Digital Humanities and Data

Francesca Giannetti

2022-10-19

Abstract

Is there a humanist way of working with data (cf. Posner, "Data Trouble")? This workshop prods at that question, examining tensions with quantification and categorization, and moving towards (we hope) repair and remediation. A hands-on portion will look at concrete ways to represent research topics as tabular data (aka a spreadsheet), one of the handiest and most portable of data formats.

Introduction

This workshop introduces the challenges and benefits of translating humanities sources into machine-accessible formats. It also provides a gentle introduction to programming in R using the RStudio environment.

Take a look around: you're in RStudio, an environment that makes coding in R more transparent. Your window is divided into four segments. In the upper right, you'll see the environment: this displays all the objects you have loaded into memory (currently none). In the upper left, you'll see the script editor: this is the place to work on code that you're currently writing (or borrowing from elsewhere!). To run code in code chunks (the grey chunks), you can either press Ctrl+Enter to run single lines or click the green arrow to run the entire chunk. In the lower left, you'll see the console: this is where code actually executes and where the output prints. In the lower right, you'll see a few different things. The "Files" tab shows whatever files live in the directory (or folder) that R is currently working in; if you run any plots, they'll show up in the "Plots" tab; you can also get help in the "Help" tab.

To start, if you're using your own computer, run the section of setup code above by clicking the green arrow in the upper right of the grey box.

Humanities Data Tensions

In her talk, "Data Trouble," Posner enumerates some of the reasons behind the data negativity in some humanities circles. It goes beyond a fondness for close reading and an aversion to positivism. Those reasons fall under the sign of categorization. E.g.:

1. demarcation
2. parameterization
3. ontological stability
4. replicability
5. boundedness
6. deracination

As an example of the first, Posner discusses the culturomics authors' paper, specifically the visualization on inventions and their adoption rate.

Demarcation – the separation of research observations into discrete items in order to perform operations on them.

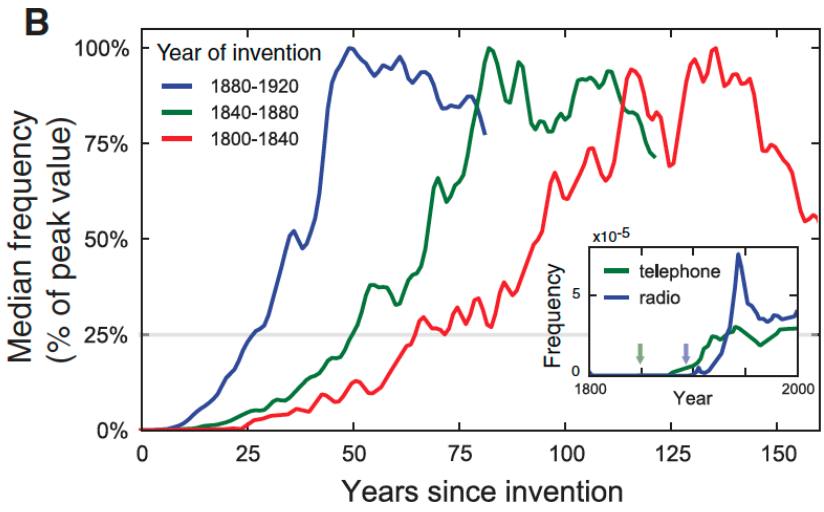

Figure 1: A figure from "Quantitative Analysis of Culture. . . "

In the above figure, the authors see proof that more recent cohorts of inventions were adopted more rapidly than previous cohorts. The visualization suggests that each invention cohort appeared like a bolt out of the blue. Humanists, on the other hand, tend to see inventions emerging from constellations of forces, integrating previous technologies in the succeeding technology. Not separate, in other words. And yet, separation is a necessary precursor to data manipulation and analysis. The computer can't do much with an undifferentiated mass of information.

Parameterization – a standard scale of measurement to facilitate comparisons

Some things lend themselves well to parameterization: word counts, canvas size, the length of a song, or the shape of a pot. But within cultural artifacts, many features that are legible to human perception are difficult to parameterize, e.g. tone of voice, posture, facial expression. And some things that seem simple, e.g. how far it is from London to Edinburgh, change considerably depending on the historical period, mode of transport, class status, gender, weather, and so on.

Ontological stability – how we organize entities to make a data model

Ontologies get at the way in which one sees the world. To humanists, perspective changes things.

replicability – the benefit of all well-structured/described data is reuse and replicability, but is that a thing in the humanities?

As Posner suggests, the interpretation of the data IS the work, and it would be foreign to most humanists to suggest that any other humanist reach the same conclusions.

boundedness – all historical records are partial, therefore no historical dataset is representative?

We have a fairly limited vocabulary for referencing gaps or absences in our data, but there are some strategies.

deracination – when creating data, some context is inevitably removed

It's not possible to capture every aspect of a given source. But fortunately, that is okay. No humanist would assert that a digitized book is the same as the print book used as its source. They both accomplish different purposes, and have different affordances and drawbacks.

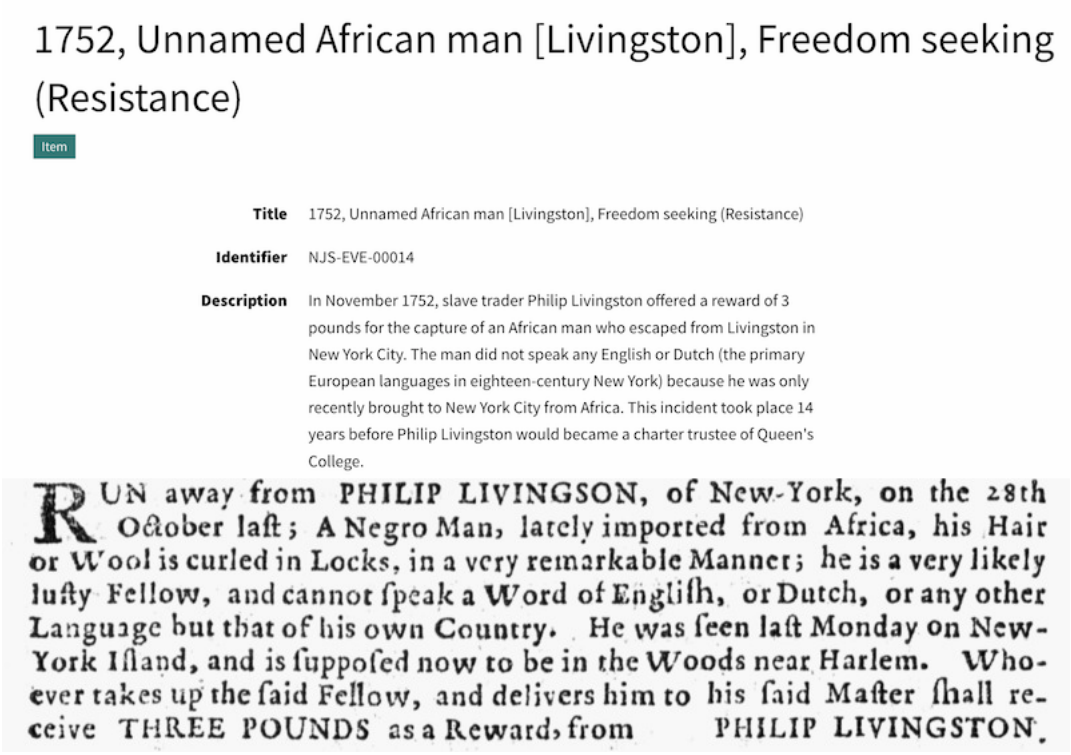

Figure 2: A runaway slave ad compared to its representation in the NJ Slavery database

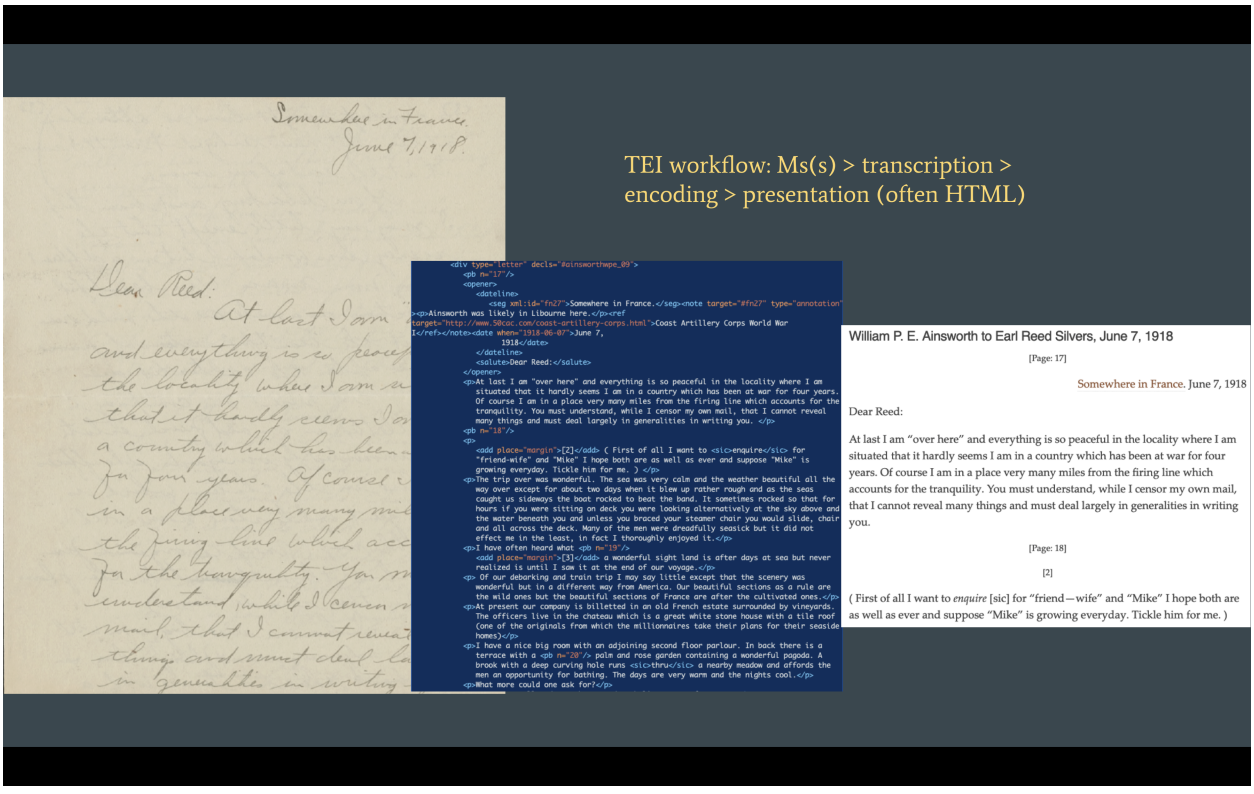

Figure 3: A letter from the Rutgers College War Service Bureau Records and its representations as TEI-XML and HTML

Data from Archival Sources

Some archival sources can be readily understood as data. Take, for example, this antebellum American newspaper's subscription book. This is already a technology for organizing and storing data; we simply want to transcribe it so that we can represent and analyze it more effectively. Depending on one's research, one may want to quantify subscribers by gender, map their locations, or calculate turnover.

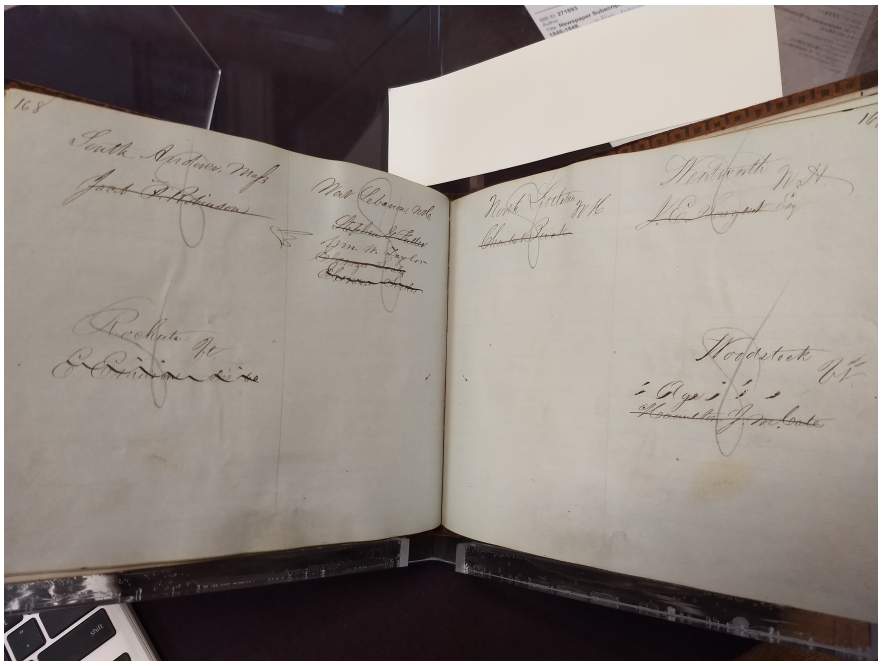

Figure 4: A page from a subscription book

One of the simplest formats for representing multidimensional data is a spreadsheet, whether in Microsoft Excel, Google Sheets, or whichever other program you prefer. In R, data in this format is called a *data frame*.

```
names <- c("Jacob Robinson", "Stephen Fuller", "William Taylor", "Ebenezer Seales", "Ebenezer Seales",
genders <- c("M", "M", "M", "M", "M", "M", "M", "M", "NA", "M")
towns <- c("South Andover", "West Lebanon", "West Lebanon", "West Lebanon", "West Lebanon", "Rochute",
states <- c("MA", "NH", "NH", "NH", "NH", "VT", "NH", "NH", "VT", "VT")
data.frame(Name=names, Gender=genders, Town=towns, State=states)
```

```
##               Name Gender            Town State
## 1   Jacob Robinson      M   South Andover    MA
## 2   Stephen Fuller      M    West Lebanon    NH
## 3   William Taylor      M    West Lebanon    NH
## 4  Ebenezer Seales      M    West Lebanon    NH
## 5  Ebenezer Seales      M    West Lebanon    NH
## 6        E Emerson      M         Rochute    VT
## 7     Charles Buck      M North Littleton    NH
## 8        J Sargent      M       Wentworth    NH
## 9              Age     NA       Woodstock    VT
## 10   Hamilton Cate      M       Woodstock    VT
```

The basic principle of organizing data into this format is that it is highly structured. In order to take full advantage of this capacity, three conditions should be met:

- Each variable must have its own column

- Each observation must have its own row
- Each value must have its own cell

Getting data into this format will ensure that any future quantitative analysis moves smoothly. It's most efficient to do so from the very start.

It's possible that you may need to combine variables, or columns, later, but with computation it's very easy to combine variables, as with the `paste`. `paste` wants three inputs: the two objects being pasted together and the method (in this case, separating with a comma and a space).

```
paste(towns, states, sep=", ")
```

```
##  [1] "South Andover, MA"   "West Lebanon, NH"    "West Lebanon, NH"
##  [4] "West Lebanon, NH"    "West Lebanon, NH"    "Rochute, VT"
##  [7] "North Littleton, NH" "Wentworth, NH"       "Woodstock, VT"
## [10] "Woodstock, VT"
```

Spreadsheet editors tend to have easy merging commands as well. In Google Sheets, for example, it's `=textjoin(", ", 1, K1, J1)` (replacing "K1" and "J1" with whichever cells are to be merged).

It's still possible to separate out one variable into two, but irregularities in your data can cause errors more readily. Here is a line of code that separates out first names, but if I had included periods after first initials ("J." instead of "J") it would require a bit more tweaking.

```
gsub("(^\\w+) \\w+$", "\\1", names)
```

```
##  [1] "Jacob"    "Stephen"  "William"  "Ebenezer" "Ebenezer" "E"
##  [7] "Charles"  "J"        "Age"      "Hamilton"
```

When inheriting spreadsheets from someone or somewhere else, these are two of the most common operations necessary for cleaning before analysis can begin. But there are more powerful functions that can be carried out in an instant in a programming language like R that are much trickier otherwise. Transposing a spreadsheet, flipping the rows and columns, just requires one `t` function; separating out alternating values in one column and spread across two columns can be done with `spread`.

But there's other data in our archival document too. For example, some entries aren't individual subscribers: they're other newspapers. When transcribing, we tend to focus on the features in which we are interested for our specific research purpose. But sometimes we transcribe something even when we don't know if we'll need it later. In this case, we've decided to do something with those newspaper titles. We will convert them into categorical variables to track their presence or absence.

```
paper <- c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
data.frame(Name=names, Gender=genders, Town=towns, State=states, Paper=paper)
```

```
##               Name Gender            Town State Paper
## 1   Jacob Robinson      M   South Andover    MA     0
## 2   Stephen Fuller      M    West Lebanon    NH     0
## 3   William Taylor      M    West Lebanon    NH     0
## 4  Ebenezer Seales      M    West Lebanon    NH     0
## 5  Ebenezer Seales      M    West Lebanon    NH     0
## 6        E Emerson      M         Rochute    VT     0
## 7     Charles Buck      M North Littleton    NH     0
## 8        J Sargent      M       Wentworth    NH     0
## 9              Age     NA       Woodstock    VT     1
## 10   Hamilton Cate      M       Woodstock    VT     0
```

There might be further implicit or noncharacter information contained in archival materials. This might include marginalia in printed documents or physical characteristics of materials. Some names are crossed out here; while it's impossible to know when they were crossed out, we know that their subscriptions ended

before the life of this ledger did. This gives us a snapshot of a single moment of the data, which otherwise spans a couple years, and could help us better understand the whole.

```
cancel <- c(1, 1, 0, 1, 1, 1, 1, 1, 0, 1)
ledger <- data.frame(Name=names, Gender=genders, Town=towns, State=states,
                     Paper=paper, Cancel=cancel)
ledger
```

```
##               Name Gender            Town State Paper Cancel
## 1   Jacob Robinson      M   South Andover    MA     0      1
## 2   Stephen Fuller      M    West Lebanon    NH     0      1
## 3   William Taylor      M    West Lebanon    NH     0      0
## 4  Ebenezer Seales      M    West Lebanon    NH     0      1
## 5  Ebenezer Seales      M    West Lebanon    NH     0      1
## 6        E Emerson      M         Rochute    VT     0      1
## 7     Charles Buck      M North Littleton    NH     0      1
## 8        J Sargent      M       Wentworth    NH     0      1
## 9              Age     NA       Woodstock    VT     1      0
## 10   Hamilton Cate      M       Woodstock    VT     0      1
```

Data frames can be indexed by individual cell, with a comma (`,`):

```
ledger[1,1]
```

```
## [1] "Jacob Robinson"
```

By row:

```
ledger[1,]
```

```
##             Name Gender          Town State Paper Cancel
## 1 Jacob Robinson      M South Andover    MA     0      1
```

Or by column, with a dollar sign (`$`):

```
ledger$Name
```

```
##  [1] "Jacob Robinson"  "Stephen Fuller"  "William Taylor"  "Ebenezer Seales"
##  [5] "Ebenezer Seales" "E Emerson"       "Charles Buck"    "J Sargent"
##  [9] "Age"             "Hamilton Cate"
```

But the real beauty of indexing is that we don't need to know the positions already. We can index the result of another function. The `which` function tells us which elements of a vector meet a particular condition.

```
which(ledger$Cancel < 1)
```

```
## [1] 3 9
```

Indexing these positions back into the ledger data frame as rows will give us the corresponding values.

```
ledger[which(ledger$Cancel < 1),]
```

```
##             Name Gender         Town State Paper Cancel
## 3 William Taylor      M West Lebanon    NH     0      0
## 9            Age     NA    Woodstock    VT     1      0
```

By calculating the `length` of our results - the number of elements in the vector produced by a function like `which` - we can do some basic arithmetic. What proportion of entries in the ledger were active subscribers upon its retirement?

```
length(which(ledger$Cancel < 1)) / length(which(ledger$Cancel==1))
```

```
## [1] 0.25
```

This is just the beginning, but it's enough to understand the basic principles at play here and, hopefully, their utility.

A note about file formats: a software agnostic, size-efficient format for spreadsheet data is a .csv file. This isn't the default for programs like Excel, but you can still open, edit, and save .csv files in Excel or Sheets, and if you start in one of those programs before shifting over to a programming language like R you'll need to save your spreadsheet as a .csv first (an option in the "Save as" menu). Writing to drive and reading from drive are each a simple function.

```
write.csv(ledger, "ledger.csv")

ledger <- read.csv("ledger.csv")
```

Using Digitized Texts

There are a number sources of digitized texts online. The most reliable of these is Project Gutenberg, which contains transcriptions of some 60,000 books, fictional and nonfictional.

The Gutenberg page for any given book, will list several format options: the most amiable is the "Plain Text UTF-8" format. Copying and pasting this into a plain notepad .txt document (not a Word .docx document) is all that's necessary to begin working with this text as data. Like a .csv file, this is the simplest and smallest format for text data, which makes it the most reliable to work with. And again, any text file in, say, Microsoft Word can be turned into a .txt format in the "Save as" menu.

Even more easily, you might scrape this text directly off the web and into an object in your computer's memory with `readLines`. Here is *The Theory of the Leisure Class* (1899) by economist and sociologist Thorstein Veblen.

```
book <- readLines("http://www.gutenberg.org/cache/epub/833/pg833.txt")
```

Type "book" into your console and click enter. If you scroll up to the top, you'll see a bunch of stuff that isn't part of the book proper: website metadata, text metadata, an introduction. We'll want to trim that off.

If you went to the webpage we got it from, you could see that the book begins with the line "THE THEORY OF THE LEISURE CLASS" and ends with the line "use and the need of direct and forcible speech." If we search the book object for elements containing those exact character strings, we'll find the exact position for the start and the end of our desired text. The `grep` function does this. I've included a `^` and a `$` because `grep` is otherwise inclusive and will return any longer character strings that happen to include our desired string as well: these serve as metacharacters specifying a hard start and a hard end.

```
start_line <- grep("^THE THEORY OF THE LEISURE CLASS$", book)
end_line <- grep("^use and the need of direct and forcible speech.$", book)

book <- book[start_line:end_line]
```

What exactly is the proper unit of data here? That depends on the question at hand. We could leave it in lines, as it currently stands, which would tell us about the format (in this case the Gutenberg format, but in the next example it would tell us about the print format). So let's remove the line breaks by collapsing all the elements of our book vector together into one big character string with `paste`.

```
book <- paste(book, collapse=" ")
```

We could break this book down to individual characters. There probably isn't much we can say about characters like "e" or "f", but we might learn something from the distribution of "?" or "!" characters. Most likely, though, the most useful unit of data is the word. So we'll split up the book string with `strsplit`,

with a break every time there's a non-letter character. Of course, there isn't a character on the keyboard for "non-letter character." To express this regular yet inexact pattern, we'll need what's called a *regular expression*. `\\w` denotes any single letter; `\\W` denotes any single non-letter; `+` denotes one or more of whatever precedes it. Thus we get:

```
book_words <- unlist(strsplit(book, "\\W+"))

book_words
```

We could just as easily split our text up into sentences. The only thing to change would be the regular expression, which now includes all the characters that signal the end of a sentence as triggers for splitting the character string. A period needs to be designated as `\\.` because it is a regular expression metacharacter (namely, any character whatsoever). Before doing so, it would be necessary to substitute out periods that don't end sentences. This might understandably get tricky, and will probably require lots of circular data work. "Mr." and "Mrs." are obvious places to start. Where else?

Exercises

Exercise #1: NJ recommendations

Let's take a few minutes as a group to add our personal recommendations of places to visit in New Jersey. These can be restaurants, places of historical interest, hiking trails, bodies of water, museums, etc., well-known or otherwise. Go to https://docs.google.com/spreadsheets/d/1P8xWaqCpGYJJj5cRCvPRTA9HQKEjrBnWBKRSunmWLlA/edit?usp=sharing and add yours. Notice that I've made some assumptions about the useful categories for our data.

Next, we'll load our data into R in order to visualize it as a map. An important intermediate step to visualization is geocoding, or adding latitude and longitude coordinates to our `city` column.

I'm making the assumption that a map will help us to browse/query our data. Here, we place markers on a map and add popups with some data from our Google sheet.

Questions:

1. In what way is a map useful, or not? What is lost and gained with this representation of our data?
2. Can you imagine alternative ways of visualizing our data that would tell a different story or suggest another interpretation?

Exercise #2: Creating a data model

What kind of data model could a scholar working with rare book auction catalogs create for their research project? What categories would this person need? Essential ones? Useful ones? Some catalogs to consider https://www.rarebookhub.com/catalogues/.

Sources

Drucker, Johanna. 2021. *The Digital Humanities Coursebook: An Introduction to Digital Methods for Research and Scholarship*. Milton, UK: Taylor & Francis.

Leslie, Alex. 2020. "What can be data in the humanities?" https://github.com/azleslie/DatainHumanities.

Michel, Jean-Baptiste, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, Joseph P. Pickett, Dale Hoiberg, et al. 2011. "Quantitative Analysis of Culture Using Millions of Digitized Books." *Science* (American Association for the Advancement of Science) 331 (6014): 176–82. https://doi.org/10.1126/science.1199644.

Posner, Miriam. 2022. "Data Trouble." Open Data: Reuse, Redistribution, and Risk, American Philosophical Society, Philadelphia, PA. https://www.youtube.com/watch?v=ggGXKSK-UP8.