Data Cleaning & Scraping

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

October, 2023

Data Cleaning

Data Cleaning

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

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: Domains

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` on the column.)

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

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

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” → “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

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/u/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

```
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>

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

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

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

Beautiful Soup: Example Features (continued)

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

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

Beautiful Soup Example

```
import requests
from bs4 import BeautifulSoup

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

# Printing all dates of activities in IMADA's Conference Room
samples = soup.find_all("td")
for sample in samples:
    if sample.string == "IMADA's Conference Room":
        print(sample.parent.td.string)
```

```
6/11
13/11
14/11
29/11
30/11
21/12
```

Beautiful Soup: Extras

- 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