Camelot Documentation

Release 0.7.2

Vinayak Mehta

Mar 08, 2019

Contents

1 Why Camelot? 3

2 The User Guide 5
- 2.1 Introduction . . . 5
- 2.2 Installation of dependencies . . . 6
- 2.3 Installation of Camelot . . . 7
- 2.4 How It Works . . . 8
- 2.5 Quickstart . . . 11
- 2.6 Advanced Usage . . . 14
- 2.7 Command-Line Interface . . . 44

3 The API Documentation/Guide 45
- 3.1 API Reference . . . 45

4 The Contributor Guide 53
- 4.1 Contributor's Guide . . . 53

Python Module Index 57

Release v0.7.2. (*Installation*) chat on gitter **Camelot** is a Python library that makes it easy for *anyone* to extract tables from PDF files!

Note: You can also check out Excalibur, which is a web interface for Camelot!

Here's how you can extract tables from PDF files. Check out the PDF used in this example here.

```
>>> import camelot
>>> tables = camelot.read_pdf('foo.pdf')
>>> tables
<TableList n=1>
>>> tables.export('foo.csv', f='csv', compress=True) # json, excel, html
>>> tables[0]
<Table shape=(7, 7)>
>>> tables[0].parsing_report
{
    'accuracy': 99.02,
    'whitespace': 12.24,
    'order': 1,
    'page': 1
}
>>> tables[0].to_csv('foo.csv') # to_json, to_excel, to_html
>>> tables[0].df # get a pandas DataFrame!
```

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

There's a *command-line interface* too!

Note: Camelot only works with text-based PDFs and not scanned documents. (As Tabula explains, "If you can click and drag to select text in your table in a PDF viewer, then your PDF is text-based".)

CHAPTER 1

Why Camelot?

- **You are in control.** Unlike other libraries and tools which either give a nice output or fail miserably (with no in-between), Camelot gives you the power to tweak table extraction. (This is important since everything in the real world, including PDF table extraction, is fuzzy.)
- *Bad* tables can be discarded based on **metrics** like accuracy and whitespace, without ever having to manually look at each table.
- Each table is a **pandas DataFrame**, which seamlessly integrates into ETL and data analysis workflows.
- **Export** to multiple formats, including JSON, Excel and HTML.

See comparison with other PDF table extraction libraries and tools.

CHAPTER 2

The User Guide

This part of the documentation begins with some background information about why Camelot was created, takes a small dip into the implementation details and then focuses on step-by-step instructions for getting the most out of Camelot.

2.1 Introduction

2.1.1 The Camelot Project

The PDF (Portable Document Format) was born out of The Camelot Project to create "a universal way to communicate documents across a wide variety of machine configurations, operating systems and communication networks". The goal was to make these documents viewable on any display and printable on any modern printers. The invention of the PostScript page description language, which enabled the creation of *fixed-layout* flat documents (with text, fonts, graphics, images encapsulated), solved this problem.

At a high level, PostScript defines instructions, such as "place this character at this *x,y* coordinate on a plane". Spaces can be *simulated* by placing characters relatively far apart. Extending from that, tables can be *simulated* by placing characters (which constitute words) in two-dimensional grids. A PDF viewer just takes these instructions and draws everything for the user to view. Since a PDF is just characters on a plane, there is no table data structure that can be extracted and used for analysis!

Sadly, a lot of today's open data is trapped in PDF tables.

2.1.2 Why another PDF table extraction library?

There are both open (Tabula, pdf-table-extract) and closed-source (smallpdf, PDFTables) tools that are widely used to extract tables from PDF files. They either give a nice output or fail miserably. There is no in between. This is not helpful since everything in the real world, including PDF table extraction, is fuzzy. This leads to the creation of ad-hoc table extraction scripts for each type of PDF table.

Camelot was created to offer users complete control over table extraction. If you can't get your desired output with the default settings, you can tweak them and get the job done!

Here is a comparison of Camelot's output with outputs from other open-source PDF parsing libraries and tools.

2.1.3 What's in a name?

As you can already guess, this library is named after The Camelot Project.

Fun fact: In the British comedy film Monty Python and the Holy Grail (and in the Arthurian legend depicted in the film), "Camelot" is the name of the castle where Arthur leads his men, the Knights of the Round Table, and then sets off elsewhere after deciding that it is "a silly place". Interestingly, the language in which this library is written (Python) was named after Monty Python.

2.1.4 Camelot License

MIT License

Copyright (c) 2018 Peeply Private Ltd (Singapore)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

2.2 Installation of dependencies

The dependencies Tkinter and ghostscript can be installed using your system's package manager. You can run one of the following, based on your OS.

2.2.1 OS-specific instructions

For Ubuntu

```
$ apt install python-tk ghostscript
```

Or for Python 3:

```
$ apt install python3-tk ghostscript
```

For macOS

```
$ brew install tcl-tk ghostscript
```

For Windows

For Tkinter, you can download the ActiveTcl Community Edition from ActiveState. For ghostscript, you can get the installer at the ghostscript downloads page.

2.2.2 Checks to see if dependencies were installed correctly

You can do the following checks to see if the dependencies were installed correctly.

For Tkinter

Launch Python, and then at the prompt, type:

```
>>> import Tkinter
```

Or in Python 3:

```
>>> import tkinter
```

If you have Tkinter, Python will not print an error message, and if not, you will see an `ImportError`.

For ghostscript

Run the following to check the ghostscript version.

For Ubuntu/macOS:

```
$ gs -version
```

For Windows:

```
C:\> gswin64c.exe -version
```

Or for Windows 32-bit:

```
C:\> gswin32c.exe -version
```

If you have ghostscript, you should see the ghostscript version and copyright information.

2.3 Installation of Camelot

This part of the documentation covers the steps to install Camelot.

2.3.1 Using conda

The easiest way to install Camelot is to install it with conda, which is a package manager and environment management system for the Anaconda distribution.

```
$ conda install -c conda-forge camelot-py
```

Note: Camelot is available for Python 2.7, 3.5 and 3.6 on Linux, macOS and Windows. For Windows, you will need to install ghostscript which you can get from their downloads page.

2.3.2 Using pip

After *installing the dependencies*, which include Tkinter and ghostscript, you can simply use pip to install Camelot:

```
$ pip install camelot-py[cv]
```

2.3.3 From the source code

After *installing the dependencies*, you can install from the source by:

1. Cloning the GitHub repository.

```
$ git clone https://www.github.com/socialcopsdev/camelot
```

2. Then simply using pip again.

```
$ cd camelot
$ pip install ".[cv]"
```

2.4 How It Works

This part of the documentation includes a high-level explanation of how Camelot extracts tables from PDF files.

You can choose between two table parsing methods, *Stream* and *Lattice*. These names for parsing methods inside Camelot were inspired from Tabula.

2.4.1 Stream

Stream can be used to parse tables that have whitespaces between cells to simulate a table structure. It is built on top of PDFMiner's functionality of grouping characters on a page into words and sentences, using margins.

1. Words on the PDF page are grouped into text rows based on their y axis overlaps.
2. Textedges are calculated and then used to guess interesting table areas on the PDF page. You can read Anssi Nurminen's master's thesis to know more about this table detection technique. [See pages 20, 35 and 40]
3. The number of columns inside each table area are then guessed. This is done by calculating the mode of number of words in each text row. Based on this mode, words in each text row are chosen to calculate a list of column x ranges.

4. Words that lie inside/outside the current column *x* ranges are then used to extend extend the current list of columns.
5. Finally, a table is formed using the text rows' *y* ranges and column *x* ranges and words found on the page are assigned to the table's cells based on their *x* and *y* coordinates.

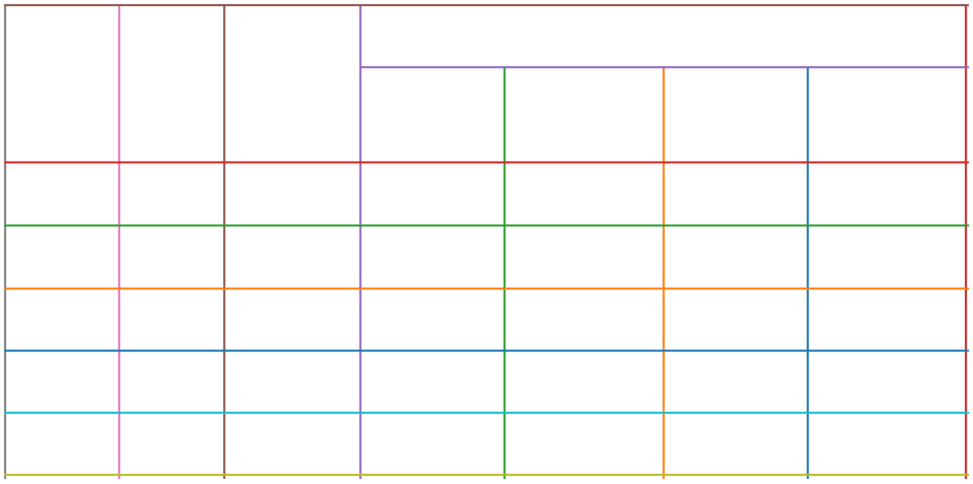
2.4.2 Lattice

Lattice is more deterministic in nature, and it does not rely on guesses. It can be used to parse tables that have demarcated lines between cells, and it can automatically parse multiple tables present on a page.

It starts by converting the PDF page to an image using ghostscript, and then processes it to get horizontal and vertical line segments by applying a set of morphological transformations (erosion and dilation) using OpenCV.

Let's see how Lattice processes the second page of this PDF, step-by-step.

1. Line segments are detected.



2. Line intersections are detected, by overlapping the detected line segments and "and"ing their pixel intensities.

Table 2-1. Simulated fuel savings from isolated cycle improvements

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

2-1 extends the analysis from eliminating stops for the five example cycles and exam

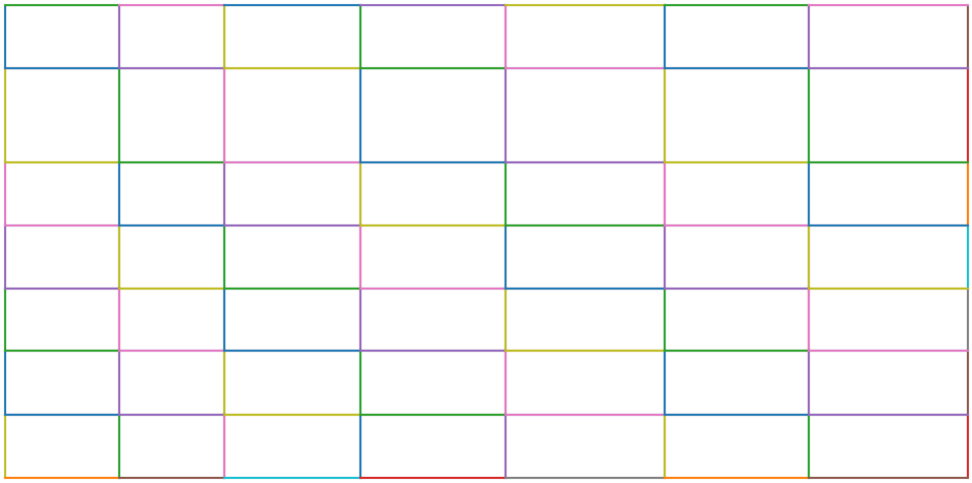
3. Table boundaries are computed by overlapping the detected line segments again, this time by "or"ing their pixel intensities.

Table 2-1. Simulated fuel savings from isolated cycle improvements

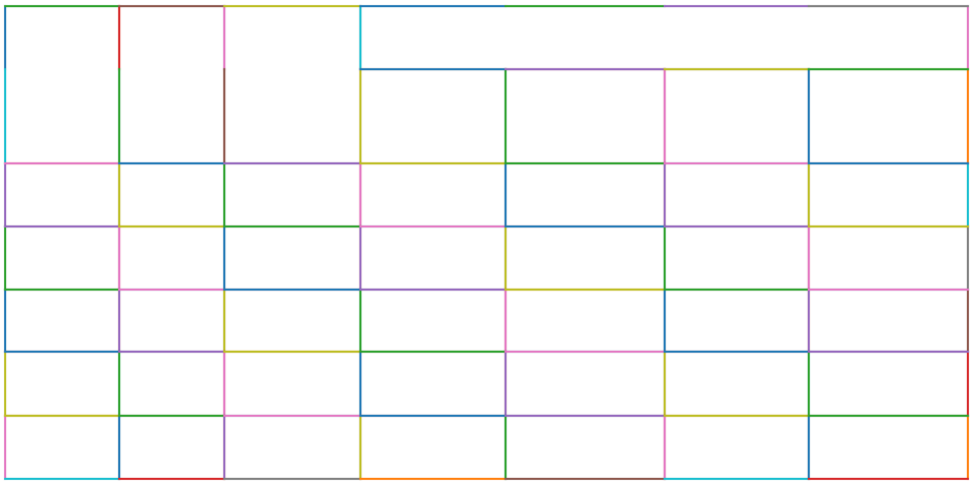
Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

2-1 extends the analysis from eliminating stops for the five example cycles and exam

4. Since dimensions of the PDF page and its image vary, the detected table boundaries, line intersections, and line segments are scaled and translated to the PDF page's coordinate space, and a representation of the table is created.



5. Spanning cells are detected using the line segments and line intersections.



6. Finally, the words found on the page are assigned to the table's cells based on their *x* and *y* coordinates.

2.5 Quickstart

In a hurry to extract tables from PDFs? This document gives a good introduction to help you get started with Camelot.

2.5.1 Read the PDF

Reading a PDF to extract tables with Camelot is very simple.

Begin by importing the Camelot module:

```
>>> import camelot
```

Now, let's try to read a PDF. (You can check out the PDF used in this example here.) Since the PDF has a table with clearly demarcated lines, we will use the *Lattice* method here.

Note: *Lattice* is used by default. You can use *Stream* with `flavor='stream'`.

```
>>> tables = camelot.read_pdf('foo.pdf')
>>> tables
<TableList n=1>
```

Now, we have a *TableList* object called `tables`, which is a list of *Table* objects. We can get everything we need from this object.

We can access each table using its index. From the code snippet above, we can see that the `tables` object has only one table, since `n=1`. Let's access the table using the index `0` and take a look at its `shape`.

```
>>> tables[0]
<Table shape=(7, 7)>
```

Let's print the parsing report.

```
>>> print tables[0].parsing_report
{
    'accuracy': 99.02,
    'whitespace': 12.24,
    'order': 1,
    'page': 1
}
```

Woah! The accuracy is top-notch and there is less whitespace, which means the table was most likely extracted correctly. You can access the table as a pandas DataFrame by using the *table* object's `df` property.

```
>>> tables[0].df
```

Cycle Name	KI (1/km)	Distance (mi)	Percent Fuel Savings			
			Improved Speed	Decreased Accel	Eliminate Stops	Decreased Idle
2012_2	3.30	1.3	5.9%	9.5%	29.2%	17.4%
2145_1	0.68	11.2	2.4%	0.1%	9.5%	2.7%
4234_1	0.59	58.7	8.5%	1.3%	8.5%	3.3%
2032_2	0.17	57.8	21.7%	0.3%	2.7%	1.2%
4171_1	0.07	173.9	58.1%	1.6%	2.1%	0.5%

Looks good! You can now export the table as a CSV file using its *to_csv()* method. Alternatively you can use *to_json()*, *to_excel()* *to_html()* or *to_sqlite()* methods to export the table as JSON, Excel, HTML files or a sqlite database respectively.

```
>>> tables[0].to_csv('foo.csv')
```

This will export the table as a CSV file at the path specified. In this case, it is `foo.csv` in the current directory.

You can also export all tables at once, using the *`tables`* object's *`export()`* method.

```
>>> tables.export('foo.csv', f='csv')
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot --format csv --output foo.csv lattice foo.pdf
```

This will export all tables as CSV files at the path specified. Alternatively, you can use `f='json'`, `f='excel'`, `f='html'` or `f='sqlite'`.

Note: The *`export()`* method exports files with a `page-*-table-*` suffix. In the example above, the single table in the list will be exported to `foo-page-1-table-1.csv`. If the list contains multiple tables, multiple CSV files will be created. To avoid filling up your path with multiple files, you can use `compress=True`, which will create a single ZIP file at your path with all the CSV files.

Note: Camelot handles rotated PDF pages automatically. As an exercise, try to extract the table out of this PDF.

2.5.2 Specify page numbers

By default, Camelot only uses the first page of the PDF to extract tables. To specify multiple pages, you can use the `pages` keyword argument:

```
>>> camelot.read_pdf('your.pdf', pages='1,2,3')
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot --pages 1,2,3 lattice your.pdf
```

The `pages` keyword argument accepts pages as comma-separated string of page numbers. You can also specify page ranges — for example, `pages=1,4-10,20-30` or `pages=1,4-10,20-end`.

2.5.3 Reading encrypted PDFs

To extract tables from encrypted PDF files you must provide a password when calling *`read_pdf()`*.

```
>>> tables = camelot.read_pdf('foo.pdf', password='userpass')
>>> tables
<TableList n=1>
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot --password userpass lattice foo.pdf
```

Currently Camelot only supports PDFs encrypted with ASCII passwords and algorithm code 1 or 2. An exception is thrown if the PDF cannot be read. This may be due to no password being provided, an incorrect password, or an unsupported encryption algorithm.

Further encryption support may be added in future, however in the meantime if your PDF files are using unsupported encryption algorithms you are advised to remove encryption before calling *read_pdf()*. This can been successfully achieved with third-party tools such as QPDF.

```
$ qpdf --password=<PASSWORD> --decrypt input.pdf output.pdf
```

Ready for more? Check out the *advanced* section.

2.6 Advanced Usage

This page covers some of the more advanced configurations for *Lattice* and *Stream*.

2.6.1 Process background lines

To detect line segments, *Lattice* needs the lines that make the table to be in the foreground. Here's an example of a table with lines in the background:

Table 6.1: The highlights of RRE-II coverage (till 11 March, 2010)

State	Date	Halt stations	Halt days	Persons directly reached (in lakh)	Persons trained	Persons counseled	Persons tested for HIV
Delhi	1.12.2009	8	17	1.29	3,665	2,409	1,000
Rajasthan	2.12.2009 to 19.12.2009						
Gujarat	20.12.2009 to 3.1.2010	6	13	6.03	3,810	2,317	1,453
Maharashtra	4.01.2010 to 1.2.2010	13	26	1.27	5,680	9,027	4,153
Karnataka	2.2.2010 to 22.2.2010	11	19	1.80	5,741	3,658	3,183
Kerala	23.2.2010 to 11.3.2010	9	17	1.42	3,559	2,173	855
Total		**47**	**92**	**11.81**	**22,455**	**19,584**	**10,644**

It includes visitors to train exhibition and those reached through outreach activities

Source: PDF

To process background lines, you can pass `process_background=Tru`

```
↪'background_
↪lines.pdf
↪',␣
↪process_
↪background=True)
```

(continues on next page)

(continued from previous page)

Tip: Here's how you can do the same with the *command-line interface*.

↪background_
↪lines.pdf

State	Date	Health sta-tions	Health days	Per-sons di-rectly reached	Per-sons trained	Per-sons code-selected (in lakh)	Per-sons test-ed for HIV
Delhi	1.18.2009	1	1	1.29	9,625	40,000	
Ra-jasthan	2.12.2009 to 19.12.2009						
Gu-jarat	20.12.2009 to 3.1.2010	1	1	0.903	3,820	17,453	
Ma-ha-rash-tra	4.01.2010 to 1-2.2010	1	1	1.25	7,680	47,153	
Kar-natta	2.21.2010 to 22.2.2010	1	1	1.80	5,741	58,183	
Ker-ala	23.2.2010 to 11.3.2010	1	1	1.43	3,529	18,355	
To-tal		47	92	11.82	1,495	58,644	

2.6.2 Visual debugging

Note: Visual debugging using `plot()` requires matplotlib which is an optional dependency. You can install it using `$ pip install camelot-py[plot]`.

You can use the `plot()` method to generate a matplotlib plot of various elements that were detected on the PDF page while processing it. This can help you select table areas, column separators and debug bad table outputs, by tweaking

different configuration parameters.

You can specify the type of element you want to plot using the `kind` keyword argument. The generated plot can be saved to a file by passing a `filename` keyword argument. The following plot types are supported:

- 'text'
- 'grid'
- 'contour'
- 'line'
- 'joint'
- 'textedge'

Note: 'line' and 'joint' can only be used with *Lattice* and 'textedge' can only be used with *Stream*.

Let's generate a plot for each type using this PDF as an example. First, let's get all the tables out.

```
>>> tables = camelot.read_pdf('foo.pdf')
>>> tables
<TableList n=1>
```

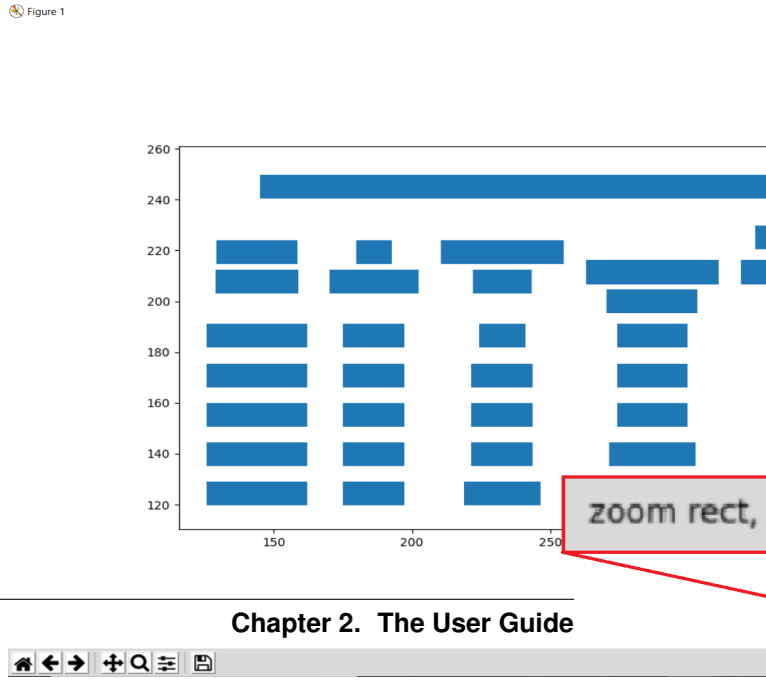
text

Let's plot all the text present on the table's PDF page.

```
>>> camelot.plot(tables[0], kind='text')
>>> plt.show()
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -plot text foo.pdf
```



rators, in case Stream does not guess them correctly.

Note: The *x-y* coordinates shown above change as you move your mouse cursor on the image, which can help you note coordinates.

table

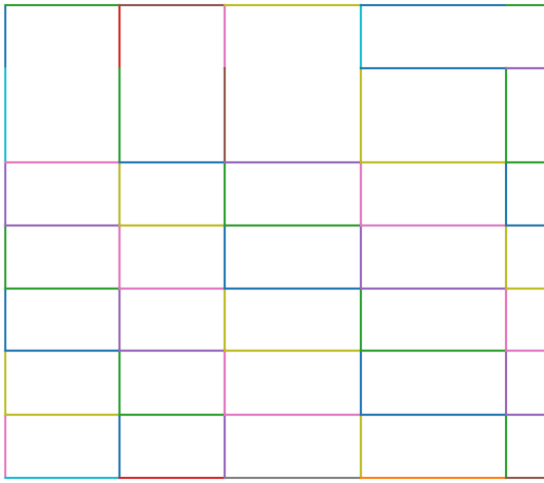
Let's plot the table (to see if it was detected correctly or not). This plot type, along with contour, line and joint is useful for debugging and improving the extraction output, in case the table wasn't detected correctly. (More on that later.)

```
>>> camelot.plot(tables[0], kind='grid')
>>> plt.show()
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -plot grid foo.pdf
```

contour



Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -plot contour foo.pdf
```

line

(continues on next page)

(continued from previous page)

Tip: Here's how you can do the same with the *command-line interface*.

joint

Tip: Here's how you can do the same with the *command-line interface*.

textedge

a “textedge” is, you can see pages 20, 35 and 40 of Anssi Nurminen’s master’s thesis.

(continues on next page)

(continued from previous page)

Tip: Here's how you can do the same with the *command-line interface*.

can plot the text on this page and note the top left and bottom right coordinates of the table.

strings to *read_pdf()*, using the `table_areas` keyword argument.

```
↪'316,499,566,337'])
```

Tip: Here's how you can do the same with the *command-line interface*.

(continues on next page)

(continued from previous page)

	One Withholding
Payroll Period	Allowance
Weekly	$71.15
Biweekly	142.31
Semimonthly	154.17
Monthly	308.33
Quarterly	925.00
Semiannually	1,850.00
Annually	3,700.00
Daily or Miscellaneous	14.23
(each day of the payroll period)	

coordinates every time but in an approximate region.

specified, Camelot will only analyze the specified regions to look for tables.

```
↪'])
```

Tip: Here's how you can do the same with the *command-line interface*.

(continues on next page)

(continued from previous page)

Età dell'Assicuratoall'epoca del decesso	Misura % dimaggiorazione
18-75	1,00%
76-80	0,50%
81 in poi	0,10%

may guess the column separators' coordinates incorrectly. To correct this, you can explicitly specify the *x* coordinate for each column separator by plotting the text on the page.

ing the `columns` keyword argument.

is specified, the separators will be applied to the whole page. When a list of table areas is specified and you need to specify column separators as well, **the length of both lists should be equal**. Each table area will be mapped to each column separators' string using their indices.

`'20,67,55,33']`, and only want to specify column separators for the first table, you can pass an empty string for

the second table in the column separators' list like this, `columns=['10,120,200,400', '']`.

on this PDF, and get the table out!

```
↪95,209,327,442,529,566,606,683'])
```

Tip: Here's how you can do the same with the *command-line interface*.

(continues on next page)

(continued from previous page)

…	…	…	…	…	…	…	…	…	…
LICENSE				PREMISE					
NUMBER TYPE DBA NAME			LICENSEE NAME	AD-DRESS	CITY	ST	ZIP	PHONE NUMBER	EX-PIRES
…	…	…	…	…	…	…	…	…	…

same cell. Let's see how we can fix this in the next section.

read_pdf(), which will split any strings that lie in different cells but have been assigned to a single cell (as a result of being merged together by PDFMiner).

```
↪95,209,327,442,529,566,606,683'], split_text=True)
```

Tip: Here's how you can do the same with the *command-line interface*.

(continues on next page)

(continued from previous page)

…	…	…	…	…	…	…	…	…	…
LI-CENSE				PREMISE					
NUM-BER	TYPE	DBA NAME	LICENSEE NAME	AD-DRESS	CITY	ST	ZIP	PHONE NUM-BER	EX-PIRES
…	…	…	…	…	…	…	…	…	…

scripts or subscripts, like this PDF.

Jammu and Kashmir	11.72	4.49	-	-	7.23	0.66
Jharkhand	-	-	-	-	-	-
Karnataka	22.44	19.59	-	-	2.86	1.22
Kerala	29.03	24.91[2]	-	-	4.11	1.77
Madhya Pradesh	27.13	23.57	-	-	3.56	0.38
Maharashtra	30.47	26.07	-	-	4.39	0.21
Manipur	2.17	1.61	-	0.26	0.29	0.08

In this case, the text that other tools return, will

be 24.912. This is relatively harmless when that decimal point is involved. But when it isn't there, you'll be left wondering why the results of your data analysis are 10x bigger!

You can solve this by passing `flag_size=True`, which will enclose the superscripts and subscripts with `<s></s>`, based on font size, as shown below.

```
>>> tables = camelot.read_pdf('superscript.pdf', flavor='stream', flag_size=True)
>>> tables[0].df
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot -flag stream superscript.pdf
```

…	…	…	…	…	…	…	…	…	…	…
Karnataka	22.44	19.59	.	.	2.86	1.22	.	0.89	.	0.69
Kerala	29.03	24.91<s>2</s>	.	.	4.11	1.77	.	0.48	.	1.45
Madhya Pradesh	27.13	23.57	.	.	3.56	0.38	.	1.86	.	1.28
…	…	…	…	…	…	…	…	…	…	…

2.6.8 Strip characters from text

You can strip unwanted characters like spaces, dots and newlines from a string using the `strip_text` keyword argument. Take a look at this PDF as an example, the text at the start of each row contains a lot of unwanted spaces, dots and newlines.

```
>>> tables = camelot.read_pdf('12s0324.pdf', flavor='stream', strip_text=' .\n')
>>> tables[0].df
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot -strip ' .\n' stream 12s0324.pdf
```

…	…	…	…	…	…	…	…	…	…
Forcible rape	17.5	2.6	14.9	17.2	2.5	14.7	–	–	–
Robbery	102.1	25.5	76.6	90.0	22.9	67.1	12.1	2.5	9.5
Aggravated assault	338.4	40.1	298.3	264.0	30.2	233.8	74.4	9.9	64.5
Property crime	1,396 .4	338 .7	1,057 .7	875 .9	210 .8	665 .1	608 .2	127 .9	392 .6
Burglary	240.9	60.3	180.6	205.0	53.4	151.7	35.9	6.9	29.0
…	…	…	…	…	…	…	…	…	…

2.6.9 Improve guessed table areas

While using *Stream*, automatic table detection can fail for PDFs like this one. That's because the text is relatively far apart vertically, which can lead to shorter textedges being calculated.

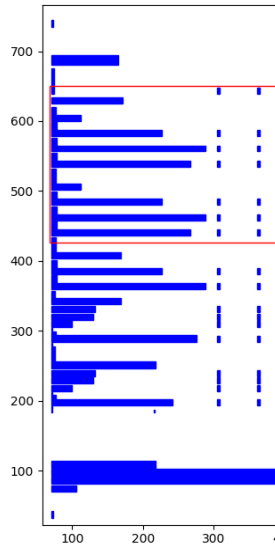
Note: To know more about how textedges are calculated to guess table areas, you can see pages 20, 35 and 40 of Anssi Nurminen's master's thesis.

Let's see the table area that is detected by default.

```
>>> tables = camelot.read_pdf('edge_tol.pdf', flavor='stream')
>>> camelot.plot(tables[0], kind='contour')
>>> plt.show()
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot stream -plot contour edge.pdf
```

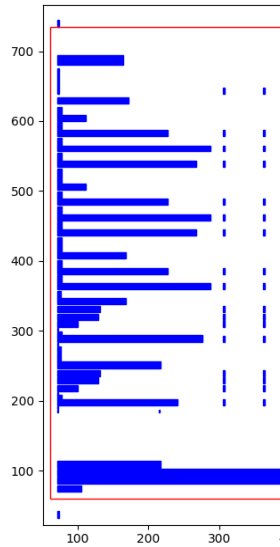


being placed relatively far apart vertically. Larger `edge_tol` will lead to longer textedges being detected, leading to an improved guess of the table area. Let's use a value of 500.

```
>>> tables = camelot.read_pdf('edge_tol.pdf', flavor='stream', edge_tol=500)
>>> camelot.plot(tables[0], kind='contour')
>>> plt.show()
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot stream -e 500 -plot contour edge.pdf
```



rows closer together, as shown below.

```
>>> tables = camelot.read_pdf('group_rows.pdf', flavor='stream')
>>> tables[0].df
```

Clave		Clave			Clave	
	Nombre Entidad			Nombre Municipio		Nombre Localidad
Entidad		Municipio			Localidad	
01	Aguascalientes	001	Aguascalientes		0094	Granja Adelita
01	Aguascalientes	001	Aguascalientes		0096	Agua Azul
01	Aguascalientes	001	Aguascalientes		0100	Rancho Alegre

```
>>> tables = camelot.read_pdf('group_rows.pdf', flavor='stream', row_tol=10)
>>> tables[0].df
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot stream -r 10 group_rows.pdf
```

Clave	Nombre Entidad	Clave		Nombre Municipio	Clave	Nombre Localidad
Entidad		Municipio			Localidad	
01	Aguascalientes	001	Aguascalientes		0094	Granja Adelita
01	Aguascalientes	001	Aguascalientes		0096	Agua Azul
01	Aguascalientes	001	Aguascalientes		0100	Rancho Alegre

2.6.11 Detect short lines

There might be cases while using *Lattice* when smaller lines don't get detected. The size of the smallest line that gets detected is calculated by dividing the PDF page's dimensions with a scaling factor called `line_scale`. By default, its value is 15.

As you can guess, the larger the `line_scale`, the smaller the size of lines getting detected.

Warning: Making `line_scale` very large (>150) will lead to text getting detected as lines.

Here's a PDF where small lines separating the the headers don't get detected with the default value of 15.

Investigations	No. of HHs	Age/Sex/ Physiological Group	Prevalence	C.I*	Relative Precision	Sample size per State
Anthropometry	2400	All the available individuals				
Clinical Examination						
History of morbidity						
Diet survey	1200	All the individuals partaking meals in the HH				
Blood Pressure #	2400	Men (≥ 18yrs)	10%	95%	20%	1728
		Women (≥ 18 yrs)				1728
Fasting blood glucose	2400	Men (≥ 18 yrs)	5%	95%	20%	1825
		Women (≥ 18 yrs)				1825
Knowledge & Practices on HTN & DM	2400	Men (≥ 18 yrs)	-	-	-	1728
	2400	Women (≥ 18 yrs)	-	-	-	1728

Let's plot the table for this PDF.

```
>
↪>
↪>
↪␣
↪tables␣
↪=␣
↪camelot.
↪read_
↪pdf(
↪'short_
↪lines.
↪pdf
↪')
>
↪>
↪>
↪␣
↪camelot.
↪plot(table
↪␣
↪kind=
↪'grid
↪')
```

(continued from previous page)

Clearly, the smaller lines separating the headers, couldn't be detected.

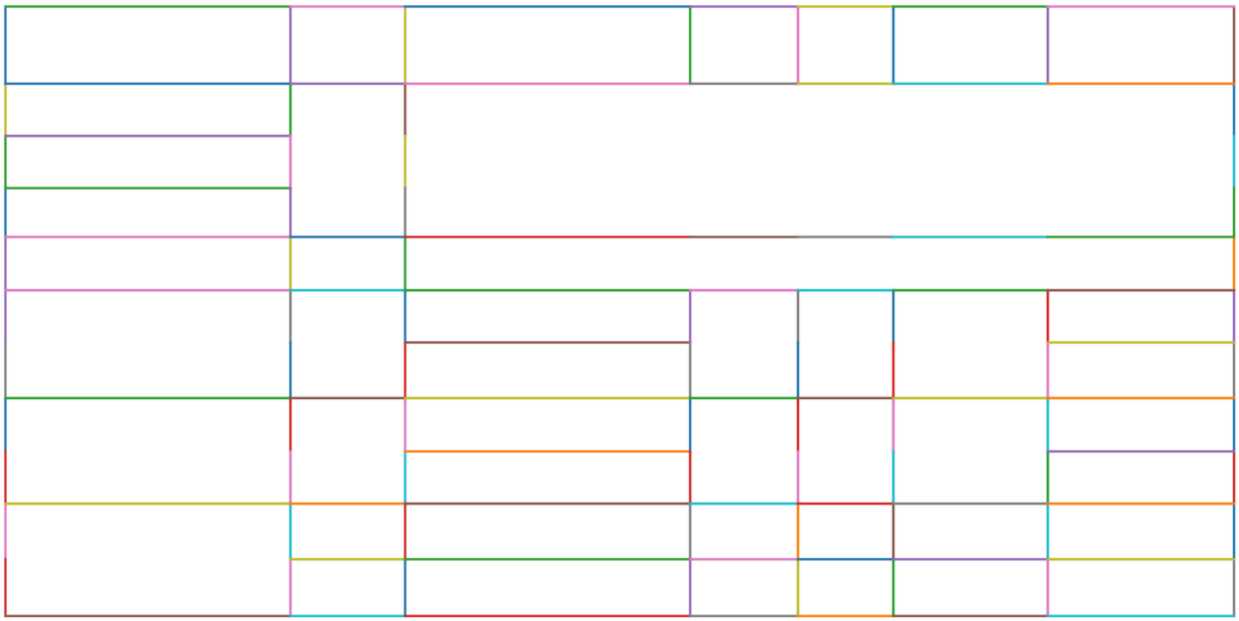
Let's try with `line_scale=40`, and plot the table again.

```
>>> tables = camelot.read_pdf('short_lines.pdf', line_scale=40)
>>> camelot.plot(tables[0], kind='grid')
>>> plt.show()
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -scale 40 -plot grid short_lines.pdf
```

Voila! Camelot can now see those lines. Let's get our table.



Investigations of HTN	Age Group	Sex	Relative Precision	Relative Precision	Sizeper State
Anthropometry	2400	All …			
Clinical Examination					
History of morbidity					
Diet survey	1200	All …			
Blood Pressure #	2400	Men (18yrs)	10%	95%	20%	1728
		Women (18 yrs)				1728
Fasting blood glucose	2400	Men (18 yrs)	5%	95%	20%	1825
		Women (18 yrs)				1825
Knowledge &Practices on HTN &DM	2400	Men (18 yrs)	.	.	.	1728
	2400	Women (18 yrs)	.	.	.	1728

2.6.12 Shift text in spanning cells

By default, the *Lattice* method shifts text in spanning cells, first to the left and then to the top, as you can observe in the output table above. However, this behavior can be changed using the `shift_text` keyword argument. Think of it as setting the *gravity* for a table — it decides the direction in which the text will move and finally come to rest.

`shift_text` expects a list with one or more characters from the following set: `('', 'l', 'r', 't', 'b')`, which are then applied *in order*. The default, as we discussed above, is `['l', 't']`.

We'll use the PDF from the previous example. Let's pass `shift_text=['']`, which basically means that the text will experience weightlessness! (It will remain in place.)

Investigations	No. of HHs	Age/Sex/ Physiological Group	Preva-lence	C.I*	Relative Precision	Sample size per State
Anthropometry	2400	All the available individuals				
Clinical Examination						
History of morbidity						
Diet survey	1200	All the individuals partaking meals in the HH				
Blood Pressure #	2400	Men (≥ 18yrs)	10%	95%	20%	1728
		Women (≥ 18 yrs)				1728
Fasting blood glucose	2400	Men (≥ 18 yrs)	5%	95%	20%	1825
		Women (≥ 18 yrs)				1825
Knowledge & Practices on HTN & DM	2400	Men (≥ 18 yrs)	-	-	-	1728
	2400	Women (≥ 18 yrs)	-	-	-	1728

Investigations of Hidden	Age Group	Sex	Relative Precision	Relative Precision	Sizeper State
Anthropometry					
Clinical Examination	2400	All	…		
History of morbidity					
Diet survey	1200	All	…		
		Men (18yrs)			1728
Blood Pressure #	2400	Women (18 yrs)	95%	20%	1728
		Men (18 yrs)			1825
Fasting blood glucose	2400	Women (18 yrs)	95%	20%	1825
Knowledge & Practices on HTN &	2400	Men (18 yrs)	.	.	1728
DM	2400	Women (18 yrs)	.	.	1728

No surprises there — it did remain

in place (observe the strings "2400" and "All the available individuals"). Let's pass `shift_text=['r', 'b']` to set the *gravity* to right-bottom and move the text in that direction.

```
>>> tables = camelot.read_pdf('short_lines.pdf', line_scale=40, shift_text=['r', 'b'])
>>> tables[0].df
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -scale 40 -shift r -shift b short_lines.pdf
```

Investigations	No. ofHHs	Age/Sex/Physiological Group	Prevalence	C.I*	RelativePrecision	Sample sizeper State
Anthropometry						
Clinical Examination						
History of morbidity	2400					All . . .
Diet survey	1200					All . . .
		Men (18yrs)				1728
Blood Pressure #	2400	Women (18 yrs)	10%	95%	20%	1728
		Men (18 yrs)				1825
Fasting blood glucose	2400	Women (18 yrs)	5%	95%	20%	1825
	2400	Men (18 yrs)	•	•	•	1728
Knowledge &Practices on HTN &DM	2400	Women (18 yrs)	•	•	•	1728

2.6.13 Copy text in spanning cells

You can copy text in spanning cells when using *Lattice*, in either the horizontal or vertical direction, or both. This behavior is disabled by default.

`copy_text` expects a list with one or more characters from the following set: `('v', 'h')`, which are then applied *in order*.

Let's try it out on this PDF. First, let's check out the output table to see if we need to use any other configuration parameters.

```
>>> tables = camelot.read_pdf('copy_text.pdf')
>>> tables[0].df
```

Sl. No.	Name of State/UT	Name of District	Disease/ Illness	No. of Cases	No. of Deaths	Date of start of out-break	Date of report-ing	Current Status	...
1	Kerala	Kollam	1. Food Poi-son-ing	19	0	31/12/13	03/01/14	Under control	...
2	Maharashtra	Beed	1. Dengue & Chikun-gunya i	11	0	03/01/14	04/01/14	Under control	...
3	Odisha	Kalahandi	3. Food Poi-son-ing	42	0	02/01/14	03/01/14	Under control	...
4	West Bengal	West Me-dinipur	4. Acute Di-ar-rhoeal Dis-ease	145	0	04/01/14	05/01/14	Under control	...
		Birbhum	22. Food Poi-son-ing	199	0	31/12/13	31/12/13	Under control	...
		Howrah	6. Viral Hep-ati-tis A &E	85	0	26/12/13	27/12/13	Under surveil-lance	...

We don't need anything else. Now, let's pass `copy_text=['v']` to copy text in the vertical direction. This can save you some time by not having to add this step in your cleaning script!

```
>>> tables = camelot.read_pdf('copy_text.pdf', copy_text=['v'])
>>> tables[0].df
```

Tip: Here's how you can do the same with the *command-line interface*.

```
$ camelot lattice -copy v copy_text.pdf
```

Sl. No.	Name of State/UT	Name of District	Disease/ Illness	No. of Cases	No. of Deaths	Date of start of out-break	Date of report-ing	Current Status	...
1	Kerala	Kollam	1. Food Poi-son-ing	19	0	31/12/13	03/01/14	Under control	...
2	Maharashtra	Beed	1. Dengue & Chikun-gunya i	11	0	03/01/14	04/01/14	Under control	...
3	Odisha	Kalahandi	3. Food Poi-son-ing	42	0	02/01/14	03/01/14	Under control	...
4	West Bengal	West Me-dinipur	4. Acute Di-ar-rhoeal Dis-ease	145	0	04/01/14	05/01/14	Under control	...
4	West Bengal	Birbhum	22. Food Poi-son-ing	199	0	31/12/13	31/12/13	Under control	...
4	West Bengal	Howrah	6. Viral Hep-ati-tis A &E	85	0	26/12/13	27/12/13	Under surveil-lance	...

2.6.14 Tweak layout generation

Camelot is built on top of PDFMiner's functionality of grouping characters on a page into words and sentences. In some cases (such as #170 and #215), PDFMiner can group characters that should belong to the same sentence into separate sentences.

To deal with such cases, you can tweak PDFMiner's LAParams kwargs to improve layout generation, by passing the keyword arguments as a dict using `layout_kwargs` in *`read_pdf()`*. To know more about the parameters you can tweak, you can check out PDFMiner docs.

```
>>> tables = camelot.read_pdf('foo.pdf', layout_kwargs={'detect_vertical': False})
```

2.7 Command-Line Interface

Camelot comes with a command-line interface.

You can print the help for the interface by typing `camelot --help` in your favorite terminal program, as shown below. Furthermore, you can print the help for each command by typing `camelot <command> --help`. Try it out!

```
Usage: camelot [OPTIONS] COMMAND [ARGS]...

  Camelot: PDF Table Extraction for Humans

Options:
  --version                       Show the version and exit.
  -q, --quiet TEXT                Suppress logs and warnings.
  -p, --pages TEXT                Comma-separated page numbers. Example: 1,3,4
                                  or 1,4-end.
  -pw, --password TEXT            Password for decryption.
  -o, --output TEXT               Output file path.
  -f, --format [csv|json|excel|html]
                                  Output file format.
  -z, --zip                       Create ZIP archive.
  -split, --split_text            Split text that spans across multiple cells.
  -flag, --flag_size              Flag text based on font size. Useful to
                                  detect super/subscripts.
  -M, --margins <FLOAT FLOAT FLOAT>...
                                  PDFMiner char_margin, line_margin and
                                  word_margin.
  --help                          Show this message and exit.

Commands:
  lattice  Use lines between text to parse the table.
  stream   Use spaces between text to parse the table.
```

CHAPTER 3

The API Documentation/Guide

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

3.1 API Reference

3.1.1 Main Interface

camelot.**read_pdf**(*filepath*, *pages='1'*, *password=None*, *flavor='lattice'*, *suppress_stdout=False*, *layout_kwargs={}*, ***kwargs*)

Read PDF and return extracted tables.

Note: kwargs annotated with ^ can only be used with flavor='stream' and kwargs annotated with * can only be used with flavor='lattice'.

Parameters

- **filepath** (*str*) – Filepath or URL of the PDF file.
- **pages** (*str, optional (default: '1')*) – Comma-separated page numbers. Example: '1,3,4' or '1,4-end' or 'all'.
- **password** (*str, optional (default: None)*) – Password for decryption.
- **flavor** (*str (default: 'lattice')*) – The parsing method to use ('lattice' or 'stream'). Lattice is used by default.
- **suppress_stdout** (*bool, optional (default: True)*) – Print all logs and warnings.
- **layout_kwargs** (*dict, optional (default: {})*) – A dict of pdfminer.layout.LAParams kwargs.
- **table_areas** (*list, optional (default: None)*) – List of table area strings of the form x1,y1,x2,y2 where (x1, y1) -> left-top and (x2, y2) -> right-bottom in PDF coordinate space.

- **columns^** (*list, optional (default: None)*) – List of column x-coordinates strings where the coordinates are comma-separated.
- **split_text** (*bool, optional (default: False)*) – Split text that spans across multiple cells.
- **flag_size** (*bool, optional (default: False)*) – Flag text based on font size. Useful to detect super/subscripts. Adds <s></s> around flagged text.
- **strip_text** (*str, optional (default: '')*) – Characters that should be stripped from a string before assigning it to a cell.
- **row_tol^** (*int, optional (default: 2)*) – Tolerance parameter used to combine text vertically, to generate rows.
- **column_tol^** (*int, optional (default: 0)*) – Tolerance parameter used to combine text horizontally, to generate columns.
- **process_background*** (*bool, optional (default: False)*) – Process background lines.
- **line_scale*** (*int, optional (default: 15)*) – Line size scaling factor. The larger the value the smaller the detected lines. Making it very large will lead to text being detected as lines.
- **copy_text*** (*list, optional (default: None)*) – {'h', 'v'} Direction in which text in a spanning cell will be copied over.
- **shift_text*** (*list, optional (default: ['l', 't'])*) – {'l', 'r', 't', 'b'} Direction in which text in a spanning cell will flow.
- **line_tol*** (*int, optional (default: 2)*) – Tolerance parameter used to merge close vertical and horizontal lines.
- **joint_tol*** (*int, optional (default: 2)*) – Tolerance parameter used to decide whether the detected lines and points lie close to each other.
- **threshold_blocksize*** (*int, optional (default: 15)*) – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.

  For more information, refer OpenCV's adaptiveThreshold.
- **threshold_constant*** (*int, optional (default: -2)*) – Constant subtracted from the mean or weighted mean. Normally, it is positive but may be zero or negative as well.

  For more information, refer OpenCV's adaptiveThreshold.
- **iterations*** (*int, optional (default: 0)*) – Number of times for erosion/dilation is applied.

  For more information, refer OpenCV's dilate.
- **resolution*** (*int, optional (default: 300)*) – Resolution used for PDF to PNG conversion.

Returns **tables**

Return type *camelot.core.TableList*

3.1.2 Lower-Level Classes

class camelot.handlers.**PDFHandler**(*filepath*, *pages='1'*, *password=None*)

Handles all operations like temp directory creation, splitting file into single page PDFs, parsing each PDF and then removing the temp directory.

Parameters

- **filepath** (*str*) – Filepath or URL of the PDF file.
- **pages** (*str, optional (default: '1')*) – Comma-separated page numbers. Example: '1,3,4' or '1,4-end' or 'all'.
- **password** (*str, optional (default: None)*) – Password for decryption.

parse(*flavor='lattice'*, *suppress_stdout=False*, *layout_kwargs={}*, ***kwargs*)

Extracts tables by calling parser.get_tables on all single page PDFs.

Parameters

- **flavor** (*str (default: 'lattice')*) – The parsing method to use ('lattice' or 'stream'). Lattice is used by default.
- **suppress_stdout** (*str (default: False)*) – Suppress logs and warnings.
- **layout_kwargs** (*dict, optional (default: {})*) – A dict of pdfminer.layout.LAParams kwargs.
- **kwargs** (*dict*) – See camelot.read_pdf kwargs.

Returns **tables** – List of tables found in PDF.

Return type *camelot.core.TableList*

class camelot.parsers.**Stream**(*table_regions=None*, *table_areas=None*, *columns=None*, *split_text=False*, *flag_size=False*, *strip_text=''*, *edge_tol=50*, *row_tol=2*, *column_tol=0*, ***kwargs*)

Stream method of parsing looks for spaces between text to parse the table.

If you want to specify columns when specifying multiple table areas, make sure that the length of both lists are equal.

Parameters

- **table_regions** (*list, optional (default: None)*) – List of page regions that may contain tables of the form x1,y1,x2,y2 where (x1, y1) -> left-top and (x2, y2) -> right-bottom in PDF coordinate space.
- **table_areas** (*list, optional (default: None)*) – List of table area strings of the form x1,y1,x2,y2 where (x1, y1) -> left-top and (x2, y2) -> right-bottom in PDF coordinate space.
- **columns** (*list, optional (default: None)*) – List of column x-coordinates strings where the coordinates are comma-separated.
- **split_text** (*bool, optional (default: False)*) – Split text that spans across multiple cells.
- **flag_size** (*bool, optional (default: False)*) – Flag text based on font size. Useful to detect super/subscripts. Adds <s></s> around flagged text.
- **strip_text** (*str, optional (default: '')*) – Characters that should be stripped from a string before assigning it to a cell.

- **edge_tol** (*int, optional (default: 50)*) – Tolerance parameter for extending textedges vertically.
- **row_tol** (*int, optional (default: 2)*) – Tolerance parameter used to combine text vertically, to generate rows.
- **column_tol** (*int, optional (default: 0)*) – Tolerance parameter used to combine text horizontally, to generate columns.

class camelot.parsers.**Lattice**(*table_regions=None, table_areas=None, process_background=False, line_scale=15, copy_text=None, shift_text=['l', 't'], split_text=False, flag_size=False, strip_text='', line_tol=2, joint_tol=2, threshold_blocksize=15, threshold_constant=-2, iterations=0, resolution=300, **kwargs*)

Lattice method of parsing looks for lines between text to parse the table.

Parameters

- **table_regions** (*list, optional (default: None)*) – List of page regions that may contain tables of the form x1,y1,x2,y2 where (x1, y1) -> left-top and (x2, y2) -> right-bottom in PDF coordinate space.
- **table_areas** (*list, optional (default: None)*) – List of table area strings of the form x1,y1,x2,y2 where (x1, y1) -> left-top and (x2, y2) -> right-bottom in PDF coordinate space.
- **process_background** (*bool, optional (default: False)*) – Process background lines.
- **line_scale** (*int, optional (default: 15)*) – Line size scaling factor. The larger the value the smaller the detected lines. Making it very large will lead to text being detected as lines.
- **copy_text** (*list, optional (default: None)*) – {'h', 'v'} Direction in which text in a spanning cell will be copied over.
- **shift_text** (*list, optional (default: ['l', 't'])*) – {'l', 'r', 't', 'b'} Direction in which text in a spanning cell will flow.
- **split_text** (*bool, optional (default: False)*) – Split text that spans across multiple cells.
- **flag_size** (*bool, optional (default: False)*) – Flag text based on font size. Useful to detect super/subscripts. Adds <s></s> around flagged text.
- **strip_text** (*str, optional (default: '')*) – Characters that should be stripped from a string before assigning it to a cell.
- **line_tol** (*int, optional (default: 2)*) – Tolerance parameter used to merge close vertical and horizontal lines.
- **joint_tol** (*int, optional (default: 2)*) – Tolerance parameter used to decide whether the detected lines and points lie close to each other.
- **threshold_blocksize** (*int, optional (default: 15)*) – Size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.

  For more information, refer OpenCV's adaptiveThreshold.
- **threshold_constant** (*int, optional (default: -2)*) – Constant subtracted from the mean or weighted mean. Normally, it is positive but may be zero or negative as well.

  For more information, refer OpenCV's adaptiveThreshold.

- **iterations** (*int, optional (default: 0)*) – Number of times for erosion/dilation is applied.

  For more information, refer OpenCV's dilate.

- **resolution** (*int, optional (default: 300)*) – Resolution used for PDF to PNG conversion.

3.1.3 Lower-Lower-Level Classes

class camelot.core.**TableList**(*tables*)

Defines a list of camelot.core.Table objects. Each table can be accessed using its index.

n

int – Number of tables in the list.

export(*path*, *f='csv'*, *compress=False*)

Exports the list of tables to specified file format.

Parameters

- **path** (*str*) – Output filepath.
- **f** (*str*) – File format. Can be csv, json, excel, html and sqlite.
- **compress** (*bool*) – Whether or not to add files to a ZIP archive.

class camelot.core.**Table**(*cols*, *rows*)

Defines a table with coordinates relative to a left-bottom origin. (PDF coordinate space)

Parameters

- **cols** (*list*) – List of tuples representing column x-coordinates in increasing order.
- **rows** (*list*) – List of tuples representing row y-coordinates in decreasing order.

df

pandas.DataFrame

shape

tuple – Shape of the table.

accuracy

float – Accuracy with which text was assigned to the cell.

whitespace

float – Percentage of whitespace in the table.

order

int – Table number on PDF page.

page

int – PDF page number.

data

Returns two-dimensional list of strings in table.

parsing_report

Returns a parsing report with %accuracy, %whitespace, table number on page and page number.

set_all_edges()

Sets all table edges to True.

set_border()
Sets table border edges to True.

set_edges(*vertical*, *horizontal*, *joint_tol=2*)
Sets a cell's edges to True depending on whether the cell's coordinates overlap with the line's coordinates within a tolerance.

Parameters

- **vertical** (*list*) – List of detected vertical lines.
- **horizontal** (*list*) – List of detected horizontal lines.

set_span()
Sets a cell's hspan or vspan attribute to True depending on whether the cell spans horizontally or vertically.

to_csv(*path*, ***kwargs*)
Writes Table to a comma-separated values (csv) file.

For kwargs, check pandas.DataFrame.to_csv().

Parameters **path** (*str*) – Output filepath.

to_excel(*path*, ***kwargs*)
Writes Table to an Excel file.

For kwargs, check pandas.DataFrame.to_excel().

Parameters **path** (*str*) – Output filepath.

to_html(*path*, ***kwargs*)
Writes Table to an HTML file.

For kwargs, check pandas.DataFrame.to_html().

Parameters **path** (*str*) – Output filepath.

to_json(*path*, ***kwargs*)
Writes Table to a JSON file.

For kwargs, check pandas.DataFrame.to_json().

Parameters **path** (*str*) – Output filepath.

to_sqlite(*path*, ***kwargs*)
Writes Table to sqlite database.

For kwargs, check pandas.DataFrame.to_sql().

Parameters **path** (*str*) – Output filepath.

class camelot.core.**Cell**(*x1*, *y1*, *x2*, *y2*)
Defines a cell in a table with coordinates relative to a left-bottom origin. (PDF coordinate space)

Parameters

- **x1** (*float*) – x-coordinate of left-bottom point.
- **y1** (*float*) – y-coordinate of left-bottom point.
- **x2** (*float*) – x-coordinate of right-top point.
- **y2** (*float*) – y-coordinate of right-top point.

lb
tuple – Tuple representing left-bottom coordinates.

lt
tuple – Tuple representing left-top coordinates.

rb
tuple – Tuple representing right-bottom coordinates.

rt
tuple – Tuple representing right-top coordinates.

left
bool – Whether or not cell is bounded on the left.

right
bool – Whether or not cell is bounded on the right.

top
bool – Whether or not cell is bounded on the top.

bottom
bool – Whether or not cell is bounded on the bottom.

hspan
bool – Whether or not cell spans horizontally.

vspan
bool – Whether or not cell spans vertically.

text
string – Text assigned to cell.

CHAPTER 4

The Contributor Guide

If you want to contribute to the project, this part of the documentation is for you.

4.1 Contributor's Guide

If you're reading this, you're probably looking to contributing to Camelot. *Time is the only real currency*, and the fact that you're considering spending some here is *very* generous of you. Thank you very much!

This document will help you get started with contributing documentation, code, testing and filing issues. If you have any questions, feel free to reach out to Vinayak Mehta, the author and maintainer.

4.1.1 Code Of Conduct

The following quote sums up the **Code Of Conduct**.

> **Be cordial or be on your way**. *–Kenneth Reitz*

Kenneth Reitz has also written an essay on this topic, which you should read.

As the Requests Code Of Conduct states, **all contributions are welcome**, as long as everyone involved is treated with respect.

4.1.2 Your first contribution

A great way to start contributing to Camelot is to pick an issue tagged with the help wanted or the good first issue tags. If you're unable to find a good first issue, feel free to contact the maintainer.

4.1.3 Setting up a development environment

To install the dependencies needed for development, you can use pip:

```
$ pip install camelot-py[dev]
```

Alternatively, you can clone the project repository, and install using pip:

```
$ pip install ".[dev]"
```

4.1.4 Pull Requests

Submit a pull request

The preferred workflow for contributing to Camelot is to fork the project repository on GitHub, clone, develop on a branch and then finally submit a pull request. Here are the steps:

1. Fork the project repository. Click on the 'Fork' button near the top of the page. This creates a copy of the code under your account on the GitHub.
2. Clone your fork of Camelot from your GitHub account:

   ```
   $ git clone https://www.github.com/[username]/camelot
   ```

3. Create a branch to hold your changes:

   ```
   $ git checkout -b my-feature
   ```

Always branch out from `master` to work on your contribution. It's good practice to never work on the `master` branch!

Note: `git stash` is a great way to save the work that you haven't committed yet, to move between branches.

---

4. Work on your contribution. Add changed files using `git add` and then `git commit` them:

   ```
   $ git add modified_files
   $ git commit
   ```

5. Finally, push them to your GitHub fork:

   ```
   $ git push -u origin my-feature
   ```

Now it's time to go to the your fork of Camelot and create a pull request! You can follow these instructions to do the same.

Work on your pull request

We recommend that your pull request complies with the following guidelines:

- Make sure your code follows pep8.
- In case your pull request contains function docstrings, make sure you follow the numpydoc format. All function docstrings in Camelot follow this format. Following the format will make sure that the API documentation is generated flawlessly.
- **Make sure your commit messages follow the seven rules of a great git commit message:**
  - Separate subject from body with a blank line

  - Limit the subject line to 50 characters
  - Capitalize the subject line
  - Do not end the subject line with a period
  - Use the imperative mood in the subject line
  - Wrap the body at 72 characters
  - Use the body to explain what and why vs. how

- Please prefix your title of your pull request with [MRG] (Ready for Merge), if the contribution is complete and ready for a detailed review. An incomplete pull request's title should be prefixed with [WIP] (to indicate a work in progress), and changed to [MRG] when it's complete. A good task list in the PR description will ensure that other people get a fair idea of what it proposes to do, which will also increase collaboration.
- If contributing new functionality, make sure that you add a unit test for it, while making sure that all previous tests pass. Camelot uses pytest for testing. Tests can be run using:

```
$ python setup.py test
```

4.1.5 Writing Documentation

Writing documentation, function docstrings, examples and tutorials is a great way to start contributing to open-source software! The documentation is present inside the `docs/` directory of the source code repository.

The documentation is written in reStructuredText, with Sphinx used to generate these lovely HTML files that you're currently reading (unless you're reading this on GitHub). You can edit the documentation using any text editor and then generate the HTML output by running *make html* in the `docs/` directory.

The function docstrings are written using the numpydoc extension for Sphinx. Make sure you check out how its format guidelines before you start writing one.

4.1.6 Filing Issues

We use GitHub issues to keep track of all issues and pull requests. Before opening an issue (which asks a question or reports a bug), please use GitHub search to look for existing issues (both open and closed) that may be similar.

Questions

Please don't use GitHub issues for support questions. A better place for them would be Stack Overflow. Make sure you tag them using the `python-camelot` tag.

Bug Reports

In bug reports, make sure you include:

- Your operating system type and Python version number, along with the version numbers of NumPy, OpenCV and Camelot. You can use the following code snippet to find this information:

```
import platform; print(platform.platform())
import sys; print('Python', sys.version)
import numpy; print('NumPy', numpy.__version__)
import cv2; print('OpenCV', cv2.__version__)
import camelot; print('Camelot', camelot.__version__)
```

- The complete traceback. Just adding the exception message or a part of the traceback won't help us fix your issue sooner.
- Steps to reproduce the bug, using code snippets. See Creating and highlighting code blocks.
- A link to the PDF document that you were trying to extract tables from, telling us what you expected the code to do and what actually happened.

Python Module Index

C

camelot, 45

Index

A

accuracy (camelot.core.Table attribute), 49

B

bottom (camelot.core.Cell attribute), 51

C

camelot (module), 45
Cell (class in camelot.core), 50

D

data (camelot.core.Table attribute), 49
df (camelot.core.Table attribute), 49

E

export() (camelot.core.TableList method), 49

H

hspan (camelot.core.Cell attribute), 51

L

Lattice (class in camelot.parsers), 48
lb (camelot.core.Cell attribute), 50
left (camelot.core.Cell attribute), 51
lt (camelot.core.Cell attribute), 50

N

n (camelot.core.TableList attribute), 49

O

order (camelot.core.Table attribute), 49

P

page (camelot.core.Table attribute), 49
parse() (camelot.handlers.PDFHandler method), 47
parsing_report (camelot.core.Table attribute), 49
PDFHandler (class in camelot.handlers), 47

R

rb (camelot.core.Cell attribute), 51
read_pdf() (in module camelot), 45
right (camelot.core.Cell attribute), 51
rt (camelot.core.Cell attribute), 51

S

set_all_edges() (camelot.core.Table method), 49
set_border() (camelot.core.Table method), 49
set_edges() (camelot.core.Table method), 50
set_span() (camelot.core.Table method), 50
shape (camelot.core.Table attribute), 49
Stream (class in camelot.parsers), 47

T

Table (class in camelot.core), 49
TableList (class in camelot.core), 49
text (camelot.core.Cell attribute), 51
to_csv() (camelot.core.Table method), 50
to_excel() (camelot.core.Table method), 50
to_html() (camelot.core.Table method), 50
to_json() (camelot.core.Table method), 50
to_sqlite() (camelot.core.Table method), 50
top (camelot.core.Cell attribute), 51

V

vspan (camelot.core.Cell attribute), 51

W

whitespace (camelot.core.Table attribute), 49