PLSC31101 Final Project Narrative

Emily Tallo

December 2019

R Markdown

Background on the project

In this final project, my aim was to webscrape data from roughly 19 tables located on the South Asia Terrorism Portal (SATP). SATP is a website run out of New Delhi, India by the Institute for Conflict Management, an organization tracking terrorism developments in South Asia. These tables contain data on major incidents involving terrorist organizations by year from 2000-2018, such as terrorist attacks, counterinsurgency operations, etc.

I wanted to use these data for an ongoing project on the role that state-sponsored terrorist attacks can play in provoking interstate crises. Precise, event-level data on terrorist attacks successfully carried out by Pakistan-sponsored terrorist groups would help my co-author and I test a theory of why interstate crises sometimes do – but oftentimes do not – arise from state-sponsored terrorist attacks in the India-Pakistan context.

Due to the dearth of data publicly available and especially official data on terrorist incidents in India, SATP is the best source of data on such incidents. However, their data is difficult to extract and not available to download into spreadsheets. Knowing this, I came into this project with two key goals: 1. Webscrape the data;
2. Extract the locations, dates, and terrorist organizations attributed to the incidents using a combination of string manipulation and Named Entity Recognition (NER); and
3. Visualize the data gathered.

Challenge #1: Webscraping the data

Collecting the data from SATP presented a few issues. I wanted to scrape the information from the date and incidents columns that you can see in the above image. After reading in the html text

from the website, I realized immediately that there were a variety of duplication, formatting, and whitespace issues with this information.

To start the webscraping process, I wrote a loop that would iterate over a vector of years to create the vector of the urls I wanted to draw from.

```
  #1. Create a vector of the 19 years SATP has data on, 2000-2018
urlyears <- c(2000:2018)
  #2. Create an empty vector of 19 years
output <- vector(mode = "character", length(urlyears))
  #3. Iterate over each of the values in the "urlyears" vector we created
for (i in seq_along(urlyears)) {
  #4. Concatenate the string and the years #of each url
  output[i] <- str_c("https://www.satp.org/other-data/india/major-incidents_",
                     as.character(urlyears[i]))

}
  list_urls <- output
```

I created a function to parse the data from any input URL. Unfortunately, this is what the date and incident columns looked like after I attempted to use my parse_url function.

```
[1] "A fresh encounter broke out in the same district on April 10 when security forces conducting search
operations in Nathapora came under fire from militants. Four militants were killed and one civilian was
injured.\r\n                                    Read more...\r\n                                    \r\n
\r\n                                    A fresh encounter broke out in the same district on April 10 when
security forces conducting search operations in Nathapora came under fire from militants. Four militants were
killed and one civilian was injured.\r\n                                    Read less..."
```

```
[[1]]
 [1] ""                  "April - 10 "   "April - 10 "   "April - 10 "
 [5] "April - 12 "   "April - 12 "   "April - 13 "   "April - 14 "
 [9] "April - 15 "   "April - 16 "   "April - 16 "   "April - 16 "
[13] "April - 16 "   "April - 19 "   "April - 19 "   "April - 21 "
[17] "April - 21 "   "April - 22 "   "April - 23 "   "April - 26 "
[21] "April - 26 "   "April - 27 "   "April - 28 "   "April - 29 "
[25] "April - 3 "    "April - 4 "    "April - 7 "    "April - 7 "
[29] "April - 7 "    "April - 9 "    "April - 9 "    "April - 9 "
```

As you can see, the HTML text on the SATP website includes both the "long" and "short" version of the incident description. Additionally, the date column does not contain the year of the incident and has unnecessary characters.

To resolve the issue with the incident column, I had to create a "condense" function to remove duplication and trim whitespace. The condense function makes abundant use of regular expressions to split the string into two parts (before and after the "Read more…" in the above text), takes only the second half, removes the "Read less…" section, and trims whitespace. The condense function worked when I inserted it as part of a map function into the parse_url function before "return(df)".

```
condense <- function(full_incident) {
  part1 <- str_split(full_incident, "(?<=Read more\\.\\.\\.)")[[1]]
  part2 <- tail(part1, 1)
  part3 <- str_remove(part2, "Read less\\.\\.\\.")
  part4 <- str_replace_all(part3, "[\\r\\n]", "")
  final_text <- trimws(part4)
  return(final_text)
}
```

To format the date column for data manipulation/visualization later, I needed to alter the date strings using regular expressions, add in the year from the page's header, and use the lubridate package to format the column as a date.

The entire parse_url function is below:

```
parse_url <- function(input_url){
doc <- read_html(input_url)

#create year variable for page
year <- html_nodes(doc, "h1") %>%
  html_text() %>%
  str_extract("[0-9]+")

#create date column
date <- html_nodes(doc, "td:nth-child(1)") %>%
  html_text(trim=T) %>%
  str_replace_all("\\ - ", ",") %>%
  str_extract_all(".+(?=\\ )") %>%
  unlist() %>%
  paste(",") %>%
  paste(year) %>%
  mdy()

#create incident column
incident <- html_nodes(doc, "td:nth-child(2)") %>%
  html_text(trim=T)

#remove rows that do not contain any relevant incident info (blank or otherwise)
incident <- incident[grep("Read more\\.\\.\\.", incident)]

#create dataframe with date and incident
df <- data.frame(date, incident, stringsAsFactors = FALSE)
names(df) <- c("date", "incident")

#map condense function (which condenses text and trims whitespace) over all the incidents
df$incident <- map_chr(df$incident, condense)

#return a dataframe of date and incident info for the URL
return(df)
}
```

Challenge #2: Named Entity Recognition

To extract named entities (districts, states, and terrorist organizations) from the code, I turned to two R packages. I tried to use the openNLP and MonkeyLearn packages, but openNLP failed to recognize most of the non-English entities and MonkeyLearn had a query limit that I soon ran into. I had to solve the problem manually, so I decided to manually create dictionaries of the named entities I wanted to extract.

At first I wrote them directly into the text of the function, but this made my code unwieldy. I also

wanted to be able to change the content of the lists without touching the code, so I exported the lists into csv files and changed the format slightly: I created two columns, one that lists a way of referencing the organization (e.g. Maoist, Naxal, Naxalite) and another that lists the final output (i.e. Communist Party of India-Maoist) (see image below). The upside of doing this was that I was able to capture multiple ways of referencing a single named entity and produce a consistent set of spellings in what is ultimately inserted into my data. However, I had to create a three input function that would detect anything from the first column and output the adjacent value in the second column.

Al Mansurian	Al Mansoorian
Al Mansoorian	Al Mansoorian
Al Shuda Brigade	Al Shuda Brigade
All Tripura Tiger Force	All Tripura Tiger Force
Al Qaeda	Al Qaeda
Al Umar Mujahideen	Al Umar Mujahideen
Babbar Khalsa International	Babbar Khalsa International
Communist Party of India (Maoist)	Communist Party of India (Maoist)
Maoist	Communist Party of India (Maoist)
Naxal	Communist Party of India (Maoist)
Naxalite	Communist Party of India (Maoist)
Naxalites	Communist Party of India (Maoist)

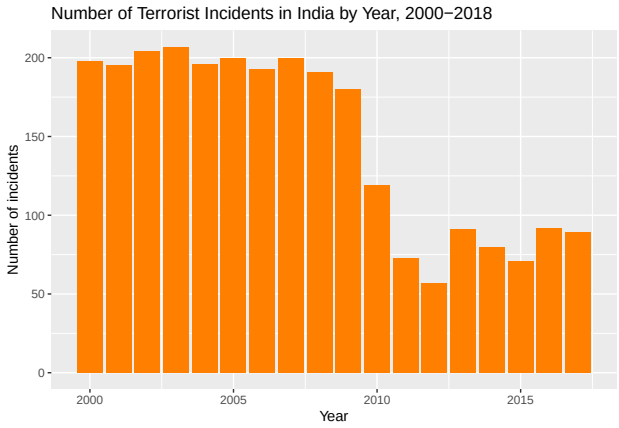
If combined, the next two functions do the following: 1. For each of the elements in the vector of named entities, check to see if they appear in a given text.
2. Then, if an element appears, return the element of the same index in the final_entities vector.
3. If the character vector is empty after that, print NA.
(Note that exists_in_string is called in the extract_string function, and is separated here only for simplicity.)

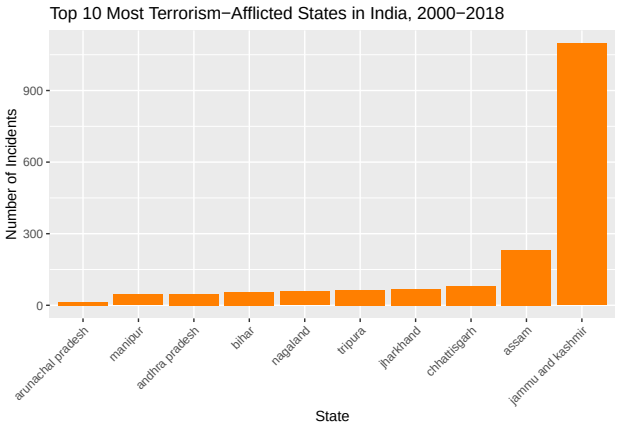
```
exists_in_string <- function(s1, s2){
  return(grepl(s1, s2))
}

extract_string <- function(text, entities, final_entities) {
  for (i in seq_along(1:length(entities))) {
    if(exists_in_string(entities[i], text)) {
      return(final_entities[i])
    }
  }
  return(NA)
}

#Drawing upon the processed text of the incidents and my list of
#terrorist organziations, I can now map the function

attr_org_col <- map(processed_incident_col, extract_string, org, final_org) %>%
  unlist()
```

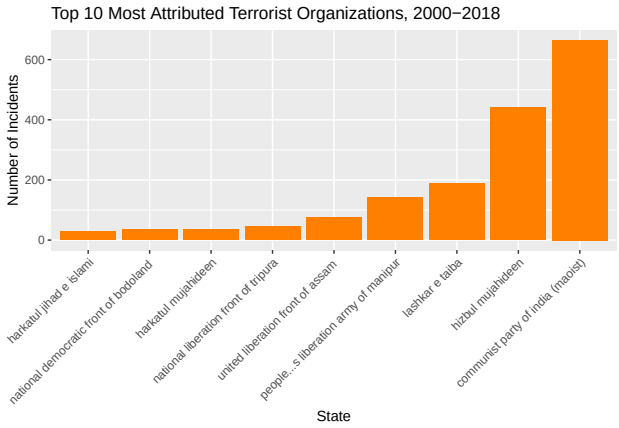
Analysis & Visualization

The final step was to analyze/visualize the data. I ended up with 2636 observations at the incident level, for many of which I was unable to locate a named district, state, and/or terrorist organization. There were 771 of 2636 missing terrorist organizations, 784 missing states, and (as expected) 1552 missing districts (since I only matched districts for the most clearly terrorism-afflicted state in India, Jammu and Kashmir).

Visualizations of the processed data after extracting the named entities are below (NAs removed):







Future work will seek to decrease the number of NAs by adding more ways of referencing terrorist organizations to the lists, inserting missing terrorist organizations that are not currently on my list, and locating detailed geodata that names all districts in the country.