messytables Documentation

Release 0.3

Friedrich Lindenberg

Aug 19, 2017

Contents

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, any_tableset

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

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

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

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

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

```
# 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. Also note that *type_guess* is done after adding the *offset_processor* so that the headers are not part of the sample that we use for type guessing.

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).

class messytables.core.**Cell**(*value*, *column=None*, *type=None*)

A cell is the basic value type. It always has a `value` (that may be `None` and may optionally also have a type and column name associated with it. If no `type` is set, the String type is set but no type conversion is set.

value

The actual content of the cell.

column

The name of the column this cell is in.

type

`CellType` of this cell.

empty

Stringify the value and check that it has a length.

class messytables.core.**TableSet**(*fileobj*)

A table set is used for data formats in which multiple tabular objects are bundled. This might include relational databases and workbooks used in spreadsheet software (Excel, LibreOffice).

For each format, we derive from this abstract base class, providing a constructor that takes a file object and tables() that returns each table. This means you can stream a table set directly off a web site or some similar source.

On any fatal errors, it should raise messytables.ReadError

tables

Return a listing of tables (i.e. RowSets) in the `TableSet`. Each table has a name.

class messytables.core.**RowSet**(*typed=False*)

A row set (aka: table) is a simple wrapper for an iterator of rows (which in turn is a list of `Cell` objects). The main table iterable can only be traversed once, so on order to allow analytics like type and header guessing on the data, a sample of `window` rows is read, cached, and made available.

On any fatal errors, it should raise messytables.ReadError

__iter__(*sample=False*)
Apply processors to the row data.

dicts(*sample=False*)
Return a representation of the data as an iterator of ordered dictionaries. This is less specific than the cell format returned by the generic iterator but only gives a subset of the information.

register_processor(*processor*)
Register a stream processor to be used on each row. A processor is a function called with the `RowSet` as its first argument and the row to be processed as the second argument.

class `messytables.types.`**CellType**
A cell type maintains information about the format of the cell, providing methods to check if a type is applicable to a given value and to convert a value to the type.

cast(*value*)
Convert the value to the type. This may throw a quasi-random exception if conversion fails.

test(*value*)
Test if the value is of the given type. The default implementation calls `cast` and checks if that throws an exception. True or False

CHAPTER 3

CSV support

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

class messytables.commas.**CSVTableSet**(*fileobj*, *delimiter=None*, *quotechar=None*, *name=None*, *encoding=None*, *window=None*, *doublequote=None*, *lineterminator=None*, *skipinitialspace=None*, ***kw*)

A CSV table set. Since CSV is always just a single table, this is just a pass-through for the row set.

tables

Return a listing of tables (i.e. RowSets) in the TableSet. Each table has a name.

class messytables.commas.**CSVRowSet**(*name*, *fileobj*, *delimiter=None*, *quotechar=None*, *encoding='utf-8'*, *window=None*, *doublequote=None*, *lineterminator=None*, *skipinitialspace=None*)

A CSV row set is an iterator on a CSV file-like object (which can potentially be infinetly large). When loading, a sample is read and cached so you can run analysis on the fragment.

CHAPTER 4

Excel support

The library supports workbooks in the Microsoft Excel 2003 format.

class `messytables.excel.`**`XLSTableSet`**(*fileobj=None*, *filename=None*, *window=None*, *encoding=None*, *with_formatting_info=True*, ***kw*)

An excel workbook wrapper object.

`tables`

Return a listing of tables (i.e. RowSets) in the `TableSet`. Each table has a name.

class `messytables.excel.`**`XLSRowSet`**(*name*, *sheet*, *window=None*)

Excel support for a single sheet in the excel workbook. Unlike the CSV row set this is not a streaming operation.

`raw`(*sample=False*)

Iterate over all rows in this sheet. Types are automatically converted according to the excel data types specified, including conversion of excel dates, which are notoriously buggy.

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

CHAPTER 5

HTML file support

The library supports HTML documents, using lxml as a parser.

Removes the content of nested tables from the parent table. The order of the tables is ill-defined.

class `messytables.html.`**`HTMLTableSet`**(*fileobj=None*, *filename=None*, *window=None*, ***kw*)
A TableSet from a HTML document.

`tables`
Return a listing of tables (i.e. RowSets) in the `TableSet`. Each table has a name.

class `messytables.html.`**`HTMLRowSet`**(*name*, *sheet*, *window=None*)
A RowSet representing a HTML table.

CHAPTER 6

PDF file support

The library supports PDF documents, using pdftables to extract tables.

Works only for PDFs which contain text information: somewhat erratic in quality.

class `messytables.pdf.`**`PDFTableSet`**(*fileobj=None*, *filename=None*, ***kw*)
A TableSet from a PDF document.

`tables`
Return a listing of tables (i.e. RowSets) in the `TableSet`. Each table has a name.

class `messytables.pdf.`**`PDFRowSet`**(*name*, *table*)
A RowSet representing a PDF table.

CHAPTER 7

ZIP file support

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

class `messytables.zip.`**`ZIPTableSet`**(*fileobj*, ***kw*)
Reads TableSets from inside a ZIP file

`tables`
Return a listing of tables (i.e. RowSets) in the `TableSet`. Each table has a name.

CHAPTER 8

Auto-detecting file format

The library supports loading files in a generic way.

any.**any_tableset** (*fileobj*, *mimetype=None*, *extension=''*, *auto_detect=True*, ***kw*)

Reads any supported table type according to a specified MIME type or file extension or automatically detecting the type.

Best matching TableSet loaded with the fileobject is returned. Matching is done by looking at the type (e.g mimetype='text/csv'), then the file extension (e.g. extension='tsv'), then autodetecting the file format by using the magic library which looks at the first few bytes of the file BUT is often wrong. Consult the source for recognized MIME types and file extensions.

On error it raises messytables.ReadError

CHAPTER 9

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.

types.**type_guess**(*rows, types=[<class 'messytables.types.StringType'>, <class 'messytables.types.DecimalType'>, <class 'messytables.types.IntegerType'>, <class 'messytables.types.DateType'>, <class 'messytables.types.BoolType'>, <class 'messytables.types.TimeType'>, <class 'messytables.types.CurrencyType'>, <class 'messytables.types.PercentageType'>], strict=False*)

The type guesser aggregates the number of successful conversions of each column to each type, weights them by a fixed type priority and select the most probable type for each column based on that figure. It returns a list of `CellType`. Empty cells are ignored.

Strict means that a type will not be guessed if parsing fails for a single cell in the column.

The supported types include:

class messytables.types.**StringType**
A string or other unconverted type.

class messytables.types.**IntegerType**
An integer field.

class messytables.types.**FloatType**
FloatType is deprecated

class messytables.types.**DecimalType**
Decimal number, `decimal.Decimal` or float numbers.

class messytables.types.**BoolType**(*true_values=None*, *false_values=None*)
A boolean field. Matches true/false, yes/no and 0/1 by default, but a custom set of values can be optionally provided.

class messytables.types.**DateType**(*format*)
The date type is special in that it also includes a specific date format that is used to parse the date, additionally to the basic type information.

class messytables.types.**DateUtilType**

The date util type uses the dateutil library to parse the dates. The advantage of this type over DateType is the speed and better date detection. However, it does not offer format detection.

Do not use this together with the DateType

CHAPTER 10

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.

`headers`**`.headers_guess`**(*rows*, *tolerance=1*)

Guess the offset and names of the headers of the row set. This will attempt to locate the first row within `tolerance` of the mode of the number of columns in the row set sample.

The return value is a tuple of the offset of the header row and the names of the columns.

CHAPTER 11

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:

`types`.**`types_processor`**(*types*, *strict=False*)
: Apply the column types set on the instance to the current row, attempting to cast each cell to the specified type.

  Strict means that casting errors are not ignored

`util`.**`offset_processor`**(*offset*)
: Skip `offset` from the given iterator. This can be used in combination with the `headers_processor` to apply the result of a header scan to the table.

  Parameters **`offset`** (*`int`*) – Offset to be skipped

`util`.**`null_processor`**(*nulls*)
: Replaces every occurrence of items from *nulls* with None.

  Parameters **`nulls`** (*`list`*) – List of items to be replaced

`headers`.**`headers_processor`**(*headers*)
: Add column names to the cells in a row_set. If no header is defined, use an autogenerated name.

CHAPTER 12

JSON table schema

Messytables can convert guessed headers and types to the JSON table schema.

jts.**rowset_as_jts**(*rowset*, *headers=None*, *types=None*)
Create a json table schema from a rowset

jts.**headers_and_typed_as_jts**(*headers*, *types*)
Create a json table schema from headers and types as returned from *headers_guess()* and *type_guess()*.

CHAPTER 13

License

Index

Symbols

A

B

C

D

E

F

H

I

N

O

P

R

S

T

V

X

Z