messytables Documentation

Release 0.3

Friedrich Lindenberg

January 13, 2017

Tabular data as published on the web is often not well formatted and structured. Messytables tries to detect and fix errors in the data. Typical examples include:

- Finding the header of a table when there are explanations and text fragments in the first few rows of the table.
- Guessing the type of columns in CSV data.
- Guessing the format of a byte stream.

This library provides data structures and some heuristics to fix these problems and read a wide number of different tabular abominations.

CHAPTER 1

Example

messytables offers some commands and data structures to read and evaluate data. A typical use might look like this:

```
from messytables import CSVTableSet, type_guess, \
  types_processor, headers_guess, headers_processor, \
  offset_processor

fh = open('messy.csv', 'rb')

# Load a file object:
table_set = CSVTableSet.from_fileobj(fh)

# If you aren't sure what kind of file it is, you can use
# AnyTableSet instead.
#table_set = AnyTableSet.from_fileobj(fh)

# A table set is a collection of tables:
row_set = table_set.tables[0]

# A row set is an iterator over the table, but it can only
# be run once. To peek, a sample is provided:
print row_set.sample.next()

# guess column types:
types = type_guess(row_set.sample)

# and tell the row set to apply these types to
# each row when traversing the iterator:
row_set.register_processor(types_processor(types))

# guess header names and the offset of the header:
offset, headers = headers_guess(row_set.sample)
row_set.register_processor(headers_processor(headers))

# add one to begin with content, not the header:
row_set.register_processor(offset_processor(offset + 1))

# now run some operation on the data:
for row in row_set:
  do_something(row)
```

As you can see in the example above, messytables gives you a toolbox of independent methods. There is no ready-made `row_set.guess_types()` because there are many ways to perform type guessing that we may implement in the future. Therefore, heuristic operations are independent of the main data structures.

CHAPTER 2

Core entities

Messytables uses a few core entities to avoid the nesting depth involved in generic data types (a dict in a list in a dict).

CHAPTER 3

CSV support

CSV support uses Python's dialect sniffer to detect the separator and quoting mechanism used in the input file.

CHAPTER 4

Excel support

The library supports workbooks in the Microsoft Excel 2003 format.

The newer, XML-based Excel format is also supported but uses a different class.

CHAPTER 5

ZIP file support

The library supports loading CSV or Excel files from within ZIP files.

CHAPTER 6

Auto-detecting file format

The library supports loading files in a generic way.

CHAPTER 7

Type detection

One aspect missing from some tabular representations (in particular the CSV format) is type information on the individual cells in the table. We can brute-force guess these types by attempting to convert all members of a given column into all types and searching for the best match.

The supported types include:

CHAPTER 8

Headers detection

While the CSV convention is to include column headers as the first row of the data file. Unfortunately, many people feel the need to put titles, general info etc. in the top of tabular data. Therefore, we need to scan the first few rows of the data, to guess which one is actually the header.

CHAPTER 9

Stream processors

Stream processors are used to apply transformations to the row set upon iteration. In order to apply transformations to a `RowSet` you can register a stream processor. A processor is simply a function that takes the `RowSet` and the current row (a list of `Cell`) as arguments and returns a modified version of the row or `None` to indicate the row should be dropped.

Most processors are implemented as closures called with some arguments:

CHAPTER 10

License