Have Your Cake and Eat It Too: A Case Study in Updated Modular Workflows for a Longitudinal Research Project

Cassia Smith
University of Wisconsin-Madison

Abstract

As datasets have become a more significant aspect of Open Science, attention has turned to the data transformations that drive their creation. Li and Ludäscher have pointed out the importance of identifying data cleaning workflows as a series of modular transformations that can be extracted for reuse. This modular approach aids reproducibility and allows for transparency in data provenance. However, the constantly evolving nature of data science technology means that even once these modules have been identified and implemented, their functionality must be ported to new platforms as old ones become less applicable or less common in a field of study. When these transformations take place, it is important to consider not only practicality and functionality, but also transparency within a data processing team. Clarity of communication within a team is the first step towards providing clear and transparent documentation to the end user. This case study of an updated workflow process for a long-running longitudinal health and well-being study provides practical examples of these principles.

Submitted 9 February 2024 ~ *Accepted* 22 February 2024

Correspondence should be addressed to Cassia Smith, Institute on Aging, 1300 University Avenue, 2245 Medical Sciences Center, Madison, WI 53706. Email: Cassia.Smith@ed.ac.uk

This paper was presented at the International Digital Curation Conference IDCC24, 19-21 February 2024

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Introduction and MIDUS Overview

As research datasets get larger, the need for automation and data standardization through quality control practices and data cleaning becomes increasingly clear. For smaller datasets, quality control can be manual and even in some cases indistinguishable from field editing or other standard publication prep. However, as datasets become larger and are recognized as valuable in their own right, the need to standardize and automate even basic data quality checks becomes more urgent and more identifiable as a curation activity separate from, though integral to, the analysis required for research publication. As Li and Ludäscher have pointed out (2022), these initial cleaning activities are important provenance information in the lifecycle of the published dataset, and tracking these actions is important to data curation transparency. As the value and lifecycle of data communities becomes an established model in the data curation field (Ruediger et al., 2022), it is important to consider the internal transparency of data curation practices as a part of maintaining community access to consistent, high-quality data. Combining the modular data cleaning model with the continuing demands of a mature data community allows our data curation processes to grow with the scope of our data collection and delivery mechanisms. This detailed case study provides an example of using an existing modular understanding of the cleaning process to allow a workflow to be updated in a way that still leverages the skillset of the existing data cleaning team.

Midlife in the United States (MIDUS) Creates Public Datasets

The Midlife in the United States (MIDUS) study is a longitudinal study run out of the Institute on Aging at the University of Wisconsin. It has been following several large national cohorts of subjects as they age, with the first (Core) cohort being recruited in the 1990s and the second major cohort (Refresher) recruited in 2012. As the study has continued, the range of data collected has also grown. The initial sample participated in a psychological assessment in the first wave of data collection. Subsequent waves have broken out large sub-samples that have participated in various additional projects collecting daily stress, biomarker, genetic, and neurological data. After each wave of data collection, data gathered by each sub-project is cleaned, metadata and data quality flags are set, and the data is aggregated, de-identified, and released as a public dataset. Depending on the metric, de-identified raw data is also available on request.

This paper focuses on the cleaning process used by the data processing staff working for the Biomarker sub-sample. The Biomarker project sub-sample typically includes around a thousand subjects per wave of data collection, each of whom is associated with approximately 3,000 variables in the Biomarker project alone. This sub-sample has been running for almost 20 years.

Because the MIDUS study has been in the field for so long, and because, as a longitudinal study, the data from each wave of collection needs to be as directly comparable as possible, a number of best practices have been established over time. The goal of these best practices was to ensure that data collection and cleaning operations were applied consistently. An outcome of that goal has been modular, reusable data cleaning workflows from one wave of data collection to the next. This cleaning has largely been a semi-manual process. As the volume of data has increased (along with a recent increase in the speed of data collection), the project has come to an inflection point where more automated data processing is desirable.

Using SPSS to Create a Transparent Modular Workflow

The pre-existing MIDUS data cleaning procedure has principally involved a semi-manual process that involves using SPSS syntax to identify anomalous cases. These anomalies are either logged for later cleaning or addressed immediately. The SPSS data workflow provides two major "modes" of interacting with data:

- A point-and-click mode in which data manipulation options are selected from a series of menus.
- A code-like mode in which a proprietary SPSS language (syntax) is used to manipulate the data. This syntax is stored in its own file separate from the data file and can import, combine, and manipulate multiple datasets. The code is run by selecting the section of syntax the user wishes to execute and clicking a Run button. Syntax files are not meant to be run all at once like a traditional computer script. Instead, they contain discrete actions and queries that are related to a specific workflow.

Both modes of SPSS data manipulation are capable of executing query-like operations on the dataset that produce a non-transforming output dataset. These query results are sent to a third file, called the output file. The syntax mode also logs all executed syntax, even non-query language, and all error messages in the output file. The point-and-click mode also sends some output to this file, but it is not as important to non-query point-and-click operations. MIDUS uses the syntax mode of SPSS data manipulation almost exclusively. This means that the standard MIDUS SPSS data cleaning workflow involves three SPSS files working in concert (Figure 1).

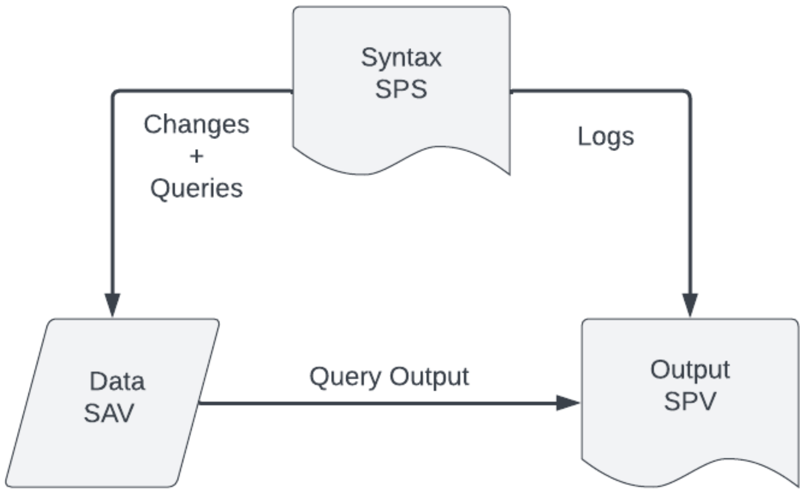
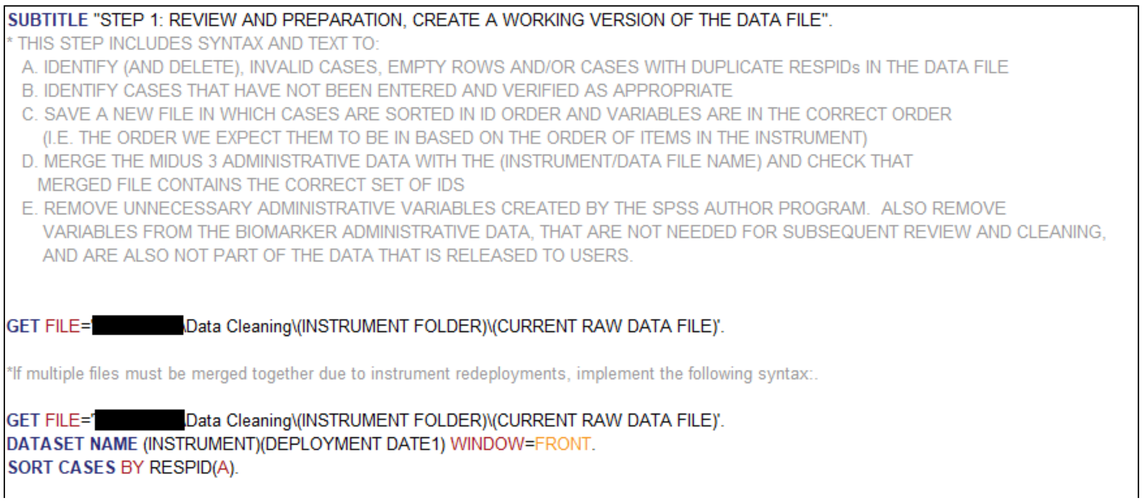


Figure 1. SPSS syntax-based data workflow.

MIDUS has been using SPSS for data cleaning from a very early point in its existence, and the team leverages this three-file system extensively to provide provenance information throughout the cleaning process. Relevant query results from the output file can be copied into the syntax file and annotated for immediate or eventual cleaning, with curator initials and query date required for each annotation. A major step in the final cleaning process involves printing key query results from the output file onto paper and annotating cleaning decisions in longhand. The annotations are then used to build a final cleaning file consisting mostly of case-specific flags and corrections. The printed and annotated paper documents would be retained until the cleaned data was finalized and released, when they would be discarded.

Standard cleaning syntax, such as setting variables to the project's missing codes or reformatting time variables, can also be easily copied and pasted from syntax used in previous waves. This syntax typically needs minor updates to allow for changes to the instruments or new conventions in SPSS syntax or data formatting, but the basic structure of the cleaning module could typically be kept. To ensure that these cleaning steps were applied logically and consistently, a standard cleaning workflow was established as a skeleton SPSS syntax file that could be expanded with a combination of new and reused syntax appropriate to each assessment (Figure 2).



```
SUBTITLE "STEP 1: REVIEW AND PREPARATION, CREATE A WORKING VERSION OF THE DATA FILE".
* THIS STEP INCLUDES SYNTAX AND TEXT TO:
  A. IDENTIFY (AND DELETE), INVALID CASES, EMPTY ROWS AND/OR CASES WITH DUPLICATE RESPIDs IN THE DATA FILE
  B. IDENTIFY CASES THAT HAVE NOT BEEN ENTERED AND VERIFIED AS APPROPRIATE
  C. SAVE A NEW FILE IN WHICH CASES ARE SORTED IN ID ORDER AND VARIABLES ARE IN THE CORRECT ORDER
     (I.E. THE ORDER WE EXPECT THEM TO BE IN BASED ON THE ORDER OF ITEMS IN THE INSTRUMENT)
  D. MERGE THE MIDUS 3 ADMINISTRATIVE DATA WITH THE (INSTRUMENT/DATA FILE NAME) AND CHECK THAT
    MERGED FILE CONTAINS THE CORRECT SET OF IDS
  E. REMOVE UNNECESSARY ADMINISTRATIVE VARIABLES CREATED BY THE SPSS AUTHOR PROGRAM.  ALSO REMOVE
     VARIABLES FROM THE BIOMARKER ADMINISTRATIVE DATA, THAT ARE NOT NEEDED FOR SUBSEQUENT REVIEW AND CLEANING,
     AND ARE ALSO NOT PART OF THE DATA THAT IS RELEASED TO USERS.


GET FILE=[redacted]Data Cleaning\(INSTRUMENT FOLDER)\(CURRENT RAW DATA FILE)'.

*If multiple files must be merged together due to instrument redeployments, implement the following syntax:.

GET FILE='[redacted]Data Cleaning\(INSTRUMENT FOLDER)\(CURRENT RAW DATA FILE)'.
DATASET NAME (INSTRUMENT)(DEPLOYMENT DATE1) WINDOW=FRONT.
SORT CASES BY RESPID(A).
```

Figure 2. Excerpt from MIDUS Biomarker cleaning template, with network locations redacted.

Advantages of this workflow include:

- deep and broad knowledge of SPSS syntax among MIDUS staff, making the data cleaning process highly transparent internally;
- reusable SPSS syntax from wave to wave of data cleaning; and
- the dated and initialed annotations bundled with the syntax for easy reference.

Disadvantages include:

- the use of SPSS syntax, which is not familiar to younger incoming staff;
- discarding the annotated paper documents after a round of cleaning (loss of provenance); and
- the highly manual review process, which becomes less practical as the volume and speed of incoming data increases.

Growing Project Prompts Need for Further Automation

The MIDUS Biomarker project was introduced during the data collection wave in the early 2000s. This was the second data collection wave for the initial cohort (see Figure 3). A sub-sample of the total MIDUS sample was assigned to one of three locations in the US (Los Angeles, California; Madison, Wisconsin; or Washington, D.C.) and brought in for a thorough clinical assessment. This initial wave of collection already involved extensive metrics, from psychological questionnaires to multiple assays performed on blood, urine, and saliva samples. The final dataset contains just under 3,000 variables per participant, including administrative variables and various data quality flags.

While this initial data collection wave was extensive, subsequent waves have provided even more data. For example, the collection of activity and sleep data using a device called an Actiwatch was initially restricted to participants who came to the clinic in Wisconsin. (All locations used a self-reported sleep diary.) In the most recent round of data collection, completed in 2022, additional funding allowed us to expand Actiwatch data collection to all of the data collection sites. Another source of increased data volume has come from the addition of new categories of metrics, many of them collected digitally and allowing for very detailed and technical datasets. For example, we have now completed two waves of data collection (one for each cohort) that include measurements of gait data collected using sensors attached to the ankles and lower back. This produces data on a number of gait characteristics that are then exported from the sensors' proprietary software and incorporated into our dataset.

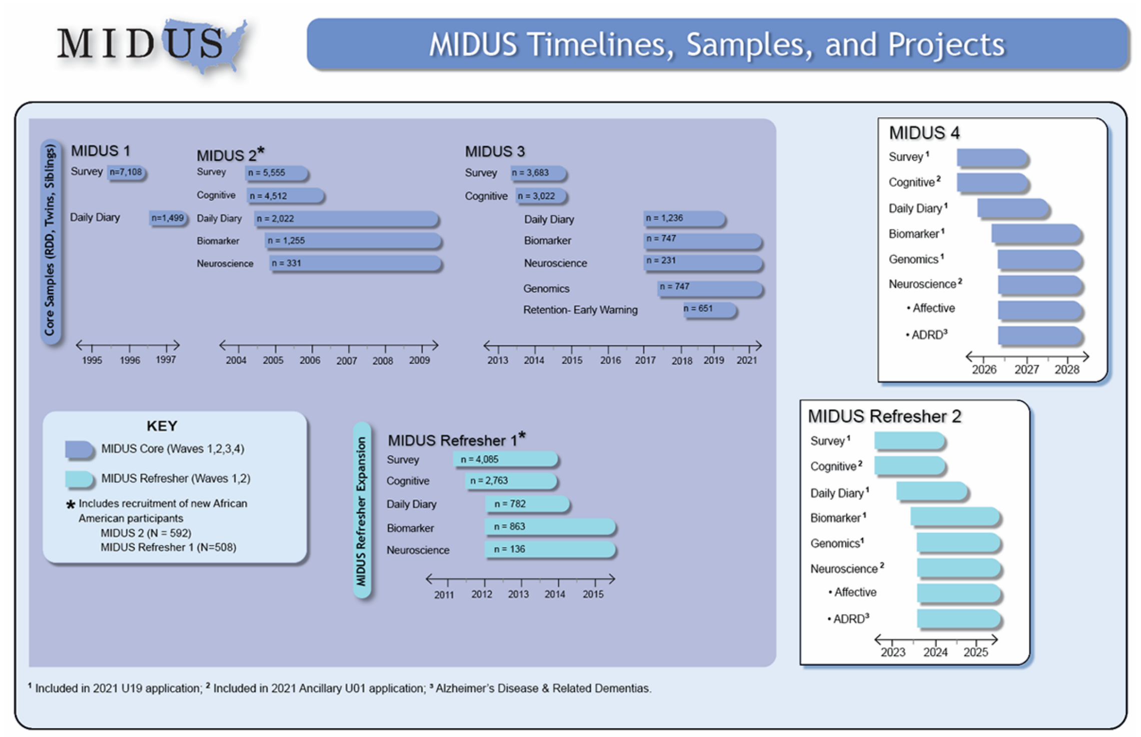


Figure 3. MIDUS data collection timelines.

The dataset we released earlier this year contained roughly 200 more variables than the initial Biomarker dataset. In addition, more participants had valid data for nearly the full dataset, rather than only the subset of data available at their assigned data collection location. Add a current collection schedule that almost doubles the pace from previous waves of data collection, and the pressures on Biomarker data staff are increasing.

This growth has put data curation for the project at an inflection point. The semi-manual process of running discrete SPSS tasks has worked well. However, the combination of more and larger digital datasets and a faster data collection schedule has brought things to the point where this process is no longer a sane or viable way of processing incoming data. Some degree of more hands-off data processing must be introduced.

Extending Cleaning Modularity into Python Automation

As the data collection and cleaning process has been reviewed for opportunities to improve efficiency, it has been important to consider the strengths of the old process when implementing a new workflow. Because SPSS has been so integral to MIDUS data for over 20 years, there is a deep and meaningful well of institutional knowledge on the software. Not only has SPSS syntax and output been woven into the fabric of the data cleaning process, but our final datasets are also always released in SPSS SAV data file format, which allows for more extensive metadata than, for example, a CSV file would provide on its own.
Keeping this in mind, it was important to be strategic when considering how and when to eliminate SPSS portions of the workflow. As the previous wave of data collection was wrapping up, it was agreed that Python could be introduced in a limited capacity, to batch-apply a handful of transformations that needed very little oversight. (For example, file aggregation or one-to-one ID swaps.) This has been used to prepare a significant percentage of our less complex native digital data for cleaning.

As our current wave of data collection has been planned and gone into the field, a major goal has been to convert more of our initial quality checks to an automated Python system. Our final cleaning workflow is too extensive to convert within our current collection timeframe, especially at the cost of losing the established workflow, institutional knowledge, and standard output file format from past waves of data cleaning. However, much of our incoming data is delivered on a monthly or semi-annual basis and goes through a set of initial quality control checks before being stored for later cleaning. The results of these checks inform regular feedback to field staff to ensure they are collecting data accurately. The checks are also used to populate a preliminary list of data issues to address during cleaning. This is the task most strongly impacted by the increased speed of data collection. It is also the least complex portion of data cleaning, with minimal batch-applied formatting making up the majority of data transformations.

The challenge, then, was threefold:

- To convert this step of the data curation process to automation without losing the provenance that had been bound up in the SPSS syntax workflow;
- To track transformations and anomalies from outside of SPSS in ways that are still transparent to a team whose core competency is SPSS and Excel;
- To develop a Python Quality Control (QC) system that allows for highly flexible and modular code that can be reused in future waves of data collection.

Much of the remainder of this section will discuss the specific technical considerations that went into adapting this workflow for a specific instrument, using the Li-Ludäscher model for identifying and reusing modular workflows. However, it is important to note that the considerations listed above also impacted the design and implementation of this case study, outside of the Li-Ludäscher model.

Medical History Interview Quality Control: An Overview

One of the prime candidates for a more automated quality control system was the lengthy Medical History Interview, a staff-administered questionnaire that examines the study participant's medical history, from reproductive health to previous surgeries, alcohol and tobacco use, family medical conditions, preventive health practices, and significant deaths in the participant's social circle. The Medical History Interview tends to receive the most extensive revisions per wave of data collection, as questions that have already been answered are dropped and questions of new physiological and psychological concern are added. It is also typically one of the longest and most complex assessments administered by the Biomarker team, with around a thousand variables and many complex skips based on participant responses.

In previous waves of data collection, the monthly QC for the Medical History was conducted manually in SPSS. The variables for each set of skips were retrieved one subject block at a time using the syntax query language and then most of the quality control was done by visually reviewing the aggregated results, with some very light automation to make sure years were not outside of range. Any significant errors (usually blank variables) were then logged in a separate Microsoft Excel file for follow-up with field staff and for the data cleaners' later reference. When performed conscientiously, these manual checks could easily take four to six hours to complete each month, and the visual review process could be cognitively taxing. When considering assessments to review using automation, the Medical History was an obvious candidate.

Automating Medical History QC: Planning

Li and Ludäscher (2022) have outlined the basic conceptual model (the Li-Ludäscher model) for producing reusable workflows in data cleaning. Technically, the quality control for these assessments is a pre-cleaning step, but applying this model is still instructive to understanding how the project was planned and implemented.

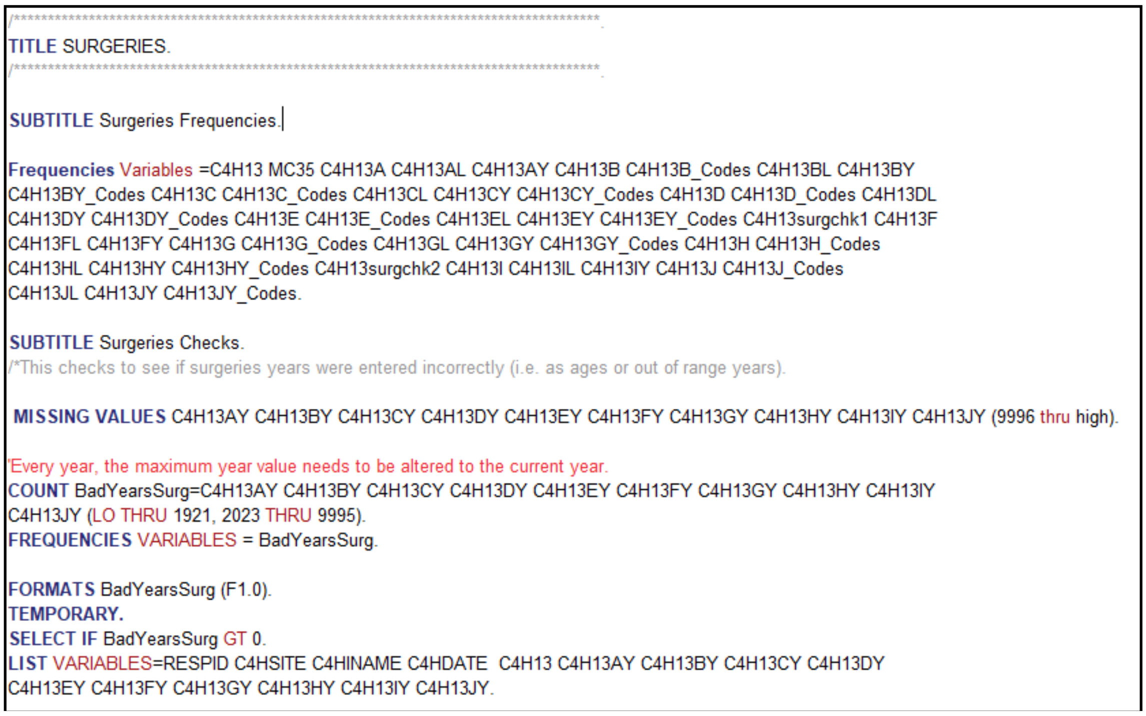
The first three steps of the Li-Ludäscher model are identifying the data analysis *purpose* and *questions* applied to the dataset and then using them to define data cleaning *objectives* (2022, p. 4). Since MIDUS datasets are constructed for open-domain public release, these terms are less clearly defined for this project. In broad terms, they could be described as follows:

- Purpose: Creating a complete and usable dataset for researchers.
- Question (Query): Are the variables completed as fully as they should be, given the skip logic activated for each participant?
- Objective: Evaluate all of the skips for each study participant, ensuring that they are completed correctly and that any anomalies have a documented justification.

The next step of the Li-Ludäscher model involves using provenance from previous cleaning to develop a recipe for achieving the data cleaning objective (2022, p. 4). This is available through the SPSS syntax files used to clean the Medical History in the previous wave.

It is not, in the language of Li and Ludäscher (2022, p.3), safe or meaningful to apply this data QC recipe directly to the current wave of data collection, for a number of reasons:

- This is in SPSS syntax and we wish to convert to Python;
- The naming convention for these variables is different for the current wave of data collection;
- Many of the Medical History variables have been moved, deleted, added, or reformatted since the previous wave of data collection.



```
/****************************************************************************************.
TITLE SURGERIES.
/****************************************************************************************.

SUBTITLE Surgeries Frequencies.

Frequencies Variables =C4H13 MC35 C4H13A C4H13AL C4H13AY C4H13B C4H13B_Codes C4H13BL C4H13BY
C4H13BY_Codes C4H13C C4H13C_Codes C4H13CL C4H13CY C4H13CY_Codes C4H13D C4H13D_Codes C4H13DL
C4H13DY C4H13DY_Codes C4H13E C4H13E_Codes C4H13EL C4H13EY C4H13EY_Codes C4H13surgchk1 C4H13F
C4H13FL C4H13FY C4H13G C4H13G_Codes C4H13GL C4H13GY C4H13GY_Codes C4H13H C4H13H_Codes
C4H13HL C4H13HY C4H13HY_Codes C4H13surgchk2 C4H13I C4H13IL C4H13IY C4H13J C4H13J_Codes
C4H13JL C4H13JY C4H13JY_Codes.

SUBTITLE Surgeries Checks.
/*This checks to see if surgeries years were entered incorrectly (i.e. as ages or out of range years).

 MISSING VALUES C4H13AY C4H13BY C4H13CY C4H13DY C4H13EY C4H13FY C4H13GY C4H13HY C4H13IY C4H13JY (9996 thru high).

'Every year, the maximum year value needs to be altered to the current year.
COUNT BadYearsSurg=C4H13AY C4H13BY C4H13CY C4H13DY C4H13EY C4H13FY C4H13GY C4H13HY C4H13IY
C4H13JY (LO THRU 1921, 2023 THRU 9995).
FREQUENCIES VARIABLES = BadYearsSurg.

FORMATS BadYearsSurg (F1.0).
TEMPORARY.
SELECT IF BadYearsSurg GT 0.
LIST VARIABLES=RESPID C4HSITE C4HINAME C4HDATE  C4H13 C4H13AY C4H13BY C4H13CY C4H13DY
C4H13EY C4H13FY C4H13GY C4H13HY C4H13IY C4H13JY.
```

Figure 4. SPSS Medical History QC syntax from previous data collection wave.

The previous wave's data QC recipe is clearly not reusable as-is for the current project. It is, however, possible to adapt and decompose (Li & Ludäscher 2022, p. 4) this existing recipe into a highly extensible and usable Python framework.

One particularly prominent feature of the original data QC recipe is the organization of the data into blocks of variables by topic, with the skips reviewed block-by-block. While there is no particular operational need to do this from a technical perspective, skip errors typically need to be resolved via review and communication with field staff by the data curation team, and having the variables organized by topic makes this process more convenient. This structure also provides a convenient bridge to Python programming: Declaring a Medical History Section class object is a digestible and logical way of approaching data review from a programmatic viewpoint as well. Using objects, or an Object-Oriented Programming approach, in Python overall meshes particularly well with the Li- Ludäscher model, as the reusable parts, or modules, that they advocate identifying in existing data cleaning recipes translate very well to the abstracted versions of operations used in Python models. In fact, an abstracted or module-based approach is particularly appropriate for the Medical History, as it is the most extensively revised assessment for each wave of data collection. The specific content of Medical History sections will change from wave to wave, but the abstracted concept of a Medical History Section will not. All Medical History sections of any wave will have variables, skips, and stems.

A second key feature of the existing Medical History recipe is the Excel file where skip errors and corrections were logged. In the SPSS workflow, these files were created, populated, and updated by hand, based on the SPSS syntax results in the SPSS output file. Since in a Python system, the program would be flagging skip errors for curator review, it made sense for Python to create and populate this spreadsheet, while data curators would review and update its content. It was important that any errors detected by the QC system be output in a stable format that was accessible to both the system developer and to staff with SPSS specialization, since both could be referencing it at different points in the data cleaning process. The developer chose to use CSV for these files, as they could be reviewed and edited using Excel but did not require extra Python packages to be installed, as would be the case for an XLSX file.

There were two additional factors not accounted for by the old recipe that needed to be taken into consideration while planning the new quality control system. First, the method of data collection for this assessment was changed, from using an SPSS Interviewer file to using the secure online health data portal REDCap. The reasons behind this change are too extensive to recount here, but one practical effect is that REDCap allows for more granular validation of data entry fields. This eliminated the need to check data formats as stringently in the new QC system. Second, for this wave of data collection, the Medical History was undergoing continued revision right up until almost the moment that data collection started. Because it is difficult to renumber variables within REDCap, this meant that the internal numbering of the variables for the assessment became non-sequential. It is standard practice for our public datasets to be released with the variables within each assessment organized sequentially, so the entire assessment would need to be re-numbered from top to bottom as part of the monthly formatting process.

Because of this renumbering, it was important that any formatted data file output by the QC system be in SPSS-ingestible format, since final data cleaning would still take place in SPSS. Again, CSV was selected for this output, because it is SPSS-ingestible and because native creation of SPSS data (SAV) files in Python requires the installation of even more precarious packages than Excel formats. A downside of this system is that the standard metadata associated with the Medical History dataset will have to be applied via SPSS syntax post-file creation rather than during the Python formatting process. However, this was already a feature of the SPSS data QC recipe, due to the constraints of the previous data collection software. Fortunately, dataset metadata creation is a task that SPSS syntax handles well.

Automating Medical History QC: Python Implementation

This paper is not intended to be an in-depth code review of the Medical History QC system as it was ultimately implemented. However, a discussion of a few key features of the Python objects defined for the system can be useful to other practitioners, as they are suggestive of the overall approach.

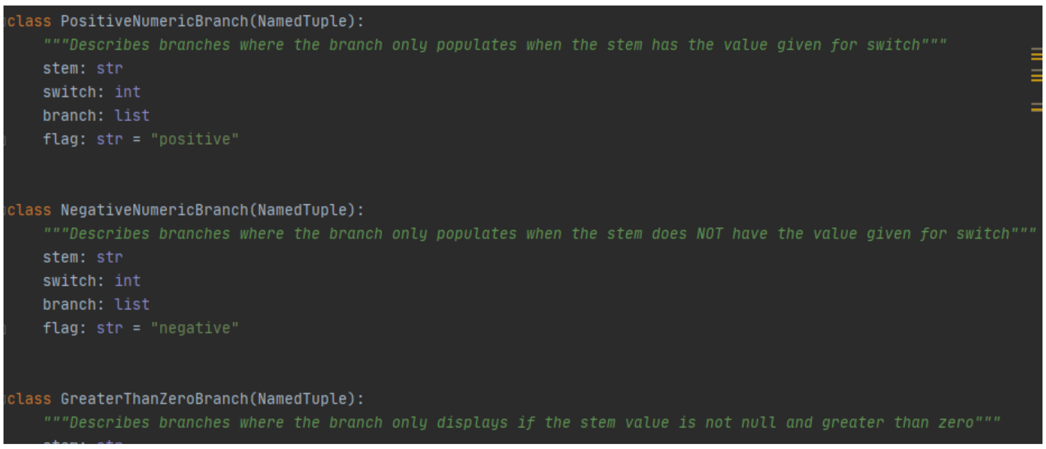
There were two basic types of generic model defined for the Python QC system: 1) Generic skip definitions based on Named Tuples, and 2) a generic Medical History Section. The Medical History Section child classes were then used to break out the Medical History Interview content into blocks based on topic and size of skips. (For example, the Reproductive Health section was defined in smaller chunks depending on whether the respondent had been assigned male or female at birth, whether people with uteruses were still menstruating or had ever been pregnant, etc.) These divisions were somewhat arbitrary but were instituted mostly so that identifying and trouble-shooting data issues would be more manageable and intuitive for data staff, and so that skip logic would not need to be overly complex in order to describe the data.

Defining skips

Skips, in the context of this dataset, typically have three basic components:

- A stem question that must be answered in order to determine if the skip is triggered or not;
- A trigger condition for that question, that must exist in order for the skip to be activated (for example, selecting “Yes” for a yes/no variable, or entering a number greater than zero for a numeric variable); and
- A list of variables that should only be displayed if the stem question receives the right input.

These key components were given a standard vocabulary that was used in nearly every skip type that was defined in this system. (More on the exception shortly.) Every skip was written to expect a stem variable, a “switch” (the trigger condition), and a “branch” (the variables that are revealed or hidden by the skip). A “flag” was also included in each skip to help make the type of trigger condition more accessible when the skips are evaluated. The skips themselves were also referred to as branches, to maintain linguistic continuity with REDCap, which refers to skips as "branching logic". Because REDCap defines skips by which trigger conditions *reveal* variables rather than hide them, the skip definitions in Python followed the same approach.



```
class PositiveNumericBranch(NamedTuple):
    """Describes branches where the branch only populates when the stem has the value given for switch"""
    stem: str
    switch: int
    branch: list
    flag: str = "positive"


class NegativeNumericBranch(NamedTuple):
    """Describes branches where the branch only populates when the stem does NOT have the value given for switch"""
    stem: str
    switch: int
    branch: list
    flag: str = "negative"


class GreaterThanZeroBranch(NamedTuple):
    """Describes branches where the branch only displays if the stem value is not null and greater than zero"""
```

Figure 5. Generic skip logic.

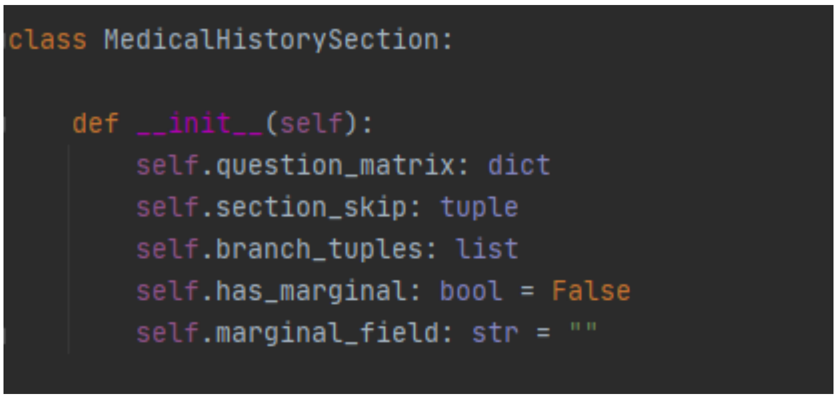
While most skips followed the format given above, two additional types of skips were defined that allowed for nested skips. *Any* type skips have the same format as the one given above but are evaluated with the expectation that they will contain a list of sub-skip definitions rather than a list of variables. This skip type is used to define instances where, once the skip is activated, at least one of the sub-branches should be filled out. (For example, if the participant indicated they had food allergies, at least one and up to three sets of variables providing details on those allergies should be valid.) *Any* skips also include a "charge" (which can be positive ["pos"] or negative ["neg"]) that is used to indicate if the sub-skips should be evaluated when the trigger condition is present or when it is absent.

Or type skips are the one kind of skip with a different structure: They assume that either of two individual skip tuples could be correct. This allows for both skips where different responses to the same stem can reveal different variables and skips where two different stems can reveal overlapping variables depending on the value given. The two tuples are listed as "condition1" and "condition2", with the stem and flag hard-coded to "or". Finding a way to better standardize this skip type is tech debt reserved for future development of this tool.

Defining sections

The skip definitions discussed in the previous section are, in turn, used in combination with the Medical History Section child classes to define the patterns for completing sections as a whole.

The basic structure of the Medical History Section was defined as a class (Figure 6) and children of that class were used to define the Medical History content section by section (Figure 7). The question matrix was used to store the crosswalk from the REDCap variable numbering scheme to a sequential one. Tuples were used to define section-level skips (where the entire section was or was not populated based on a stem question) as well as skips within the section (for example, checking to see if year variables were populated for corresponding events such as surgeries). Marginal fields in MIDUS datasets are used for field staff to record unusual circumstances surrounding nonstandard or edge case answers. Some sections that frequently inspire caveats have their own marginal notes field, while many others do not.

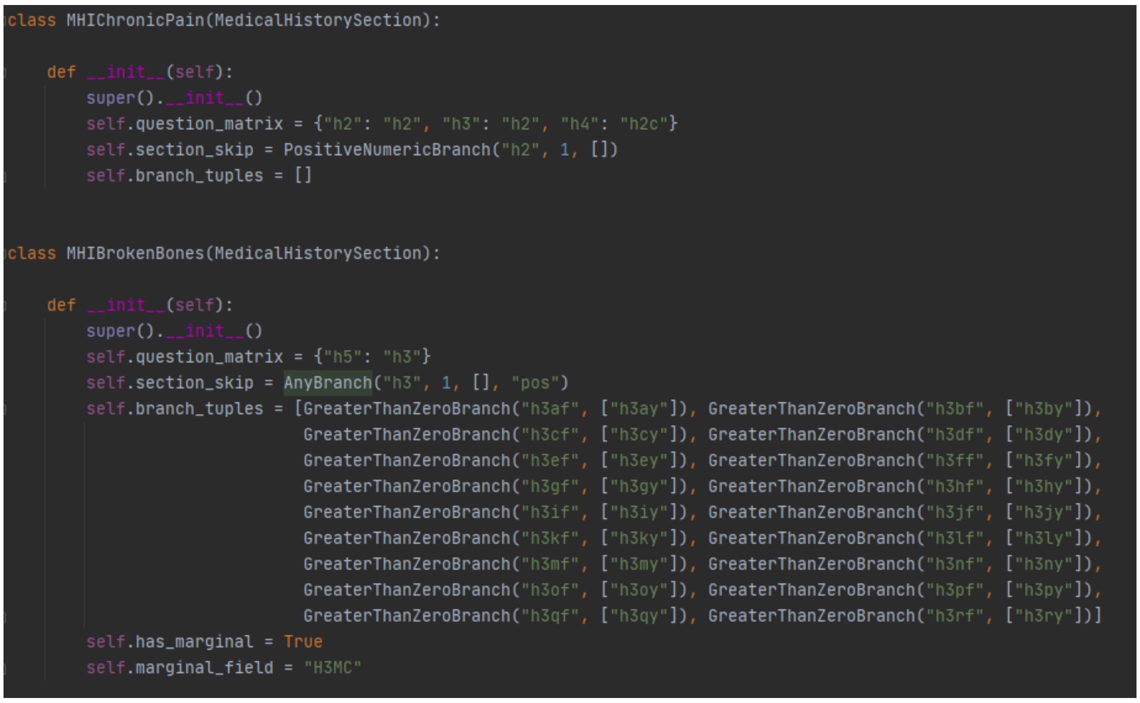


```python
class MedicalHistorySection:

    def __init__(self):
        self.question_matrix: dict
        self.section_skip: tuple
        self.branch_tuples: list
        self.has_marginal: bool = False
        self.marginal_field: str = ""
```

Figure 6. Generic Medical History Section definition.

Properly populating the section-specific mappings was by far the most time-consuming and challenging part of the model definitions, and the skip logic in particular required extensive testing and comparison with the REDCap skip logic. However, now that the generic Medical History Section class has been defined and implemented, those mappings are the only part of the QC system that should need to be re-written with each wave of data collection. All of the remaining code relies on the basic structure and methods defined on the parent class.



```python
class MHIChronicPain(MedicalHistorySection):

    def __init__(self):
        super().__init__()
        self.question_matrix = {"h2": "h2", "h3": "h2", "h4": "h2c"}
        self.section_skip = PositiveNumericBranch("h2", 1, [])
        self.branch_tuples = []


class MHIBrokenBones(MedicalHistorySection):

    def __init__(self):
        super().__init__()
        self.question_matrix = {"h5": "h3"}
        self.section_skip = AnyBranch("h3", 1, [], "pos")
        self.branch_tuples = [GreaterThanZeroBranch("h3af", ["h3ay"]), GreaterThanZeroBranch("h3bf", ["h3by"]),
                              GreaterThanZeroBranch("h3cf", ["h3cy"]), GreaterThanZeroBranch("h3df", ["h3dy"]),
                              GreaterThanZeroBranch("h3ef", ["h3ey"]), GreaterThanZeroBranch("h3ff", ["h3fy"]),
                              GreaterThanZeroBranch("h3gf", ["h3gy"]), GreaterThanZeroBranch("h3hf", ["h3hy"]),
                              GreaterThanZeroBranch("h3if", ["h3iy"]), GreaterThanZeroBranch("h3jf", ["h3jy"]),
                              GreaterThanZeroBranch("h3kf", ["h3ky"]), GreaterThanZeroBranch("h3lf", ["h3ly"]),
                              GreaterThanZeroBranch("h3mf", ["h3my"]), GreaterThanZeroBranch("h3nf", ["h3ny"]),
                              GreaterThanZeroBranch("h3of", ["h3oy"]), GreaterThanZeroBranch("h3pf", ["h3py"]),
                              GreaterThanZeroBranch("h3qf", ["h3qy"]), GreaterThanZeroBranch("h3rf", ["h3ry"])]
        self.has_marginal = True
        self.marginal_field = "H3MC"
```

Figure 7. Using Named Tuple skip logic classes to populate Medical History Section classes.

Extending skips and sections

While the current Python QC definitions are focused on the demands of the Medical History, the recipe established by the Python code is highly transferrable. Named Tuple skips are easily directly ported to evaluate the skips of a different assessment with no alteration. The Medical History Section class can be easily split into an even more abstracted Assessment Section class, with the question matrix and renumbering methods remaining on the Medical History class and the other attributes and methods moved up to the Assessment Section. This would allow the Assessment Section to be assigned child classes from any other assessment that needs skip review.

Conclusions

FAIR (findable, accessible, interoperable, and reusable) data principles are rightly spoken of in the context of promoting communication between researchers. And modular data cleaning workflows do promote the interoperability and reusability of data. For example, the consistency of MIDUS cleaning protocols across waves allows us to provide consistently applied data quality flags and data cleaning operations across our datasets: An asset to researchers seeking to use, reuse and recombine our data. However, it is important to remember that FAIR and transparent data practices are just as important to internal communication as they are to external communication. Promoting clear and accessible communication within a team is a first step to providing clear documentation to external researchers.

As Boeckhout et al. (2018) have pointed out, machine-actionability is a key aspect of FAIR data principles (pp. 932-933). This is true in the sense of creating datasets that are machine-ingestible for user analysis. But it also interlocks with the Li-Ludäscher modular data review module: A machine-actionable cleaning module is a module uniquely suited to reuse, and not just to reuse, but to clear, consistent, describable reuse. This allows for better internal documentation and refinement of data cleaning practices, completely aside from the benefit to outside researchers. It is important to implement FAIR practices for the benefit of the wider data community, but in this case, FAIR data management benefits the research team itself as well. In this case study, the clear documentation of the existing SPSS data QC recipe allowed the Python QC system to be designed more easily. Applying interoperability to the design of the Python recipe will allow it to be extended for future QC tasks within the project.

However, when implementing the Li-Ludäscher model, it is important to keep in mind other aspects of FAIR beyond the conceptual and technical concepts the model employs. Accessibility to research staff is the first step towards accessibility for the wider data community. Ruediger et al. (2022) have noted that metadata and ontologies are a "social infrastructure", "embody[ing] community norms and shared… frameworks of understanding" (p. 15). While this is true of metadata, it is also important to note that data cleaning workflows are also a social infrastructure, particularly in the context of a longitudinal project such as MIDUS. Not only is it important to be aware of the baseline assumptions that underpin cleaning choices in long-running or community standard cleaning workflows, but even the act of engaging in data cleaning itself assumes certain levels of technical skill and data understanding on the part of the project curators. Perfectly translating a cleaning module into a format too opaque to be used or parsed by existing cleaning staff would not be a truly successful module reuse, even if in terms of data modification or evaluation it functioned in precisely the same way.

As more data communities become established and mature (Ruediger et al., 2022), it will become increasingly important to consider any changes in data management technology not only in the context of what is new and future-oriented, but also in terms of the existing data and domain expertise contained within the community as a whole and within the research teams that comprise it. This does not have to mean that data management cannot be fully or partially updated as appropriate, but it does suggest supplemental concerns to add to Li and Ludäscher's modular cleaning reuse approach. In particular, this curator suggests adding the following questions to reuse considerations, particularly when the technology used to implement the cleaning recipe will be changed or upgraded:

- Which research staff (*who*) implements this recipe? Is it employed by field staff, by data cleaning staff, by lab staff, by the researchers themselves?

- Does that staff person or team have the *capacity* to update this workflow in the way being considered? *Capacity* can refer to technical ability, hours available to devote to the project, cleaning technologies they are familiar with, available hardware and software, and any other broader factors that may complicate the technical implementation of the update or reuse as conceived.

- Who will use this data *next?* What is the *data pipeline* for this data? Will the next person to use this data after the cleaning step under review have specific data format or software compatibility needs that are not strictly required by this cleaning process? Do modifications need to be made to the current or projected recipe in order to help the data flow to its next user more smoothly? In some cases, it may be useful to review the entire pipeline in order to ensure, for example, that a specific variable that is not useful for intermediate cleaning steps is still available in the final shared dataset. But even in cases where mapping the entire pipeline is not necessary, understanding the needs of the next step in the chain is a useful exercise, especially on teams with multiple individuals cleaning or analysing the same data. This could be seen as an aspect of the cleaning objectives from the Li-Ludäscher model, but it is worth highlighting as a sometimes less-intuitive consideration.

The ultimate goal of FAIR data principles should be an open science environment, where data can be shared freely and meaningfully within, at minimum, the community of practice where it was created. However, it is worth investigating the benefits these principles can offer within a research team, before the data has a chance to be shared elsewhere. This paper provides one example; the author looks forward to seeing others from the broader data curation community.

Acknowledgements

I would like to thank Linzhu Tian, my colleague at MIDUS who helped develop the Biomarker SPSS data cleaning workflow along with Dr. Gayle Love. I would also like to acknowledge principal investigator Dr. Carol Ryff for her support and tireless work on the MIDUS project.

References

Boeckhout, M., Zielhuis, G., & Bredenoord, A. (2018). The FAIR guiding principles for data stewardship: Fair enough? *European Journal of Human Genetics*, 26, 931-936. doi:10.1038/s41431-018-0160-0

Li, L., & Ludäscher, B. (2022). On the reusability of data cleaning workflows. *International Journal of Digital Curation*, 17(1), 6pp. doi:10.2218/ijdc.v17i1.828

Ruediger, D., MacDougall, R., Cooper, D., Carlson, J., Herndon, J., & Johnston, L. (2022). *Leveraging data communities to advance open science: Findings from an incubation workshop series.* ITHAKA S+R. doi:10.18665/sr.317145