EDA for Humanities Data

awk.dev/eda.html

Exploratory Data Analysis for Humanities Data

Circular reference: there's a *Hacker News* thread on this essay here.

Introduction

In the spring of 2023, I co-taught a course called *Literature as Data* with my friend and colleague Meredith Martin, professor of English at Princeton, and director of the Center for Digital Humanities.
The course was very much an experiment, an attempt to combine literary study with computing. The 17 surviving students were largely sophomores. About two thirds of them were hard-core humanities majors; the rest were potential or actual CS majors or other STEM concentrators. The class met once a week for three hours, 12 weeks in all.

One of the goals of the course was to try to teach enough computing to a mostly non-technical and definitely not computer-experienced population that they could use computers to do an interesting and new (to them) exploration of some dataset that they found intriguing.

After much discussion, Meredith and I decided that for the programming aspects, we would devote half of each class meeting to a "studio" where I would lead the students through hands-on computing exercises on some small dataset, followed by having the students do similar exercises on larger datasets outside the class.

We planned to spend one week on Unix file system and command-line basics, a week on grep and similar tools (including a real text editor), a week on Awk, and a couple of weeks on Python. Not surprisingly, everything took longer than expected, and a fair number of students struggled mightily in spite of help from their peers, Meredith and myself, and two exceptional colleagues from the CDH, Sierra Eckert and Ryan Heuser.

More about the course can be found elsewhere (at least when we get around to writing it). The course had a web site at hum307.com, though after a while we stopped using that in favor of Google Docs -- information was spread out over too many sites, and in addition we had to limit access since some of our datasets were not public.

Awk vs Pandas

Arguably, most humanities computing uses Python and the Pandas library to analyze structured data that often began life as a CSV file. If you're looking for a single language solution, this is an excellent way to go. But in my experience, admittedly biased, Unix command-line tools and Awk can be especially useful in the initial stages of exploring a new dataset. They are better at counting things, looking for anomalies and outliers, and coping with data that isn't quite in the right format. And, again a biased view, looping over a set of input lines seems more natural than the dataframe selectors that Pandas favors.

Awk for EDA

The purpose of this essay is to talk about how we used Awk in a different EDA setting from the examples in the book. In keeping with the "literature" theme of the course, in the first few weeks we asked students to look at poetry, specifically the *Sonnets from the Portuguese* by Elizabeth Barrett Browning, and Shakespeare's sonnets. We used grep and wc to count things like the number of times specific words appeared, and also to observe something unexpected: even though sonnets always have 14 lines, Shakespeare's Sonnet 126 has only 12 lines, and Sonnet 99 has 15 lines. This was a good lesson about literature but also about how real data does not always behave as expected.

This essay parallels the EDA discussion in Chapter 3 of the new Awk book, but using different data, a metadata file about 18-th century sonnets originally curated by Professor Mark Algee-Hewitt of the Stanford Literary Lab. We are very grateful to Mark for kindly allowing us to use this metadata for the class and for this page.

What we did in class and the outside class exercises can be found at `hum307.com`.

The metadata file is 18.csv. The name is not strictly accurate: we converted the original CSV into a file with fields separated by slashes so that it would work with any version of Awk. We also removed some attributes from Mark's original dataset, leaving us with 7 columns.

```
$ wc 18.csv
     508    3499   36876 18.csv
$ sed 1q 18.csv
author_dob/author_dod/author_fname/author_lname/num_lines/title_main/title_sub
$
```

Using the head-tail code of Section 2.2 of the book, we can look at the start and end of the contents:

```
$ headtail 18.csv
author_dob/author_dod/author_fname/author_lname/num_lines/title_main/title_sub
1771/1798/E. H. (Elihu Hubbard)/Smith/8/SONNET I./Sent to Miss  , with a Braid of
Hair.
1771/1798/E. H. (Elihu Hubbard)/Smith/8/SONNET II./Sent to Mrs.  , with a Song.
...
1740/1809/Hugh/Downman/14/SONNET II./[Though here almost eternal Winter reigns]
1740/1809/Hugh/Downman/14/SONNET III./[When Recollection stirs up in the mind]
1740/1809/Hugh/Downman/14/SONNET IV./[Now is the feudal vassalage destroy'd]
$
```

Notice that the first sonnets only have 8 lines, echoing the observation above.

Validation

The first step is data validation: does the data satisfy basic criteria like the right number of fields in each record. In a separate command, we can look at the distribution of number of lines in each sonnet:

```
$ awk -F/ '{print NF}' 18.csv | sort -u
7
$ awk -F/ '{print $5}' 18.csv | sort | uniq -c | sort -nr
 483 14
  11 8
   3 15
   2 12
   1 num_lines
   1 68
   1 66
   1 55
   1 54
   1 40
   1 28
   1 24
   1 123
$
```

This is a bit surprising: although 95% of the sonnets have the standard 14 lines, 8 lines is not uncommon and some are remarkably long. I'm no scholar, so I don't know what makes these "sonnets" as opposed to some other poetic form.

Maybe printing something about the longer ones might provide more insight? We'll exclude the header line too.

```
$ awk -F/ 'NF > 1 && $5 > 14' 18.csv
1729/1777/William/Dodd/15/SONNET./OCCASIONED BY HEARING A YOUNG LADY SING
SPENSER'S AMORETTI, &c. SET TO MUSIC BY DR. GREENE.
1749/1806/Charlotte Turner/Smith/40/ODE TO DESPAIR./FROM THE NOVEL OF EMMELINE.
1749/1806/Charlotte Turner/Smith/68/ELEGY./[‘Dark gathering clouds involve the
threatening skies]
1749/1806/Charlotte Turner/Smith/24/SONG./FROM THE FRENCH OF CARDINAL BERNIS.
1749/1806/Charlotte Turner/Smith/123/THE ORIGIN OF FLATTERY./
1749/1806/Charlotte Turner/Smith/54/THE PEASANT OF THE ALPS./FROM THE NOVEL OF
CELESTINA.
1749/1806/Charlotte Turner/Smith/55/THIRTY-EIGHT./ADDRESSED TO MRS. HY.
1749/1806/Charlotte Turner/Smith/28/VERSES/INTENDED TO HAVE BEEN PREFIXED TO THE
NOVEL OF EMMELINE, BUT THEN SUPPRESSED.
_/_/G./Bent/66/To the SAME,/On receiving his Poems to Thespia with a Sonnet
prefixed.
1735/1779/John/Langhorne/15/SONNET CLXXIX./
1735/1788/William Julius/Mickle/15/SONNET TO VASCO DE GAMA: FROM TASSO./
```

This output suggests a variety of other questions and potential issues. Charlotte Turner Smith wrote the majority of the long sonnets, including the extreme outlier with 123 lines. How many did she write overall?

```
$ grep Charlotte.Turner 18.csv | wc
     101     941    8955
$
```

She certainly was prolific, representing nearly 20% of the data. This raises some other questions for scholars, in particular, how were these sonnets selected and how representative are they?

Here are some commands to investigate the authors further. For example, if we do a straightforward display of unique author names:

```
$ awk -F/ '{print $3, $4}' 18.csv | uniq | sed 10q  # display the first 10 lines
author_fname author_lname
E. H. (Elihu Hubbard) Smith
M. F. Cogswell
E. H. (Elihu Hubbard) Smith
M. F. Cogswell
John Bampfylde
Thomas Edwards
William Cowper
William Dodd
Thomas Edwards
```

we see that records for at least three authors are not contiguous, as we might naively have expected. This raises a new question: what is the order of the records?
How many unique authors are there, and how much did they write?

```
$ awk -F/ '{print $4 ", " $3}' 18.csv | sort | uniq -c | sort -n
   1 Anon., _
   1 Anstey, Christopher
   1 Bent, G.
   1 Bishop, Samuel
   1 Bradford, A. M.
     ...
  23 Russell, Thomas
  38 Downman, Hugh
  50 Edwards, Thomas
 101 Smith, Charlotte Turner
 103 Seward, Anna
```

There are 42 distinct authors; the display above shows a few of the 14 singletons, and the five most prolific. Interestingly, the top two are women. Guessing from names, there are only a handful of other female authors. One might wonder why over 40% of the sonnets are by two women. How the data was created is a very important consideration for any dataset, and particularly for data in the humanities. As Meredith puts it, **there is no such thing as raw data**; all datasets are the result of a selection and curation process. Historically, this has often been to the detriment of some classes of authors.

As a peripheral question, how many lines did Anna Seward and Charlotte Turner Smith write?

```
awk -F/ '/Anna/ { anna += $5 }; /Charlotte/ { char += $5 }; END {print anna,
char}' 18.csv
1456 1706
```

Seward wrote fewer lines because all her sonnets were 14 lines long. (Though, as Tim Taylor points out, there is one sonnet by Anna Williams that get swept up by this test.)

When did they live and die?

The first two fields are the birth and death dates for each author, so we can explore questions about the ages of the authors. The age is just `$2-$1`:

```
$ awk -F/ '{age[$3 " " $4] = $2 - $1}
    END {for (i in age) print age[i], i}' 18.csv | sort -n
-1807 M. F. Cogswell
0 A. M. Bradford
0 G. Bent
0 J. Cole
0 R. Hole
0 _ Anon.
0 author_fname author_lname
23 Henry Headley
...
79 John, Mrs. Hunter
81 Christopher Anstey
81 Thomas James Mathias
83 Anne MacVicar Grant
84 William Crowe
88 William Lisle Bowles
```

This tells us pretty clearly that the dates need to be studied further, most easily by looking for the ages that are zero or negative:

```
$ awk -F/ '$2-$1 <= 0' 18.csv
1807fl.//M. F./Cogswell/8/SONNET,/Written after hearing a SONG sung by several
SISTERS.
1807fl.//M. F./Cogswell/8/THE SMILE./SONNET TO CAROLINE.
_/_/_/Anon./14/SONNET./
_/_/A. M./Bradford/14/To the SAME./
_/_/J./Cole/14/To the SAME./
_/_/G./Bent/66/To the SAME,/On receiving his Poems to Thespia with a Sonnet
prefixed.
_/_/R./Hole/14/To the SAME,/On his Poems addressed to Thespia.
```

What do we do about entries like R. Hole and the often prolific Anon, whose date fields are marked with _?

What's with M. F. Cogswell? He flourished around 1807, with two 8-line sonnets. Actually, this entry shows one way in which Awk differs from Python, often usefully. When converting an arbitrary string to a number, Awk uses the longest prefix that looks like a number, so `1807fl.` has numeric value 1807; Awk processes that and carries on. Python, by contrast, will throw an exception that, unless explicitly caught, will terminate execution.

Neither of these deals with the real issue, which is that the data is incomplete so an age computation isn't possible. One simple solution, appropriate when unknown dates are a small fraction of the data, is to ignore those records. Regular expressions are the easiest way to select the good bits or reject the bad ones:

```
$ awk -F/ '$1 ~ /^[0-9]+$/ && $2 ~ /^[0-9]+$/ {print $2-$1, $3, $4}' 18.csv | uniq
| sort -n | uniq
23 Henry Headley
25 Richard Gall
26 Thomas Russell
27 E. H. (Elihu Hubbard) Smith
37 Robert Burns
...
79 John, Mrs. Hunter
81 Christopher Anstey
83 Anne MacVicar Grant
84 William Crowe
88 William Lisle Bowles
```

Rather than repeating the test over and over again, we could collect all the lines that have valid dates in a new temporary file, then do some further computations on that.

```
$ awk -F/ '$1 ~ /^[0-9]+$/ && $2 ~ /^[0-9]+$/' 18.csv >temp
$ wc temp
     486    3350   35265 temp
```

After that, we can compute the average age, and determine the youngest and oldest.

```
$ awk -F/ '{ ages = ages + $2 - $1 } # add up all the ages
       END { print "average age =", ages / NR }' temp
average age = 60.2942
$ awk -F/ '$2-$1 > max { max = $2 - $1; fname = $3; lname = $4 }
       END { print "oldest:", fname, lname, " age", max }' temp
oldest: William Lisle Bowles  age 88
```

As noted above, the author names are not contiguous, and the dates are not in any obvious order either, which makes one wonder what order the data has been sorted into. How would you detect such anomalies mechanically in a larger dataset, in particular if it were not related to dates)?

Associative Arrays

None of the examples so far have used associative arrays, which are Awk's only data structure. We didn't spend much if any time on associative arrays in the class, since it felt like a slightly too-advanced topic for our population.
An associative array, equivalent to a hash in Java or a dictionary in Python, is an array whose subscripts are not integers but arbitrary strings. Several of the examples above that used `sort|uniq -c` can be written equivalently with associative arrays to bring together identical items. For example, to see the distribution of sonnet lengths, we can write:

```
# awk -F/ '{print $5}' 18.csv | sort | uniq -c | sort -nr

$ awk -F/ '{ len[$5]++ }; END { for (i in len) print len[i], i }' 18.csv | sort -
nr
```

This produces the same result as the previous version. We could even embed the sort command in the Awk program:

```
$ awk -F/ '{ len[$5]++ }; END { for (i in len) print len[i], i | "sort -nr" }'
18.csv
```

This computation is essentially the same as the Pandas `unique_id` dataframe function.

Wrap-up

Awk is good for the kind of preliminary analysis we've shown here, particularly when coupled with basic Unix tools like `sort`, `uniq`, and `wc`. At some point it's time to switch to Python and some of its libraries, especially for plotting, but that can be deferred until you really understand what's in the data and where it might be flaky. null