EXTRACTING TABLES FROM PDFS USING CAMELOT

NB: with Camelot, you can extract table from PDF. But not scanned copy of PDFs.

Note: *While PDF files are great for laying out text in a way that's easy for people to print and read, they're not straightforward for software to parse into plaintext. As such, PyPDF2 might make mistakes when extracting text from a PDF and may even be unable to open some PDFs at all. There isn't much you can do about this, unfortunately. PyPDF2 may simply be unable to work with some of your particular PDF files.*

Reading the PDF

NB: Before reading a pdf with Camelot, you begin by importing the necessary modules:

Import Camelot as cm

```
>>> import camelot
```

Read the pdf file

Use:

```
tables = cm.read_pdf('foo.pdf')
```

NB: you can specify the number of pages you want to extract the table from. By default, Camelot looks into the first page only. How? *Pages = '1,2,3,4'*

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

OR

You can specify individual pages yourself

```
camelot.read_pdf('your.pdf',pages = "1,4-10,20-end")
```

OR

```
#read csv file
table = cm.read_pdf('python/data_gathering/tables/table2.pdf', pages= "1-end")
```

NB: what gets returned to 'tables' is a ***table_list*** object telling you how many tables are in the page. i.e. a list of table objects. **NB**: *The minimum value it can return is '1' i.e. if there is only one table in the pdf file.* For example:

```
>>> tables
```

```
<TableList n=1>
```

From the code snippet above, we can see that the tables object has only one table, since n=1.

NB: *We can get everything we need from this table object*

NB: ***We can access each table using its index.***

LOOKING AT THE SHAPE OF A TABLE

NB: *we can access the table using the* ***index 0*** *and take a look at its* ***shape****. E.g.*

table[0]

PRINTING THE PARSING REPORT (***this tells us how accuracy our extraction of the table was*****)**

NB: This can be done for each table (you just have to index them). ***This report is very important***. E.g. *tables[page_index].parsing_report*

```
e.g. tables[0].parsing_report
```

NB: metrics of high interest are the accuracy and number of whitespaces. i.e. the higher your accuracy and the lower your whitespaces, the better your extraction will be.

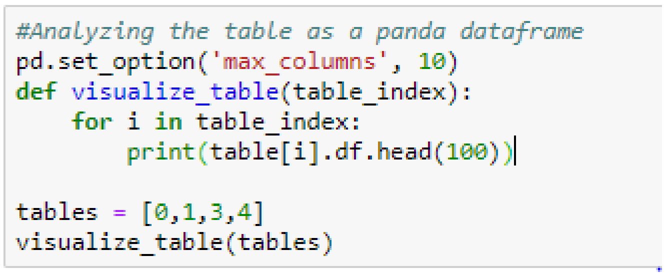
NB: any table with a parsing report of zero (0) or less than (50) accuracy should never be extracted.

Accessing/Visualizing each Table

NB: You can now access the table as a pandas data frame by using the 'table' df property. (don't forget the index) i.e.

Table[].df

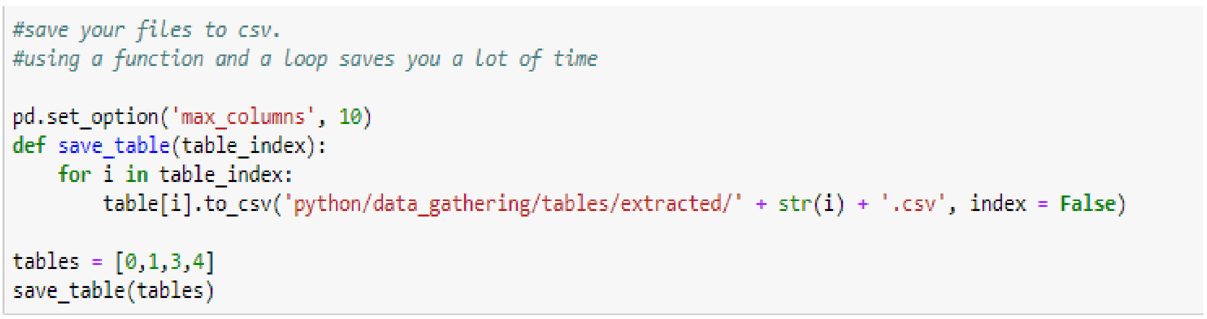
NB: You can use a loop to make it a lot simpler for you



```
#Analyzing the table as a panda dataframe
pd.set_option('max_columns', 10)
def visualize_table(table_index):
    for i in table_index:
        print(table[i].df.head(100))

tables = [0,1,3,4]
visualize_table(tables)
```

Saving your tables



```python
#save your files to csv.
#using a function and a loop saves you a lot of time

pd.set_option('max_columns', 10)
def save_table(table_index):
    for i in table_index:
        table[i].to_csv('python/data_gathering/tables/extracted/' + str(i) + '.csv', index = False)

tables = [0,1,3,4]
save_table(tables)
```

OR

Exporting all your tables at once

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

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

Path: This is the folder in which you want to save your file

F = in what kind of document you want to save your file as e.g. csv, json, html, excel, sqlite

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'.

NB: When using the '***export***' command, it will save your file in your path in multiple line. You can also save it in a 'zip file' by compressing it. NB: to achieve this, in the export() function, you specify;

Compress = True