OCR Table Extractor in Images or PDFs

- This notebook looks at how to extract tables and text using OCR from PDFs using *a simple library I made*.
  Code : https://github.com/josephmulindwa/Table-OCR-Extractor

Import files

First, we load essential libraries; *libutil*, *libtable* and *libocr*

In [1]:
```
import matplotlib.pyplot as plt

import libutil
import libtable as lt
import libocr
```

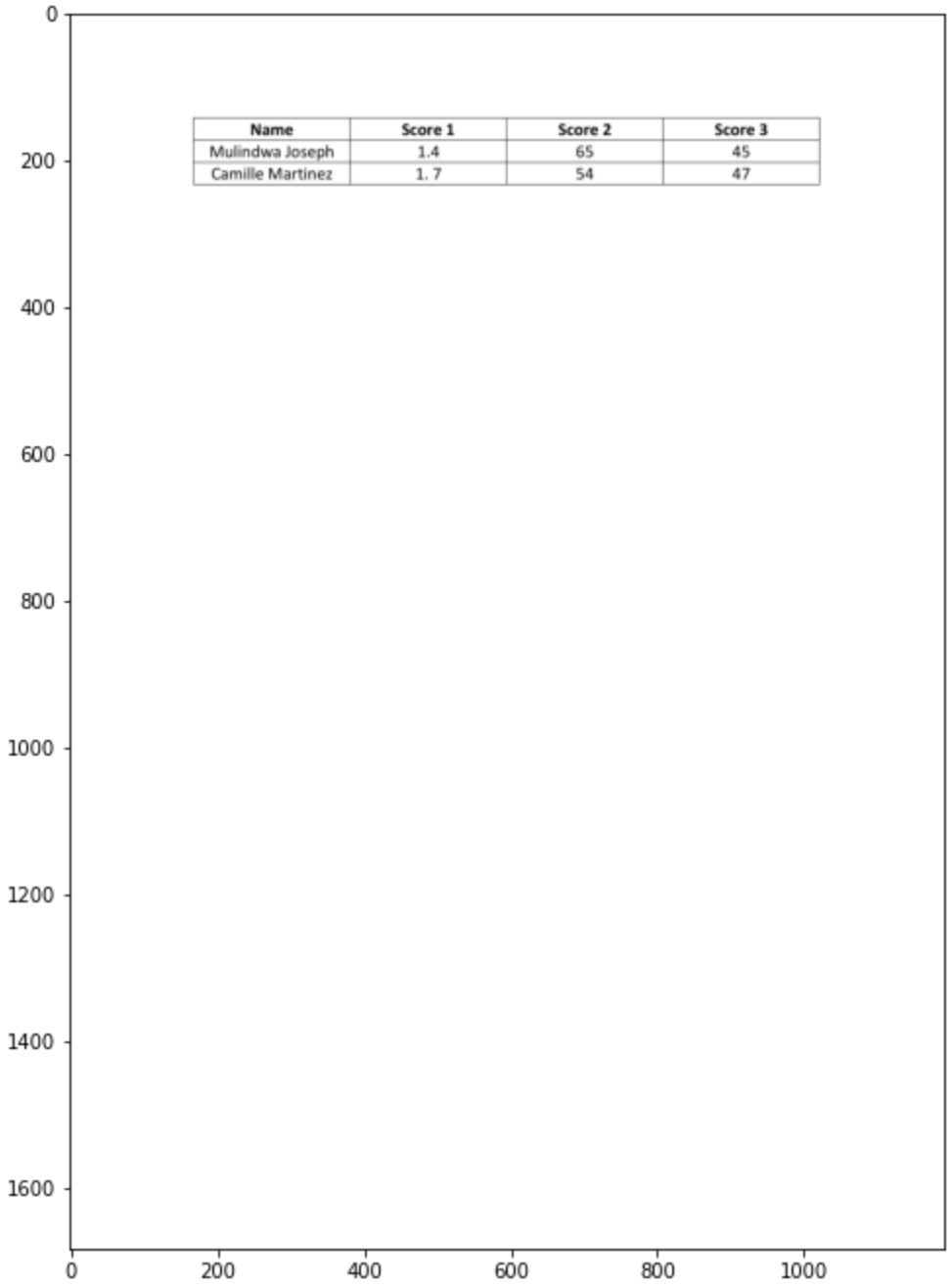
Load an Image or PDF

We can either load in a PDF or an Image. For this notebook, we shall do both, we shall load images and PDFs and convert them to images using *libutil* which relies on `fitz/mupdf` python library.To do that, we shall use the `pdf2images()` function from *libutil*.
This function takes the PDF path and optional parameters such as *zoom* which controls the clarity and size of the returned images. It returns a list containing *all* the images in the PDF by default. It can however return a list of paths to the images generated and stored on your (hard) disk if you enable saving.

Note:
This function has not yet been optimized to work on large PDFs efficiently. Small PDFs (about 10 pages) are handled efficiently.

Extracting from PDFs

In [2]:
```
images = libutil.pdf2images('data\\SIMPLE.pdf', zoom=2) # load images
image = images[0] # pick the first image

plt.figure(figsize=(8, 15))
plt.imshow(image, cmap='gray')
plt.show()
```



Extracting tables and finding good extraction parameters

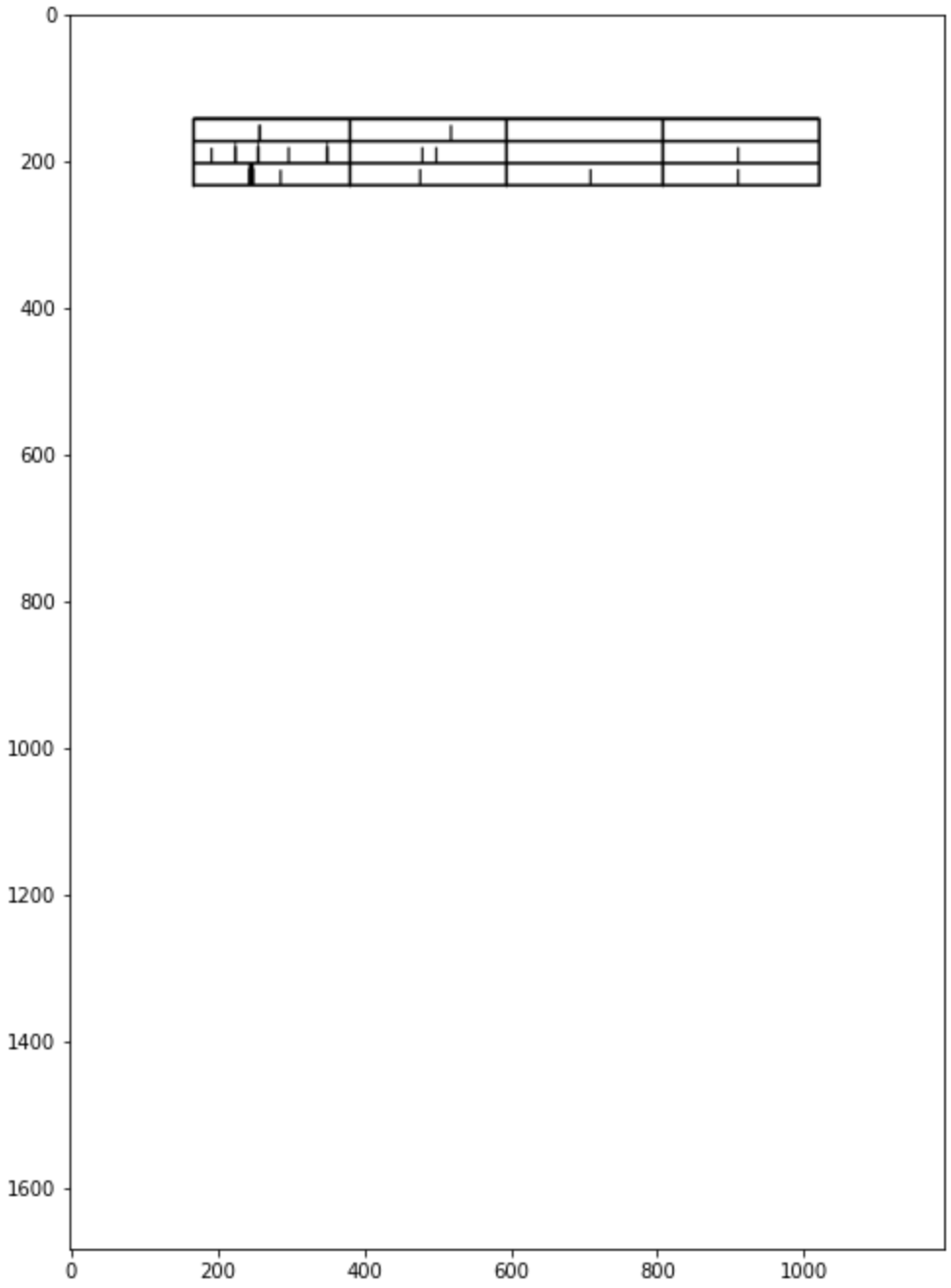
To extract tables, we use the `get_tabular_image()` function from libtable.py. We pass it an image and two main parameters; *blur_thresh* and *kernel_ratio*.
`blur_thresh` : controls the erosion to be applied to the image in which tables are to be detected. It can be seen as the parameter that blurs out/erodes everything else except the tables you want. Therefore, if the image has a table and alot of lines in it, you can set this parameter high in order to erode those lines and leave the table visible. For a clear image such as the one we are using above, we can set this value low.
`kernel_ratio` : controls the thickness of the table borders to be detected (with respect to the size of the image). The default value is 0.01 and is a perfect value for most cases and won't need adjustment.

This function can take more parameters however these are the most important. Use `help(libtable.get_tabular_image)` to see more.

Lets look at the effects of changing `blur_thresh` as we attempt to find a good value for it.

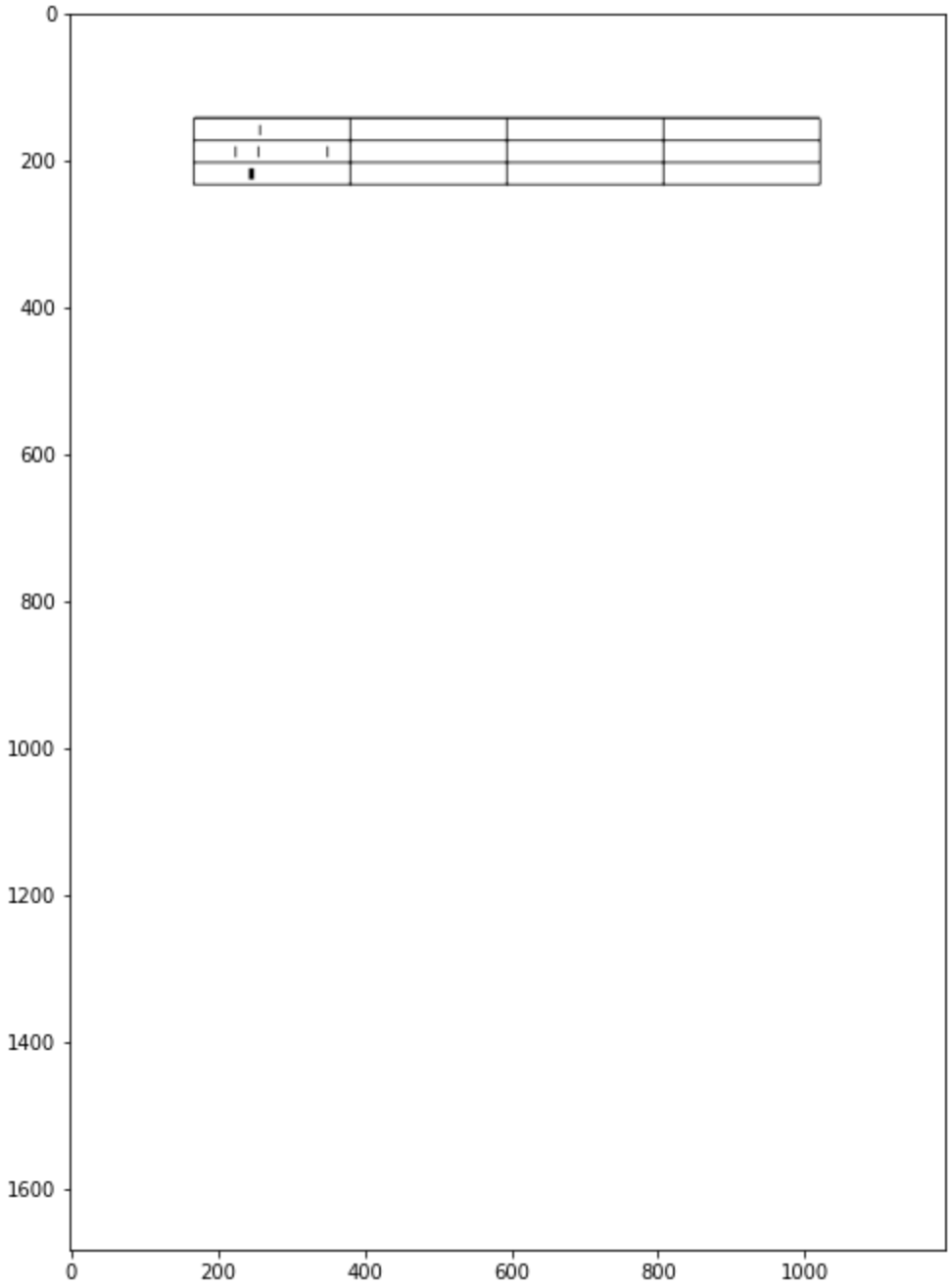
In [3]:
```
blur_thresh = 10  # <---- starting with a value of 10
```

```
kernel_ratio=0.01
table_img = lt.get_tabular_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rat

plt.figure(figsize=(8, 15))
plt.imshow(table_img, cmap='gray_r')
plt.show()
```



There are some unwanted vertical lines in the above image. To remove them and have a result similar to the wanted table, we shall increase the blur_thresh, so as to erode these lines out.

In [4]:
```
blur_thresh = 60  # <------- increased to 60
kernel_ratio=0.01
table_img = lt.get_tabular_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rat

plt.figure(figsize=(8, 15))
plt.imshow(table_img, cmap='gray_r')
plt.show()
```

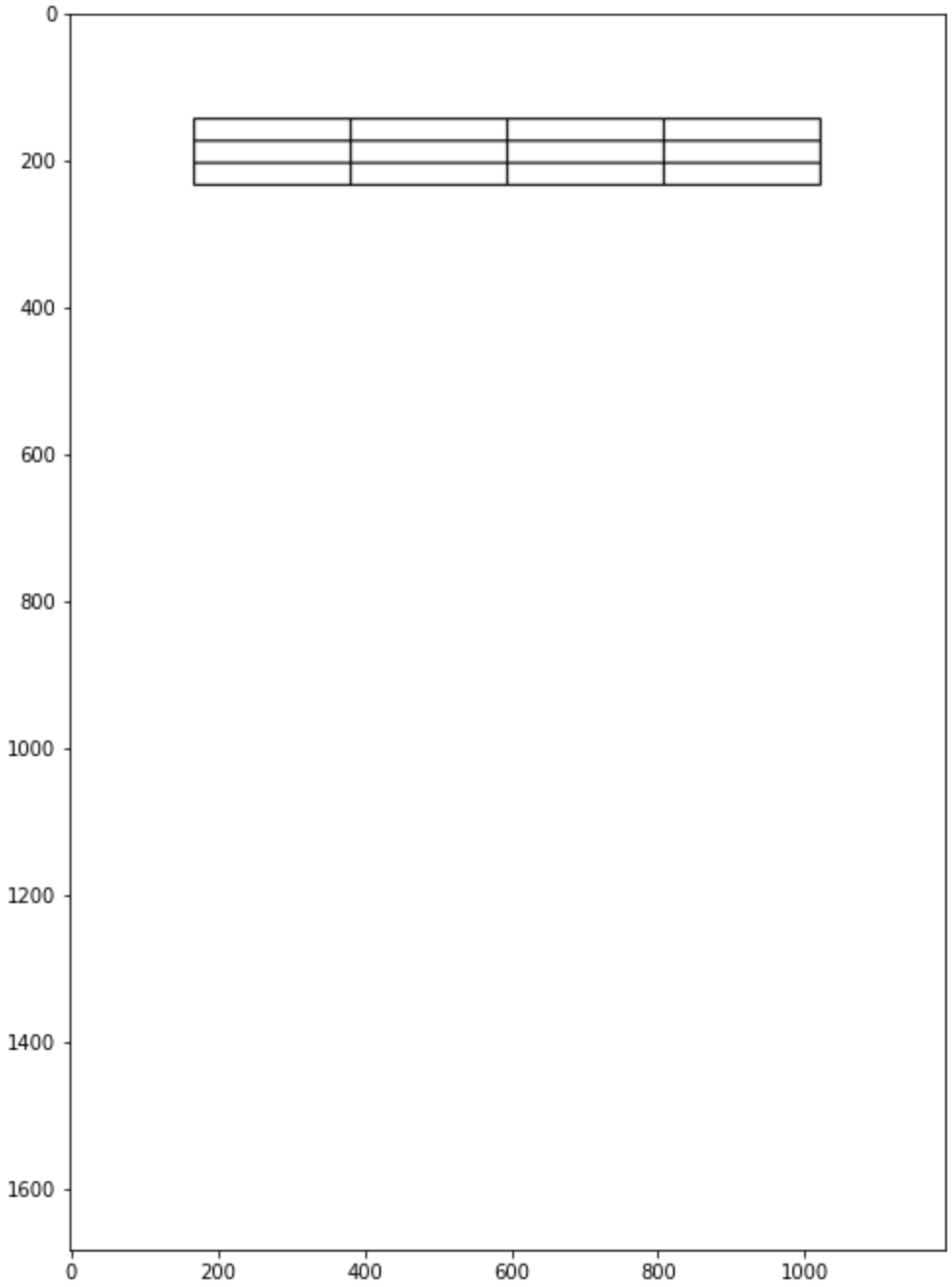


That's better. Note that we don't need to match the exact table, we just need to have something as similar to it as possible. This step only serves to guide the algorithm on where to find the tables by highlighting their positions in the image. Then, we can pass the values we have got to `libtable.get_tables_in_image` as shown below.

In [5]:
```
blur_thresh = 60
kernel_ratio=0.01
tables = lt.get_tables_in_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rati
                                        min_cell_size=10, min_table_width_ratio=0.001)
```

We can now view the detected tables using `libtable.get_tabled_mask` by passing it the image, the tables and the thickness to be used in drawing the table borders.

In [6]:
```
imm = lt.get_tabled_mask(image, tables, thickness=2)
plt.figure(figsize=(8, 15))
plt.imshow(imm, cmap='gray')
plt.show()
```



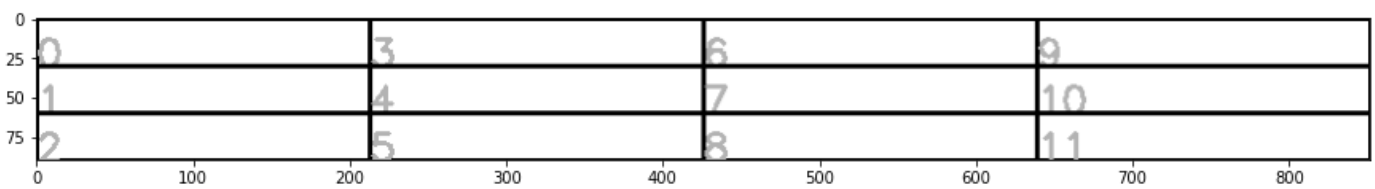
Since we have one table in the image, we can put it in a variable and do operations on it.

In [7]:
```
table = tables[0]
```

We can;

1. preview the table (see `help(table.preview)`)

In [8]:
```
imm = table.preview(numbered=True)
plt.figure(figsize=(15, 6))
plt.imshow(imm, cmap='gray')
plt.show()
```



1. perform OCR to fill the table cells using text from the image (see `table.fill_from_image`)

In [9]:
```
table.fill_from_image(image, border_pad=1)
```

1. pretty_print the table (see `table.print`)

In [10]:
```
table.print(pad=3)
```

```
|        Name        | Score 1  | Score 2  | Score 3  |
|  Mulindwa Joseph   |   1.4    |    65    |    45    |
|  Camille Martinez  |   1.7    |    54    |    47    |
```

1. get a JSON string or dictionary from the table. This dictionary or JSON string can include positions (x-y co-ordinates) of each individual word in the each table. (see `table.as_json`)

In [11]:
```
js = table.as_json(include_data=False, as_dict=False)
print(js)
```


```
{
    "type": "table",
    "table_id": 0,
    "xmin": 169,
    "ymin": 145,
    "xmax": 1021,
    "ymax": 235,
    "cells": [
        {
            "row": 0,
            "col": 0,
            "row_span": 1,
            "col_span": 1,
            "xmin": 169,
            "ymin": 145,
            "xmax": 382,
            "ymax": 175,
            "score": 96.818092,
            "text": "Name"
        },
        {
            "row": 1,
            "col": 0,
            "row_span": 1,
            "col_span": 1,
            "xmin": 169,
            "ymin": 175,
            "xmax": 382,
            "ymax": 205,
            "score": 93.91569150000001,
            "text": "Mulindwa Joseph"
        },
        {
            "row": 2,
            "col": 0,
            "row_span": 1,
            "col_span": 1,
            "xmin": 169,
            "ymin": 205,
            "xmax": 382,
            "ymax": 235,
            "score": 96.3505705,
            "text": "Camille Martinez"
        },
```

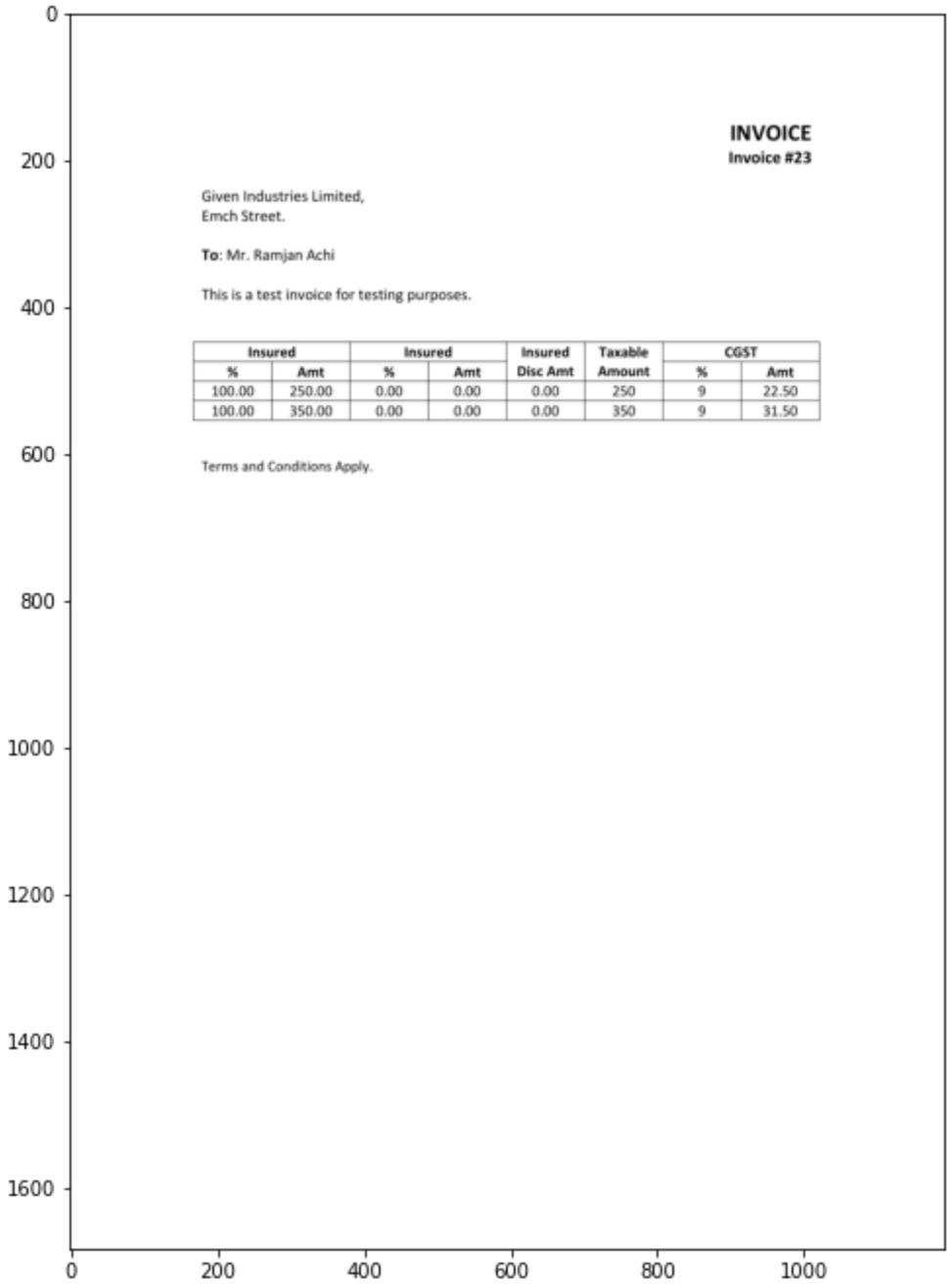
```
{
    "row": 0,
    "col": 1,
    "row_span": 1,
    "col_span": 1,
    "xmin": 382,
    "ymin": 145,
    "xmax": 595,
    "ymax": 175,
    "score": 0.0,
    "text": "Score 1"
},
{
    "row": 1,
    "col": 1,
    "row_span": 1,
    "col_span": 1,
    "xmin": 382,
    "ymin": 175,
    "xmax": 595,
    "ymax": 205,
    "score": 94.960678,
    "text": "1.4"
},
{
    "row": 2,
    "col": 1,
    "row_span": 1,
    "col_span": 1,
    "xmin": 382,
    "ymin": 205,
    "xmax": 595,
    "ymax": 235,
    "score": 86.644508,
    "text": "1.7"
},
{
    "row": 0,
    "col": 2,
    "row_span": 1,
    "col_span": 1,
    "xmin": 595,
    "ymin": 145,
    "xmax": 808,
    "ymax": 175,
    "score": 92.920273,
    "text": "Score 2"
},
{
    "row": 1,
    "col": 2,
    "row_span": 1,
    "col_span": 1,
    "xmin": 595,
    "ymin": 175,
    "xmax": 808,
    "ymax": 205,
    "score": 96.491325,
    "text": "65"
},
{
    "row": 2,
    "col": 2,
    "row_span": 1,
    "col_span": 1,
    "xmin": 595,
```

```
                "ymin": 205,
                "xmax": 808,
                "ymax": 235,
                "score": 95.224319,
                "text": "54"
            },
            {
                "row": 0,
                "col": 3,
                "row_span": 1,
                "col_span": 1,
                "xmin": 808,
                "ymin": 145,
                "xmax": 1021,
                "ymax": 175,
                "score": 78.599602,
                "text": "Score 3"
            },
            {
                "row": 1,
                "col": 3,
                "row_span": 1,
                "col_span": 1,
                "xmin": 808,
                "ymin": 175,
                "xmax": 1021,
                "ymax": 205,
                "score": 96.745003,
                "text": "45"
            },
            {
                "row": 2,
                "col": 3,
                "row_span": 1,
                "col_span": 1,
                "xmin": 808,
                "ymin": 205,
                "xmax": 1021,
                "ymax": 235,
                "score": 95.734955,
                "text": "47"
            }
        ]
}
```


For more operations and parameters, see `help(libtable.Table)`

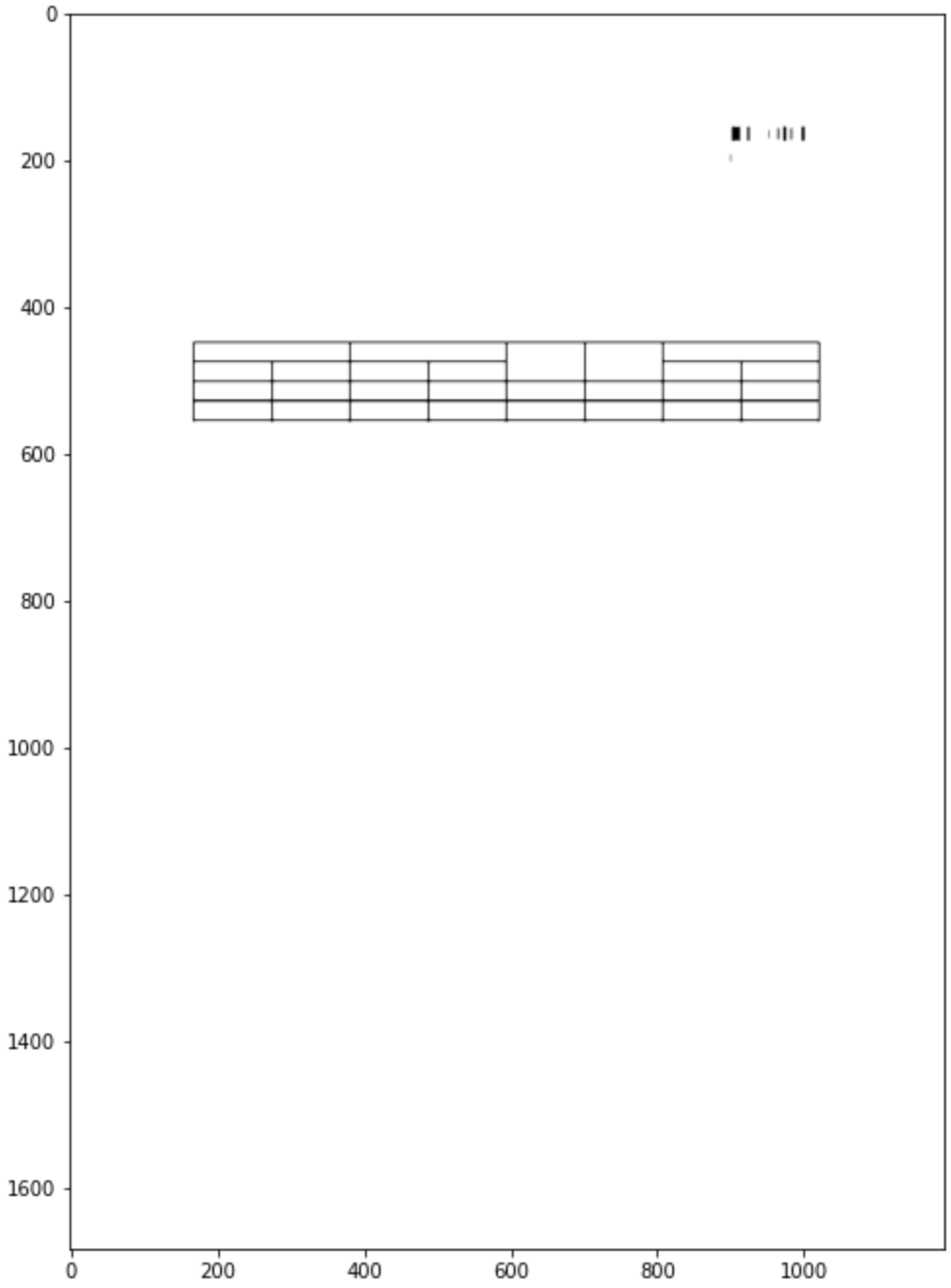
Lets try another PDF

In [12]:
```
images = libutil.pdf2images('data\\INVOICE.pdf', zoom=2)
image = images[0]

plt.figure(figsize=(8, 15))
plt.imshow(image, cmap='gray')
plt.show()
```

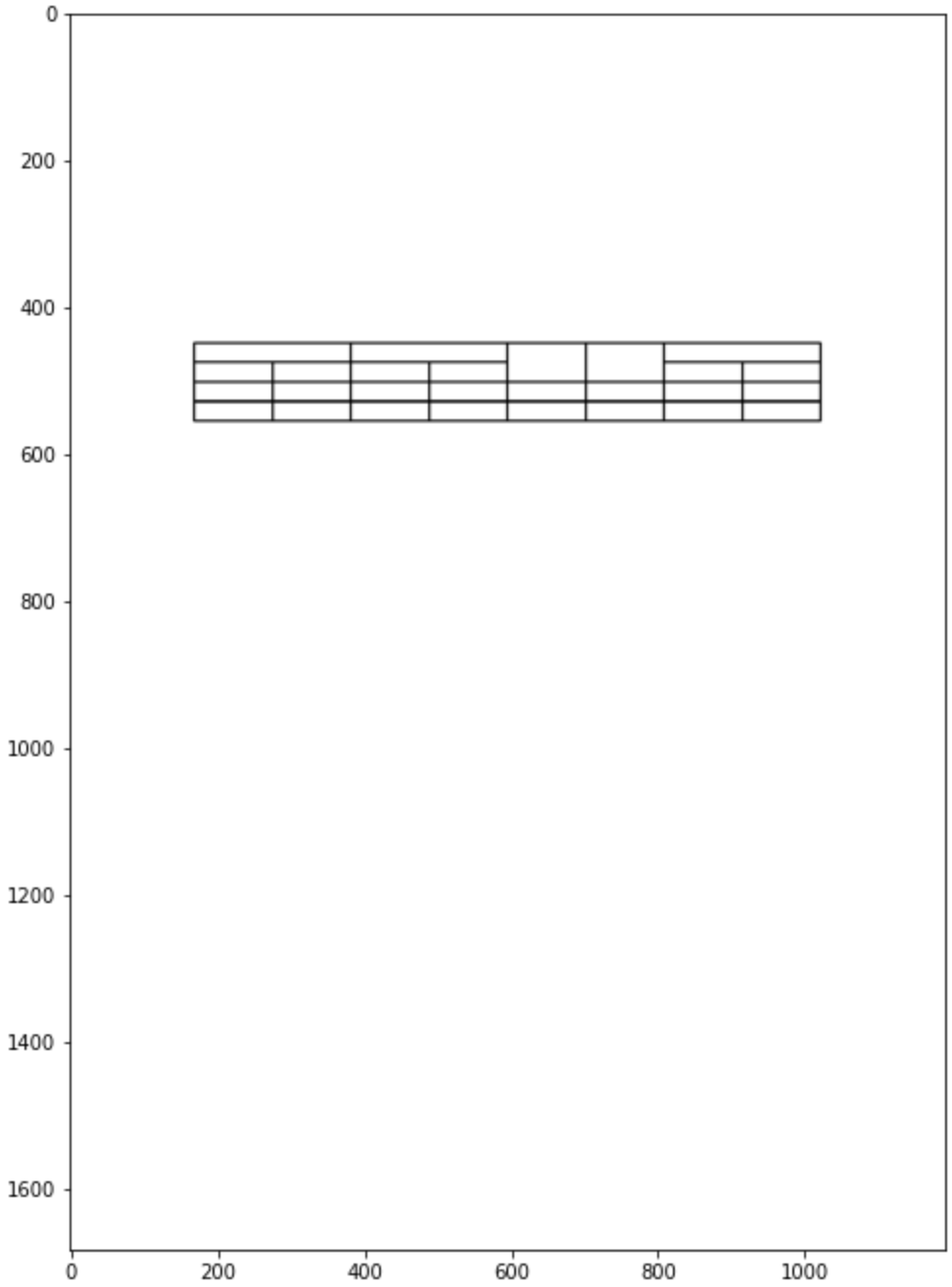


Insured		Insured		Insured	Taxable	CGST	
%	Amt	%	Amt	Disc Amt	Amount	%	Amt
100.00	250.00	0.00	0.00	0.00	250	9	22.50
100.00	350.00	0.00	0.00	0.00	350	9	31.50

In [13]:
```
blur_thresh = 50
kernel_ratio=0.01
table_img = lt.get_tabular_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rat

plt.figure(figsize=(8, 15))
plt.imshow(table_img, cmap='gray_r')
plt.show()
```



In [14]:
```
tables = lt.get_tables_in_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rati
                                        min_cell_size=10, min_table_width_ratio=0.001)

imm = lt.get_tabled_mask(image, tables, thickness=2)
plt.figure(figsize=(8, 15))
plt.imshow(imm, cmap='gray')
plt.show()
```



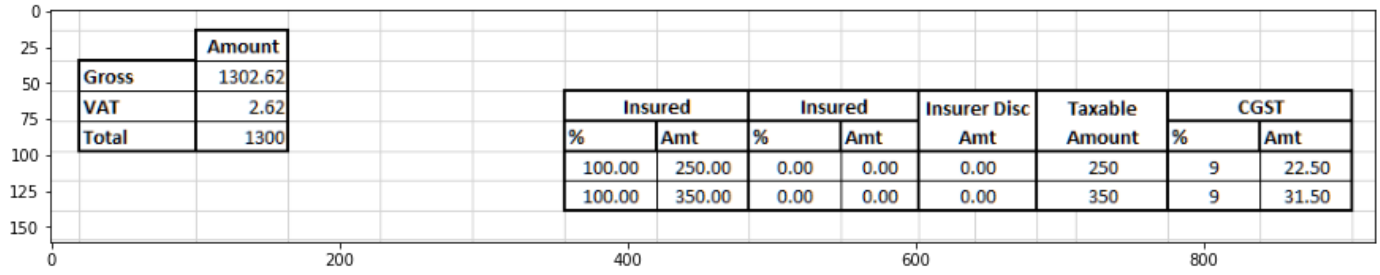
In [15]:
```
tables[0].fill_from_image(image)
tables[0].print()
```

```
|      Insured       |      Insured       |  Insured  | Taxable  |       CGST       |
|     %      |   Amt    |     %      |  Amt  |  Disc Amt |  Amount  |    %    |  Amt   |
|  100.00    |  250.00  |    0.00    |  0.00 |     0.00  |    250   |    9    | 22.50  |
|  100.00    |  350.00  |    0.00    |  0.00 |     0.00  |    350   |    9    | 31.50  |
```

Extracting directly from an image

Lets load an image as a numpy array using OpenCV (cv2)

In [16]:
```
import cv2 as cv

image = cv.imread('data\\tabular.PNG')

plt.figure(figsize=(15, 6))
plt.imshow(image, cmap='gray')
plt.show()
```

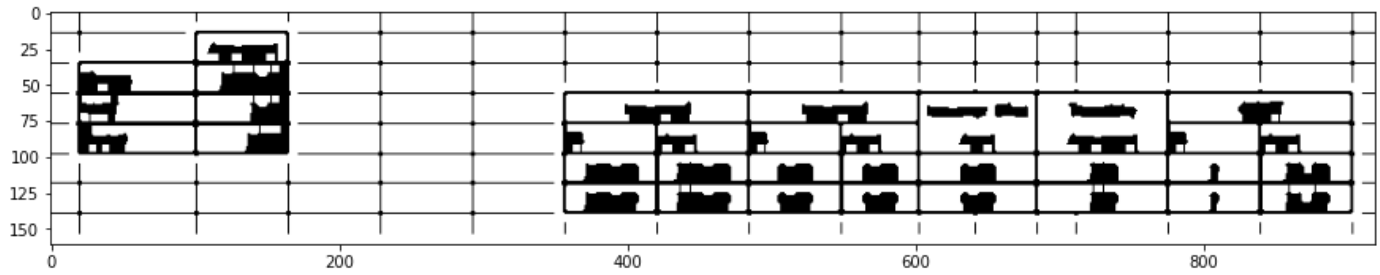


	Amount
Gross	1302.62
VAT	2.62
Total	1300

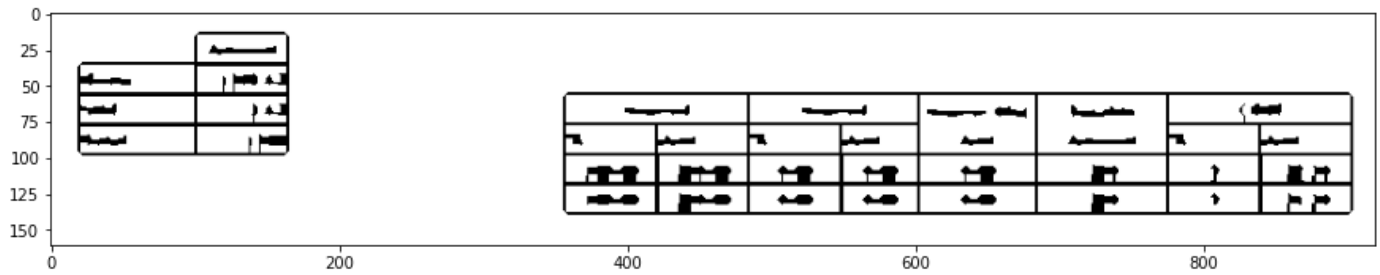
Insured		Insured		Insurer Disc	Taxable	CGST	
%	Amt	%	Amt	Amt	Amount	%	Amt
100.00	250.00	0.00	0.00	0.00	250	9	22.50
100.00	350.00	0.00	0.00	0.00	350	9	31.50

This image, which is an extract of an Excel spreadsheet, has many unwanted lines and thus will need a higher value for `blur_thresh`.

In [17]:
```
blur_thresh = 50  # <--- I set a lower value here for demonstration
kernel_ratio=0.01

table_img = lt.get_tabular_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rat

plt.figure(figsize=(15, 6))
plt.imshow(table_img, cmap='gray_r')
plt.show()
```

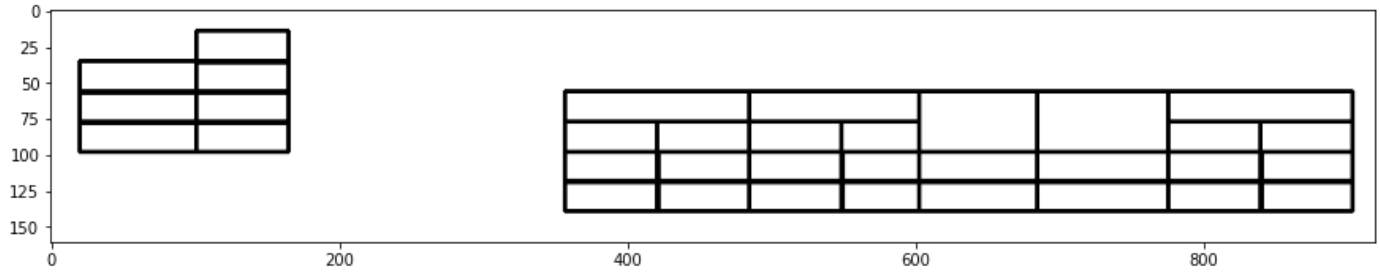


In [18]:
```
blur_thresh = 90
kernel_ratio=0.01
table_img = lt.get_tabular_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rat

plt.figure(figsize=(15, 6))
plt.imshow(table_img, cmap='gray_r')
plt.show()
```



A value of 90 highlights the tables fairly well.

In [19]:
```
tables = lt.get_tables_in_image(image, blur_thresh=blur_thresh, kernel_ratio=kernel_rati
                                        min_cell_size=10, min_table_width_ratio=0.001)

imm = lt.get_tabled_mask(image, tables, thickness=2)
plt.figure(figsize=(15, 6))
plt.imshow(imm, cmap='gray')
plt.show()
```



Filling the first detected table and printing it. Tables are normally detected from bottom to top.

In [20]:
```
tables[0].fill_from_image(image)
tables[0].print()
```

```
|       Insured.       |       Insured.       |  Insurer Disc | Taxable  |       CGST       |
|     %      |   Amt   |     %      |  Amt    |      Amt      |  Amount  |   %    |  Amt    |
|   100.00   |  250.00 |    0.00    |  0.00   |      0.00     |   250    |   9    | 22.50   |
|   100.00   |  350.00 |    0.00    |  0.00   |      0.00     |   350    |   9    | 31.50   |
```

Filling and printing the second (detected) table

In [21]:
```
tables[1].fill_from_image(image)
tables[1].print()
```

```
         |  Amount  |
| Gross  | 1302.62  |
|  VAT   |   2.62   |
| Total  |   1300   |
```