TEMPLE UNIVERSITY

Computer Services

we bring IT.

S251
STATA on the LINUX Platform
(BATCH MODE)

TECH Center
Room 106, Tech Center
215-204-8000

Document Version 3

TABLE OF CONTENTS: Page #

I. OBJECTIVES

The purpose of this document is to provide the following objectives:

1. To introduce users to STATA software on the LINUX platform when it is run in the Batch mode and the Interactive mode.
2. To explain STATA commands for data definition, retrieval, and manipulation.
3. To introduce STATA procedures
4. To provide examples of STATA syntax files

The information in this document applies to STATA Version 13.

II. REFERENCES

General

The STATA manuals can be purchased from STATA Corp LP (call 1-800-STATAPC). The manuals cannot be purchased at the Student Book Store.

Online courses in STATA are available through the STATA website (www.stata.com).
Manuals/Computer Services Documentation

1. Getting Started with STATA for LINUX, Release 9, STATA Press, 2005
2. STATA User's Guide, Release 9, STATA Press, 2005
3. STATA Reference Manuals, Release 9, Vol. 1-3, STATA Press, 2005
4. LINUX Quick Reference Guide (Go to: cs.temple.edu/linux-quick-reference-guide)

III. PREREQUISITES

1. Familiarity with LINUX including knowledge of the Nano editor
2. Prior STATA knowledge or other statistical software knowledge is helpful.

IV. PROCESSING COMPONENTS / SYNTAX RULES

A. Introduction

STATA is a modern and general command-driven package for statistical analysis, data management, and graphics. The most current release, STATA 13, is available under LINUX on the Compute server. In this document there is a brief review of some of the key elements of STATA, a description of some of the important commands, and sample programs.

STATA can be used to interactively enter and edit data, and it can perform both simple and complex statistical analysis. Commands can be executed one at a time at the STATA prompt (Interactive STATA), or groups of commands can be entered into **do-files** which can be executed (STATA in Batch or Console mode). A later document will describe STATA under X-WINDOWS (also called STATA GUI) which provides a graphical interface.

The following sections describe the Batch and Interactive modes for running STATA.

B. Running STATA under LINUX – Batch Mode

1. File Types

Before using STATA, files which contain the software instructions (i.e., STATA syntax) and the data must be created and stored in the user's LINUX account. The user may store any of the following:

a. One file consisting of the data.
b. One file consisting of the program.
c. One file consisting of the program and data (for data files < 50 observations.)

2. File Creation (Using a LINUX editor or Word Processing)

The Nano editor can be used for the creation and storage of these files. The convention is to save syntax files (i.e., b or c above) with an extension of ".do" and ASCII or text data files (i.e., a above) with an extension of ".dat" or ".raw" when the files are stored under a LINUX system.

Any editor can be used to create do files. For example, to create a file in Nano, one simply opens the Nano editor by typing, at the LINUX prompt, "nano filename.ext", where "filename" is the name of the file and ".ext" is the file extension. The user can give the file any name consisting of letters and numbers, but not special characters such as *, &, and @. (See References)

For example, to create a STATA syntax file (i.e., command file or program file) named mysyntax, using the Nano editor, the user enters the following:

```
nano mysyntax.do
```

To save (i.e., write out) the file after entering STATA syntax, the user types the Ctrl and O keys simultaneously and hits the Enter key after being prompted for the filename. The file's name can be changed, if desired, prior to hitting the enter key.

Since Nano is difficult to use for creating (or editing) large program or data files, some users may want to create these files in word processing software and transfer the completed file to their LINUX account using SSH (Secure Shell) software (See https://computerservices.temple.edu/ssh-secure-shell-file-transfer-application for more information).

3. Executing STATA

To execute a STATA syntax file in Batch Mode under LINUX, any of the following commands can be entered:

```
stata < synname.do

        or

stata –b do synname

        or

stata < synname.do > synname.log
```

Where synname is the name of the STATA command or syntax file, do is the recommended file extension, which is understood if it is omitted, and synname.log is the output file. In the first case, the STATA commands in the file synname.do will be executed and the output will appear on the screen, but will not be saved. In the second case, the commands will also be executed, no output will appear, but the output will be stored in the file synname.log. In the last case, the commands will be executed; no output will appear but will be saved in the output file, which is arbitrarily named synname.log. The output file is further described under 6. below.

Note:

1. In order to manipulate files under LINUX when Batch mode is used, the researcher may need to initiate separate LINUX sessions. This is only recommended when the STATA job(s) do not require large resources from the system.

2. If the syntax used in the STATA program requires continuation lines, only the 2nd form of STATA execution shown above should be used (i.e., using the b parameter). In this case, within the do file (i.e., the file with do as the extension), the first line that requires continuation should end with /* and the next line should begin with */.

4. STATA syntax

a. STATA syntax typically has the structure below:

Note: All commands, subcommands, etc. are shown in upper case for emphasis, but should be typed in lower case when STATA is used under LINUX.

command varlist if exp in obs

Example: summarize price IF weight > 2000

where *command* = what it is the user needs to do (e.g., summarize, list, save, etc.)

varlist = the variables in the analysis (e.g., price)

if exp (if expression) = the observations to be used in the analysis (e.g., IF weight > 2000). When „IF" is not specified, the command is carried over all observations in a variable.

in range = performs the same task as „IF" but specifies the restriction in terms of observation numbers.

Example: list price in 2/4

This example will list the prices for observations 2, 3, and 4.

b. Subcommands consist of specific directions which modify the command statement. Subcommands cannot be typed first and are often separated from the command by a comma. They can continue for as many lines as necessary. In the LIST example above, „IN" is the subcommand that specifies the range; it cannot be used without the „LIST" command.

c. Variable Names in Stata should follow the below rules:

(1) Stata is case sensitive (i.e., variable Gender is different than gender). Therefore, it is recommended that names be typed in lower case.
(2) Maximum of 32 characters in length.
(3) Can use alphabetical characters, numbers, or underscore.
(4) Cannot include spaces or other characters (e.g., *).
(5) The initial character must be a letter or underscore. However, letters are recommended to avoid conflict with built-in variable names from Stata.

5. Data

Data are available in many forms and from a variety of sources. Temple provides assistance on how to acquire data (i.e., from large data bases) through the Social Science Data Library, located in Gladfelter Hall, 8th floor. (Call: 215-204-5001). The following are the most common types of data used with STATA.

a. ASCII data
This type of data (i.e., text data) is readable and does not contain any strange symbols. A number of organizations, including the Inter Consortium for Political and Social Research (ICPSR), the Bureau of the Census, the National Opinion Research Center, and the National Center for Educational Statistics provide data in ASCII form.

b. STATA datasets
STATA can be used to create special files known as STATA data files (also referred to as datasets) which are written by the package in binary form. A STATA data file consists of data plus associated information about the data.

Once the user stores a STATA dataset, all the data-definition information becomes accessible with the data when the data file is used in subsequent computer runs. Thus, the user does not need to redefine the data. The creation and retrieval of datasets are described later in this handout.

c. Other data files
Programs such as StatTransfer or DBMS/COPY can convert a number of common statistical software data files to STATA datasets. (Contact Computer Services at (215)204-8000 for assistance with either of these programs.

STATA datasets are usually smaller in size than SPSS data files when the same data is used.

6. Output

When a STATA '.do' file is run (or executed), the results can be viewed on the screen or saved in an output file as described earlier. If the output file is created with the .log extension, it will be in text or ASCII format. Log files keep a record of the commands that were issued and the results obtained during the STATA session. The file extension is usually .log but can be some other extension (e.g., out) during the STATA session.

Example: stata < regress.do > regress.out

Here the file 'regress.do' is executed. The output is directed to the file called '.regress.out'. The output will contain the regressions requested, and if errors are made, will tell the user the nature of those mistakes.

C. Running STATA under LINUX – Interactive Mode

A STATA program can be invoked interactively by typing ‘stata’ at the LINUX prompt. At the left lower corner, a ‘.’ will appear, which is the STATA interactive prompt. At this prompt, the user can perform data definition and retrieval (e.g., input, list, save, or retrieve the data) or perform data manipulation (e.g., generate, recode) or perform statistical analysis (e.g., tabulate, regress). The user can either view output on the screen or create a log file. The log file is created by the use of the LOG command as shown below:

Example: log using filename.log

To see the contents of this file, use the type command.

Example: type filename.log

All of the output plus the commands issued will be contained in the log file. The log file created under Interactive mode is n text format. If the log file were created as described above, the output can also be viewed with the NANO editor, but its format will not be as readable. At any point, if the user wants to terminate sending output to the log file, he/she should use LOG CLOSE. If the newly created file has the name of a previous log file, add; REPLACE to overwrite the previous file.

To obtain a copy of just the commands used in interactive mode, type:

```
cmdlog using filename.log or
cmdlog using filename
```

To see the contents of these files use the TYPE command:

```
type filename.log or
type filename.txt
```

In the latter case, .txt is the default extension when the user doesn’t provide an extension. Additional aspects of interactive use with STATA under LINUX are described next:

- Lengthy lines which need to be continued onto the next line are understood if the Enter key is used at the end of a line.

- The researcher can issue standard LINUX commands (e.g., ls -s, pwd, cd dirname, etc.).

- An existing do file is executed by typing at the prompt:

```
do filename.do or
do filename
```

where the .do extension is understood in the latter case.

- The use of the set command of the form:

  ```
  set linesize 72
  ```

  will eliminate wrap around in the output.

- The set command can also be used to control the amount of lines in the output, by typing:

  ```
  set pagesize 55
  ```

- When data is entered in interactive mode, use END as the last line of data in order to terminate the prompt for more data.

- If STATA says it cannot add more observations after the USE filename command, more memory is needed for the job.

- To increase memory for large jobs, type:

  ```
  set memory #m
  ```

  where # is replaced by the number of megabytes desired (e.g., 32)

- After the analysis is done, reset the memory to a smaller value (e.g., 2m). Alternatively, if the user logs off LINUX, the memory will revert to the default value.

- If the set memory command is not able to supply enough memory, precede its use with an additional set command as shown:

  ```
  set virtual on
  set memory #m
  ```

  After running the program, use:

  ```
  set virtual off
  ```

- The memory value can be checked by typing:

  ```
  memory
  ```

- The user can exit interactive STATA by typing ‘exit’.

If STATA refuses to close because it says ‘data in memory will be lost’ and you do not create a new dataset, you can exit by going to the File menu and selecting Exit. You will then be disconnected from Euler. If you want to create a new dataset, use the SAVE command (See V.A.5 below).

V. BASIC STATA COMMANDS

Below are some basic STATA commands along with their corresponding sub-commands. Those listed under Data Definition/Retrieval describe commands that define the data, save the data in various STATA formats, and/or access previously saved data.

Commands listed under Data Manipulation describe how to create or modify variables, select subsets of observations, and sort the data. STATA provides a large variety of statistical techniques and a few of the commands used are listed under Statistical Procedures.

NOTE: *For emphasis, all commands, subcommands, etc. are shown below in upper case. When STATA is used, type in lower case, since the software is case sensitive.*

A. Data Definition /Retrieval Commands.

1. INPUT

The INPUT command is used when data is typed from the keyboard and saved within a .do file. The syntax is:

```
Example:  input varlist
          data lines
          end
```

Observations can be entered line by line as shown below:

```
Example:  input price mpg weight
          15400 25 1930
          10200 20 1750
          18500 22 2200
          end
```

2. INFIX

The INFIX command can be used with raw data files which are in ASCII text and have a fixed format. STATA assumes that fields are space delimited.

```
Example:  infix var1 1-4 var2 6-7 var3 9-11 using myfile.txt
          where myfile.txt has data in a text format.
```

3. INSHEET

Files that are created by spreadsheet programs (e.g., Excel) as well as TAB or comma delimited files can be read with the INSHEET command as shown below:

Example: insheet var1 var2 var3, etc using filename.csv
where var1, var2, var3 etc. are the names of the variables on the first row of the Excel comma delimited file that was uploaded to the LINUX account using SSH Secure Shell.

4. INFILE

The INFILE command is used to read free-formatted data, where each element is separated by one or more spaces. A free-formatted data file may contain irregular spacing. Such files are given the extension .raw.

The easiest way to read free-formatted data files in STATA is with a data dictionary, which defines the name, location, and format of each of the variables. A data dictionary is saved in a file with the extension **.dct**. and is executed using STATA‟s INFILE command. The INFILE command is included in a .do file (created with any LINUX editor such as NANO) that contains the following syntax:

Example: infile using myfile.dct

Here, INFILE is the data definition command and USING is the subcommand that points to the data dictionary (e.g., myfile.dct)

The Data Dictionary specifies the number of columns for each variable, value labels and the data type for each variable (e.g., integer). An example of a Data Dictionary file is included in the Appendix. The data files and the dictionary must be stored in the LINUX account.

5. SAVE

Once the data has either been entered using the INPUT command or infix, infile, or insheet are used to read the data, the data is loaded into memory. SAVE produces a STATA dataset. This file is specially formatted for use by STATA. It contains data, with labels (if specified) and missing-value codes (if missing data specifications are provided) for each variable. The standard extension for a STATA dataset is .dta.

A STATA dataset is created by executing a .do file that includes the following syntax:

Example: save filename.dta

6. USE

USE accesses a STATA dataset that was created by the SAVE command. USE copies all variables from the dataset into the active file. Variables in the active file are in the same order and have the same names as those in the dataset.

Example: use filename.dta

7. **LABEL VARIABLE (optional)**

The LABEL VARIABLE command provides a description of the variable. It may contain a maximum of 80 characters.

Example: label variable cars “cars in 2001”

8. **LABEL DEFINE / LABEL VALUES (optional)**

LABEL DEFINE provides the meaning of the values given to a variable. The value label can be 32000 characters in length.

If the values for the variable called „cars‟ are 1 and 2, then one could create a lblname called origin and assign the label „Foreign‟ to a value of 1 and „Domestic‟ to a value of 2. The lblname can be at most 32 characters. The LABEL VALUES command associates the lblname (e.g., origin) with the variable whose values it defines (e.g., cars). These labels will be displayed when the data is shown in the output (e.g., after the execution of the DESCRIBE command).

```
Example:   label define origin 1 “Foreign” 2 “Domestic”
           Label values cars origin
           describe
```

9. **DESCRIBE**

The DESCRIBE command allows one to list the names and formats of the variables in the active file as they appear in sequence in that file.

```
Example:   use cars.dta
           describe
```

DESCRIBE will provide details on the size of the file, the number of variables, the number of observations, labels, etc.

10. **LIST**

The LIST command, when used with a varlist, shows the values of the requested variable(s) in each observation. If no varlist is given, it provides the values of all the variables in each observation.

```
Example:   user cars.dta
           list cars
```

B. Data Manipulation Commands

1. GENERATE

The GENERATE command (which may be abbreviated g, ge, or gen) allows the user to create new variables. In the example below, variables a, b, and c are read. Then the GENERATE command is used to create variable d. Notice that the variables named on the INPUT statement must have spaces between their names in order for STATA to recognize them as separate variables. The variable d is equal to the difference between variables c and b.

```
Example:  input a b c
          1 5 10
          0 8.
          1 4 6
          end
          generate d=c-b
          describe, list, save myfile, replaced
```

If the file named by SAVE (e.g., myfile) already exists, the REPLACE command will overwrite the old file with the new file. When the REPLACE command is not used and the file already exists, the output will show a message that the file exists and will not make any changes to the file that the program would have caused. In the latter case, to save a new file, a different dataset name would have to be specified.

The user would have to designate an output file name (See IV B.1) at the time of program execution in order to store the entire output (e.g., the output from DESCRIBE and the output from LIST).

Notice that there is a period indicating a missing value on the second observation or case for the variable c. STATA will indicate that it understands the data is missing there. However, if no period is present and only a blank space is visible, for a numeric variable STATA would delete the entire case. (See Example 2 in VI for use of REPLACE with missing value codes). Spaces are allowable to designate missing data for the string or character variable.

Be cautious when using GENERATE commands with missing data. STATA will provide a missing value if the computed variable is undefined due to missing data, or invalid computation (e.g., square root of a negative number). If two variables have been used to generate a new variable, the latter will contain missing values when there are missing observations that appear in either of the two variables. Also, be careful to create unique variable names with the GENERATE command so as not to overwrite existing variables.

2. RECODE

The RECODE command allows the user to collapse the values in a variable into a discrete number of categories.

The RECODE command below assigns a value of '1' to all values of price between 0 and 4000 and a value of 2 to all values between 4001 and 8000. The data file called cars.dta is accessed in the example below:

Example:
```
user cars.dta
recode price 0/4000=1 4001/8000=2
list price
```

If a user wants to preserve the original values of the variable, but also have a variable that has categories, the user may create a new variable with the GENERATE command.

Example:
```
use cars.dta
generate price1 = price
recode price1 0/4000=1 4001/8000=2
list price price1
```

3. IF

The IF subcommand allows the user to specify „conditional‟ data transformations for selected subsets of the data.

One may want to inflate price data such that the price for the foreign cars increases by 5% and those of domestic car increase by 10%. See the following example:

Example:
```
USE cars.dta
GENERATE prep = 1.05 *price IF car = = 0
REPLACE prep = 1.1 * price IF car = = 1
LIST weight price prep car
```

NOTE: Tests of equality are specified with double equal signs. Single equal signs, are used for assignment as in the previous example using RECODE.

Here the values of price are inflated at different rates depending on whether the cars are domestic or foreign. A new variable called prep is created by using the GENERATE command. REPLACE is used to revise the value of the created variable prep. In this case the use of GENERATE a second time would not have worked.

The IF subcommand can also be used with the LIST command to obtain values for each variable in the data file according to the IF condition specified.

Example: `list if price >5000`

4. **IN**

STATA can also list or select cases which fall within a specific observation range of the dataset by using the subcommand IN.

```
Example:  use cars.dta
          list price IN 4/6
```

Only the prices for observations 4 through 6 will be displayed in the above example.

5. **KEEP**

Use the KEEP command to permanently select a subset of cases for analysis. The cases that are selected depend upon logical conditions specified in the KEEP statement.

```
Example:  use cars.dta
          keep in 3/5
```

In the above example, observations 3 through 5 are used.

```
Example:  use cars.dta
          keep if mpg <= 21
```

In this example, only the observations where mpg is at most 21 will be used.

6. **DROP**

DROP can be used the same way as KEEP, but it works in the opposite way (i.e., it deletes cases). DROP_ALL drops the dataset from memory.

```
Example:  use cars.dta
          drop if weight > 2000
```

7. **CLEAR**

Clear deletes the dataset from memory. All the variables, observations, value label definitions, constraints, equations, programs, etc. are removed.

8. **BY and SORT**

To process data by groups, the data has to be sorted first. The example below requests separate processing for each level of year and sorts the data first by year:

```
Example:  use cars.dta
          by year, sort: command
where command could be TAB price
```

C. Statistical Procedures

1. TABULATE (Univariate tabulation)

The Tabulate procedure is used to create a frequency distribution table which provides a listing of all observed values for a given variable and the number of observations that fall under each of these values.

To create a frequency distribution table in STATA, use the command TABULATE or TAB.

Example: To create a frequency distribution table for the categorical variable price, type: tab price

In addition to frequencies, the TABULATE command determines univariate statistics (e.g., means, standard deviations). Repeat the TAB command for each additional variable.

Example:
```
use cars.dta
tab price
tab weight
```

2. TABULATE (Bivariate or Crosstabulation)

TABULATE also allows users to cross-classify two or more variables.

Example:
```
use cars.dta
tabulate price weight, chi2 column taub
```

This will generate a crosstabulation of price by weight and give the column percentages, chi square value and kendall's tau-b. Notice that a comma is needed to separate the command (with its variable list) from the options (e.g., chi2), but no commas are placed between options.

If the user wishes to generate more than one crosstabulation and all possible statistics then follow the next example:

Example:
```
tab2 price weight cars, all
```

This command will generate crosstabulations of price by weight, price by cars, and weight by cars and provide various statistics. When the options ROW, COLUMN, and CELL are added, row, column, and total percentages for each cell will also be computed.

3. CODEBOOK

CODEBOOK provides the variable names, labels, and the data to produce a codebook describing the dataset. The CODEBOOK procedure provides univariate statistics (e.g., means, standard deviations, highest value, lowest value, etc.) for interval-level variables. It produces tabulations for variables that are categorical (e.g., variables that have <= 9 unique values)

```
Example:  use cars.dta
          codebook price weight
```

Here the codebook is used to describe only two variables, price and weight. When no variables are named, the codebook is done for all the variables in the dataset.

4. REGRESS

The REGRESS command allows one to run linear regression analysis.

```
Example: regress price weight
```

Here price (the dependent variable) is predicted by weight (the independent variable). REGRESS provides many summary statistics and includes the regression coefficients. Also available are prediction values, diagnostics, residual plots, etc.

Multivariate regression is set up similarly:

```
Example:  use cars.dta
          regress price weight cars, level (99)
```

In this case, price is the dependent variable and weight and cars are the predictors. LEVEL (99) indicates that a 99% confidence level will be given for each regression coefficient.

VI. EXAMPLES OF STATA SYNTAX FILES

A. Example 1: Creating a STATA Data File

```
input id age weight gender score1 score2
01 25 132 2 87 62
02 23 244 1 75 95
03 19 127 1 84 88
end
summarize
save newdata.dta
```

NOTE:

1. INPUT provides the variable names
2. END indicates that all of the data is listed.
3. SUMMARIZE gives several descriptive statistics of all variables (e.g., mean, std-dev, etc).
4. The SAVE command creates the STATA dataset.
5. The new dataset is arbitrarily named newdata.dta, where .dta is the extension used for all STATA datasets.

B. Example 2: Accessing a STATA Data File

```
use newdata.dta
replace score1=. if score1==99
tabulate score1 score2, row column cell chi2
```

NOTE:

1. The data file being accessed is named newdata.dta.
2. The REPLACE command substitutes a missing value (.) for a variable (e.g. score1) with a missing value of 99.
3. The TABULATE procedure provides 1-way to n-way tabulations.
4. Optional keywords such as ROW COLUMN CELL and CHI2 provide row, column, and total percentages plus Chi Square.

C. Example 3: Updating a STATA Data File

```
use newdata.dta
generate newscore = score1/score2
label define gender 1 “male” 2 “female”
label values sex gender
list
save newdata, replace
```

NOTE:

1. The GENERATE command is used to calculate the new variable newscore.
2. The updated STATA dataset includes all of the old variables plus the new variable newscore.
3. The updated STATA dataset (e.g., newdata.dta) has the same name as the earlier file.
4. If a user wishes to create a new file with a new filename then change the name in the SAVE command line and remove the REPLACE command. (e.g., SAVE newfile).

D. Example 4. Accessing a Free-Formatted Text File

Since several steps are required, this example was placed in the Appendix. The example reads an external data file, creates a data dictionary file, and then creates a STATA dataset.

VII. EXERCISES

1. Do Examples 1, 2, and 3 using Batch Mode
2. Do Example 1, 2, and 3 using Interactive Mode

VIII. SEQUENCE OF COMMANDS

A. Initial Steps

Step 1: Start the SSH Secure Shell program. Click "Quick Connect". Once the dialog box appears, type „compute.temple.edu" where it says "Host Name:" Type the ID where it says "User Name", then click "Connect". After that click "OK" to the next message. Then you will be prompted for your password. Type your password and click "OK".

Step 2: At the Compute prompt, decide whether to use STATA in Interactive mode or Batch mode. If Interactive mode is desired, type stata at the Compute prompt and look at Section C below. If Batch mode is desired, follow Steps 1-7 in Section B below.

B. Batch Mode

Step 1: Initiate the Nano Editor, by typing the

following: nano filename.do

The above command creates a file called "filename" with an extension of "do." Filenames are arbitrary. Extensions should relate to the type of file involved. Files that have the .do or **.**STATA extension should contain STATA commands.

Step 2: Entering Commands

STATA program commands can be typed at the cursor position after Nano has been initiated. Use the Enter key to advance to the next line. Specific Nano commands are shown at the bottom of the screen. For example, to move forward, type the 'control' and 'v' keys together; to search for text within the document, use the 'control' and 'w' keys, and then enter the text for which you are searching. Make any changes to the program as needed.

Step 3: Save changes

To save changes, type the Ctrl and O keys together. Next, at the "File Name to write:" prompt, provide a new filename or accept the current file name and hit the Enter key. The changes to the file will now be stored.

Step 4: Print the STATA program file

Once the file has been saved by Nano, the user may want to have a hardcopy of the program. Use the mouse to select the program lines. Then select Print from the File menu to have the Stata program file printed.

Step 5: Exit Nano

Type the Ctrl and X keys together to exit from Nano

Step 6: Execute the STATA program

At the LINUX prompt type:

```
stata < filename.do > filename.log
```

This executes the STATA program (i.e., syntax) file named filename.do that was created in Nano and puts the output in a file called "filename.log". It is recommended that the syntax file and the log file have the same filename (e.g., myfile.do and myfile.log).

NOTE:

1. A file containing STATA statements should have either .do or .STATA as the extension.
2. A data file could have .raw, .dat, or .data as its extension.
3. Both .log and .lst are extensions which can conflict with SAS output files that are similarly named.
4. If the researcher keeps the files associated with STATA in a separate directory, there will be no conflict with any of the file extensions used.

Step 7: Viewing the Output

Use the MORE command to view the STATA output.

Example: more filename.log | more
where | more allows each screen to pause

Step 8: Printing the Output

The typical way to print the STATA output is described below:

Download the file to be printed using SSH Secure Shell software. Then access the file with a word processing package (e.g., Word) from which the file can be printed.

C. Interactive Mode

After stata is typed at the Compute (or Storm) prompt, the software's initial screen will appear. At the left lower corner a '.' will be visible as the interactive prompt. All STATA commands are entered at that prompt. All output will appear on the screen. See Section IV for a description of many of STATA's features and their use in Interactive Mode. In particular Section IV.C describes creating an output file when the user is in STATA interactive mode. To stop an executing program, type quit or q.

To exit from STATA, type exit.

IX. APPENDIX: Using STATA with Free-formatted Text Files

Before writing a STATA program to read a raw data file that is free-formatted, one needs to know the following about the data.

A. Raw data file definitions

1. The **Variable names**: Select the variables needed and create names (e.g., sex, college, status, etc.)

2. The variable names can be at most 32 characters and can contain combinations of letters, numbers and/or the underscore character (dob, _var1, yr8st2a, DOB.) No spaces or other characters are allowed. STATA is also case sensitive (i.e., Level and level are different variable names).

3. A variable name must not be one of STATA's *reserved* names. See the STATA *User's Guide* for a list.

4. The **start position** of each variable is the first column of the variable's position in the record.

5. The **variable's width** equals the number of columns the variable occupies. If a variable is in column 5-8, its width is 4

6. Each **variable's type**. The variable types are named:

    a) byte - integers whose interval storage is between -127 and 126
    b) int - integers whose interval storage is between -32,768 and 32766
    c) long - integers whose interval storage is between -2,147,483,648 and 2,147,483,646
    d) float - real numbers with 8.5 digits of precision
    e) double - real numbers with 16.5 digits of precision
    f) str[n] - alphanumeric variable with length n. n can be from 1 to 80

7. The **number of decimal places** required, if any, for numeric variables.
8. The **input file name** and **location** of the raw data file.

B. Raw data file

Consider the small dataset below, which is contained in the file **d1999.raw.**

```
1999 DATA SAMPLE #1
61000691 BALBOA     5765 2 no
61000691 20216 96
60213072 CORNER     2080 4 no
60213072   399100
60198303 WARNER     3427 8 no
60198303 10389 80
60213494 SILVER     381010 no
60213494   422 96
60225295 VALENTINE  4010 6 no
60225295 11114100
60180636 MARQUEZ    3459 5yes
60180636 10424 94
60212997 MONTEMALAGA242017yes
60212997 19232 81
60224958 CARSVER    4090 8yes
60224958 12413 97
60189649 ROSCOMIR   2575 4 no
60189649  9128 79
602001010WONDER     325110 no
60200101 17407 85
602183611GOLDEN     4861 1 no
60218361111323 88
602032512PENNEKAMP  325012yes
602032512 8    96
602135613SOLEADO    296021 no
602135613  407 95
602255214FRANKLIN   5236 5 no
60225521419343 95
601463315LA CANADA  5350 4 no
60146331515451 97
602040816ROBINSON   2270 8 no
60204081610450100
602035817GRAND VIEW 354011 no
602035817      76
602137218VISTA      377025yes
602137218  445 96
601465819PALM       5050 5 no
601465819 3247100
```

This data file begins with a comment line at the top line. Each observation requires two lines, and some of the values are placed together with no spaces between them. Further, the string variables are not enclosed in quotes.

C. Codebook

To read the data, one needs to create a Codebook (See Table 1). The user manually creates the Codebook in his/her notes. It is not stored as a file.

Table 1: Example of Codebook

lin	variabl	label	col	form
1	schcod	school code 1	1	7.0
1	rec1	record 1	8	2.0
1	school	school name	10	11s
1	numbe	number tested	21	3.0
1	pctmis	percent missing	24	1.0
1	pctel	percent eligible	25	2.0
1	yrrnd	year round	27	3s
2	schcod	school code 2	1	7.0
2	rec2	record 2	8	2.0
2	mobili	mobility rate	10	2.0
2	avged	avg parent ed	12	3.2
2	pcteme	percent emer	15	3.0

NOTE:

If a variable does not occupy the full field, STATA will recognize and read the data correctly as long as a space separates that variable from the next variable.

D. The Data Dictionary

The method that is used in STATA to read a new data file is with a data dictionary. The data dictionary defines the name, location, and format of each of the variables. A data dictionary for the above data is saved in a file named **d1999.dct** and is shown in Table 2 below. The dictionary command, which is shown on the first line, accesses the data file (e.g., **d1999.raw)**.

Table 2: Example of Data Dictionary

```
dictionary using d1999.raw {
_first(2)
_lines(2)
  line(1)
long   schcod1   %7.0f     "school code"
long   rec1       %2.0f     "record1"
str11  school     %11s      "school name"
int     number    %3.0f     "number tested"
byte   pctmiss    %1.0f     "percent missing"
byte   pctel       %2.0f     "percent eligible"
str3    yrrnd       %3s      "year round
school"
_line(2)
long  schcod2    %7.0f  "school code 2"
long  rec2        %2.0f  "record 2"
byte  mobility   %2.0f  "mobility rate"
float avged       %3.2f  "avg parent ed"
byte  pctemer    %3.0f  "percent emer cred"
}
```

A description of the terms used above is given next:

input position controls
_first(2) -- begin reading the data with the second line
_lines(2) -- there are two lines of data for each observation
_line(1) -- read data from the first line of each observation
_line(2) -- read data from the second line of each observation
_skip(8) -- skip 8 columns or
_column (9) -- read data in the ninth column

input formats
%7.0f -- fixed numeric format 7 columns wide with no decimal values
%3.2f -- fixed numeric format 3 columns wide with 2 decimal values
%11s -- string format 11 columns wide

variable labels -- enclosed in double quotes (e.g. "school name")

variable type – see Section A above for definitions of the terms: byte, int, long, float, double, and str.

E. Using the Data Dictionary

In order to use the dictionary, the two file, **d1999.raw** and **d1999.dct** must be in the same directory as the directory to be used for the STATA data files. The data is then read with the INFILE command which is one of the commands within the .do file:

```
Example: infile using d1999.dct
         describe
         list
         save d1999.dta, replace
```

The dataset d1999.dta is created by the SAVE command. After execution of the above .do file, the output will be seen in the .log file and this is described next.

Example: Output from accessing a Data Dictionary program

The first part of the output will show the data dictionary as seen in Table 2. Next the output for DESCRIBE, LIST, and SAVE is provided.

.describe

```
Contains    data
obs:         20
vars:        12
size:        880 (99.8% of memory free)
-------------------------------------------------------------------------------
                          storage   display    value
 variable name            type       format     label     variable label
-------------------------------------------------------------------------------
 schcod1                  long      %12.0g                school code
 rec1                     long      %12.0g                 record1
 school                   str11     %11s                  school name
 number                   int       %8.0g                 number tested
 pctmiss                  byte      %8.0g                 percent missing
 pctel                    byte      %8.0g                 percent eligible
 yrrnd                    str3      %9s                   year round school
 schcod2                  long      %12.0g                school code 2
 rec2                     long      %12.0g                record 2 mobility
                          byte      %8.0g                 mobility rate
 avged                    float     %9.0g                 avg parent ed
 pctemer                  byte      %8.0g                 percent emer cred
-------------------------------------------------------------------------------
Sorted by:
    NOTE:  dataset has changed since last saved
```

. list

```
Observation 1

schcod1   6100069    rec1      1         school    BALBOA
number    576        pctmiss   5         pctel     2
yrrnd     no         schcod2   6100069   rec2      1
mobility  20         avged     2.16      pctemer   96

Observation 2

schcod1   6021307    rec1      2         school    CORNER
number    208        pctmiss   0         pctel     4
yrrnd     no         schcod2   6021307   rec2      2
mobility  .          avged     3.99      pctemer   100

……………………………………………………………………………..
……………………………………………………………………………..
……………………………………………………………………………..
……………………………………………………………………………..

Observation 19

   schcod1   6014658    rec1      19        school    PALM
   number    505        pctmiss   0         pctel     5
   yrrnd     no         schcod2   6014658   rec2      19
   mobility  3          avged     2.47      pctemer   100

   . save d1999.dta, replace
   file d1999.dta saved
```

NOTE: In case the output generates an additional observation, which has no values, as the last line in the .dta file (as shown in the .log output file), use the KEEP command in the .do file to eliminate it and resave the file.

```
Example:  use d1999.dta
          keep in 1/19
          save d1999.dta, replace
```

This example will retain the first 19 observations.

STATA on LINUX Platform