readme

Sven Maurice Morlock

2022-08-26

Aim of this project is to analyze the development of Population size and church appointments in Great Britain during the time period of 1525 until 1850. Church appointments are documented in a database that can be accessed at "https://theclergydatabase.org.uk/jsp/search/index.jsp". The organizational structe of this project is structured as follows:

- data will be written in a data folder.
- scripts are stored in a separate source file.
- the web scraper functions are stored in a script called 'functions.R'.
- the data is scraped via the script 'write_data.R' and written into the data folder.
- the final analysis is done in a script called 'main.R' where necessary scripts are run via `source` command.

Some webpages did not contain relevant data. The details can be looked up in the corresponding logfiles which can be accessed at 'https://github.com/mms1410/english_church_appointment/tree/master/log', as well as the whole project.

The scraped web data contains 168945 observations and is joined with the given population data resulting in `data.table` with 14153 observations. Note that for joining the data an 'inner join' was used setting the argument `nonmatch` to zero, so only data where corresponding keys where found where maintained.

In detail the functions used for web scraping are stored in `functions.R` and are structured as follows:

The function `scrape.html.R` takes a string representing a url as input and returns a data.table object consisting of scraped data. First the webpages html content is parsed using R's curl interface written in the package Rcurl. Since scrape.html uses regular expressions, white spaces might at some occasions be unwanted, therefore the variable page.trim contains the parsed html page with removed whitespaces. Note that here the flag perl must be set to TRUE since the regex pattern contains perl syntax that would otherwise not be interpreted appropriately. The webpages requested in this project typically contains the following pattern: There are three major parts, 'Evidence', 'Summary' and 'History'. The data in the 'Evidence' and 'History' section are embedded in an html tabel while the 'Summary' data is simply written as list items. Not all webpages contain the necessary data, therefore some checks are done to assure certain list items do exists using the fact that list items in html are encoded with the assure certain list items do exists using the fact that list items in html are encoded with the `<li>` tag. Inspecting the html code one can see that 'Evidence', 'Summary' and 'History' are actually represented as third level headers on the webpages encoded in the `<h3>` tag. Data contained in the 'Summary' section have to be scraped individually and are refrenced in the 'tbl.smry' variable. The regular expressions used to get the necessary data contains two patterns worthwile to mention: First when I want t find data contained in html tags I want to perform a non greedy search since otherwise closing html tags of the last found "html-tag-section" will be used. The syntax for this is a '?' operator after any quantifier, here the '*' operator which qunatifies to none or arbitrary many times. The '.' character encodes all alphanumerical symbols including whitespaces and special symbols like exclamation marks. The second important syntax used here are 'lookahead' and 'lookbehind' symbols ecoded in "(?='some_text') and"(?<='some_text')". Lookaheads (Lookbehinds) assert certain letters after (before) the actual text of interest and are excluded from the match. Note that Lookaheads and Lookbehinds are Perl syntax, so the appropriate flag must be set to true, further note that the special characters '^' and '$' encode the beginning or end of a text. Certain errors that could happen during the scraping process are intercepted using the `warning` function, returning an object of appropriate type instead of"NULL" that can be intercepted by the caller. Since `scrape.html.R` only processes a single url the function `assemble.data` is used to sequentially scrape several urls given by the `url.base` argumet and the locKey contained in the `idx.seq` argument. The function takes several other arguments that can be looked up in the functions docstring.

The script `write_data.R` finally handles the actual data processing, calling `assemble.data` from the script `functions.R` The scraped data is written into the data folder (data_web.csv) and merged with the given population data and once more written into the data folder (final_data.csv). The following table shows head and tail of the final data:

```
##        time C_ID latitude  longitude Appointments Population    lnPop     lnApp
##     1: 1665  458 51.14510  0.8739631            1       1090 6.993933 0.0000000
##     2: 1565  483 51.41729  0.7470207            6        100 4.605170 1.7917595
##     3: 1665  483 51.41729  0.7470207           17        270 5.598422 2.8332133
##     4: 1680  483 51.41729  0.7470207            5        160 5.075174 1.6094379
##     5: 1815  483 51.41729  0.7470207            2        805 6.690842 0.6931472
##    ---
## 14149: 1665  471 51.49337  0.0098214            1       2500 7.824046 0.0000000
## 14150: 1815  471 51.49337  0.0098214            2      16947 9.737846 0.6931472
## 14151: 1680  372 51.21120 -1.4919233            2       1450 7.279319 0.6931472
## 14152: 1815  372 51.21120 -1.4919233            1       3295 8.100161 0.0000000
## 14153: 1815  880 51.23727 -0.2058830            2       1128 7.028201 0.6931472
```

The structure of the final data is as follows:

```
## Classes 'data.table' and 'data.frame':   14153 obs. of  8 variables:
##  $ time        : int  1665 1565 1665 1680 1815 1680 1815 1665 1815 1815 ...
##  $ C_ID        : int  458 483 483 483 483 504 498 458 458 502 ...
##  $ latitude    : num  51.1 51.4 51.4 51.4 51.4 ...
##  $ longitude   : num  0.874 0.747 0.747 0.747 0.747 ...
##  $ Appointments: int  1 6 17 5 2 3 3 3 3 2 ...
##  $ Population  : int  1090 100 270 160 805 360 2786 1090 2532 1901 ...
##  $ lnPop       : num  6.99 4.61 5.6 5.08 6.69 ...
##  $ lnApp       : num  0 1.792 2.833 1.609 0.693 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```