Elizabeth Wickes

Data Cleaning

Final Project Report

## The Problem to Solve

I am trying to solve the problem of overcoming the workload barrier of writing documentation for datasets. Simple and small datasets can be relatively easy to skim over and pick out columns for description. However, datasets can easily increase in complexity and quickly move out of reach for the average researcher to handle manually. The barrier between a researcher and robust documentation is one of time and detail.

Documentation will always need a human eye and attention, but a tool to create a template representing information that can be programmatically aggregated would allow the researcher to focus much needed efforts on the elements that truly require their personal knowledge. Tools like Yes Workflow tap into this philosophy, allowing the authors of scripts to add low-level markup within the code to programmatically produce a high-level description and visualization of the data process.

This project has a potentially unlimited scope when it comes to features. My goal with this project will be to produce a framework that could easily add in custom analysis for specific data types and file formats. The initial pass for this class would be something that could look over a set of CSV files, produce high level descriptions and aggregate statistics on that data set, and export out a SQL db file that could have queries run on it. This database could then also be run through a report system that automatically creates something like a rudimentary codebook for the researcher to review and fill in the contextual information that would be impossible to infer from the looking solely at the numbers.

A good example of this would be attempting to detect Likert scale questions. A detection algorithm would look for columns of data containing only integer values between 1 and 5, 6, or 7. Only these numbers would be seen, and we would expect to see all of them. However, a given survey, particularly with a low number of respondents, may only have values within limited clusters. This means that ranges on a given 1 to 7 Likert scale may only be observed between 2 and 4. Additionally, the test of 1 as an observed minimum could be true while the observed maximum in the dataset is incorrectly lower than what was offered on the instrument. Accompanying text matching these values may or may not be present within the data file. Certainly it would have been stripped and replaced by the respective integer value if the data file had been used for analysis. The tool could produce the core observed descriptive statistics of minimum, maximum, mean, standard deviation, and median. However, the researcher would need to use their knowledge of the survey instrument to complete the context of the actual responses and question text. Checking that Likert scale questions have correct balance is vital for appropriate analysis and reporting.

## The Approach

I will be using Python tools to create this tool, with a preference of standard library modules. However, as I’ve been exploring techniques, it appears that I cannot move away from using the Pandas module to arrange and output the data into a transportable format, such as a SQL database. I have also created a tool that will produce a set of CSV files with random values and lengths. This simulated data will be the files that I develop my script with. Again, the scope of the project is quite large and handling of special cases or messy data are features that could be added in at a later point.

To keep the scope in range of a single semester project my core plan is the following:

1. Create an input framework
2. Establish a rough function framework that can easily have analysis function added in
3. Establish a method of dumping the output into a SQL database for analysis and report generation
4. Develop a basic codebook generation example

## Input framework

An important part of the input framework is to have something to actually input. In order to test the many possible tools to add I wrote a small program to create a set of CSV files with random content. As a first pass, I wanted to handle purely numerical data over files with homogeneous columns. This would represent several runs of the same analysis on differing source files, similar to what may be created by brain measurement data or online survey data that is reported back using numerical IDs for the categories. Adding in support to detect and analyze textual data would be a second step.

My tool did successfully work on the dummy data, but I wanted to attempt and find some real data. I chose a dataset that I’ve worked with before, “Vagrant Lives: 14,789 Vagrants Processed by Middlesex County, 1777-1786 (version 1.1)”[[1]](#footnote-1). This is a dataset from crowd source transcriptions of arrest records from London. The dataset attempted to capture the geolocation and other demographic datapoints for people arrested as vagrants within London between 1777 and 1786. The single CSV file contains a mix of geolocations, text, and a bit of quantitative information.

## Function framework

One of the more complex issues to consider was how to handle the granularity of reporting. The file metadata makes the most sense to report at the level of folder collection, while the column level belongs at the reporting level of each file. The JSON file can easily contain this information, but JSON is not easily converted to a CSV, SQL table, or other easily analyzed format. As much as I hated doing it, I decided to make these be separate reports: one for file level metadata and separate column metadata reports for each file found. I will first focus on developing the tool to produce these individual reports. Moving forward, handling multiple files with the same headers and handling multiple files with differing headers will be future development goals.

## Dump out to SQL

This part got very complex very fast. In my minds eye I had hope that this would be quite simple. Just load up the data in to a pandas dataframe and dump it out to a db file. Pandas has a function called to\_sql(), so this certainly appeared promising. However, attempting to integrate it revealed that I’d need to install SQLAlchemy and a few other SQL packages for Python. The higher level goal for this project was to make something that researchers could use without much overhead, suddenly doubling the installation requirements moved this feature outside of scope. Instead, I will focus on just a CSV dump of the dataframe table and allow any users of this tool to import that into their database of choice. This file should be easily compatible with sqlite3 but also more manual inspection and manipulation within Excel. Having reviewed the possibilities of this, I actually prefer this method more because working knowledge of SQL and sqlite3 are not required.

## Codebook generation

Once the desire data was being generated I needed to choose how to output it into a codebook text file. I have considered a few things: output to an XML document for XSLT transformation, output as a pre-formatted HTML file, output to PDF, output as plain text, and output with markdown. A plain text file would be the most valuable from a preservation and machine-readable perspective. This decision is also mediated by human readability.

1. https://zenodo.org/record/31026?ln=en#.Vx1IrZMrJXg [↑](#footnote-ref-1)