Data Scraping

Scraping: download_data.py

Libraries: urllib2, BeautifulSoup, Selenium

```
usage: download_data.py [-h] [-o dir] -d {MX,TW,PE,Rio,PAHO,TN, CME}

optional arguments:
  -h, --help            show this help message and exit
  -o dir, --output dir  the output directory
  -d {MX,TW,PE,Rio,PAHO,TN,CME}, --dataset {MX,TW,PE,Rio,PAHO,TN,CME}
                        the data set to download

Example:
    download_data.py -d PAHO -o paho/PDFs/
```

Converting: PDF2CSV.jar

Libraries: PDFBox, Tabula-java, OpenCV

```
usage: tabula [-b <DIRECTORY>] [-d] [-f <FORMAT>] [-g] [-h] [-i] [-l] [-o
       <OUTFILE>] [-p <PAGES>] [-s <PASSWORD>] [-t] [-u] [-v]

Tabula helps you extract tables from PDFs

 -b,--batch <DIRECTORY>     Convert all .pdfs in the provided directory.
 -d,--debug                 Print detected table areas instead of
                            processing.
 -f,--format <FORMAT>       Output format: (CSV,TSV,JSON). Default: CSV
 -g,--guess                 Guess the portion of the page to analyze per
                            page.
 -h,--help                  Print this help text.
 -i,--silent                Suppress all stderr output.
 -l,--lattice               Force PDF to be extracted using lattice-mode
                            extraction (if there are ruling lines
                            separating each cell, as in a PDF of an Excel
                            spreadsheet)
 -o,--outfile <OUTFILE>     Write output to <file> instead of STDOUT.
                            Default: -
 -p,--pages <PAGES>         Comma separated list of ranges, or all.
                            Examples: --pages 1-3,5-7, --pages 3 or
                            --pages all. Default is --pages 1
 -s,--password <PASSWORD>   Password to decrypt document. Default is empty
 -t,--stream                Force PDF to be extracted using stream-mode
                            extraction (if there are no ruling lines
                            separating each cell)
 -u,--use-line-returns      Use embedded line returns in cells. (Only in
                            spreadsheet mode.)
 -v,--version               Print version and exit.
```

Summary: Tabula is a PDF table extractor. Their library tabula-java uses PDFBox. They have a command line application, but I found it lacking in the automatic table detection algorithm. Their algorithm used overly simplistic image processessing methods in order to detect vertical and horizontal lines on the page. So in PDF2CSV.jar I use OpenCV to accomplish the line detection much more effectively. I copied most of the code from Tabula-java's command line app, changing the image processing methods and eleminating a few unneeded command line options.

Tabula-java has an ‘-a,–area’ option which allows the user to indicate by coordinates the region of interest on the page. But the whole reason I wrote this program was to automate that process. So that option has been removed. It has a similar ‘-c,–column’ option which I removed for the same reason. I removed two depricated options ‘-r’ and ‘-n’ which have been replaced with ‘-l,–lattice’ and ‘-t,–stream’. Option ‘-g,–guess’ should always be used, as that is where the table detection algorithm is called.

Comparison: This is the most general, fast, and accurate of the three conversion programs I have made.

- PDF2CSV.py is very slow (but okay on small tables)
- The first version of PDF2CSV.jar (with no Tabula) is not nearly general enough and would require lots more work (which Tabula has already done!).

Small adjustments needed:

- Handle certain fonts
- Detect inversely colored tables. (Right now it only detects dark lines).

Converting: HTML2CSV.py

Libraries: HTMLParser

```
usage: HTML2CSV.py [-h] [-o dir] [-r] file [file ...]

A coarse HTML tables to CSV (Comma-Separated Values) converter.

positional arguments:
  file                  the HTML file(s) you want to convert to CSV.

optional arguments:
  -h, --help            show this help message and exit
  -o dir, --output dir  the output directory
  -r, --recursive       To get all leaf tables from HTML with nested tables

Examples   : HTML2CSV.py -i mypage.html
           : HTML2CSV.py -i *.html -o CSVs/
```

Automating: crontab

Crontab is a list of commands that you want to run on a regular schedule.

```
Crontab:
- use 'env EDITOR=vim crontab -e' to add/change a scheduled script
- use 'crontab -l' to display all scheduled events

*     *     *   *    *        command to be executed
-     -     -   -    -
|     |     |   |    |
|     |     |   |    +----- day of week (0 - 6) (Sunday=0)
|     |     |   +------- month (1 - 12)
|     |     +--------- day of        month (1 - 31)
|     +----------- hour (0 - 23)
+------------- min (0 - 59)

Example:
Including this line in your crontab will run command every day at noon:

0 12 * * * cd ~/Dengue/Peru/Code/Python && ./download_PDFs_weekly.py
```

Converting: PDF2CSV.py (depricated in favor of Java)

Libraries: pdfminer

```
usage: PDF2CSV.py [-h] [-o dir] [-s] (-b | -c) file [file ...]

positional arguments:
  file                  the input file(s)

optional arguments:
  -h, --help            show this help message and exit
  -o dir, --output dir  the output directory
  -s, --separate        Each page of PDF is a separate table
  -b, --boundary        differentiate cells by cell borders
  -c, --coordinate      differentiate cells by text element coordinates

Examples   : PDF2CSV.py country/PDFs/* -o country/CSVs/ -b
           : PDF2CSV.py country/PDFs/weekly_pages.pdf -o country/weekly-CSVs/ -c -s
```

This script uses the pdfminer python library. Pdfminer extracts the contents of PDFs. From (pg. 12-15): https://media.readthedocs.org/pdf/pdfminer-docs/latest/pdfminer-docs.pdf

A layout analyzer returns a LTPage object for each page in the PDF document. This object contains child objects within the page, forming a tree structure.

```
pdfminer LT objects:
                       ____________________________LTPage____________________
                      /                    /          |         \           \
                     /                    /           |          \           \
              LTTextBox               LTFigure     LTLine      LTRect      LTImage
             /          \            /        \
            /            \          /          \
       LTTextLine        ...     LTCurve       ...
      /     |     \
     /      |      \
 LTChar    LTChar  LTText
```

LTPage

- Represents an entire page. May contain child objects like LTTextBox, LTFigure, LTImage, LTRect, LTCurve and LTLine.

LTTextBox

- Represents a group of text chunks that can be contained in a rectangular area. Note that this box is created by geometric analysis and does not necessarily represents a logical boundary of the text. It contains a list of LTTextLine objects. get_text() method returns the text content.

LTTextLine

- Contains a list of LTChar objects that represent a single text line. The characters are aligned either horizontaly or vertically, depending on the text's writing mode. get_text() method returns the text content.

LTChar

- Represents a single character of text.

LTAnno

- Represent an actual letter in the text as a Unicode string. Note that, while a LTChar object has actual boundaries, LTAnno objects does not, as these are "virtual" characters, inserted by a layout analyzer according to the relationship between two characters (e.g. a space).

LTFigure

- Represents an area used by PDF Form objects. PDF Forms can be used to present figures or pictures by embedding yet another PDF document within a page. Note that LTFigure objects can appear recursively

LTImage

- Represents an image object. Embedded images can be in JPEG or other formats, but currently PDFMiner does not pay much attention to graphical objects.

LTLine

- Represents a single straight line. Could be used for separating text or figures.

LTRect

- Represents a rectangle. Could be used for framing another pictures or figures.

LTCurve

- Represents a generic Bezier curve

LT-elements all share some of these public attributes:

- bbox: (x0, y0, x1, y1) <- coordinates of the element's bounding box.
- width
- height
- text (or get_text()) <- for character and text elements only.
- ...

Implementation

PDF2CSV.py has two methods for extracting tabular structures from PDFs.

1. By Cell Boundaries [-b]

This method uses LTLine(s) and LTRect(s) along with their bbox, width, and height attributes to determine cell boundaries. It works well if there are lines/rectangles that divide every cell. See this tutorial for an explanation.

2. By Cell Coordinates [-c]

This method uses LTTextLine(s) and LTChar(s) along with their bbox attribute to determine cell positions. Basically each LTTextLine is sorted first by its y-coordinate (i.e. by row), and then by its x-coordinate (i.e. by column). The text attribute is then extracted from this 2-dimensional, ordered list.

To convert this README to PDF, use: $ pandoc README.md -o README.pdf -V geometry:margin=0.8in