Creating Self-Validating Datasets

Bill Rising

StataCorp

2007 West Coast Stata Users Group meeting
26 October 2007
Outline I

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes
Outline II

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Validation Should Be in Dataset

- Currently, validation is contained in
  - Outside documentation
  - Outside programs (do/ado files)
- Can be separated from data too easily
  - Not shared well, either
Validation Should Be Persistant

- Validation must follow variables through manipulation.
  - Merges
  - Subsetting variables
  - Subsetting observations
  - Appending
- Validation rules must be attached to variables themselves.
Validation Should Be Easy

- Can attach most validation knowing no Stata
- Can attach most of what is left knowing minimal Stata
- Do not need to know a lot of programming tricks
- Not Easy == Not Used
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Making Friendly, Part 1

- Use simple syntax for simple checks.
  - When possible use syntax(es) familiar to both experienced and new Stata users.
- Most checks use ranges or lists, so these are of top priority.
- Try to avoid using any kind of Stata programming.
- Make this somewhat odd method invisible to the casual user and clear to the aficiando.
Making Friendly, Part 2

- Use a simple interface for simple needs.
  - Be sure that users cannot get lost.
  - Protect against inadvertent undesirable changes.
- Try to use a simple interface for complex needs.
- Perhaps a dialog box as the main interface?
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Tools

- A dialog box, `ckvaredit`, which takes care of attaching the characteristics,
- A command, `ckvar`, which runs through the variables and does the validation,
- A helper command, `ckvardo`, which turns the characteristics into a do-file which could be used with other, similar datasets.
The Dialog Box

Here it is:
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Valid Validation Rules

- Simple rules—no Stata knowledge needed
  - Bounds
  - Ranges
  - Sets

- Complicated rules—for complicated validation
  - Full-fledged do-files or complicated commands
Validation Using Bounds

- For one-sided bounds on the values of a variable
- Syntax: `{>= | > | == | < | <=} #
- Examples:
  - >=0
  - <5
Validation Using `in` and Sets

- For more complicated sets, such as ranges or individual values
- Syntax `in set [& | | | ! set ...]`
- Sets can be specified in a number of ways.
- Logic works, using Stata’s operators
  - Parentheses do **not** work, unfortunately
Specifying Sets

- For discrete sets of numbers or strings:
  - Set notation works.
  - Stata’s `numlists` work for numbers.

- For continuous ranges of numbers:
  - Set notation works: round brackets: ( and ) **do not** include endpoints, square brackets: [ ] **do include** endpoints
  - Use . to denote infinity, and –. to denote minus infinity

---

Goals
Implementation
Demo of Package
Finishing Up
Post-Conclusion

The Interface
Validation Rules
Other Tools

Bill Rising
Self-Validating Datasets
in Examples

- in \{1,2,3,4,5\}
- in 1/5 is the same as above
- in [0,5] is any number between 0 and 5, inclusive
- in [0,1) is any number from 0 to under 1
- in [0,.) is the same as \geq 0
How to Enter Validation Rules (Complex)

- These are simply Stata commands with some slight twists which keep everything functioning.
  - Use `self` to refer to the variable being checked
  - Use `valid` for valid values, and `error` for invalid values
- These are entered using the do-file editor, as we’ll see.
How to Avoid Reentering Rules

- Can use `like varname` to check just like another variable.
- One big reason for using `self`!
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Keeping Track of Dependencies

- Using `like` or programs makes new dependencies among variables.
- Should not be able to drop or rename needed variables.
- Be sure to put the variables in the **Other Variables Needed** ... box.
- Use `ckdrop`, `ckkeep`, and `ckrename`. 

Self-Validating Datasets
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
The `example.dta` Dataset

- `use example` brings in an example dataset.
- `describe` is enough to set up the validation rules!
  - Ha! How often does that happen?
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Entering the Rules

- Type `ckvaredit` at the Stata prompt, and start
  - `id` already has a rule, so we’ll skip and come back later.
- It would be nice to have a way to step through all the variables.
  - Surprise! The `stepthru` option will go from one variable to the next.
- When finished, the dataset has been marked as dirty, so that it is harder to throw away the validation work.
- Save this—"save example cked"
Document the Validation Rules

- Try ckcodebook
- Shows all the error checks
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Simple Use of \texttt{ckvar}

- Try \texttt{ckvar}
- Done!
ckvar can be used to be sure that identifiers are distinct.

Drop the all the error variables

  drop error*

ckvar, key(id)

  Aha! There are duplicates

Drop the error* variables, again

ckvar, key(id) markdups(duplicate)
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Make a Do-file for Future Datasets

- **Try** `ckvar do using example.do, replace`
- **To see the do-file:** `doedit using example.do`
  - Notice the backslashes in front of the open-quotes!
  - Can see how it works: Characteristics
- **To see it in action:**
  - `ckvarclear` to clear out all the characteristics.
  - `do example`
  - `drop error*`
  - `ckvar`
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
We need some protection to keep variables needed for validation from disappearing or being renamed.

- `ckkeep`, `ckdrop`, and `ckrename` try to take care of this.

**Examples**

- `ckdrop rating1` *does nothing*, because `rating1` is needed for checking the other `rating` variables, as well as `best`.
- `ckrename rating3` *fails because `best` needs `rating3` for validation*.
- `ckkeep id best` *keeps some extra variables*.
Clean Slate?

- The `ckvarclear` command will clean out all the characteristics.
- Should really be used only for debugging!
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
This package can also be used for scoring instruments.

Instead of generating error markers, will generate a score for each variable and each observation, as well as a maximum possible score.

The distinction between scoring and validating is small: two values (for validation) or many values (for scoring).
Automation Through Templates

- Can use `ckvardo` to generate do-files.
- Can make dataset templates for standard datasets, instead.
  - Make an empty dataset with the proper variable names, and then add the rules.
  - Use a dictionary (or some other mechanism) to make sure data have proper variable names.
  - Append data set to template to implement validation or scoring.
Can keep working through the face of problems by using the `keepgoing` option.

- Good for big datasets

For debugging, the `loud` option is good for echoing lots of esoteric output.
Could attach keys directly via a `char _dta[key]`, perhaps.

- There is some danger, because this should not survive a merge.

- Need ways to run corruption checks more easily.
**Commands Which Need Modification**

- `reshape` could be OK in many cases.
  - Going wide to long: keep the rules for the first variable
  - Going long to wide: put the rule in the first variable, make the rest use `like`

- Need checks when appending, so that conflicting rules do not overwrite each other.
Possible Trickiness

- Need way to keep validation or scoring if there are many types of rules attached to each variable, all with their own dependencies.
- It would be nice to automatically detect other variables needed, instead of relying on the user to notify the dataset.
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Questions?

Ask away!
Outline

1. Goals
   - Goals for Validation
   - Implementation Goals

2. Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3. Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4. Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5. Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
Characteristics Store the Rules

- Characteristics . . .
  - . . . allow attaching large amounts of text to any variable or the dataset.
  - . . . follow variables through data manipulations.
  - . . . can have arbitrary names.

- These are one of the most undervalued pieces of Stata.
The characteristics’ names have 2 pieces.

- A **prefix** which often conveys its grand purpose, but which can be overridden.
- A **suffix** which conveys its specific purpose within the grand purpose.
- The two are separated by an underscore (\_).

**Examples:**

- `valid_rule` holds a validation rule.
- `score_required` says there should be no missing values when scoring.
Prefixes

- **Defaults**
  - `valid` is the default for validation.
  - `score` is the default for scoring.

- Any prefix can be used in place of these.
  - Useful for scoring, if there are multiple scores which can result.
  - Otherwise should be avoided, because others will expect the defaults.
rule contains the rule to be evaluated.

required for whether valid values are required. If not specified, this is treated as “no”.

other_vars_needed lists those variables needed for the validation or scoring.

missval contains the value used to tag missing values when missing values are invalid.

These may not be changed.
Outline

1 Goals
   - Goals for Validation
   - Implementation Goals

2 Implementation
   - The Interface
   - Validation Rules
   - Other Tools

3 Demo of Package
   - The Data
   - Adding Rules
   - Checking the Data
   - Reusing Your Work
   - Other Notes

4 Finishing Up
   - Extensions
   - Unfinished Business
   - Questions?

5 Post-Conclusion
   - What Holds the Rules?
   - How Are the Rules Used?
ckvar calls a program called dochar, which is a utility for running code stored in characteristics. This could be used by others for their own embedded code.

dochar looks for in or like, and failing that, saves the commands to a temporary do-file, which is then executed.

dochar has arguments for passing through temporary macro names.

This is the one place where things get a bit sticky.

More detail available through help dochar.