Stata Cheat Sheet for Midterm

Large Scale Data Analysis - Part 2

1. Basic Setup & Data Import

Standard Setup

```
// Clear memory and set options
clear all
set more off

// Set working directory
cd "path/to/folder"

// Start logging
capture log close
log using "analysis.log", replace
```

Import Data

```
// Import CSV
import delimited "file.csv", clear

// Import Excel
import excel "file.xlsx", ///
    sheet("Sheet1") firstrow clear

// Import SAS XPT
import sasxport5 "file.xpt", clear

// Load Stata file
use "file.dta", clear

// Save data
save "filename.dta", replace
```

⚠ Warning

Always use `clear` or `, clear` option when loading new data to avoid ”no; data in memory would be lost” error.

2. Data Exploration

Basic Viewing

```
// View first 10 rows
list var1 var2 var3 in 1/10

// View specific observations
list if condition

// Browse data (opens viewer)
browse

// Describe variables
describe
describe var1 var2

// View variable type
codebook varname
```

Summary Statistics

```
// Basic summary
summ varname

// Detailed summary (with percentiles)
summ varname, d
// Shows: p1, p5, p10, p25, p50, p75,
//        p90, p95, p99

// Multiple variables
summ var1 var2 var3, d
```

Frequency Tables

```
// Frequency table (include missing)
tab varname, m

// Show numeric codes (not labels)
tab varname, nolabel

// Two-way table with row %
tab var1 var2, row m

// Two-way table with column %
tab var1 var2, col m

// Both row and column %
tab var1 var2, row col
```

Check Data Size

```
// Number of observations
display _N

// Number of variables
describe, short
```

3. Variable Creation

Generate New Variable

```
// Create empty variable
gen newvar = .

// Create with value
gen age_squared = age^2

// Create constant
gen constant = 1
```

Replace Values

```
// Replace all values
replace varname = new_value

// Conditional replace
replace var = value if condition
```

⚠ Warning

CRITICAL: Always protect missing values!
Use: `if var != .` in conditions
Missing (.) is treated as infinity in Stata.

Binary Variables (0/1 Flags)

```
// Method 1: Standard approach
gen flag = 1 if condition
replace flag = 0 if !condition

// Method 2: One-liner
gen flag = (condition) if var != .

// Example: Age >= 18
gen adult = 1 if age >= 18 & age != .
replace adult = 0 if age < 18

// Example: High cost (>$50,000)
gen expensive = 1 if cost > 50000 ///
    & cost != .
replace expensive = 0 if cost <= 50000

// VERIFY binary variable
summ flag
// mean should be 0-1, min=0, max=1
```

Categorical Variables

```
// Method 1: Manual creation
gen category = .
replace category = 1 if condition1
replace category = 2 if condition2
replace category = 3 if condition3

// Add value labels
label define cat_lbl 1 "Low" ///
    2 "Medium" 3 "High"
label values category cat_lbl

// Method 2: recode
recode age (0/29=1) (30/49=2) ///
    (50/max=3), gen(age_group)

// Method 3: encode string variable
encode string_var, gen(numeric_var)
```

Top-coding (Capping Outliers)

```
// Create clean version
gen cost_clean = cost

// Top-code at $1,000,000
replace cost_clean = 1000000 ///
    if cost > 1000000 & cost != .

// Verify
summ cost, d
summ cost_clean, d
// Check: max of clean version = cap
```

Logarithmic Transformation

```
// Handle right-skewed data
// Step 1: Add small value to avoid log(0)
gen cost_plus1 = cost + 1

// Step 2: Take log
gen log_cost = log(cost_plus1)

// Alternative: only for positive values
gen log_cost = log(cost) if cost > 0
```

EGEN - Extended Generation

```
// Row operations
egen total = rowtotal(var1 var2 var3)
egen mean = rowmean(var1 var2 var3)
egen max = rowmax(var1 var2 var3)
egen min = rowmin(var1 var2 var3)

// Grouped statistics
egen mean_by_group = mean(var), ///
    by(group_var)

// Example: County-level averages
egen avg_income_county = mean(income), ///
    by(county_name)

// Count non-missing
egen count_nonmiss = count(var), ///
    by(group)

// String concatenation
egen fullname = concat(first last), ///
    punct(" ")
```

4. Missing Values

Identify Missing Values

```
// Count missing
count if varname == .

// Summary shows N
summ varname
// Total N vs variable N

// Show missing in table
tab varname, m
```

Recode Missing Values

```
// Set specific values to missing
replace var = . if var == 99
replace var = . if var == 999
replace var = . if var < 0

// Example from BRFSS
replace height = . if height == 7777
replace height = . if height == 9999
```

⚠ Warning

Common Mistake:
`replace var = 100 if var > 100`
This will set missing to 100!
Correct:
`replace var = 100 if var > 100 & var != .`

Create Missing Indicator

```
// Flag for missing
gen miss_flag = (varname == .)

// Or explicitly
gen miss_flag = 0
replace miss_flag = 1 if varname == .
```

5. String Variables

String to Numeric

```
// Basic conversion (force ignores errors)
destring string_var, gen(num_var) force

// Check what couldn't be converted
tab string_var if num_var == .

// Example: Handle "120 +"
destring lengthofstay, gen(los_num) force
replace los_num = 120 ///
    if lengthofstay == "120 +"
```

Numeric to String

```
// Convert number to string
gen str_var = string(numeric_var)

// With formatting
```

```
gen str_var = string(num_var, "%9.2f")
```

String to Categorical (encode)

```
// Create numeric with labels
encode string_var, gen(categorical_var)

// Example: Admission type
encode typeofadmission, ///
    gen(admission_type_num)

// Verify
tab typeofadmission admission_type_num
```

String Manipulation

```
// Case conversion
gen upper = strupper(string_var)
gen lower = strlower(string_var)
gen proper = strproper(string_var)

// Extract substring (pos starts at 1!)
gen first5 = substr(string_var, 1, 5)
gen char2to4 = substr(string_var, 2, 3)

// Find substring position
gen pos = strpos(string_var, "keyword")
// Returns 0 if not found

// String length
gen length = strlen(string_var)

// Replace text
gen new = subinstr(string_var, ///
    "old", "new", .)
// Last argument: . = replace all

// Split string
split string_var, gen(part) parse("_")
// Creates: part1, part2, part3, ...
```

⚠ Warning

substr() position starts at 1!
`substr(str, 1, 2)` = first 2 chars
`substr(str, 2, 1)` = 2nd char only

6. Data Merging

Merge Types

```
// 1:1 - Both datasets: 1 row per ID
merge 1:1 id_var using "file.dta"

// 1:M - Master: 1 row/ID, Using: many rows/ID
merge 1:m id_var using "file.dta"

// M:1 - Master: many rows/ID, Using: 1 row/ID
merge m:1 id_var using "file.dta"

// M:M - Both: many rows/ID (rare, avoid)
// Use joinby instead if needed
```

M:1 Merge Example (Most Common)

```
// Merge county data to individual records
// Main: Individual hospitalizations
//       (many records per county)
// Using: County characteristics
//        (one record per county)

use "hospital_data.dta", clear

// Rename if needed to match
rename hospitalcounty County_Name

// Perform M:1 merge
merge m:1 County_Name ///
    using "county_data.dta"

// CHECK merge results
tab _merge
/*
_merge values:
  1 = master only (hospital records
      with no county match)
  2 = using only (counties with no
      hospital records)
  3 = matched successfully
*/

// Keep what you want
keep if _merge == 1 | _merge == 3
// Keeps all hospital records

// Clean up
drop _merge
```

Merge Workflow

```
// Step 1: Check merge variable exists
describe merge_var

// Step 2: Check if unique (if "1" side)
duplicates report merge_var
// Should show 0 duplicates

// Step 3: Ensure variable names match
// If not, rename in one dataset
rename old_name new_name

// Step 4: Check variable types match
describe merge_var
// Both should be numeric or string

// Step 5: Perform merge
merge type merge_var using "file.dta"

// Step 6: Always check _merge!
tab _merge

// Step 7: Keep desired records
keep if _merge == 1 | _merge == 3

// Step 8: Drop _merge
drop _merge
```

Tip

Quick merge type decision:
Ask: "How many rows per ID?"
Master(M) : Using(1) → M:1
Master(1) : Using(M) → 1:M
Master(1) : Using(1) → 1:1

Append (Stack Datasets)

```
// Combine datasets with same structure
use "data2020.dta", clear
append using "data2021.dta"
append using "data2022.dta"

// All rows are kept, stacked vertically
```

7. Data Reshaping

Wide to Long

```
// Wide format:
// id  bp1  bp2  bp3  hr1  hr2  hr3
// 1   120  118  115  72   70   68

reshape long bp hr, i(id) j(round)

// Long format:
// id  round  bp   hr
// 1   1      120  72
// 1   2      118  70
// 1   3      115  68
```

Long to Wide

```
// Long format:
// id  round  bp   hr
// 1   1      120  72
// 1   2      118  70

reshape wide bp hr, i(id) j(round)

// Wide format:
// id  bp1  bp2  hr1  hr2
// 1   120  118  72   70
```

⚠ Warning

reshape permanently changes data.
Always **save** before reshaping!

8. Regression Analysis

Linear Regression

```
// Simple regression
reg outcome predictor

// Multiple regression
reg y x1 x2 x3

// With robust standard errors
reg y x1 x2, robust
// or
reg y x1 x2, r
```

Categorical Variables in Regression

```
// Use i. prefix for categorical
reg outcome continuous_var i.category

// Example
reg totalcosts lengthofstay ///
    i.agegroup i.admission_type

// Stata automatically:
// - Creates dummy variables
// - Omits first category (reference)
// - Shows each category coefficient
```

Continuous Variables

```
// Default: continuous
reg y x1 x2

// Explicit: c. prefix (optional)
reg y c.x1 c.x2
```

Interaction Terms

```
// Categorical x Categorical
reg y i.var1##i.var2
// ## includes main effects + interaction

// Categorical x Continuous
// IMPORTANT: Use c. for continuous!
reg y i.category##c.continuous

// Example: Does income effect vary by sex?
reg health i.sex##c.income age

// Only interaction (no main effects)
reg y i.var1#i.var2
```

⚠ Warning

Interaction with continuous:
MUST use `c.` prefix!
Wrong: `i.sex##age`
Right: `i.sex##c.age`

Logistic Regression

```
// Binary outcome (0/1)
logistic binary_y x1 x2 i.category
// Reports Odds Ratios (OR)

// Alternative: logit (reports log-odds)
logit binary_y x1 x2 i.category

// Example
logistic expensive_stay ///
    County_Income lengthofstay ///
    i.agegroup i.ED_flag
```

Linear vs Logistic Interpretation

```
// LINEAR regression on binary outcome
reg expensive_stay County_Income, r
// Coefficient: Percentage point difference
// Example: coef = 0.04
// Interpretation: "County income increase
// of $1000 associated with 4 percentage
// point increase in probability of
// expensive stay (e.g., 10% to 14%)"

// LOGISTIC regression
logistic expensive_stay County_Income
// Coefficient: Odds Ratio
// Example: OR = 1.02
// Interpretation: "County income increase
// of $1000 associated with 2% increase
// in the odds of expensive stay"
```

Survey Weights

```
// Set survey design
svyset [pweight = weight_var]

// Weighted regression
```

```
svy: reg y x1 x2 i.category

// Weighted logistic
svy: logistic binary_y x1 x2

// Why use weights?
// - Make results representative
// - Account for survey design
// - Adjust for non-response
```

Post-Regression Commands

```
// Test joint significance
reg y x1 x2 x3
test x1 x2
// Tests: x1 = x2 = 0

// Predicted values
predict yhat

// Residuals
predict resid, residuals

// Margins (adjusted predictions)
reg y x1 i.group
margins group
// Shows predicted y for each group
```

Individual Fixed Effects

```
// For panel/longitudinal data
// Controls for all time-invariant
// individual characteristics

// Set panel structure
xtset person_id time_var

// Fixed effects regression
xtreg y x1 x2, fe

// Why use FE?
// - Within-person analysis
// - Control for unmeasured confounders
// - Stronger causal inference
```

9. Verification & Validation

Verify Binary Variables

```
// Create binary flag
gen flag = 1 if cost > 50000 & cost != .
replace flag = 0 if cost <= 50000

// CHECK 1: Summary statistics
summ flag
// mean: 0-1, min: 0, max: 1, N correct?

// CHECK 2: Cross-tabulation
tab flag, m
// Should show: 0, 1, and . only

// CHECK 3: Verify cutoff
summ cost if flag == 1
// min should be > 50000
summ cost if flag == 0
// max should be <= 50000
```

Verify Categorical Variables

```
// After encoding or recoding
tab old_var new_var
// Check mapping is correct

// With percentages
tab old_var new_var, row col
```

Verify Continuous Variables

```
// After transformation
summ original_var, d
summ clean_var, d
// Compare: mean, min, max, N

// Grouped summary
bysort group: summ var
// or
summ var if group == 1
summ var if group == 0
```

Check for Missing

```
// Count missing
count if var == .

// Identify observations with missing
list id var if var == .

// Missing by group
tab group, m
bysort group: count if var == .
```

Verify Merge Success

```
// After merge
tab _merge

// List unmatched from master
list id if _merge == 1

// List unmatched from using
list id if _merge == 2

// Check merged variable
summ merged_var if _merge == 3
// Should have valid values
```

10. Common Workflows

Clean Outcome Variable

```
// Step 1: Explore
codebook outcome
summ outcome, d
tab outcome, m

// Step 2: Identify issues
// - Missing values?
// - Outliers?
// - Correct range?

// Step 3: Create clean version
gen outcome_clean = outcome

// Step 4: Handle missing
replace outcome_clean = . if outcome == 99
replace outcome_clean = . if outcome < 0

// Step 5: Handle outliers (top-code)
replace outcome_clean = 1000000 ///
    if outcome > 1000000 & outcome != .

// Step 6: Verify
summ outcome_clean, d
tab outcome outcome_clean, m
```

Prepare Covariates

```
// Continuous variable
// - Check range
summ age, d
// - Handle missing
replace age = . if age == 99
// - Create squared term if needed
gen age_squared = age^2

// Categorical variable (numeric)
// - Check values
tab category, m
// - Create labeled version
label define cat_lbl 1 "A" 2 "B" 3 "C"
label values category cat_lbl

// Categorical variable (string)
// - Encode to numeric
encode string_var, gen(category_num)
// - Verify
tab string_var category_num

// Binary flag
// - Create 0/1
gen flag = 1 if condition & var != .
replace flag = 0 if !condition
// - Verify
summ flag
tab flag, m
```

Complete Analysis Example

```
// Research Q: County income effect on
// hospitalization costs?

// Step 1: Load and check data
use "hospital_data.dta", clear
describe
summ totalcosts, d

// Step 2: Clean outcome
gen cost_clean = totalcosts
replace cost_clean = 1000000 ///
    if totalcosts > 1000000 & totalcosts != .

// Step 3: Clean covariates
encode agegroup, gen(age_num)
gen ED_flag = (ed_indicator == "Y") ///
    if ed_indicator != ""

// Step 4: Merge county data
rename county County_Name
merge m:1 County_Name ///
    using "county_income.dta"
keep if _merge == 1 | _merge == 3
drop _merge

// Step 5: Check merged data
summ County_Income, d
// Report min and max

// Step 6: Run regression
reg cost_clean County_Income ///
    lengthofstay i.age_num i.ED_flag, r

// Step 7: Interpret
// "Controlling for length of stay, age,
// and ED status, each $1000 increase in
// county income is associated with
// $XX increase in hospital costs.
// This is statistically significant
// (p<0.05)."
```

11. Important Reminders

Critical Points

⚠ Warning

Top 5 Common Mistakes:

1. **Missing values:** Always use `& var != .`
2. **substr():** Position starts at 1, not 0
3. **Interaction:** Use `c.` for continuous
4. **_merge:** Always `tab _merge` after merge
5. **Binary range:** Check `summ` shows 0-1

Statistical Significance vs Practical

```
// Example: p=0.007, diff=3 minutes
// Statistical: YES (p<0.05)
// Practical: MAYBE
// - 3 min might be too small
// - But >10% relative difference
// - Large sample = "overpowered"
//   (can detect tiny differences)

// Always discuss BOTH in interpretation
```

Regression Interpretation

```
// Linear regression coefficient:
// "Each 1-unit increase in X is
// associated with beta-unit change in Y,
// controlling for other variables."

// Logistic regression OR:
// "Each 1-unit increase in X is
// associated with ORx change in the
// odds of Y, controlling for others."

// Binary outcome + linear regression:
// "Each 1-unit increase in X is
// associated with beta percentage point
// change in probability of Y."
```

When to Use What

Data Types:

- Cross-sectional survey: Prevalence, associations
- Longitudinal survey: Within-person changes, causality
- Claims data: Utilization, costs, readmission
- EHR data: Clinical details, single system

Merge Types:

- M:1: Individual records + area-level data
- 1:M: Visits + medications per visit
- 1:1: Same IDs in both datasets

Regression Types:

- Linear: Continuous outcome
- Linear: Binary outcome (percentage points)
- Logistic: Binary outcome (odds ratios)
- Fixed effects: Panel data, within-person

Cheat Sheet Usage Tips

Tip

During the exam:

1. Start with exploration (summ, tab, describe)
2. Create clean versions of variables
3. Verify each step before moving on
4. Check merge with `tab _merge`
5. Verify binary variables with `summ`
6. Write complete interpretations

Quick Reference

Task	Command
Load data	use ”file.dta”, clear
Import CSV	import delimited ”file.csv”
Summary stats	summ var, d
Frequency	tab var, m
Create var	gen newvar = expr
Binary flag	gen flag = (condition)
Top-code	replace var = cap if var ¿ cap & var != .
Encode string	encode str, gen(num)
String to num	destring str, gen(num) force
M:1 merge	merge m:1 id using ”file.dta”
Check merge	tab _merge
Linear reg	reg y x1 x2, r
With category	reg y x1 i.cat
Interaction	reg y i.cat1##c.cont
Logistic	logistic binary_y x1 x2
Verify binary	summ flag (should be 0-1)
Cross-check	tab oldvar newvar