IISS Data Wrangling: OCR with Tesseract

Christiana Moore

September 2019

Introduction[1]

This paper provides an overview of my optical character recognition (OCR) work with the `tesseract` R package and can, in addition, serve as a guide for converting PNG tables into dataframes and LaTeXfiles. For this particular task, the original images were tables sourced from a multitude of scholars, spanning the years of 1980-2016, and the topics of alliances, military equipment, weaponry, and more. In light of this variation within my source data, the creation of a completely standardized process appeared difficult, as each table presented a unique set of challenges.

Nevertheless, through this project, I discovered the nuances of OCR and the factors influencing successful text extraction and conversion, including font, formatting, and overall image quality. I find that the process has the potential to be relatively standardized, and that this standardization is dependent on both the overall number of columns within the original file and the original image quality.

The remainder of this document is organized as follows: I first provide an overview of helpful resources and packages. I then proceed by walking the reader through conversions for a double column table and a low resolution table. Finally, I conclude by discussing my final thoughts and reflections on the PDF conversion project.

Resources and Helpful Packages

I began my work on this task with the resources provided to me by Andres, as follows:

- Bruno Rodrigues' **blog post** is an incredibly helpful starting point for OCR of both PDFs and images in R, specifically in regards to working code examples.
- The `magick` package is necessary for reading in and processing the image. As I discuss later in length, image preprocessing is crucial to ensuring the quality of text extraction, especially for older tables.
- The `tesseract` package provides our OCR tool.
- The `tidyverse` package is practical for cleaning and creating the dataframe once the text is read into R, especially the `dplyr %>%` operator and `purrr`.
- Since I was previously unfamiliar with regular expressions, **regex guides** were helpful for string extractions.
- Finally, the `xtable` helps transform the final dataframe into a .tex table.

OCR for Double/Repetitive Columns: Haesebrouck 2016

Our first example file is "Democratic Members Coalition Against IS" (Haesebrouck 2016), a table with repetitive/double columns. As depicted in Figure 1, the columns are placed in a wide format, which can

[1]Thank you to J. Andres Gannon for your help with the PDF conversion project and to everyone at cPASS for your support this summer!

make string extraction difficukt. Otherwise, this table features relatively clear text and few unusual symbols, making image preprocessing unnecessary.

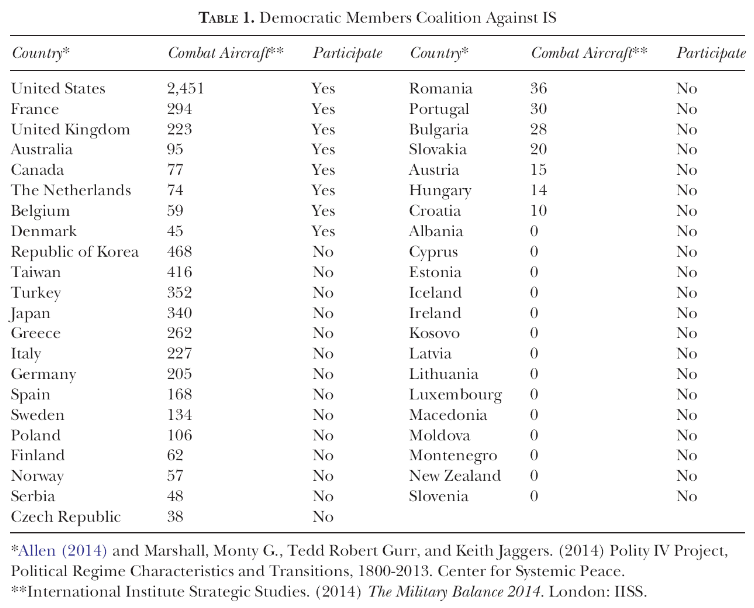

Table 1. Democratic Members Coalition Against IS

*Country**	*Combat Aircraft***	*Participate*	*Country**	*Combat Aircraft***	*Participate*
United States	2,451	Yes	Romania	36	No
France	294	Yes	Portugal	30	No
United Kingdom	223	Yes	Bulgaria	28	No
Australia	95	Yes	Slovakia	20	No
Canada	77	Yes	Austria	15	No
The Netherlands	74	Yes	Hungary	14	No
Belgium	59	Yes	Croatia	10	No
Denmark	45	Yes	Albania	0	No
Republic of Korea	468	No	Cyprus	0	No
Taiwan	416	No	Estonia	0	No
Turkey	352	No	Iceland	0	No
Japan	340	No	Ireland	0	No
Greece	262	No	Kosovo	0	No
Italy	227	No	Latvia	0	No
Germany	205	No	Lithuania	0	No
Spain	168	No	Luxembourg	0	No
Sweden	134	No	Macedonia	0	No
Poland	106	No	Moldova	0	No
Finland	62	No	Montenegro	0	No
Norway	57	No	New Zealand	0	No
Serbia	48	No	Slovenia	0	No
Czech Republic	38	No			

*Allen (2014) and Marshall, Monty G., Tedd Robert Gurr, and Keith Jaggers. (2014) Polity IV Project, Political Regime Characteristics and Transitions, 1800-2013. Center for Systemic Peace.
**International Institute Strategic Studies. (2014) *The Military Balance 2014*. London: IISS.

Figure 1: Example of double column raw table (Haesebrouck 2016)

Reading image in as a *magick* item

In line with the resources listed above, the process begins by reading in the image as an R item, as outlined in the **Bruno Rodrigues** blog.

```r
image <- purrr::map(paste0(here::here(), '/data/raw_tables/'),
                    magick::image_read)
```

Dividing *magick* item into halves

The following lines of code allow you to split the image into two halves, such that each set of headers (in this case: Country, Combat_Aircraft, and Participate) is a separate R item. Although the image can be read in and converted without this step, dividing the image into two makes string extraction and cleaning easier.[2]

```r
first_half <- purrr::map(image,
                         ~image_crop(., geometry = "710x1076"))
second_half <- purrr::map(image,
                          ~image_crop(., geometry = "689x1076+710.5+0"))
```

The process of cropping is largely dependent on trial and error; thus, I began by setting the dimensions at roughly half of the image's overall size, with 700x1076 and 700x1076+700+0, respectively, but I eventually arrived at the solution of 710x1076 for `first_half` and 689x1076+710.5+0 for `second_half`.[3]

[2]However, many tables in this project were not in double-column format, so this step is context-dependent.
[3]Since I wanted the second half of this image, the +700 signals R to crop the image beginning at the width of 700, and the +0 signals R to not crop the image's height

Joining lists before OCR

Again referencing the Rodrigues blog, the next step is to join the halves back together to prepare for text extraction.

```r
merged_list <- purrr::prepend(first_half, NA) %>%
  purrr::reduce2(second_half, c) %>%
  purrr::discard(is.na)
```

The purrr package provides functionality to combine the two lists, but the lists must be of unequal lengths, so we add NAs to `first half`, which are later discarded.

```r
purrr::prepend(first_half, NA)
```

```r
purrr::discard(is.na)
```

Optical Character Recognition with *Tesseract*

Now that we have our `merged_list`, we can convert it to strings with the `tesseract::ocr` function.

```r
text_list <- purrr::map(merged_list, tesseract::ocr)
```

As I will discuss later, the `tesseract::ocr` output can potentially be imprecise; however, in this case everything was accurate, so we move onto the next step.[4]

```r
text_list <- text_list %>%
  purrr::map(., ~str_split(., "\n"))
```

The lines of strings in our `text_list` are split by `\n`, so we use this as our indicator for `str_split`.

Dataframe Conversion and Cleaning

Since there were 58 rows in our `text_list`, we use the following code to convert the list into a dataframe.

```r
df <- data.frame(matrix(unlist(text_list), nrow=58, byrow=T),
                        stringsAsFactors=FALSE)
```

Once our text is in a dataframe, we then remove all unnecessary lines, such as column names (we add these back in later, as seen in our next step) headers, footers, and any other anomalies. I chose to just remove these rows by their position and turn the character vector back into a dataframe, as it seemed like the simplest solution.

```r
df <- df[-c(1:2, 25:30, 52:58), ]
df <- df(df)
```

Our next step is to split the strings into separate variables (using regular expressions), naming our variables/columns in line with the source image. We then delete the original column, leaving us with our desired dataframe.

```r
df <- df %>%
  dplyr::mutate(country = stringr::str_extract(df, "[aA-zZ ]+"),
```

[4]Important things to look out for that don't always read in properly include: special characters, decimal points, similar letters/numbers, etc.

```
  combat_aircraft = stringr::str_extract(df, "[0-9,?]+"),
  participate = stringr::str_extract(df, "[aA-zZ]+$"))

df$df <- NULL
```

Notably, this table featured distinct patterns within the strings, making extraction a simpler process. However, the space in 'United States' and the comma in 2,451 had to be accounted for, as seen in the code below:

```
(country = stringr::str_extract(df, "[aA-zZ ]+"
combat_aircraft = stringr::str_extract(df, "[0-9,?]+")
```

Finally, we convert this dataframe into a .tex file with the xtable package.[5]

```
print(xtable::xtable(df, type = "latex"), include.rownames = F,
                     file = paste0(here::here(),'/data/cleaned_tables/'".tex")
```

This returns the following output, as seen in the sample below:[6]

country	combat_aircraft	participate
United States	2,451	Yes
France	294	Yes
United Kingdom	223	Yes
Australia	95	Yes
Canada	77	Yes
The Netherlands	74	Yes

OCR for Lower Resolution Images: Nelson 1984

Although our first table did not require additional cleaning, many tables need extra cleaning/processing due to poor image quality and a lack of patterns within strings. "Distribution of Involvement" (Nelson 1984), in line with many older files, initially resulted in a less precise read without preprocessing. However, due to the table's lack of double/repetitive columns, we can skip the image division.

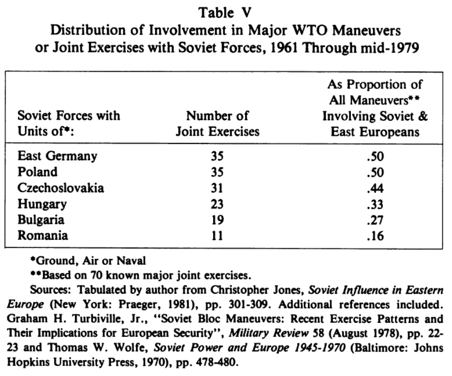

Table V
Distribution of Involvement in Major WTO Maneuvers or Joint Exercises with Soviet Forces, 1961 Through mid-1979

Soviet Forces with Units of*:	Number of Joint Exercises	As Proportion of All Maneuvers** Involving Soviet & East Europeans
East Germany	35	.50
Poland	35	.50
Czechoslovakia	31	.44
Hungary	23	.33
Bulgaria	19	.27
Romania	11	.16

*Ground, Air or Naval
**Based on 70 known major joint exercises.
Sources: Tabulated by author from Christopher Jones, *Soviet Influence in Eastern Europe* (New York: Praeger, 1981), pp. 301-309. Additional references included. Graham H. Turbiville, Jr., "Soviet Bloc Maneuvers: Recent Exercise Patterns and Their Implications for European Security", *Military Review* 58 (August 1978), pp. 22-23 and Thomas W. Wolfe, *Soviet Power and Europe 1945-1970* (Baltimore: Johns Hopkins University Press, 1970), pp. 478-480.

Figure 2: Example of raw table with lower resolution (Nelson 1984)

[5]Both xtable and stargazer returned similar tables, but the xtable version was slightly cleaner.
[6]Full table available in '/IISS/data/cleaned_tables'

Reading image in as a *magick* item and OCR

As with our previous example, we read in the image as a `magick` item.

```
image <- magick::image_read(paste0(here::here(),
'/data/raw_tables/training-frequency_WarsawPact_1961-1979_nelson-1984.png'))
```

However, for poorer quality (generally older) images, preprocessing is needed prior to conducting our OCR process. Through `magick` functions, we can resize, brighten, saturate, and enhance our image. The extent to which this is needed depends on the quality of your image file, but I successfully utilized the below code for the majority of tables.[7] We then apply `ocr` and `str_split` functions, resulting in our desired list item.

```
image <- image %>%
  magick::image_resize("2000x") %>%
  magick::image_crop(., geometry = "3000x3000") %>%
  magick::image_modulate(brightness = 75, saturation = 200) %>%
  magick::image_trim(fuzz = 40) %>%
  magick::image_contrast(sharpen = 100) %>%
  magick::image_enhance() %>%
  magick::image_write(format = 'png', density = '300x300') %>%
  tesseract::ocr() %>%
  purrr::map(., ~str_split(., "\n"))
```

In spite of our preprocessing, this solution proved imperfect, prompting further cleaning. Decimal points, spacing, and font all had varying effects on the quality of the text extraction; consequently, I had to manually clean some data (again deleting headers and footers, but I also recoded some data). For ease of string extraction, I temporarily coded East Germany as EastGermany so I could later split the strings by spaces.

```
df <- data.frame(matrix(unlist(df),nrow=25, byrow=T),
                              stringsAsFactors=FALSE)

df[8,] <- "EastGermany 35 .50"
df[9,] <- "Poland 35 .50"
df[11,] <- "Hungary 23 .33"
df[13,] <- "Romania 11 .16"

df <- df[-c(1:7, 14:25), ]
```

Finally, we split the strings by spaces, conduct our final cleaning, and convert the data frame into a LaTeXtable, as seen below:[8]

```
df <- reshape2::colsplit(df, " ", c("soviet_force_unit",
                                    "joint_exercises",
                                    "proportion_maneuvers"))
```

[7]The image preprocessing quantities were mostly determined by trial and error.
[8]As with the previous table, the original column names are added back in this step.

```
df[df == "EastGermany"] <- "East Germany"

print(xtable::xtable(df, type = "latex"), include.rownames = F,
      file = paste0(here::here()'/data/cleaned_tables'))
```

soviet_force_unit	joint_exercises	proportion_maneuvers
East Germany	35	0.50
Poland	35	0.50
Czechoslovakia	31	0.44
Hungary	23	0.33
Bulgaria	19	0.27
Romania	11	0.16

Discussion

The Haesebrouck and Nelson table conversions exemplify that, while each image presents its own set of challenges, certain aspects remain fairly standard. Important features, such as the presence of double/repetitive columns or the raw image quality, can affect the process, but tables sharing specific aspects can be converted with similar code. Although the `ocr` tool proves imprecise at times, image preprocessing can reduce cleaning to a few short lines of code. These solutions may not be as elegant as they could be (especially compared to those an advanced R user would produce), but I am grateful to have learned and improved throughout both this project and the creation of this document. [9]

Sources

- https://www.brodrigues.co/blog/2019-03-31-tesseract/
- https://cran.r-project.org/web/packages/magick/index.html
- https://cran.r-project.org/web/packages/tesseract/tesseract.pdf
- https://cran.r-project.org/web/packages/tidyverse/index.html
- https://cran.r-project.org/web/packages/xtable/index.html
- https://www.rexegg.com/regex-quickstart.html

[9]Full code for all tables can be found in IISS/docs/01c_PrepTables_external.rmd