TableDataExtractor Documentation

Release 2019

Juraj Mavračić

Dec 17, 2021

Contents:

Input a table as unstructured **.csv** file, **python list**, **.html** file, or **url** and output a standardized table where each row corresponds to a single data entry in the original table. TableDataExtractor will take care of complicated header structures in row and column headers, which includes:

- spanning cells
- nested column/row headers
- titles within the table
- note cells
- footnotes
- prefixing of row and column headers if non-unique
- multiple tables within one

Tip: TableDataExtractor can output to Pandas and will automatically create complex MultiIndex DataFrame structures.

Example

Importing Table 2 from 'https://link.springer.com/article/10.1007%2Fs10853-012-6439-6':

	Rutile	
	$a = b$ (Å)	c (Å)
This study	4.64	2.99
GGA [25]	4.67	2.97
GGA [26]	4.63	2.98
HF [27]	–	–
Expt. [23]	4.594	2.958

```
from tabledataextractor import Table
table = Table('https://link.springer.com/article/10.1007%2Fs10853-012-6439-6',2)
print(table)
```

```
+---------+------------------+--------------------------------------+
|   Data  |  Row Categories  |          Column Categories           |
+---------+------------------+--------------------------------------+
|   4.64  | [' This study '] |  [' Rutile ', '  a = b (Å) ']        |
|   2.99  | [' This study '] |       [' Rutile ', '  c (Å) ']       |
|  0.305  | [' This study '] |        [' Rutile ', '  u  ']         |
```

(continues on next page)

(continued from previous page)

```
|   3.83  | [' This study '] | [' Anatase ', '  a = b (Å) ']       |
|   9.62  | [' This study '] |      [' Anatase ', '  c (Å) ']      |
|  0.208  | [' This study '] |        [' Anatase ', '  u  ']       |
|   4.67  |  [' GGA [25] ']  |  [' Rutile ', '  a = b (Å) ']       |
|   2.97  |  [' GGA [25] ']  |       [' Rutile ', '  c (Å) ']      |
|  0.305  |  [' GGA [25] ']  |        [' Rutile ', '  u  ']        |
|   3.80  |  [' GGA [25] ']  | [' Anatase ', '  a = b (Å) ']       |
|   9.67  |  [' GGA [25] ']  |      [' Anatase ', '  c (Å) ']      |
|  0.207  |  [' GGA [25] ']  |        [' Anatase ', '  u  ']       |
|   4.63  |  [' GGA [26] ']  |  [' Rutile ', '  a = b (Å) ']       |
|   2.98  |  [' GGA [26] ']  |       [' Rutile ', '  c (Å) ']      |
|  0.305  |  [' GGA [26] ']  |        [' Rutile ', '  u  ']        |
|    -    |  [' GGA [26] ']  | [' Anatase ', '  a = b (Å) ']       |
|    -    |  [' GGA [26] ']  |      [' Anatase ', '  c (Å) ']      |
|    -    |  [' GGA [26] ']  |        [' Anatase ', '  u  ']       |
|    -    |  [' HF [27] ']   |  [' Rutile ', '  a = b (Å) ']       |
|    -    |  [' HF [27] ']   |       [' Rutile ', '  c (Å) ']      |
|    -    |  [' HF [27] ']   |        [' Rutile ', '  u  ']        |
|   3.76  |  [' HF [27] ']   | [' Anatase ', '  a = b (Å) ']       |
|   9.85  |  [' HF [27] ']   |      [' Anatase ', '  c (Å) ']      |
|  0.202  |  [' HF [27] ']   |        [' Anatase ', '  u  ']       |
|  4.594  | [' Expt. [23] '] |  [' Rutile ', '  a = b (Å) ']       |
|  2.958  | [' Expt. [23] '] |       [' Rutile ', '  c (Å) ']      |
|  0.305  | [' Expt. [23] '] |        [' Rutile ', '  u  ']        |
|  3.785  | [' Expt. [23] '] | [' Anatase ', '  a = b (Å) ']       |
|  9.514  | [' Expt. [23] '] |      [' Anatase ', '  c (Å) ']      |
|  0.207  | [' Expt. [23] '] |        [' Anatase ', '  u  ']       |
+---------+------------------+-------------------------------------+
```

CHAPTER 1

Installation

You can use `pip` to install TableDataExtractor:

```
pip install git+https://github.com/CambridgeMolecularEngineering/tabledataextractor
```

CHAPTER 2

Basics

2.1 Input

- from file, as .csv or .html
- from url (if there are more tables at the provided url, use the `table_number` argument)
- from python list object

[1]:
```
table_path = '../examples/tables/table_example.csv'

from tabledataextractor import Table
table = Table(table_path)
```

First we will check out the original table, which is now stored as `table.raw_table`. We can use the `print_raw_table()` function within *TableDataExtractor*:

[2]:
```
table.print_raw_table()
```

		Rutile a = b (Å)	Rutile c (Å)	Rutile u	Anatase a = b (Å)	Anatase c (Å)	Anatase u
Computational	This study	4.64	2.99	0.305	3.83	9.62	0.208
Computational	GGA [25]	4.67	2.97	0.305	3.80	9.67	0.207
Computational	GGA [26]	4.63	2.98	0.305	-	-	-
Computational	HF [27]	-	-	-	3.76	9.85	0.202
Experimental	Expt. [23]	4.594	2.958	0.305	3.785	9.514	0.207

TableDataExtractor provides a *category table*, where each row corresponds to a single data point. This is the main result of *TableDataExtractor*. We can simply `print` the table to see it:

[3]:
```
print(table)
```

```
+-------+---------------------------------+--------------------------+
|  Data |          Row Categories         |    Column Categories     |
+-------+---------------------------------+--------------------------+
|  4.64 | ['Computational', 'This study'] | ['Rutile', 'a = b (Å)']  |
|  2.99 | ['Computational', 'This study'] |   ['Rutile', 'c (Å)']    |
| 0.305 | ['Computational', 'This study'] |     ['Rutile', 'u']      |
|  3.83 | ['Computational', 'This study'] | ['Anatase', 'a = b (Å)'] |
|  9.62 | ['Computational', 'This study'] |   ['Anatase', 'c (Å)']   |
| 0.208 | ['Computational', 'This study'] |     ['Anatase', 'u']     |
|  4.67 |  ['Computational', 'GGA [25]']  | ['Rutile', 'a = b (Å)']  |
|  2.97 |  ['Computational', 'GGA [25]']  |   ['Rutile', 'c (Å)']    |
| 0.305 |  ['Computational', 'GGA [25]']  |     ['Rutile', 'u']      |
|  3.80 |  ['Computational', 'GGA [25]']  | ['Anatase', 'a = b (Å)'] |
|  9.67 |  ['Computational', 'GGA [25]']  |   ['Anatase', 'c (Å)']   |
| 0.207 |  ['Computational', 'GGA [25]']  |     ['Anatase', 'u']     |
|  4.63 |  ['Computational', 'GGA [26]']  | ['Rutile', 'a = b (Å)']  |
|  2.98 |  ['Computational', 'GGA [26]']  |   ['Rutile', 'c (Å)']    |
| 0.305 |  ['Computational', 'GGA [26]']  |     ['Rutile', 'u']      |
|   -   |  ['Computational', 'GGA [26]']  | ['Anatase', 'a = b (Å)'] |
|   -   |  ['Computational', 'GGA [26]']  |   ['Anatase', 'c (Å)']   |
|   -   |  ['Computational', 'GGA [26]']  |     ['Anatase', 'u']     |
|   -   |   ['Computational', 'HF [27]']  | ['Rutile', 'a = b (Å)']  |
|   -   |   ['Computational', 'HF [27]']  |   ['Rutile', 'c (Å)']    |
|   -   |   ['Computational', 'HF [27]']  |     ['Rutile', 'u']      |
|  3.76 |   ['Computational', 'HF [27]']  | ['Anatase', 'a = b (Å)'] |
|  9.85 |   ['Computational', 'HF [27]']  |   ['Anatase', 'c (Å)']   |
| 0.202 |   ['Computational', 'HF [27]']  |     ['Anatase', 'u']     |
| 4.594 |  ['Experimental', 'Expt. [23]'] | ['Rutile', 'a = b (Å)']  |
| 2.958 |  ['Experimental', 'Expt. [23]'] |   ['Rutile', 'c (Å)']    |
| 0.305 |  ['Experimental', 'Expt. [23]'] |     ['Rutile', 'u']      |
| 3.785 |  ['Experimental', 'Expt. [23]'] | ['Anatase', 'a = b (Å)'] |
| 9.514 |  ['Experimental', 'Expt. [23]'] |   ['Anatase', 'c (Å)']   |
| 0.207 |  ['Experimental', 'Expt. [23]'] |     ['Anatase', 'u']     |
+-------+---------------------------------+--------------------------+
```

If we want to further process the *category table*, we can access it as a list of lists:

[4]:
```
print(table.category_table)
```

```
[['4.64', ['Computational', 'This study'], ['Rutile', 'a = b (Å)']], ['2.99', [
↪'Computational', 'This study'], ['Rutile', 'c (Å)']], ['0.305', ['Computational',
↪'This study'], ['Rutile', 'u']], ['3.83', ['Computational', 'This study'], ['Anatase
↪', 'a = b (Å)']], ['9.62', ['Computational', 'This study'], ['Anatase', 'c (Å)']], [
↪'0.208', ['Computational', 'This study'], ['Anatase', 'u']], ['4.67', [
↪'Computational', 'GGA [25]'], ['Rutile', 'a = b (Å)']], ['2.97', ['Computational',
↪'GGA [25]'], ['Rutile', 'c (Å)']], ['0.305', ['Computational', 'GGA [25]'], ['Rutile
↪', 'u']], ['3.80', ['Computational', 'GGA [25]'], ['Anatase', 'a = b (Å)']], ['9.67
↪', ['Computational', 'GGA [25]'], ['Anatase', 'c (Å)']], ['0.207', ['Computational',
↪ 'GGA [25]'], ['Anatase', 'u']], ['4.63', ['Computational', 'GGA [26]'], ['Rutile',
↪'a = b (Å)']], ['2.98', ['Computational', 'GGA [26]'], ['Rutile', 'c (Å)']], ['0.305
↪', ['Computational', 'GGA [26]'], ['Rutile', 'u']], ['-', ['Computational', 'GGA␣
↪[26]'], ['Anatase', 'a = b (Å)']], ['-', ['Computational', 'GGA [26]'], ['Anatase',
↪'c (Å)']], ['-', ['Computational', 'GGA [26]'], ['Anatase', 'u']], ['-', [
↪'Computational', 'HF [27]'], ['Rutile', 'a = b (Å)']], ['-', ['Computational', 'HF␣
↪[27]'], ['Rutile', 'c (Å)']], ['-', ['Computational', 'HF [27]'], ['Rutile', 'u']],␣
↪['3.76', ['Computational', 'HF [27]'], ['Anatase', 'a = b (Å)']], ['9.85', [
↪'Computational', 'HF [27]'], ['Anatase', 'c (Å)']], ['0.202', ['Computational', 'HF␣
↪[27]'], ['Anatase', 'u']], ['4.594', ['Experimental', 'Expt. [23]'], ['Rutile', 'a␣
↪= b (Å)']], ['2.958', ['Experimental', 'Expt. [23]'], ['Rutile', 'c (Å)']], ['0.305
↪', ['Experimental', 'Expt. [23]'], ['Rutile', 'u']], ['3.785', ['Experimental',
↪'Expt. [23]'], ['Anatase', 'a = b (Å)']], ['9.514', ['Experimental', 'Expt. [23]'],␣
↪['Anatase', 'c (Å)']], ['0.207', ['Experimental', 'Expt. [23]'], ['Anatase', 'u']]]
```

(continues on next page)

(continued from previous page)

We may wish to access other elements of the table, such as the title row, the row or column headers, and the data:

[5]:
```
print ("Title row:     \n", table.title_row)
print ("Row header:    \n", table.row_header)
print ("Column header: \n", table.col_header)
print ("Data:          \n", table.data)
```

```
Title row:
 0
Row header:
 [['Computational' 'This study']
 ['Computational' 'GGA [25]']
 ['Computational' 'GGA [26]']
 ['Computational' 'HF [27]']
 ['Experimental' 'Expt. [23]']]
Column header:
 [['Rutile' 'Rutile' 'Rutile' 'Anatase' 'Anatase' 'Anatase']
 ['a = b (Å)' 'c (Å)' 'u' 'a = b (Å)' 'c (Å)' 'u']]
Data:
 [['4.64' '2.99' '0.305' '3.83' '9.62' '0.208']
 ['4.67' '2.97' '0.305' '3.80' '9.67' '0.207']
 ['4.63' '2.98' '0.305' '-' '-' '-']
 ['-' '-' '-' '3.76' '9.85' '0.202']
 ['4.594' '2.958' '0.305' '3.785' '9.514' '0.207']]
```

If needed we can transpose the whole table, which will return the same category table, with row and column categories interchanged:

[6]:
```
table.transpose()
print(table)
```

```
+-------+-------------------------+---------------------------------+
|  Data |      Row Categories     |        Column Categories        |
+-------+-------------------------+---------------------------------+
|  4.64 | ['Rutile', 'a = b (Å)'] | ['Computational', 'This study'] |
|  4.67 | ['Rutile', 'a = b (Å)'] |  ['Computational', 'GGA [25]']  |
|  4.63 | ['Rutile', 'a = b (Å)'] |  ['Computational', 'GGA [26]']  |
|   -   | ['Rutile', 'a = b (Å)'] |   ['Computational', 'HF [27]']  |
| 4.594 | ['Rutile', 'a = b (Å)'] |  ['Experimental', 'Expt. [23]'] |
|  2.99 |   ['Rutile', 'c (Å)']   | ['Computational', 'This study'] |
|  2.97 |   ['Rutile', 'c (Å)']   |  ['Computational', 'GGA [25]']  |
|  2.98 |   ['Rutile', 'c (Å)']   |  ['Computational', 'GGA [26]']  |
|   -   |   ['Rutile', 'c (Å)']   |   ['Computational', 'HF [27]']  |
| 2.958 |   ['Rutile', 'c (Å)']   |  ['Experimental', 'Expt. [23]'] |
| 0.305 |     ['Rutile', 'u']     | ['Computational', 'This study'] |
| 0.305 |     ['Rutile', 'u']     |  ['Computational', 'GGA [25]']  |
| 0.305 |     ['Rutile', 'u']     |  ['Computational', 'GGA [26]']  |
|   -   |     ['Rutile', 'u']     |   ['Computational', 'HF [27]']  |
| 0.305 |     ['Rutile', 'u']     |  ['Experimental', 'Expt. [23]'] |
|  3.83 | ['Anatase', 'a = b (Å)'] | ['Computational', 'This study'] |
|  3.80 | ['Anatase', 'a = b (Å)'] |  ['Computational', 'GGA [25]']  |
|   -   | ['Anatase', 'a = b (Å)'] |  ['Computational', 'GGA [26]']  |
|  3.76 | ['Anatase', 'a = b (Å)'] |   ['Computational', 'HF [27]']  |
| 3.785 | ['Anatase', 'a = b (Å)'] |  ['Experimental', 'Expt. [23]'] |
|  9.62 |   ['Anatase', 'c (Å)']   | ['Computational', 'This study'] |
|  9.67 |   ['Anatase', 'c (Å)']   |  ['Computational', 'GGA [25]']  |
```

(continues on next page)

(continued from previous page)

```
|   -   |   ['Anatase', 'c (Å)']   |  ['Computational', 'GGA [26]']  |
|  9.85 |   ['Anatase', 'c (Å)']   |   ['Computational', 'HF [27]']  |
| 9.514 |   ['Anatase', 'c (Å)']   |  ['Experimental', 'Expt. [23]'] |
| 0.208 |     ['Anatase', 'u']     | ['Computational', 'This study'] |
| 0.207 |     ['Anatase', 'u']     |  ['Computational', 'GGA [25]']  |
|   -   |     ['Anatase', 'u']     |  ['Computational', 'GGA [26]']  |
| 0.202 |     ['Anatase', 'u']     |   ['Computational', 'HF [27]']  |
| 0.207 |     ['Anatase', 'u']     |  ['Experimental', 'Expt. [23]'] |
+-------+--------------------------+---------------------------------+
```

2.2 Output & Pandas

- as csv file
- as Pandas DataFrame

To store the table as `.csv`:

[7]:
```
table.to_csv('./saved_table.csv')
```

The table can also be converted to a Pandas DataFrame object:

[8]:
```
import pandas
df = table.to_pandas()
df
```

[8]:

		Computational			Experimental
		This study	GGA [25]	GGA [26] HF [27]	Expt. [23]
Rutile	a = b (Å)	4.64	4.67	4.63 –	4.594
	c (Å)	2.99	2.97	2.98 –	2.958
	u	0.305	0.305	0.305 –	0.305
Anatase	a = b (Å)	3.83	3.80	– 3.76	3.785
	c (Å)	9.62	9.67	– 9.85	9.514
	u	0.208	0.207	– 0.202	0.207

We can now use all the powerful features of Pandas to interpret the content of the table. Lets say that we are interested in the experimental values for 'Anatase':

[9]:
```
df.loc['Anatase','Experimental']
```

[9]:

	Expt. [23]
a = b (Å)	3.785
c (Å)	9.514
u	0.207

The most powerful feature of TableDataExtractor is that it will automatically create a `MultiIndex` for the Pandas DataFrame, which would traditionally be done by hand for every individual table.

[10]:
```
print(df.index)
print(df.columns)
```

```
MultiIndex(levels=[['Anatase', 'Rutile'], ['a = b (Å)', 'c (Å)', 'u']],
           labels=[[1, 1, 1, 0, 0, 0], [0, 1, 2, 0, 1, 2]])
MultiIndex(levels=[['Computational', 'Experimental'], ['Expt. [23]', 'GGA [25]', 'GGA␣
↪[26]', 'HF [27]', 'This study']],
           labels=[[0, 0, 0, 0, 1], [4, 1, 2, 3, 0]])
```

Or, we might be interested in only the 'c (Å)' values from the table. Here, `ilevel_1` specifies the *index level* of *1*, which includes a=b(Å), c(Å) and u:

[11]:
```
df.query('ilevel_1 == "c (Å)"')
```

[11]:

		Computational				Experimental
		This study	GGA [25]	GGA [26]	HF [27]	Expt. [23]
Rutile	c (Å)	2.99	2.97	2.98	–	2.958
Anatase	c (Å)	9.62	9.67	–	9.85	9.514

CHAPTER 3

Features

TableDataExtractor uses a variety of algorithms to represent a table in standardized format. They work independently of the input format in which the table was provided. Thus, *TableDataExtractor* works equally as good for `.csv` files, as for `.html` files.

3.1 Standardized Table

The main feature of *TableDataExtractor* is the standardization of the input table. All algorithms and features presented herein have the goal to create a higher quality standardized table. This can subsequenlty be used for automated parsing, and automated retrieval of information from the table.

The standardized table (*category table*) can be output as a list as `table.category_table` or simply printed with `print(table)`. Table example from Embley et. al (2016):

[1]:
```
from tabledataextractor import Table

file = '../examples/tables/table_example_footnotes.csv'
table = Table(file)

table.print_raw_table()
print(table)
```

```
1 Development
Country        Million dollar  Million dollar  Million dollar  Percentage of GNI ␣
↪Percentage of GNI
               2007            2010*           2011* a.        2007               2011
First table
Australia      3735            4580            4936            0.95               1
Greece         2669            3826            4799            0.32               0.35
New Zealand    320             342             429             0.27               0.28
OECD/DAC c     104206          128465          133526          0.27               0.31
c              (unreliable)
* world bank
```

(continues on next page)

(continued from previous page)

```
a.


+--------+--------------------------+-----------------------------------------+
|  Data  |       Row Categories     |             Column Categories           |
+--------+--------------------------+-----------------------------------------+
|  3735  |       ['Australia']      |         ['Million dollar', '2007']      |
|  4580  |       ['Australia']      |   ['Million dollar', '2010 world bank ']|
|  4936  |       ['Australia']      | ['Million dollar', '2011 world bank    '] |
|  0.95  |       ['Australia']      |       ['Percentage of GNI', '2007']     |
|   1    |       ['Australia']      |       ['Percentage of GNI', '2011']     |
|  2669  |         ['Greece']       |         ['Million dollar', '2007']      |
|  3826  |         ['Greece']       |   ['Million dollar', '2010 world bank ']|
|  4799  |         ['Greece']       | ['Million dollar', '2011 world bank    '] |
|  0.32  |         ['Greece']       |       ['Percentage of GNI', '2007']     |
|  0.35  |         ['Greece']       |       ['Percentage of GNI', '2011']     |
|  320   |      ['New Zealand']     |         ['Million dollar', '2007']      |
|  342   |      ['New Zealand']     |   ['Million dollar', '2010 world bank ']|
|  429   |      ['New Zealand']     | ['Million dollar', '2011 world bank    '] |
|  0.27  |      ['New Zealand']     |       ['Percentage of GNI', '2007']     |
|  0.28  |      ['New Zealand']     |       ['Percentage of GNI', '2011']     |
| 104206 | ['OECD/DAC (unreliable)']|         ['Million dollar', '2007']      |
| 128465 | ['OECD/DAC (unreliable)']|   ['Million dollar', '2010 world bank ']|
| 133526 | ['OECD/DAC (unreliable)']| ['Million dollar', '2011 world bank    '] |
|  0.27  | ['OECD/DAC (unreliable)']|       ['Percentage of GNI', '2007']     |
|  0.31  | ['OECD/DAC (unreliable)']|       ['Percentage of GNI', '2011']     |
+--------+--------------------------+-----------------------------------------+
```

3.2 Nested Headers and Cell Labelling

The *data region* of an input table is isolated, taking complex row/column header structures into account and preserving the information about which header categories a particular data point belongs to. The table cells are labelled, according to their role in the table, as *Data*, *Row Header*, *Column Header*, *Stub Header*, *Title*, *Footnote*, *Footnote Text*, and *Note* cells.

[2]:
```
from tabledataextractor.output.print import print_table
table.print_raw_table()
print_table(table.labels)
```

```
1 Development
Country        Million dollar  Million dollar  Million dollar  Percentage of GNI ␣
↪Percentage of GNI
               2007            2010*           2011* a.        2007              2011
First table
Australia      3735            4580            4936            0.95              1
Greece         2669            3826            4799            0.32              0.35
New Zealand    320             342             429             0.27              0.28
OECD/DAC c     104206          128465          133526          0.27              0.31
c              (unreliable)
* world bank
a.


TableTitle         TableTitle  TableTitle         TableTitle                    ␣
↪TableTitle  TableTitle
```

(continues on next page)

(continued from previous page)

```
StubHeader          ColHeader   ColHeader           ColHeader                  ␣
↪ColHeader   ColHeader
StubHeader          ColHeader   ColHeader & FNref   ColHeader & FNref & FNref ␣
↪ColHeader   ColHeader
Note                /           /                   /                          /       ␣
↪   /
RowHeader           Data        Data                Data                       Data    ␣
↪   Data
RowHeader           Data        Data                Data                       Data    ␣
↪   Data
RowHeader           Data        Data                Data                       Data    ␣
↪   Data
RowHeader & FNref   Data        Data                Data                       Data    ␣
↪   Data
FNprefix            FNtext      /                   /                          /       ␣
↪   /
FNprefix & FNtext   /           /                   /                          /       ␣
↪   /
FNprefix            /           /                   /                          /       ␣
↪   /
```

3.3 Prefixing of headers

In many tables the headers are non-conclusive, meaning that they include duplicate elements that are usually highlighted in bold or italics. Due to the highlighting the structure of the table can still be understood by the reader. However, since *TableDataExtractor* doesn't take any graphical features into account, but only consideres the raw content of cells in tabular format, a *prefixing* step needs to be performed in some cases to find the header region correctly.

Since the main algorithm used to find the data region, the MIPS algorithm (*Minimum Indexing Point Search*), relies on duplicate entries in the header regions, the prefixing step is done in an iterative fashion. First, the headers are found and only afterwards the prefixing is performed. By comparison of the new results before and after a decision is made whether to accept the prefixing or not.

Two examples of prefixing are shown below, for the column and row header, respectively (examples from Embley et. al 2016). Prefixing can be turned off by setting the `use_prefixing = False` keyword argument upon creation of the `Table` instance.

[3]:
```
file = '../examples/tables/table_example8.csv'
table = Table(file)
table.print()
```

```
Table 9.
Year      Short messages  Change %  Other messages  Multimedia messages  Change %
2003      1647218         24.3      347             2314
2004      2193498         33.2      439             7386                 219.2


Table 9.
                          Short messages                                        ␣
↪Multimedia messages
Year      Short messages  Change %        Other messages  Multimedia messages  Change
↪%
2003      1647218         24.3            347             2314
```

(continues on next page)

(continued from previous page)

```
2004      2193498         33.2            439             7386                  219.2


TableTitle  TableTitle  TableTitle  TableTitle  TableTitle  TableTitle
StubHeader  ColHeader   ColHeader   ColHeader   ColHeader   ColHeader
StubHeader  ColHeader   ColHeader   ColHeader   ColHeader   ColHeader
RowHeader   Data        Data        Data        Data        Data
RowHeader   Data        Data        Data        Data        Data
```

[4]:
```
file = '../examples/tables/table_example9.csv'
table = Table(file)
table.print()
```

```
Year                          2003     2004
Short messages/thousands      1647218  2193498
Change %                      24.3     33.2
Other messages                347      439
Multimedia messages/thousands 2314     7386
Change %                               219.2


                              Year                          2003     2004
                              Short messages/thousands      1647218  2193498
Short messages/thousands      Change %                      24.3     33.2
                              Other messages                347      439
                              Multimedia messages/thousands 2314     7386
Multimedia messages/thousands Change %                               219.2


StubHeader  StubHeader  ColHeader  ColHeader
RowHeader   RowHeader   Data       Data
RowHeader   RowHeader   Data       Data
RowHeader   RowHeader   Data       Data
RowHeader   RowHeader   Data       Data
RowHeader   RowHeader   Data       Data
```

3.4 Spanning cells

Spanning cells are commonly encountered in tables. This information is easy to retreive if the table is provided in `.html` format. However, if the table is provided as `.csv` file or a python list, the content of spannig cells needs to be duplicated into each one of the spanning cells. *TableDataExtractor* does that automatically.

The duplication of spanning cells can be turned off by setting `use_spanning_cells = False` at creation of the `Table` instance. Table example from Embley et. al (2016):

[5]:
```
file = '../examples/tables/te_04.csv'
table = Table(file)
table.print()
```

```
Pupils in comprehensive schools
Year                          School  Pupils
↪    Grade 1  Leaving certificates
```

(continues on next page)

(continued from previous page)

```
                                        Pre-primary  Grades          Additional␣
↪Total
                                                     6 Jan   9 Jul
1990                            4869    2189         389410  197719          ␣
↪592920  67427    61054
1991                            4861    2181         389411  197711  3601    ␣
↪592921  67421


Pupils in comprehensive schools
Year                            School  Pupils       Pupils  Pupils  Pupils  ␣
↪Pupils  Grade 1  Leaving certificates
Year                            School  Pre-primary  Grades  Grades  Additional␣
↪Total   Grade 1  Leaving certificates
Year                            School  Pre-primary  6 Jan   9 Jul   Additional␣
↪Total   Grade 1  Leaving certificates
1990                            4869    2189         389410  197719          ␣
↪592920  67427    61054
1991                            4861    2181         389411  197711  3601    ␣
↪592921  67421


TableTitle  TableTitle  TableTitle  TableTitle  TableTitle  TableTitle  TableTitle␣
↪TableTitle  TableTitle
StubHeader  ColHeader   ColHeader   ColHeader   ColHeader   ColHeader   ColHeader  ␣
↪ColHeader   ColHeader
StubHeader  ColHeader   ColHeader   ColHeader   ColHeader   ColHeader   ColHeader  ␣
↪ColHeader   ColHeader
StubHeader  ColHeader   ColHeader   ColHeader   ColHeader   ColHeader   ColHeader  ␣
↪ColHeader   ColHeader
RowHeader   Data        Data        Data        Data        Data        Data       ␣
↪Data        Data
RowHeader   Data        Data        Data        Data        Data        Data       ␣
↪Data        Data
```

3.5 Subtables

If there are many tables nested within a single input table, and if they are of a compatible header structure, *TableDataExtractor* will automatically process them. `table.subtables` will contain a list of those subtables, where each entry will be an instance of the *TableDataExtractor* `Table` class.

[6]:
```
file = '../examples/tables/te_06.csv'
table = Table(file)
table.print_raw_table()

table.subtables[0].print_raw_table()
table.subtables[1].print_raw_table()
table.subtables[2].print_raw_table()
```

```
Material  Tc    A  Material     Tc    A  Material  Tc
Bi6Tl3    6.5   x  TiN          1.4   y  TiO2      1.1
Sb2Tl7    5.5   y  TiC          1.1   x  TiO3      1.2
```

(continues on next page)

(continued from previous page)

```
Na2Pb5    7.2   z  TaC          9.2   x  TiO4      1.3
Hg5Tl7    3.8   x  NbC          10.1  a  TiO5      1.4
Au2Bi     1.84  x  ZrB          2.82  x  TiO6      1.5
CuS       1.6   x  TaSi         4.2   x  TiO7      1.6
VN        1.3   x  PbS          4.1   x  TiO8      1.7
WC        2.8   x  Pb-As alloy  8.4   x  TiO9      1.8
W2C       2.05  x  Pb-Sn-Bi     8.5   x  TiO10     1.9
MoC       7.7   x  Pb-As-Bi     9.0   x  TiO11     1.10
Mo2C      2.4   x  Pb-Bi-Sb     8.9   x  TiO12     1.11


Material  Tc    A
Bi6Tl3    6.5   x
Sb2Tl7    5.5   y
Na2Pb5    7.2   z
Hg5Tl7    3.8   x
Au2Bi     1.84  x
CuS       1.6   x
VN        1.3   x
WC        2.8   x
W2C       2.05  x
MoC       7.7   x
Mo2C      2.4   x


Material     Tc    A
TiN          1.4   y
TiC          1.1   x
TaC          9.2   x
NbC          10.1  a
ZrB          2.82  x
TaSi         4.2   x
PbS          4.1   x
Pb-As alloy  8.4   x
Pb-Sn-Bi     8.5   x
Pb-As-Bi     9.0   x
Pb-Bi-Sb     8.9   x


Material  Tc
TiO2      1.1
TiO3      1.2
TiO4      1.3
TiO5      1.4
TiO6      1.5
TiO7      1.6
TiO8      1.7
TiO9      1.8
TiO10     1.9
TiO11     1.10
TiO12     1.11
```

3.6 Footnotes

TableDataExtractor handles footnotes by copying the footnote text into the appropriate cells where the footnotes have been referenced. This is a useful feature for automatic parsing of the *category table*. The copying of the footnote text can be prevented by using the `use_footnotes = False` keyword argument on `Table` creation.

Each footnote is a `TableDataExtractor.Footnote` object that contains all the footnote-relevant information. It can be inspected with `print(table.footnotes[0])`. Table example from Embley et. al (2016):

[7]:
```
file = '../examples/tables/table_example_footnotes.csv'
table = Table(file)
table.print()

print(table.footnotes[0])
print(table.footnotes[1])
print(table.footnotes[2])
```

```
1 Development
Country        Million dollar  Million dollar  Million dollar  Percentage of GNI ␣
↪Percentage of GNI
               2007            2010*           2011* a.        2007               2011
First table
Australia      3735            4580            4936            0.95               1
Greece         2669            3826            4799            0.32               0.35
New Zealand    320             342             429             0.27               0.28
OECD/DAC c     104206          128465          133526          0.27               0.31
c              (unreliable)
* world bank
a.


1 Development
Country                Million dollar  Million dollar    Million dollar       ␣
↪Percentage of GNI  Percentage of GNI
Country                2007            2010 world bank   2011 world bank      2007   ␣
↪           2011
First table
Australia              3735            4580              4936                 0.95   ␣
↪           1
Greece                 2669            3826              4799                 0.32   ␣
↪           0.35
New Zealand            320             342               429                  0.27   ␣
↪           0.28
OECD/DAC (unreliable)  104206          128465            133526               0.27   ␣
↪           0.31
c                      (unreliable)
* world bank
a.


TableTitle         TableTitle  TableTitle         TableTitle                ␣
↪TableTitle  TableTitle
StubHeader         ColHeader   ColHeader          ColHeader                 ␣
↪ColHeader   ColHeader
StubHeader         ColHeader   ColHeader & FNref  ColHeader & FNref & FNref ␣
↪ColHeader   ColHeader
Note               /           /                  /                          /       ␣
↪   /
```

(continues on next page)

(continued from previous page)

```
RowHeader           Data        Data              Data                      Data
↪   Data
RowHeader           Data        Data              Data                      Data
↪   Data
RowHeader           Data        Data              Data                      Data
↪   Data
RowHeader & FNref   Data        Data              Data                      Data
↪   Data
FNprefix            FNtext      /                 /                         /
↪   /
FNprefix & FNtext   /           /                 /                         /
↪   /
FNprefix            /           /                 /                         /
↪   /


Prefix: 'c'    Text: '(unreliable)'
↪Ref. Cells: [(7, 0)]   References: ['OECD/DAC c']
Prefix: '*'    Text: 'world bank'
↪Ref. Cells: [(2, 2), (2, 3)]   References: ['2010*', '2011* a.']
Prefix: 'a.'   Text: ''
↪Ref. Cells: [(2, 3)]   References: ['2011 world bank  a.']
```

CHAPTER 4

API Docs

Note: The `Table` object is everything you need to use *TableDataExtractor*. The other sections, *Input*, *Output*, *History*, *Footnotes*, *Algorithms*, *Cell Parser*, and *Exceptions* are for reference only.

4.1 Table Object

Note: This is everything you need to use *TableDataExtractor*. The other sections, *Input*, *Output*, *History*, *Footnotes*, *Algorithms*, *Cell Parser*, and *Exceptions* are for reference only.

Represents a table in a highly standardized format.

class tabledataextractor.table.table.**Table**(*file_path*, *table_number=1*, ***kwargs*)

Main *TableDataExtractor* object that includes the raw (input), cleaned (processes) and labelled tables. Represents the table input (.csv, .html, python list, url) in a highly standardized *category table* format, using the MIPS (*Minimum Indexing Point Search*) algorithm.

Optional configuration keywords (defaults):

- **use_title_row = True** A title row will be assumed if possible.
- **use_prefixing = True** Will perform the prefixing steps if row or column index cells are not unique.
- **use_spanning_cells = True** Will duplicate spanning cells in the row and column header regions if needed.
- **use_header_extension = True** Will extend the row and column header beyond the MIPS-defined headers, if needed.
- **use_footnotes = True** Will copy the footnote text into the appropriate cells of the table and remove the footnote prefix.

- **use_max_data_area = False** If *True* the max data area will be used to determine the cell *CC2* in the main MIPS algorithm. It is probably never necessary to set this to True.
- **standardize_empty_data = True** Will standardize empty cells in the *data* region to 'NoValue'
- **row_header = None** If an integer is given, it indicates the index of *row_header* columns. This overwrites the MIPS algorithm. For example, `row_header = 0` will make only the first column a row header.
- **col_header = None** If an integer is given, it indicates the index of *col_header* rows. This overwrites the MIPS algorithm. For example, `col_header = 0` will make only the first row a column header.

Parameters

- **file_path** (*str | list*) – Path to .html or .cvs file, URL or list object that is used as input
- **table_number** (*int*) – Number of table to read, if there are several at the given url, or in the html file

category_table
Standardized table, where each row corresponds to a single data point of the original table. The columns are the row and column categories where the data point belongs to.

Type list

col_header
Column header of the table.

Type numpy.ndarray

configs
Configuration keywords set at the creation of the *Table* instance.

Type dict

contains(*pattern*)
Returns true if table contains a particular string.

Parameters **pattern** – Regular expression for input

Returns True/False

data
Data region of the table.

Type numpy.ndarray

footnotes
List of footnotes in the table. Each footnote is an instance of *Footnote*.

Type list[*Footnote*]

history
Indicates which algorithms have been applied to the table by TableDataExtractor.

Type *History*

labels
Cell labels.

Type list

pre_cleaned_table
Cleaned-up table. This table is used for labelling the table regions, finding data-cells and building the category table.

Type numpy.array

pre_cleaned_table_empty
Mask array with *True* for all empty cells of the `pre_cleaned_table`.

Type numpy.array

print()
Prints the *raw table* (input), *cleaned table* (processed by *TableDataExtractor*) and *labels* (regions of the table) nicely.

print_raw_table()
Prints raw input table nicely.

raw_table
Input table, as provided to *TableDataExtractor*.

Type numpy.array

row_categories
Table where the original stub header is the first row(s) and all subsequent rows are the row categories of the original table. The assumption is made that the stub header labels row categories (that is, cells below the stub header). The *row_categories* table can be used if the row categories want to be analyzed as *data* themselves, which can occur if the header regions of the original table intentionally have duplicate elements.

Type *TrivialTable*

row_header
Row header of the table.

Type numpy.ndarray

stub_header
Stub header of the table.

Type numpy.ndarray

subtables
List of all subtables. Each subtable is an instance of `Table`.

Type list[*Table*]

title_row
Title row of the table.

Type list

to_csv(*file_path*)
Saves the *raw_table* to a *.csv* file.

to_pandas()
Converts the *Table* into a *Pandas DataFrame*, taking the complex MultiIndex structure of the table into account.

Returns pandas.DataFrame

transpose()
Transposes the *Table* and performs the analysis again. In this way, if working interactively from a *Jupyter*

notebook, it is possible to input a table and then transpose it to see how it looks like and if the results of the standardization are different.

class tabledataextractor.table.table.**TrivialTable**(*file_path*, *table_number=1*, ***kwargs*)

Trivial Table object. No high level analysis will be performed. MIPS algorithm is never run. This table doesn't have footnotes, a title row or subtables.

Optional configuration keywords (defaults):

- **standardize_empty_data = False** Will standardize empty cells in the *data* region to 'No-Value'.
- **clean_row_header = False** Removes duplicate rows that span the whole table (all columns).
- **row_header = 0** The column up to which the row header is defined.
- **col_header = 0** The row up to which the column header is defined.

col_header
Column header of the table.

Type numpy.ndarray

footnotes
None

labels
Cell labels.

Type numpy.array

row_header
Row header of the table. Enables a one-column table.

Type numpy.ndarray

subtables
None

title_row
None

4.2 Input

4.2.1 From any input

Analyzes the input and calls the appropriate input module.

tabledataextractor.input.from_any.**create_table**(*name_key*, *table_number=1*)

Checks the input and calls the appropriate modules for conversion. Returns a numpy array with the raw table.

Parameters

- **name_key** (*str | list*) – Path to *.html* or *.cvs* file, *URL* or *python list* that is used as input
- **table_number** (*int*) – Number of the table that we want to input if there are several at the given address/path

Returns table as numpy.array

tabledataextractor.input.from_any.**csv**(*name*)
Returns *True* if input is *csv* file.

Parameters **name** (*str*) – Input string

tabledataextractor.input.from_any.**html**(*name*)
Returns *True* if input is *html* file.

Parameters **name** (*str*) – Input string

tabledataextractor.input.from_any.**url**(*name*)
Returns *True* if input is *URL*. Uses `django.core.validators.URLValidator`.

Parameters **name** (*str*) – Input string

4.2.2 From .csv file

Reads a *csv* formatted table from file. The file has to be 'utf-8' encoded.

tabledataextractor.input.from_csv.**read**(*file_path*)

Parameters **file_path** (*str*) – Path to *.csv* input file

Returns numpy.ndarray

4.2.3 From .html file

Reads an *html* formatted table.

tabledataextractor.input.from_html.**configure_selenium**(*browser='Firefox'*)
Configuration for Selenium. Sets the path to `geckodriver.exe`

Parameters **browser** (*str*) – Which browser to use

Returns Selenium driver

tabledataextractor.input.from_html.**makearray**(*html_table*)
Creates a numpy array from an *.html* file, taking *rowspan* and *colspan* into account.

Modified from: John Ricco, https://johnricco.github.io/2017/04/04/python-html/, *Using Python to scrape HTML tables with merged cells*

Added functionality for duplicating cell content for cells with *rowspan/colspan*. The table has to be $n * m$, rectangular, with the same number of columns in every row.

tabledataextractor.input.from_html.**read_file**(*file_path*, *table_number=1*)
Reads an .html file and returns a numpy array.

tabledataextractor.input.from_html.**read_url**(*url*, *table_number=1*)
Reads in a table from an URL and returns a numpy array. Will try Requests first. If it doesn't succeed, Selenium will be used.

Parameters

- **url** (*str*) – Url of the page where the table is located
- **table_number** (*int*) – Number of Table on the web page.

4.2.4 From Python List

Inputs from python list object.

tabledataextractor.input.from_list.**read**(*plist*)
Creates a numpy array from a Python list. Works if rows are of different length.

Parameters **plist** (*list*) – Input List

Returns numpy.ndarray

4.3 Output

4.3.1 Print to screen

Functions for printing to screen in a nice format.

tabledataextractor.output.print.**as_string**(*table*)
Returns table as string for printing.

Parameters **table** (*numpy.array*) – input numpy array

Returns string

tabledataextractor.output.print.**list_as_PrettyTable**(*table_list*)
Turns list into PrettyTable object, ready for printing.

Parameters **table_list** – list to be printed

Returns table as PrettyTable

tabledataextractor.output.print.**print_table**(*table*)
Prints a table to screen.

Parameters **table** (*numpy.array*) – input numpy array for printing

4.3.2 Save to file

Outputs the table to cvs.

tabledataextractor.output.to_csv.**write_to_csv**(*table*, *file_path*)
Writes a numpy array table to a .csv file. Overrides existing files.

Parameters

- **table** (*ndarray*) – Array of table data
- **file_path** (*str*) – Output location

4.3.3 Convert to Pandas DataFrame

Outputs the table to a Pandas DataFrame.

tabledataextractor.output.to_pandas.**build_category_table**(*df*)
Builds the category table in form of a Python list, from *Pandas DataFrame* input

Parameters **df** (*pandas.DataFrame*) – Pandas DataFrame input

Returns category_table as Python list

tabledataextractor.output.to_pandas.**find_multiindex_level**(*row_number*, *column_number*, *df*)
Helping function for `_build_category_table()`. Finds the *Pandas MultiIndex level* in a given *Pandas DataFrame*, for a particular data value.

tabledataextractor.output.to_pandas.**print_category_table**(*df*)
Prints the category table to screen, from *Pandas DataFrame* input

Parameters **df** (*pandas.DataFrame*) – Pandas DataFrame input

tabledataextractor.output.to_pandas.**to_pandas**(*table*)
Creates a Pandas *DataFrame* object from a *Table* object.

Parameters **table** (*Table*) – Input table

Returns pandas.DataFrame

4.4 History

Indicates to the user which methods have been used on the table. This should be checked for testing on a sample dataset, to justify the choice of settings for the given domain.

class tabledataextractor.table.history.**History**
Stores *True*/*False* for each property, indicating if a method has been used on the particular *Table* instance.

footnotes_copied
Indicates whether footnotes have been copied into the table cells.

header_extended_down
Indicates whether the header has been extended downwards, beyond the result obtained by the MIPS (*Minimum Indexing Point Search*) algorithm.

header_extended_up
Indicates whether the header has been extended upwards, beyond the result obtained by the MIPS (*Minimum Indexing Point Search*) algorithm.

prefixed_rows
Indicates whether prefixing has been performed on the rows (left side).

prefixing_performed
Indicates whether prefixing has been performed on the table.

spanning_cells_extended
Indicates whether the content of cells has been duplicated into neighbouring cells, in case of cells that are merged cells (spanning cells).

table_transposed
Indicates whether the table has been transposed.

title_row_removed
Indicates whether a title row has been removed from the table.

4.5 Footnotes

Footnote handling.

class tabledataextractor.table.footnotes.**Footnote**(*table*, *prefix*, *prefix_cell*, *text*)
Defines a footnote found in the provided table. Contains elements of a footnote. Will construct the footnote and find all associated elements.

Parameters

- **table** (Table) – table to work on
- **prefix** (*str*) – Prefix that has been identified as footnote prefix
- **prefix_cell** ((*int, int*)) – Index of the cell containing the associated prefix
- **text** (*str*) – Optional. Text associated with the found footnote prefix

prefix = None
Prefix string, e.g., *"a)"*.

prefix_cell = None
Cell index of the prefix, e.g., *(7,0)*.

reference_cells = None
Cell indexes of the cells containing the footnote references within the table.

references = None
Cell content of the cells contatining the footnote references within the table.

text = None
Footnote text, e.g., *"This is the text of a footnote"*.

text_cell = None
Cell of the footnote text, e.g., *(7,1)*.

tabledataextractor.table.footnotes.**find_footnotes**(*table_object*)
Finds a footnote and yields a *Footnote* object with all the appropriate properties. A footnote is defined with:

```
FNprefix  = \*, #, ., o, †; possibly followed by "." or ")"
```

A search is performed only below the data region.

Parameters **table_object** (Table) – Input Table object

4.6 Algorithms

Algorithms for TableDataExtractor.

tabledataextractor.table.algorithms.**build_category_table**(*table*, *cc1*, *cc2*, *cc3*, *cc4*)
Build category table for given input table. Original header factorization, according to Embley et al., *DOI: 10.1007/s10032-016-0259-1*. This version is not used, instead *build_category_table* is being used.

Parameters

- **table** (*Numpy array*) – Table on which to perform the categorization
- **cc1** – key MIPS cell
- **cc2** – key MIPS cell
- **cc3** – key MIPS cell
- **cc4** – key MIPS cell

Returns category table as numpy array

tabledataextractor.table.algorithms.**categorize_header**(*header*)
Performs header categorization (calls the *SymPy fact* function) for a given table.

Parameters **header** – header region, Numpy array

Returns factor_list

tabledataextractor.table.algorithms.**clean_row_header**(*pre_cleaned_table*, *cc2*)
Cleans the row header by removing duplicate rows that span the whole table.

tabledataextractor.table.algorithms.**clean_unicode**(*array*)
Replaces problematic unicode characters in a given numpy array. :param array: input array :type array: numpy.array :return: cleaned array

tabledataextractor.table.algorithms.**duplicate_columns**(*table*)
Returns True if there are duplicate columns in the table and False if there are no duplicate columns :param table: :return: True or False

tabledataextractor.table.algorithms.**duplicate_rows**(*table*)
Returns True if there are duplicate rows in the table and False if there are no duplicate rows :param table: :return: True or False

tabledataextractor.table.algorithms.**duplicate_spanning_cells**(*table_object*, *array*)
Duplicates cell contents into appropriate spanning cells. This is sometimes necessary for *.csv* files where information has been lost, or, if the source table is not properly formatted.

Cells outside the row/column header (such as data cells) will not be duplicated. MIPS is run to perform a check for that.

Algorithm according to Nagy and Seth, 2016, in Procs. ICPR 2016, Cancun, Mexico.

Parameters

- **table_object** (Table) – Input Table object
- **array** (*Numpy array*) – Table to use as input

Returns Array with spanning cells copied, if necessary. Alternatively, returns the original table.

tabledataextractor.table.algorithms.**empty_cells**(*array*, *regex='^([\\s\\-\\–\\—\\"]+)?$'*)
Returns a mask with *True* for all empty cells in the original array and *False* for non-empty cells.

Parameters

- **regex** (*str*) – The regular expression which defines an empty cell (can be tweaked).
- **array** (*numpy array*) – Input array to return the mask for

tabledataextractor.table.algorithms.**empty_string**(*string*, *regex='^([\\s\\-\\–\\—\\"]+)?$'*)
Returns *True* if a particular string is empty, which is defined with a regular expression.

Parameters

- **string** (*str*) – Input string for testing
- **regex** (*str*) – The regular expression which defines an empty cell (can be tweaked).

Returns True/False

tabledataextractor.table.algorithms.**find_cc1_cc2**(*table_object*, *cc4*, *array*)
Main MIPS (*Minimum Indexing Point Search*) algorithm. According to Embley et al., *DOI: 10.1007/s10032-016-0259-1*. Searches for critical cells *CC2* and *CC3*. MIPS locates the critical cells that define the minimum row and column headers needed to index every data cell.

Parameters

- **table_object** (Table) – Input Table object
- **cc4** ((int, int)) – Position of *CC4* cell found with find_cc4()
- **array** (numpy array) – table to search for *CC1* and *CC2*

Returns cc1, cc2

tabledataextractor.table.algorithms.**find_cc3**(*table_object*, *cc2*)

Searches for critical cell *CC3*, as the leftmost cell of the first filled row of the data region.

Comment on implementation

There are two options on how to implement the search for *CC3*:

1. **With the possibility of *Notes* rows directly below the header (default):**
   - the first half filled row below the header is considered as the start of the data region, just like for the *CC4* cell
   - implemented by Embley et. al.
2. **Without the possibility of *Notes* rows directly below the header:**
   - the first row below the header is considered as the start of the data region
   - for scientific tables it might be more common that the first data row only has a single entry
   - this can be chosen my commenting/uncommenting the code within this function

Parameters

- **table_object** (Table) – Input Table object
- **cc2** ((int,int)) – Tuple, position of *CC2* cell found with find_cc1_cc2()

Returns cc3

tabledataextractor.table.algorithms.**find_cc4**(*table_object*)

Searches for critical cell *CC4*.

Searching from the bottom of the pre-cleaned table for the last row with a minority of empty cells. Rows with at most a few empty cells are assumed to be part of the data region rather than notes or footnotes rows (which usually only have one or two non-empty cells).

Parameters **table_object** (Table) – Input Table object

Returns cc4

tabledataextractor.table.algorithms.**find_note_cells**(*table_object*, *labels_table*)

Searches for all non-empty cells that have not been labelled differently.

Parameters

- **table_object** (Table) – Input Table object
- **labels_table** (Numpy array) – table that holds all the labels

Returns Tuple

tabledataextractor.table.algorithms.**find_row_header_table**(*category_table*, *stub_header*)

Constructs a Table from the row categories of the original table.

Parameters

- **category_table** (*list*) – ~tabledataextractor.table.table.Table.category_table
- **stub_header** (*numpy.ndarray*) – ~tabledataextractor.table.table.Table.stub_header

Returns list

tabledataextractor.table.algorithms.**find_title_row**(*table_object*)

Searches for the topmost non-empty row.

Parameters **table_object** (Table) – Input Table object

Returns int

tabledataextractor.table.algorithms.**header_extension_down**(*table_object*, *cc1*, *cc2*, *cc4*)

Extends the header downwards, if no prefixing was done and if the appropriate stub header is empty. For column-header expansion downwards, only the first cell of the stub header has to be empty. For row-header expansion to the right, the whole stub header column above has to be empty.

Parameters

- **table_object** (Table) – Input Table object
- **cc2** ((*int, int*)) – Critical cell *CC2*
- **cc1** ((*int, int*)) – Critical cell *CC1*
- **cc4** ((*int, int*)) – Critical cell *CC4*

Returns New *cc2*

tabledataextractor.table.algorithms.**header_extension_up**(*table_object*, *cc1*)

Extends the header after main MIPS run.

Algorithm according to Nagy and Seth, 2016, *"Table Headers: An entrance to the data mine"*, in Procs. ICPR 2016, Cancun, Mexico.

Parameters

- **table_object** (Table) – Input Table object
- **cc1** – *CC1* critical cell

Returns cc1_new

tabledataextractor.table.algorithms.**pre_clean**(*array*)

Removes empty and duplicate rows and columns that extend over the whole table.

Parameters **array** (*Numpy array*) – Input Table object

tabledataextractor.table.algorithms.**prefix_duplicate_labels**(*table_object*, *array*)

Prefixes duplicate labels in first row or column where this is possible, by adding a new row/column containing the preceding (to the left or above) unique labels, if available.

Nested prefixing is not supported.

The algorithm is not completely selective and there might be cases where it's application is undesirable. However, on standard datasets it significantly improves table-region classification.

Algorithm for column headers:

1. Run MIPS, to find the old header region, without prefixing.
2. **For row in table, can *meaningful* prefixing in this row been done?**

- yes –> do prefixing and go to 3, prefixing of only one row is possible; accept prefixing only if prefixed rows/cells are above the end of the header (not in the data region), the prefixed cells can still be above the header
- no –> go to 2, next row

3. run MIPS to get the new header region
4. accept prefixing only if the prefixing has not made the header region start lower than before and if it hasn't made the header region wider than before

The algorithm has been modified from Embley et al., *DOI: 10.1007/s10032-016-0259-1*.

Parameters

- **table_object** (Table) – Input Table object
- **array** (*Numpy array*) – Table to use as input and to do the prefixing on

Returns Table with added rows/columns with prefixes, or, input table, if no prefixing was done

tabledataextractor.table.algorithms.**split_table**(*table_object*)

Splits table into subtables. Yields *Table* objects.

Algorithm: If the stub header is repeated in the column header section the table is split up before the repeated element.

Parameters **table_object** (Table) – Input Table object

tabledataextractor.table.algorithms.**standardize_empty**(*array*)

Returns an array with the empty cells of the input array standardized to 'NoValue'.

Parameters **array** (*numpy.array*) – Input array

Returns Array with standardized empty cells

4.7 Cell Parser

Tools for parsing the table based on regular expressions.

class tabledataextractor.table.parse.**CellParser**(*pattern*)

Parameters **pattern** (*str*) – Regular expression pattern which defines the cell parser. Use *grouping*, since matching strings will be returned explicitly.

cut(*table*, *method='match'*)

Inputs a table and yields a tuple with the index of the next matching cell, as well as a string that is obtained from the original string by cutting out the match string.

Parameters

- **method** (*str*) – *search*, *match* or *fullmatch*; see Python Regular expressions
- **table** (*numpy.array*) – Input table to be parsed, of type 'numpy.ndarray'

Yield (int, int, str) with index of cells and the strings of the groups that were matched

parse(*table*, *method='match'*)

Inputs a table and yields a tuple with the index of the next matching cell, as well as the string that was matched.

Parameters

- **method** (*str*) – *search*, *match* or *fullmatch*; Python Regular expressions
- **table** (*numpy.array*) – Input table to be parsed

Yield (int, int, str) with index of cells and the strings of the groups that were matched

replace(*table*, *repl*, *method='match'*)
Inputs a table and yields a tuple with the index of the next matching cell, as well as a string that is obtained from the original string by cutting out the match string and replacing it with another string.

Parameters

- **method** (*str*) – *search*, *match* or *fullmatch*; see Python Regular expressions
- **table** (*numpy.array*) – Input table to be parsed
- **repl** (*str*) – Replacement string that will be included instead of the patters

Yield (int, int, str) with index of cells and the strings of the groups that were matched

class tabledataextractor.table.parse.**StringParser**(*pattern*)

Parameters **pattern** (*str*) – Regular expression pattern that defines the string parser.

cut(*string*)
Inputs a string and returns the same string with the pattern cut out

Parameters **string** (*str*) – Input string

Returns string with *pattern* cut out

parse(*string*, *method='match'*)
Inputs a string and returns *True* if pattern matches.

Parameters

- **string** (*str*) – Input string
- **method** (*str*) – *search*, *match* or *fullmatch*; see Python Regular expressions

Returns True/False

4.8 Exceptions

Exceptions defined for TableDataExtractor.

exception tabledataextractor.exceptions.**InputError**(*message*)
Exception raised for errors in the input.

exception tabledataextractor.exceptions.**MIPSError**(*message*)
Exception raised for failure of the main MIPS algorithm. Usually signals that the table is broken or not well structured.

exception tabledataextractor.exceptions.**TDEError**
Base class for exceptions in TableDataExtractor.

CHAPTER 5

License

The MIT License (MIT)

CHAPTER 6

Acknowledgements

TableDataExtractor
Molecular Engineering Group
Cavendish Laboratory,
University of Cambridge

This software was written by Juraj Mavračić supported by a grant from the UK Engineering and Physical Sciences Research Council (EPSRC) EP/L015552/1 for the Centre for Doctoral Training (CDT) in Computational Methods for Materials Science.

It forms part of a PhD project supervised by Jacqueline M. Cole and Stephen R. Elliott. The PhD was completed in the Molecular Engineering Group at the University of Cambridge, which is financially supported by the Engineering and Physical Sciences Research Council (EPSRC, EP/L015552/1), Science and Technology Facilities Council (STFC) and the Royal Academy of Engineering (RCSRF1819710).

Core algorithms used and modified in *TableDataExtractor* have originally been developed by Embley et al. This is the MIPS (*Minimum Indexing Point Search*) algorithm that is used to find the row/column headers and the data region, as well as algorithms for prefixing header cells. Also, some of the examples in this documentation are based on examples from Embley et al:

> Embley, D.W., Krishnamoorthy, M.S., Nagy, G., and Seth, S. (2016) Converting heterogeneous statistical tables on the web to searchable databases. *Int. J. Doc. Anal. Recognit.*, *19* (2), 119–138.

Algorithms for duplicating spanning cells and extending headers, that are used in *TableDataExtractor*, have been developed by Nagy and Seth:

> Nagy, G., and Seth, S. (2017) Table headers: An entrance to the data mine. *Proc. - Int. Conf. Pattern Recognit.*, 4065–4070.

The algorithm for converting *html* files to *Numpy arrays* has been modified from John Rico:

> John Ricco, (2017) Using Python to scrape HTML tables with merged cells, https://johnricco.github.io/2017/04/04/python-html/

Please cite these works where appropriate.

CHAPTER 7

Indices and tables

- genindex
- modindex

Python Module Index

t

Index

I

L

M

P

R

S

T

U

W