The Stata Journal (2023)
23, Number 3, pp. 875–883 DOI: 10.1177/1536867X231196496

iefieldkit: Commands for primary data collection and cleaning (update)

Kristoffer Bjärkefur
World Bank Group
Development Impact Evaluation
Washington, DC
kbjarkefur@worldbank.org

Luíza Cardoso de Andrade
University of Chicago
Development Innovation Lab
Chicago, IL
luizaandrade@uchicago.edu

Benjamin Daniels
World Bank Group
Development Impact Evaluation
and
Georgetown University
Initiative on Innovation, Development and Evaluation
Washington, DC
benjamin.daniels@georgetown.edu

Abstract. Data collection and cleaning workflows implement highly repetitive but extremely important processes. In this article, we describe an update to `iefieldkit`, a package developed to standardize and simplify best practices for high-quality primary data collection across the World Bank's Development Impact Evaluation department. The first release of `iefieldkit` provided workflows to automate error checking for Open Data Kit-based survey modules, duplicate management, data cleaning, and codebook creation. This update to the package includes improved commands to document and implement data point corrections, verify the structure or contents of data using codebooks, and create replication-ready data through automated variable subsetting.

Keywords: dm0105_1, iefieldkit, iecorrect, iecodebook, primary data collection, Open Data Kit, SurveyCTO, data cleaning, survey harmonization, duplicates, codebooks

1 Introduction

The `iefieldkit` package, first published in Bjärkefur, Cardoso de Andrade, and Daniels (2020), is a set of commands designed to simplify a series of tedious and repetitive tasks for Stata users who are in the process of collecting original data, especially survey data. This package now supports four major components of that workflow: survey design, data collection and quality assurance, data point correction, and data cleaning and survey harmonization. This update describes the new data point correction command, `iecorrect`, and new functionality extensions for data documentation and verification in `iecodebook`.

All `iefieldkit` commands use spreadsheet-based workflows so that their inputs and outputs are significantly more human readable than Stata do-files completing the same tasks, and these tasks can be supported and reviewed by personnel who specialize in field work rather than code tools. The increasing diversity and specialization of research teams has made accessibility to non-Stata-proficient personnel an essential component of data management workflows, and the `iefieldkit` package takes this development seriously.

2 The iecorrect command

This section describes `iecorrect`, a new command in `iefieldkit` that allows teams to collaborate on making corrections (changes) to datasets at the level of the individual data point. Like other commands in `iefieldkit`, `iecorrect` operates by using the `iecorrect template` subcommand to create a standardized spreadsheet template ("changelog") that users fill out without using Stata. The `iecorrect apply` subcommand then reads the contents of that changelog and applies the specified corrections to data in memory.

This workflow is designed with three advantages in mind compared with coding changes to individual data points in Stata do-files. First, it improves the human readability of the changes such that they can be implemented and reviewed by non-Stata users and quickly scanned by any reader without decoding Stata logic. Second, it increases the accuracy of corrections by building in multiple confirmations of the data points to edit. Finally, it reduces the amount of Stata code required to implement all data point corrections to one command, making data preparation do-files significantly shorter and easier to read (and preventing hard-to-detect issues that may arise from the order in which corrections are applied).

The general syntax is

```
iecorrect template using "filename.xlsx", idvar(varlist)
```

or

```
iecorrect apply using "filename.xlsx", idvar(varlist) [noisily
    save("filename.do") replace sheets(typelist) break]
```

2.1 Creating a corrections changelog template

To use this command, the user will first load the data and use the `iecorrect template` subcommand to create a spreadsheet changelog template at the desired location. The `idvar()` option should include a list of any variables that will be used to uniquely identify observations to which corrections should be made.

The template will contain three sheets called `string`, `numeric`, and `drop`. Each type of correction requires slightly different inputs from the user, which are discussed below. These sheets may not be deleted or their headers altered, or the rest of the command will not run properly. In addition, the template changelog will have columns for `initials` and `notes` that have no Stata functionality but allow the team to keep records of who made each change to the data and why. We recommend that these be completed in every case.

With the `iecorrect apply` subcommand, there are additional options allowing the user to examine, confirm, or archive complex corrections to data points. The `noisily` option will return all the executed operations in the Stata console with additional details, which is intended for examination of logic or recording to log files. The `save()` option requests that the complete code for all corrections be written to a do-file at the indicated location for archive or review; `replace` allows this to be overwritten. The `sheets()` option accepts a list containing the words `string`, `numeric`, or `drop` that indicates which of these changes should be implemented; this option is rarely used in the final code because the command will automatically ignore unfilled sheets, but it can be useful while implementing specific changes to review subsets of corrections in close detail. Finally, the option `break` will cause the command to return an error message when the variables listed in `idvar()` do not fully identify observations or when any line in the template does not change any observations—by default, these inconsistencies will result only in a warning message.

2.2 Types of corrections that can be performed

2.2.1 Corrections to string and numeric variables

String and numeric variable corrections can be made through the `string` and `numeric` sheet tabs of the changelog spreadsheet, respectively. The instructions for both types of corrections are identical. Take care to separate corrections for each type of variable into the correct sheet, however; although the command will flag variables if they appear to be in the wrong sheet, fixing this manually so that the command can run can be time consuming.

On these two sheets, the spreadsheet will begin with one column for each specified `idvar()`. These must be filled with the value of the ID variable in the observation to be corrected. In most cases, these columns should be used to uniquely identify observations to be edited; however, they can also be used to select observations more generally using nonuniquely identifying characteristics (for example, cluster or strata IDs). Either fill each variable with the intended value of the observations to be corrected or use the wildcard asterisk (`*`) to select all observations regardless of the value of that ID variable. Use the wildcard carefully.

After the columns for the specified identifying variables, the sheet contains columns to indicate the correction to be made to the selected observations. The `varname` column should be filled with the complete name of the variable to be corrected (lists, abbreviations, and wildcards are not supported). Filling this column is required for this type of correction to run. Next the `value` column should be filled with the desired corrected value of the specified variable to be replaced in the selected observations. This value will replace the current value once `iecorrect apply` is run. Filling this column is required for this type of correction to run properly.

The `valuecurrent` column operates as a confirmation that the correct observations and values are being corrected. It should be filled with the current, incorrect value of the variable in the observation, which, combined with the ID variables, will prevent a wide range of hard-to-detect errors from, say, multiple corrections that affect overlapping groups of observations, or misspecified variable names. Filling this column is required for this type of correction to run; it will also accept the wildcard asterisk (`*`) to select all observations regardless of their current values for the indicated variable. Use the wildcard carefully, and note that if you fill the `valuecurrent` and `all` columns representing ID variables with the wildcard, `iecorrect` will return an error because that would simply change all the data points for the specified variable.

2.2.2 Dropping observations

Corrections to data that require Stata to drop observations are implemented through the `drop` tab of the changelog spreadsheet. This tab has two required columns. One column will appear for each specified `idvar()`. These must be filled with the values of the ID variables in each observation to be corrected. It will also accept the wildcard asterisk (`*`) to select all observations regardless of their values for the indicated ID variable. Use the wildcard carefully.

The `n_obs` column must be filled with the exact number of observations that should be dropped corresponding to the ID values specified in the corresponding line. This column will not accept the wildcard asterisk (`*`); the number of observations to drop must be exactly specified. If the number of observations indicated to drop does not match the number that will actually be dropped by the command, `iecorrect` will return an error indicating the actual number of observations that were detected matching the ID pattern so that the user can check against the data for the source of the inaccuracy. This check makes it intentionally difficult to drop observations mistakenly.

2.3 Reviewing information in the changelog

Once `iecorrect apply` is used to apply corrections to the data, two new columns will be added to the `string` and `numeric` sheets, and one new column will be added to the `drop` sheet. A column named `date_last_changed` will be added to all sheets. It contains a timestamp indicating when `iecorrect apply` was last run. This means that when users enter new lines into the template after running `iecorrect apply`, they can tell which corrections are already reflected in the data.

A column named `n_changes` will be added only to the `string` and `numeric` sheets once `iecorrect apply` is called. It shows the number of observations that were modified by this correction. If users make a mistake when filling the template by, for example, entering values for the identifying variables that do not occur in the data, this column will show the value 0, and the command will return a warning saying that some corrections were not successfully implemented. Users can then refer back to this column to investigate what caused the issue. They may also choose to break the code should such an issue be detected, in which case they can specify the option `break`. This column is not added to the sheet `drop`, because it already requires the exact number or modified observations to be entered in the column title `n_obs`.

Through these columns, the information available in the changelog once corrections are applied to the data offers transparent documentation for any changes made to individual data points. This information can be more easily read by team members who are not familiar with code than the do-files that implement the changes. It can also be shared as part of the data documentation with published datasets or reproducibility packages.

3 New functionality in iecodebook: Verifying data and creating replication-ready data

The initial release of `iecodebook` included an `export` command that provided a simple utility for documenting the current state of a dataset through the creation of a spreadsheet "codebook". In this release, the `iecodebook export` subcommand adds two major functionalities through combinations of the options `signature`, `reset`, `verify`, and `trim()`. In addition, it now allows faster handling of multiple datasets with a built-in syntax to use data if desired through a main argument. The full syntax is

```
iecodebook export [ "data.dta" ] using "filename.xlsx" [ , replace
    plaintext(compact|detailed) noexcel save saveas("filename.dta")
    signature reset verify trim("filename.do" [ "filename.do" ] [ ... ]) ]
```

The base command—the `using` argument and optionally `replace`—will produce a spreadsheet containing a detailed description of the variables and labels in the dataset-specified location. If the optional main argument is specified, the command will `use`

the indicated data before taking action. Otherwise, it will export the codebook for the dataset that is currently in memory. The `plaintext()` option will additionally create a text-only (Git-compatible) codebook at the `using` file path with the same name and the `.txt` file extension. The contents of this file correspond to the outputs of the built-in `codebook` command if `plaintext(detailed)` is specified and `codebook, compact` if `plaintext(compact)` is specified. The `noexcel` option will suppress the spreadsheet output. The `replace` option will apply to all outputs from the command.

3.1 Verifying the content or structure of data before saving

The next set of options provides several functionalities for workflows where data contents and structures must be verified before saving data. `iecodebook export` handles two main use cases. First, a user may want to ensure that the data are totally unchanged, such as in the case of reproducibility packages or for code that imports occasionally updated remote data. Second, a user may want to ensure that the structure of the data is unchanged while allowing for the data themselves to change, such as in the case of continuous data collection and processing or the addition of further rounds of data collection that are intended to append seamlessly with existing data.

In both cases, the user will typically intend to save data at some location after confirming that they are correct. For example, the user might be accessing data from a shared drive or archival depository and copying them into a version-controlled code directory that does not contain (and does not track) the desired data files. The command would then ensure the data are appropriate for use with the code there after creating a trackable record of what the data should be. To do so, the user should specify the `save()` option, which will save the data at the `using` file path with the same name and the `.dta` file extension. Alternatively, the `saveas()` option will accept any file path. If all requested checks are completed and codebooks are created without error, the data will be saved to the specified location (again, the main `replace` option will be applied here).

3.1.1 Verifying data contents before saving

To verify that the contents of the data are unchanged, the user should first run the command with the `signature` and `reset` options. This will call the built-in `datasignature` command, writing a data signature file at the `using` file path with the filename ending in `-sig.txt` (to enable Git version control). In later runs, the user should remove the `reset` option; this will call the `datasignature confirm, strict` command targeting the expected file path. If the current data match the expected data signature (that is, their contents are exactly the same up to the order of each column), the command will run without error. Otherwise, the command will stop and return an error, reporting that the signature is not found or is different. This is intended to stop the accidental overwriting of data in place. It can be used in version control systems or replication packages, where data themselves are not stored, to ensure that other users are placing the correct data in the specified location. If the user later wants to ignore the saved signature and change the data, respecifying the `reset` option will allow overwriting the signature to reflect new data.

3.1.2 Verifying data structure before saving

To verify that the structure of the data matches a desired specification, a user should first run the base command, creating a spreadsheet codebook from the reference data. This will not work with the `noexcel` option. In later runs of `iecodebook export`, using data from the same source or from another source, the `verify` option should be specified with the `using` argument targeting the reference codebook. The codebook itself will never be overwritten in this use case (by definition, it must be identical). However, the user may want to create a copy of this codebook in a different location. The `saveas()` option will usually be sufficient to avoid filename collisions if data are being saved in a different location than the reference codebook for some reason (for example, if the reference is used for multiple rounds of data collection on the same project).

When `verify` is requested, the data in memory will be compared with the `using` codebook to ensure they are exactly compatible. The following checks will be run, and any differences will result in the command exiting with an error and listing all differences:

1. All variable names appear in both the codebook and the data, and all variable labels are identical.
2. The type of each variable is identical in the codebook and the data. String types of different lengths (including `strL`) are considered to be identical.
3. Value labels for labeled variables and the contents of each value label are identical in the codebook and the data.

These checks will ensure that different datasets will work with the same code as the code is intended; code outputs, of course, may change arbitrarily with the actual

contents of the data. This type of check will prevent potential nonbreaking errors that are difficult to detect, such as from using commands like `encode` when the contents of the underlying string variable changes between datasets (as long as the `encode` command is run before `iecodebook export, verify`). Of course, if the `verify` option is not specified, the contents of the data will not be checked, and the existing codebook will simply be overwritten if the `replace` option is specified.

3.2 Creating replication-ready data with trim()

In addition to the above checks for data integrity, `iecodebook export` can be used to automatically remove variables from data before saving them. The `trim()` option instructs the command to import a set of do-files, scan them for variable names, and retain only those variables used in the specified do-files before performing its other functions. To be conservative, the command retains only explicitly named variables. Therefore, do-files relying on variable lists that are not fully written out—such as abbreviations, wildcards, and some loop structures—are unlikely to work with `trim()` as the user expects; desired variables will in fact be dropped from the original data.

When the `trim()` option is specified with a list of do-files, `iecodebook export` will read the contents of the specified do-files and `drop` any variables that do not match the contents. It will then write codebooks and verify or save the resulting data as requested. For example, suppose the user has a do-file titled `analysis.do` containing only the line `sum foreign mpg trunk`. After loading `auto.dta`, the command

```
iecodebook export using "codebook-trim.xlsx",   ///
      trim("analysis.do") save replace
```

creates a codebook called `codebook-trim.xlsx` as well as a dataset called `codebook-trim.dta` in the same directory. Both contain only the variables `foreign`, `mpg`, and `trunk` because they are the only variables explicitly mentioned in the do-file.

Any number of do-files can be listed in the `trim()` option, and there are various ways to use the extended macro function `:dir` to create locals containing, for example, a list of all do-files in a given directory. The command does not check that all mentioned variables exist in the dataset. Therefore, it can be used with the optional main argument to load, trim, save, and document many datasets in a simple loop (as long as retaining different variables with the same name across datasets is not an issue). This is frequently useful, for example, when a project has many datasets and the user wants to trim all of them using the same set of analysis do-files.

4 Programs and supplemental materials

In addition to the Statistical Software Components Archive, the `iefieldkit` commands are hosted on GitHub and can be installed by typing

```
. net install iefieldkit,
> from("https://raw.githubusercontent.com/worldbank/iefieldkit/main/src")
```

5 Reference

Bjärkefur, K., L. Cardoso de Andrade, and B. Daniels. 2020. iefieldkit: Commands for primary data collection and cleaning. *Stata Journal* 20: 892–915. https://doi.org/10.1177/1536867X20976321.

About the authors

Kristoffer Bjärkefur is a data science consultant with the Development Impact Evaluation department (DIME) at the World Bank. He has previously worked in development research in the agricultural sector but is now working full time on supporting other researchers in their data work. This includes developing packages like `iefieldkit` and `ietoolkit`, training research teams in different programming methodologies, and providing teams with general data work advice when planning large data projects.

Luíza Cardoso de Andrade is the Data Analytics Lead at the University of Chicago's Development Innovation Lab. Her work focuses on incorporating nontraditional data sources into development research, promoting transparency and reproducibility in social sciences, and developing software tools to simplify research data work. Prior to joining the Development Innovation Lab, she was a junior data scientist with DIME at the World Bank.

Benjamin Daniels is a research fellow at the Georgetown University Initiative on Innovation, Development and Evaluation and a consultant with DIME at the World Bank. His research focuses on the delivery of high-quality primary healthcare in developing contexts. His work has highlighted the importance of direct measurement of healthcare provider knowledge, effort, and practice. To that end, he has supported some of the largest research studies to date using clinical vignettes, provider observation, and standardized patients. He works with the QuTUB Project.

All three authors were members of the DIME analytics team when these updates were made. This team creates tools and workflows that improve the quality and reproducibility of development research. It also supports best practices in econometrics, statistical programming, and research reproducibility across the i2i portfolio with DIME at the World Bank. This work comprises code and process development, research personnel training, and direct support for data analysis and survey development. The findings, interpretations, and conclusions expressed here are those of the authors and do not necessarily represent the views of the World Bank, its executive directors, or the governments they represent.