R Notebook

I had a client ask me to analyze mentions of the political party Vox in Spanish newspapers in June of 2023. They were writing a paper about attitudes toward the far right party and had a discontinuity in their results on June 13, the day that Vox entered a regional coalition with a center right party. My client wished to argue that entering a coalition with an established political party had changed Vox's public image from a maverick pariah party to a mainstream party, and that they could detect the change with their survey.

The only problem with that argument is that careful reviewers might accuse them of p-hacking by choosing a particular day for the regression discontinuity in their results. An unscrupulous researcher could run a survey experiment then keep trying discontinuities over each day in the survey to find one which gave them a significant result. Try June 1st, if no stars, then try June 2nd and so on until you find a significant result. My clients wanted to argue that they had not done this, so they needed to show that June 13th had a highly important event that could plausibly change the survey. They needed an argument about the date's importance that could not be made for just any day in the survey. Moreover, we needed an argument that would convince political scientists all over the world, and could not rely on a deep knowledge of Spanish politics.

I proposed to the professors that we look at mentions of Vox in Spanish newspapers before and after the coalition announcement. If June 13th was a big deal then all the papers should be talking about Vox on June 14th. The argument is simple and requires no deep knowledge of Spanish politics, so they commissioned me to build the dataset.

The plan

To meet my clients needs, I needed to scrape a set of spanish newspapers once a day throughout June, automatically count the number of times Vox is mentioned, and create visually appealing plots to go in their paper. For ease of use, I chose to use archive.org as the source of my scrapes because they have the most comprehensive coverage of Spanish newspapers and provide free access. More controvertially, I decided to use both Python and R. Python has more functionality for webscraping than R, but R is better at massaging data and visualizations (using both also lets me show off more). In general I would not recommend this approach; working purely in R would have been easier.

Code documentation

First I gather my python packages.

```
#loading packages
import sys
import requests as rq
from bs4 import BeautifulSoup as bs
from time import sleep
from time import time
from random import randint
from warnings import warn
import json
```

```
import pandas as pd
import re
```

Scraping Archive

This section first creates a list of the base urls for each of the newspapers, with a string afterward that tells archive.org to give me the first scrape each hour for that month of June 2023 in a csv format. This provides a spreadsheet that gives a link to each backup of the website that archive.org has saved. The for loops below then go to that site, grab each spreadsheet, and combine them into a dataframe that contains the link to each spreadsheet.

```
#creating a list of urls to search. The syntax specifies the days.
url = []
url.append('http://web.archive.org/cdx/search/cdx?url=www.elmundo.es/&collapse=timestamp:10&from=202306
url.append('http://web.archive.org/cdx/search/cdx?url=https://www.larazon.es//&collapse=timestamp:10&fr
url.append('http://web.archive.org/cdx/search/cdx?url=https://www.eldiario.es/&collapse=timestamp:10&fr
url.append('http://web.archive.org/cdx/search/cdx?url=https://elpais.com/?ed=es&collapse=timestamp:10&f
url.append('http://web.archive.org/cdx/search/cdx?url=https://www.lavanguardia.com/&collapse=timestamp:


#initial parse_url length is 1333
parse_url = []
# Gets a list of urls for archived versino of the sites in the url list
for i in url:
  print(i)
  urls = rq.get(i).text
  parse_url.extend(json.loads(urls))

## Extracts timestamp and original columns from urls and compiles a url list.
url_list = []
for i in range(1,len(parse_url)):
    orig_url = parse_url[i][2]
    tstamp = parse_url[i][1]
    waylink = tstamp+'/'+orig_url
    url_list.append(waylink)

d = {'full_url': url_list}
df = pd.DataFrame(data=d)
```

Now that I have a spreadsheet that lists each of the addresses of scrapes, I need to convert it into useful data. This next piece uses R because that makes the data management a bit easier. The urls that archive.org gives me have a specific format where the date and time of the scrape come first and the website comes second, so I split those up to get variables for the date and time of day of each scrape.

I only want one scrape each day from the 10:00 am paper, so I run a filter to remove all the others. Finally in the comments there are scripts that check for any newspapers that have too many missing days or days that have too few newspapers.

```
#now I want to massage the url data to give us the time taken, which I will use later for plotting
setwd("C:/Users/liptr/Box/TL_LBV_APP/vox_analysis")

library(reticulate)
library(tidyverse)
```

```r
url_list <- py$url_list

df <- data.frame(url_full = url_list)

#note that archive.org times are in UTC and spain uses UTC + 1 because they hate me, so the time thresh
df <- df %>% mutate(time_code = substr(url_full, start = 1, stop = 14),
                    date = substr(url_full, start = 1, stop = 8),
                    time = substr(url_full, start = 9, stop = 14),
                    site =substr(url_full, start = 16, stop = 500),
                    site = str_remove(site, 'http://'),
                    site = str_remove(site, 'https://'),
                    site = str_remove(site, 'www.'),
                    site = str_remove(site, '/$')) %>%#,
#                     vox_mentions=NA) %>%
#  filter(date == "20230623") %>% view()
  filter(as.numeric(time)>90000) %>%
  group_by(date,site) %>%
  slice(which.min(as.numeric(time))) %>%
  ungroup()

#code to check for missing day
#df_together %>% group_by(date) %>%
#  summarise(count = n(),
#            nas = sum(is.na(vox_mentions))) %>% view()

#df %>% arrange(desc(as.numeric(time_code))) %>%
#                 view()

#df_together %>% group_by(site) %>%
#  summarise(count = n(),
#            nas = sum(is.na(vox_mentions))) %>% view()
```

This section loads the Document Object Model for each website for each day (that is the code which generates the newspapers front page). Then I run a very simple algorithm to count the number of times the string "Vox" appears. "Vox" is an unusual string so I don't need to check that I have the right one, except to exclude a webservice called "librivox" which can give false positives.

This is a 'quick and dirty' way of scraping because we get mentions from article titles, image titles, article text samples and all the other stuff on the front page. The other way of doing this would involve scraping each site and finding the article title class in each DOM, but this approach is hard to scale across different sites.

```python
df = r.df

df['url_full']

vox_counts = []

for i in range(0, len(df['url_full'])):
  url = df['url_full'][i]
  final_url = 'https://web.archive.org/web/'+url
  req = rq.get(final_url).text
  print(final_url)
  print(len(re.findall("vox", req, re.IGNORECASE)))
```

```
  count = len(re.findall("vox", req, re.IGNORECASE))
  false_positive = len(re.findall("librivox", req, re.IGNORECASE))
  count = count - false_positive
  vox_counts.append(count)

df['vox_mentions'] = vox_counts
```

Massaging the data

The one problem with my quick and dirty approach is that sites vary in how much text is on their site. If I graph the raw mentions the verbose sites like Diario will drown out the others. To fix this I adjust each total by dividing each sites total by the total mentions over june. This ensures an equal weighting for each site (and makes the graph more visually appealing).

```
library(tidyverse)
#normally I would take the data back from Python, but the scrape takes a long time so when writing this
#df <- py$df

#reading in backup
df <- readRDS(file = "newspaper_mentions_raw_upload.rds")

library(lubridate)

#set this for the last day included in the plot
last_day = ymd('20230630')

df <- df %>%
  mutate(date = ymd(date)) %>%
  filter(date <= last_day)

total_mentions <- df %>% group_by(site) %>%
  summarize(total_mentions = sum(vox_mentions))

df <- df %>% mutate(mentions_adj = NA)

#Different sites put different amounts of text on their frontpage, so I adjust each daily total by the
for (i in 1:nrow(df)) {
  adjustor = total_mentions[total_mentions$site == df[i, 'site'][[1,1]],2][[1,1]]
  df[i, "mentions_adj"] <- df[i, "vox_mentions"] / adjustor
}
```

Visualizing the data

This section gives a straightforward stacked barchart visualization of the results. I adda red line for the 13th to represent the day of the coalition formation.

```
df %>% filter(site != 'abc.es') %>% ggplot(aes(x=date,y=mentions_adj)) +
  geom_bar(stat = 'identity', position = 'stack',
           aes(fill = site)) +
  scale_fill_brewer(palette = "Dark2", name = "Digital Frontpage") +
  scale_x_continuous(breaks=c(ymd(20230601,20230613,20230630))) +
```

```
  #  xlim(ymd(20230601),last_day) +
  geom_vline(xintercept = ymd(20230613), linetype = "dotted",
             color = "red", size = 1) +
  labs(title = "Mentions of Vox in Spanish newspapers in June of 2023",
       x= "Date",
       y= "Number of mentions each day
(normalized by newspaper)") +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "black"))
```