Data Cleaning, Scraping, and Keyboard Macros

a topic in

DM565 – Formal Languages and Data Processing

Kim Skak Larsen

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark (SDU)

kslarsen@imada.sdu.dk

November, 2019

Data Cleaning

Data Cleaning

SDU

We discuss frequently occurring problems in tabular data.

Realize that there are errors in most data – usually lots of errors – due to programming problems, data transfer/format problems, or data entered by many different people, most of which are incompetent or do not care.

Your command-line toolbox can be very helpful in this process.

Before really starting,

- check character encoding so you know what it is and can treat it correctly in your tools – and/or recode to your favorite, and
- cut down on the noise by getting rid of columns and rows you do not need – bigger problems if there are missing column separators!

Data Cleaning: Format Examples

SDU

Perform a Manual Inspection of Column Types

Not just `string` or `int`, but more narrowly,

- names (person, city, product)
- zip codes
- Y/N entries or other true/false equivalents
- email addresses
- dates

Data Cleaning Format Examples

For each column, perform type checking as accurately as possible, i.e.,

- a name is a string, but more precisely a string without digits and (most) special characters with a certain capitalization pattern; there may be downloadable lists of given names, city names, etc.
- a zip code is a string (or integer), but more precisely exactly four digits (if Danish); in fact you can download a file from postnord with all existing zip codes
- for Y/N like entries, check if there is anything else
- email addresses have well-defined formats they have to adhere to
- for fields of potentially unlimited length, consider finding the longest

Data Cleaning: Small Domains

SDU

For small domains,

- collect all values in the *i*th field (`cut -fi` or `gawk '{print $i}'` – new delimiter ex. `-d ","` or `-F ","`, respectively),
- sort (can specify sorting type),
- remove duplicates (`uniq`), and
- inspect the result.

For larger domains,

- count the number of occurrences of each value, and
- inspect the histogram profile (possibly graphically),
- with special focus on values occurring seldomly or very frequently.

For columns appearing to be unique identifiers, check for duplicates!
(As a first check, see if the file size changes after running `uniq`.)

Data Cleaning: Missing Values

In the process above, you may have found missing values; especially blanks. However, be aware of common "missing value" notation such as

- – (or more)
- ? (or more)
- NA, N/A (not applicable)
- NaN (not a number)
- None, null, nil, void, etc.

– with or without capitalization, but unfortunately also "will be provided later", "currently missing", and lots of other options.

Data Cleaning: Fixes and Accepts

- It may be clear that a blank in a Y/N column is an N and not a missing value.
- It may be clear that 1/1 70 is a date and should be replaced with 1/1 1970; or maybe more appropriately with 1970-01-01.
- We wanted full names, but maybe we can live with Kim S. Larsen, which is clearly not a full name.
- We may be able to deduce the full email address of `kslarsen@imada`.
- City names not appearing in the official list can maybe be changed to the nearest match; spelling-correction style.
- Zip code/city name inconsistencies may be fixable because of the redundant information.
- Values may be logically deducible; if we collected survey information and a couple had 2 children in 2015 and 2 children in 2017, a missing number of children in 2016 is likely correctable.

Data Cleaning: Replacement

Some missing data is unfixable – how can we get something we can live with?

- Missing street number – there is no reasonable way of guessing: can we live without it, maybe use additional databases, or just delete the row?
- Missing number of kids - it is tempting to use the average, but listing 3.14 kids may cause problems later on. For some purposes, making up a believable number (the median, for instance) may work (*imputation*).
- Missing distance from home to birth place - again, the average might be tempting, but it is often a bad idea. Ranging over a few people, where most live close to their birth place, but one is from another continent gives an average that places the person in one of the oceans! Using the median may again be preferable.
- One may want to consider using regression analysis to impute the most reasonable values.
- Be careful if data is not missing at random, e.g., people are less likely to deliver possibly embarrassing information such as voting for an extreme party, having an extreme income, etc.

Data Cleaning Tools

There are tools worth considering if problems are even worse or you have to do this frequently, e.g.,

- pandas, for Python
- tidyverse, for R

Web Scraping

Web Scraping

SDU

Web scraping (web harvesting, web data extraction) is about extracting data from a web page that is not yours.

You may want to

- scrape information from a lot of pages, or
- scrape information from the same page repeatedly (because it updates).

The former is often referred to as *web crawling*, and since many pages are involved and formats are unknown, it is hard to extract exact information other than keywords, links, modification times, etc.

Focusing on scraping from the same page(s) repeatedly, check if the home page provides an API (most do not); otherwise, you could use tools you have already seen.

Web Scraping by Searching

SDU

Using tools you already have, you could

- download the html page,
- find the interesting information by searching,
- identify the textual surroundings, and
- create regular expression searches for extracting exactly the information you want, possibly via a programming language and its built-in searching facilities.

To get the page the first time, you could, for example, choose "More tools" $\rightarrow$ "Save page as..." in Google Chrome; when you automatize, you may want to use GNU `wget`, if you use command-line tools, or the appropriate package/library from your programming language.

Note that `wget` will get the raw page, whereas browsers may change the content some before saving.

Web Scraping Tools

SDU

If you want to extract a lot information from a page or you want to be immune to minor textual changes, you can use a tool.

html is much like xml, except there is some sloppiness, especially regarding closing the parenthesis-like structures.

Thus, it is a tree where nodes can be annotated and have a varying number of children.

Each programming language offers its own tools; one such tool for Python is *Beautiful Soup*.

Beautiful Soup

Beautiful Soup needs a parser for html; we prefer `lxml`.

You can obtain a page to be used as input for BeautifulSoup by

```
import requests
from bs4 import BeautifulSoup

page = requests.get("https://imada.sdu.dk/~kslarsen/dm565/notes.php")

soup = BeautifulSoup(page.content, "lxml") # "html.parser" can also be used
```

In the rest, for speed and reproducibility, we will just write example data directly in the Python program.

Beautiful Soup and lxml

`lxml` tries to fix errors and return a nicely structured document; other parsers work similarly, but may produce different trees.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup("Kim Skak Larsen", "lxml")
print(soup)
# <html><body><p>Kim Skak Larsen</p></body></html>

soup = BeautifulSoup("<ul><li>First<li>Second</ul>", "lxml")
print(soup)
# <html><body><ul><li>First</li><li>Second</li></ul></body></html>
```

Beautiful Soup: Example Features

SDU

```
from bs4 import BeautifulSoup

soup = BeautifulSoup("<ul><li>First<li>Second</ul>", "lxml")
print(soup)
# <html><body><ul><li>First</li><li>Second</li></ul></body></html>

print(soup.body.ul)
# <ul><li>First</li><li>Second</li></ul>

print(soup.ul)
# <ul><li>First</li><li>Second</li></ul>

print(soup.li)
# <li>First</li>

print(soup.find_all('li'))
# [<li>First</li>, <li>Second</li>]

for child in soup.ul:
    print(child)
# <li>First</li>
# <li>Second</li>
```

Beautiful Soup: Example Features (continued)

```
soup = BeautifulSoup("<ul><li>First<li>Second</ul>", "lxml")
print(soup)
# <html><body><ul><li>First</li><li>Second</li></ul></body></html>

print(soup.li.next_sibling)
# <li>Second</li>

print(soup.li.next_sibling.parent.li)
# <li>First</li>

print(soup.find_all(string="Second")[0].parent)
# <li>Second</li>

soup = BeautifulSoup("<ul><li>First<li><ol><li>Second A<li>Second B</ol></ul>",
print(soup.li.li)
# None

soup = BeautifulSoup("<ul><li><ol><li>First A<li>First B</ol><li>Second</ul>", "
print(soup.li.li)
# <li>First A</li>

print(soup.li.li.string)
# First A
```

Beautiful Soup: Example Features (continued)

```
soup = BeautifulSoup('<a href="https://imada.sdu.dk/~kslarsen/">important</a>',
print(soup.a['href'])
# https://imada.sdu.dk/~kslarsen/

print(soup.a.get_attribute_list('href'))
# ['https://imada.sdu.dk/~kslarsen/']
```

Beautiful Soup Example

```
import requests
from bs4 import BeautifulSoup

page = requests.get("https://imada.sdu.dk/~kslarsen/dm565/notes.php")
soup = BeautifulSoup(page.content, "lxml")

# Printing all dates with DM565 activities
samples = soup.find_all("h3")
for sample in samples:
   print(" ".join(sample.string.split()[1:]))
```

```
October 24
October 24
October 11
October 9
October 7
October 4
...
```

Beautiful Soup: Extras

SDU

- regular expressions can be used instead of string search
- many find-variants
- support for CSS selectors
- operators for modifying the tree
- pretty printing
- support for various encodings

Keyboard Macros

Utilizing Editors

For data processing/transformation of very large amounts of data, we may prefer command-line tools for efficiency.

If data is not too large, *but* the process needs to be repeated often (when there is new data), we prefer programming: command-line tools in a script or some programming language with an appropriate API.

For one-time data transformation, your favorite editor may be the easiest – either using regular expressions or using macros. You already know regular expressions.

Keyboard Macros

SDU

If an editor offers macros, then you can do the following:

1. Start recording
2. Execute some editing commands
3. Stop recording
4. Replay the recorded editing sequence

To be really useful, one should be able to give an argument, saying how many times the macro should be replayed.

It is useful to include the appropriate navigation commands in the macro, so one does not have to reposition the cursor before replaying.

Emacs Example

Assumed we scraped the following off of `elearn.sdu.dk`:

```
Abyayananda Maiti Email: abmai13@student.sdu.dk
Anders Nicolai Knudsen Email: anknu18@student.sdu.dk
Bárður Árantsson Email: baara07@student.sdu.dk
Jens Svalgaard Kohrt Email: jeko404@student.sdu.dk
Lars Jacobsen Email: lajac01@student.sdu.dk
Marie Gabriele Christ Email: machr14@student.sdu.dk
Martin Rud Ehmsen Email: maehm10@student.sdu.dk
Morten Nyhave Nielsen Email: monie02@student.sdu.dk
Sushmita Gupta Email: sugup13@student.sdu.dk
```

We want to make a LaTeX table from this with the family name first and leaving off the repetitive `@student.sdu.dk`, i.e., using the format

```
FAMILY NAME & GIVEN NAMES & USER NAME \\
```

We can of course use regular expressions, but editor macros might be faster.

Emacs Example Considerations

If you have never used macros before, you may wonder how this will work, since editing the first line (manually) is rather different (position-wise) from editing the next.

The idea is to *program* using your usual editor commands, i.e., you edit the first line in a manner that would also be valid for the others. This makes it a little more cumbersome to edit the first line, but you see the result incrementally, so it is fairly easy to verify correctness as you go along. And you can always give up, fix the rest of the first line manually and try again from scratch on the second line.

Emacs Command Notation

`C-` means press and hold the Ctrl key while typing the character following the hyphen.

`M-` has a similar interpretation, where M is the meta key. This is often the key Alt, i.e., it `M-x` means `Alt-x`. As a fall back (for instance when editing through an ssh connection, where many keyboard sequences are not recognized or caught by the ssh protocol), one can use Esc, but then one has to let go of Esc before pressing the next key, e.g., `M-x` should then be interpreted as `Esc x`.

SPC denotes the space bar.

Emacs Example Macro

SDU

Command	Explanation
`C-x (`	Start recording
`C-s E m a i l :`	Search for "Email:"
`M-←`	Go left one word (across "Email:")
`M-←`	Go left one word (across the family name)
`C-SPC`	Set mark
`M-→`	Go right one word (across the family name)
`C-w`	Delete and buffer from mark (the family name)
`C-a`	Go to beginning of line
`C-y`	Yank from buffer (insert the family name)
`␣ & ␣`	Inserting the three characters space, ampersand, and space
`C-s E m a i l :`	Search for "Email:"
`C-SPC`	Set mark
`M-←`	Go left one word (across "Email:")

Emacs Example Macro (continued)

Command	Explanation
←	Go one position left
`C-w`	Delete and buffer from mark (" Email:")
`&`	Insert ampersand
`C-s @`	Search for "@"
←	Go one position left
`C-k`	Delete the rest of the line
␣ \ \	Insert the three characters space, backslash, backslash
`C-n`	Go to next line
`C-a`	Go to beginning of line
`C-x )`	End recording
`C-x e`	Execute recorded macro - yes, it worked!
`C-u 2 C-x e`	Execute the recorded macro twice
`C-u 0 C-x e`	Execute the recorded macro until error (rest of file)

Keyboard Macros Lesson

SDU

Lesson 1

If you use an editor that does not support regular expressions, macros, block indentation, etc., you may want to upgrade.

Lesson 2

You may not like the cryptic Emacs commands, but that is not the point. The point is the programming indicated in the Explanation column.

The text makes it clear it is programming using searches, word skips, beginning-of-line, cut to end-of-line, next line, etc.

The commands are specific to Emacs, but you can just replace them with the equivalent actions in your favorite editor.

Disclaimer

If you switch to a new editor (Emacs, for instance), you will initially find it hard to use macros because you ruin the sequence if you make just one mistake.

Keyboard Macros

A good editor may have

- a buffer system so that temporary cuts can be saved and used later
- possibilities of using counters (to make enumerated lists, for instance)
- asking for user input during macro execution
- saving (and naming) a macro
- possibilities for editing an already recorded macro