairwise correlation matrix.

```
pwcorr var1 var2, sig
```

tabstat: Summarize statistics by groups.
```
tabstat var1, by(groupvar)
```

---

5. Graphics and Visualization Commands

histogram: Create a histogram.
```
histogram var1
```
scatter: Generate a scatter plot.
```
scatter y x
```
graph bar: Create a bar chart.
```
graph bar var1, over(groupvar)
```
graph box: Create a box plot.
```
graph box var1, over(groupvar)
```
twoway: Create a two-way graph.
```
twoway (line y x) (scatter y x)
```

---

6. Programming and Automation Commands

do: Execute a do-file.
```
do myfile.do
```
log: Start and close a log file.
```
log using mylogfile.log, replace
log close
```
foreach: Loop over variables or values.
```
foreach var in var1 var2 var3 {
    summarize `var'
}
```
forvalues: Loop over a range of numbers.
```
forvalues i = 1/10 {
    display `i'
}
```
while: Run a loop while a condition is true.
```
while x < 10 {
    display x
    replace x = x + 1
}
```

---

7. File Management Commands

dir: List files in the working directory.
```
dir
```
cd: Change the working directory.
```
cd "C:\MyFolder"
```
pwd: Display the current working directory.
```
pwd
```

---

8. Help and Documentation Commands

help: Open help for a command.
```
help regress
```
search: Search for a topic or command.

```
search regression
```
view: Open a manual or guide.
```
view manual entry [R] regress
```

Session D3S3: Data Analysis: Descriptive Statistics

Descriptive data analysis in Stata involves summarizing and exploring data using various commands.

Summary Statistics

1. **Basic Summary**
   - `summarize` (or `sum`): Provides mean, standard deviation, min, max, and observations.
   - `summarize var1 var2`
   - `summarize, detail`
   - `summarize age if ethnicity ==3`
2. **Frequency Table**
   - `tabulate` (or `tab`): Creates frequency tables for categorical variables.
   - `tabulate var1`
   - `tabulate var1 var2`
3. **Percentiles and Quartiles**
   - `centile`: Calculates percentiles for a variable.
   - `centile var1, centile(25 50 75)`
   - `centile age, centile(25 40 50 60 75 90)`
4. **Summary of All Variables**
   - `codebook`: Provides detailed metadata and descriptive statistics.
   - `codebook var1`
5. **Summary Statistics by Groups**
   - `egen`: Generates group statistics.
   - `egen mean_var1 = mean(var1), by(var2)`
   - `egen mean_age = mean(age), by(ethnicity)`

```
6. tab  mean_age ethnicity
```

Exploratory Analysis

1. **Distribution**
   - `histogram`: Displays a histogram for a variable.
   - `histogram var1, percent`
   - `histogram var1, normal`
   - `kdensity`: Displays kernel density estimation.
   - `kdensity var1`
2. **Boxplot**
   - `graph box`: Creates a boxplot for visualizing the spread and outliers.
   - `graph box var1`
   - `graph box var1, over(var2)`
   - `graph box age, over(ethnicity)`
3. **Tabulations**
   - `table`: Generates customizable tables of summary statistics.
   - `table var1, contents(mean var2)`
   - `table ethnicity, statistic(mean age)`
   - `table ethnicity gender, statistic(mean age)`

Summaries for Subgroups

1. **By Group Analysis**
    - `by`: Provides summaries by groups.
    - `by var2, sort: summarize var1`
    - `by ethnicity, sort: summarize age`
    - `tabstat`: Summary statistics for subgroups.
    - `tabstat var1, by(var2) statistics(mean sd min max)`
    - `tabstat age, by( ethnicity) stat(mean sd min max)`
    - `tabstat age, by(ethnicity) stats(mean)`
    - `collapse (mean) age, by(ethnicity)`

`summarize`: Get summary statistics for variables.

```
summarize var1 var2
```

`inspect`: Examine the distribution of values.

```
inspect var1
```

`codebook`: Display detailed information about variables.

```
codebook var1
```

`tabulate`: Create frequency tables.

```
tabulate var1
tabulate var1 var2
```

`mean`: Compute means of variables.

```
mean var1
mean age, over( gender)me
```

`pctile`: Calculate percentiles.

```
pctile newvar = var1, p(25 50 75)
summarize var1, detail

sysuse auto, clear

* Calculate median using summarize
summarize price, detail

* Generate median using egen
egen median_price = median(price)
egen md_price = median( age)


* Calculate the 25th percentile using egen
egen p25_price = pctile(price), p(25)
egen p25_age = pctile(age), p(25)

* Display results
list median_price p90_price p25_price if _n == 1
```

- `Other statistics can also be calculated using disply command.`

Correlations

- `correlate`: Calculates correlation between variables.
- `correlate var1 var2 var3`
- `pwcorr`: Displays pairwise correlations with significance levels.
- `pwcorr var1 var2, sig`
- `pwcorr age yearsedu ethnicity married , sig`

Major data cleaning commands in Stata

```
//******Major Data cleaning commands****

//1. Inspecting the Dataset
browse                  // Open the data in spreadsheet view
list                    // List data in the Results window
describe                // Overview of variables
codebook                // Detailed info on variables, including unique values
summarize               // Summary statistics for all or selected variables
summarize var1, detail // More descriptive stats, incl. percentiles

//2. Checking for Missing Values
misstable summarize     // Summary of missing values for all variables
count if missing(var1) // Count missing values in a specific variable
list if missing(var1)  // List observations with missing var1

// 3. Handling Missing or Invalid Data
replace var1 = . if var1 == 999      // Recode invalid code 999 to missing
replace var2 = 1 if var2 == .        // Replace missing with a default value
drop if missing(var3)                // Drop observations with missing var3

// 4. Dropping or Keeping Observations
drop if age < 0                     // Drop invalid ages
keep if sex == 1                    // Keep only males (sex=1)
drop if inlist(var1, 999, -9)       // Drop observations with specific values

// 5. Dropping or Renaming Variables

drop var3 var4                      // Drop variables
rename oldvar newvar                // Rename a variable
order id age sex, first             // Reorder variables in dataset

// 6. Creating or Recoding Variables
gen bmi = weight / (height^2)       // Create new variable
gen age_cat = .
replace age_cat = 1 if age < 18
replace age_cat = 2 if age >= 18 & age < 65
replace age_cat = 3 if age >= 65

recode var1 (0=.) (999=.)           // Recode invalid values to missing
label define agegrp 1 "Child" 2 "Adult" 3 "Senior"
label values age_cat agegrp         // Assign value labels

// 7. Dealing with Outliers

gen z_score = (var1 - r(mean)) / r(sd)    // After summarize, detail
list var1 if abs(z_score) > 3             // List potential outliers

// Or use IQR method
sum var1, detail
gen iqr = r(p75) - r(p25)
gen lower = r(p25) - 1.5*iqr
gen upper = r(p75) + 1.5*iqr
gen outlier = var1 < lower | var1 > upper

// 8. Dealing with Duplicates

duplicates list id                      // Show duplicates based on ID
duplicates report id                    // Summary of duplicates
duplicates drop id, force               // Drop duplicate observations
duplicates examples                     //reports all duplicates if available
```

```
// 9. String Cleaning and Conversion
trim(name)                                  // Remove leading/trailing spaces
gen name_clean = lower(name)                // Convert to lowercase
replace name = subinstr(name, "  ", " ", .) // Replace double spaces
gen name_length = strlen(name)              // Length of string
destring var1, replace ignore(",")          // Convert string to numeric
tostring var2, replace format(%9.0g)        // Convert numeric to string

// 10. Checking Variable Types and Fixing
describe                                    // Shows type: string or numeric
encode str_var, gen(num_var)                // Convert string categories to numeric
destring str_var, replace                   // Convert string numbers to numeric

// 11. Sorting and Saving
sort id                                     // Sort by id
save cleaned_data, replace                  // Save cleaned data
************************************
```

In Stata, there are several ways to `detect outliers` depending on your approach (e.g., statistical thresholds, visualization, robust methods).

◇ 1. Using Standard Deviations from the Mean

```
summarize varname, detail
```

Then manually look at mean, sd, and calculate z-scores:

```
gen z_var = (varname - r(mean)) / r(sd)
list varname z_var if abs(z_var) > 3
```

This flags values more than 3 standard deviations from the mean.

◇ 2. Using the IQR Method

```
sum varname, detail
```

Check the 25th and 75th percentiles (p25, p75) and compute IQR:

```
gen iqr = r(p75) - r(p25)
gen lower = r(p25) - 1.5*iqr
gen upper = r(p75) + 1.5*iqr
gen outlier_iqr = (varname < lower | varname > upper)
list varname if outlier_iqr
```

◇ 3. Using the mvoutlier Package (for Multivariate Outliers)

Install it first:

```
ssc install mvoutlier
```

Then use:

```
mvoutlier varlist, method(classical)
```

This uses Mahalanobis distance for outlier detection.

◇ 4. Visual Techniques

- **Boxplot** (for univariate detection):

```
graph box varname
```

- **Scatter plot** (for bivariate):

```
scatter yvar xvar
```

- **Histogram**:

```
histogram var, normal
```

◇ 5. Detecting with extremes Command

Install first:

```
ssc install extremes
```

Then use:

```
extremes varname, n(10)
```

This shows the top and bottom 10 values of varname.

◇ 6. Ivr2plot – Leverage vs. residual squared plot(after regression)

```
Regress y x1 x2
Ivr2plot
```

```
***Merge- Simple example

webuse autosize, clear
list
webuse autoexpense
list
//Perform 1:1 match merge
webuse autosize, clear
merge 1:1 make using https://www.stata-press.com/data/r18/autoexpense
list
******************

// Perform 1:1 match merge, requiring there to be only matches
// (The merge command intentionally causes an error message.)
webuse autosize, clear
merge 1:1 make using            https://www.stata-press.com/data/r18/autoexpense,
assert(match)
tab _merge
list
******************
// Perform 1:1 match merge, keeping only matches and squelching the _merge
variable
webuse autosize, clear
merge 1:1 make using            https://www.stata-press.com/data/r18/autoexpense,
keep(match)            nogen
list
*****************************
 webuse dollars, clear
 list
 webuse sforce
 list

// Perform m:1 match merge with sforce in memory
merge m:1 region using https://www.stata-press.com/data/r18/dollars
list
*********************************

// Perform 1:m match merge, illustrating update replace option
webuse overlap2, clear
merge 1:m id using https://www.stata-press.com/data/r18/overlap1,update replace
list

***************
// Perform sequential merge
webuse sforce, clear
```

```
merge 1:1 _n using https://www.stata-press.com/data/r18/dollars
list
*******************

***Joinby
// joinby -- Form all pairwise combinations within groups
webuse child,clear
describe
list
webuse parent
describe
list, sep(0)

// Join information on parents from data in memory with information on children
from data at https://www.stata-press.com
joinby family_id using https://www.stata-press.com/data/r18/child

// Describe the resulting dataset
describe

// List the resulting data
list, sepby(family_id) abbrev(12)

 ***************************
***Append****
sysuse citytemp, clear
 keep if region == 4
 save west,replace
 sysuse citytemp
 keep if region == 3
 save south,replace
 sysuse citytemp
 keep if region == 1
//Append temperature data for the West region (region==4) and the South region
(region==3) to the end of the data for the New England region    (region==1), and
generate new variable filenum to indicate from which file each observation came.
Do not load value-label definitions from west.dta  or south.dta.
append using west south, generate(filenum) nolabel

// List the results
list

***********merge and joinby practice NLSS**************
clear
use poverty.dta, clear

use xh05_s05.dta, clear
merge m:m xhpsu using poverty.dta
tabstat v05_04 [aw= wt_hh],stat (sum) f(%15.0f)
clear
use poverty.dta, clear
joinby xhpsu using xh05_s05.dta
tabstat v05_04 [aw= wt_hh],stat (sum) f(%15.0f)
clear
use xh05_s05.dta, clear
joinby xhpsu using poverty.dta
tabstat v05_04 [aw= wt_hh],stat (sum) f(%15.0f)
clear

tabstat v05_04 [aw= wt_hh],stat (sum) f(%15.0f)
total v05_04
tabstat v05_04 [aw= wt_hh],stat (sum) f(%15.0f)
clear
```

Cross-Tabulations (In detail)

- `tabulate`: Two-way tables with optional statistics.
- `tabulate var1 var2, row col`
- `tabulate edutype ethnicity`
- `tabulate edutype ethnicity, row col`
- `tabstat`
- `table`

```
//using sysuse data: tabstat
sysuse auto, clear

// Show the mean (by default) of price, weight, mpg, and rep78
tabstat price weight mpg rep78

// Show the mean (by default) of price, weight, mpg, and rep78 by categories of
foreign
tabstat price weight mpg rep78, by(foreign)

// In addition to mean, show standard deviation, minimum, and maximum
tabstat price weight mpg rep78, by(foreign) stat(mean sd min max)

// Suppress overall statistics
tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal

// Include names of statistics in body of table
tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal long

// Format each variable's statistics using the variable's display format
tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal long
format

tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal long
f(%5.02f)

tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal long
f(%9.2fc)
// Show statistics horizontally and variables vertically
tabstat price weight mpg rep78, by(foreign) stat(mean sd min max) nototal long
col(stat)


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


****Using web data
webuse nhanes2, clear

*One-way table of frequencies
table agegrp

*One-way table of percentages
table agegrp, stat(percent)

// Three-way table of frequencies
table (agegrp) (sex race)
```

```
// Two-way tables for each level of sex
table (agegrp) (race) (sex)

// Table with percentage of observations in each category of sex and race,
for each category of region
table region, stat(fvpercent sex race) nformat(%5.2f)

// Table with pairwise correlations
table (rowname) (colname), command(r(C): pwcorr age bmi weight)

// Table of regression coefficients
table colname, command(regress bpsystol age bmi)

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

****Cross tab tabulation using census simulation data**

```
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
```

Sample selection using STATA

In Stata, you can draw different types of samples—simple random, stratified, cluster, or systematic—using various commands and approaches.

1. Simple Random Sampling

The `sample` command is used to draw a simple random sample.

Command:

```
sample n [, count]
sample 20, count
```

- **`n`**: Percentage of observations to sample (default) or the number of observations if `count` is specified.
- **Options**:
  - `count`: Indicates `n` is the number of observations instead of a percentage.

Example:

- Draw a 10% random sample:
- `sample 10`
- Draw 100 random observations:
- `sample 100, count`

---

2. Stratified Sampling

Stratified sampling requires dividing the dataset into strata (groups) and sampling within each stratum.

Steps:

1. Sort the data by the stratifying variable.
2. `sort stratavar`
3. Use the `runiform()` function to assign random values for sampling:
4. `generate random = runiform()`
5. Draw the sample within strata:
6. `by stratavar (random): gen sample_flag = _n <= floor(_N * 0.1)`
   - Here, `0.1` is the proportion to sample from each stratum.
7. Keep only the sampled observations:
8. `keep if sample_flag`

```
sample 20, by(ethnicity)
```

---

3. Cluster Sampling

Cluster sampling involves selecting groups (clusters) instead of individual observations.

Steps:

1. Identify or create the cluster variable.
2. Generate a random number for each cluster:
3. `generate random = runiform()`
4. `bysort clustervar (random): keep if _n == 1`
5. Select a fixed number or proportion of clusters:
6. `sample 50, count`

---

4. Systematic Sampling

Systematic sampling selects observations at regular intervals.

Steps:

1. Sort the dataset:
2. `sort varname`
3. Generate a sequence number:
4. `generate seq = _n`
5. Select every kth observation:
6. `keep if mod(seq, k) == 0`
    - Replace `k` with the sampling interval (e.g., `k = 5` selects every 5th observation).

Key Notes

- Ensure data integrity:
- `preserve`
    - Use this before sampling to save the original dataset.
    - Restore the dataset later:
    - `restore`
- For large datasets, stratified and cluster sampling may require additional memory.

To export result in excel:

collect export filename.xlsx, replace