EC999: Sourcing Data

Thiemo Fetzer

University of Chicago & University of Warwick

March 30, 2017

Plan

Basic Manipulating of Strings in R

After introducing the R-basics in the last lecture, we will turn to the basics of string manipulation.

- Basic string manipulation functions
- Regular Expressions, search and replace

Top string manipulation functions: - `tolower` (also `toupper`, capitalize) - `nchar` - `grep` - `gsub` - `substring` - `paste` and `paste0` - and the following from library stringr: `strtrim` , `str_extract`, `str_match`, `str_split` but many more.

Case-folding for dimensionality reduction

As we will see, due to the curse of dimensionality, we will often simply ignore the case o individual words, as depending on the task at hand, the word case does not carry a lot of information.

```
# USArrests is a data frame that R comes shipped with
states <- rownames(USArrests)
tolower(states[0:4])

## [1] "alabama"  "alaska"   "arizona"  "arkansas"

toupper(states[0:4])

## [1] "ALABAMA"  "ALASKA"   "ARIZONA"  "ARKANSAS"

# smarter way to do case folding is not to replace acronyms like US or IBM, for that use
# regular expressions (see later)
WORDS <- c("IBM", "Chicago")

WORDS[grep("[^A-Z]{2,}[a-z]*", WORDS)] <- tolower(WORDS[grep("[^A-Z]{2,}[a-z]*", WORDS)])

WORDS

## [1] "IBM"     "chicago"
```

Number of Characters and Substr

Note - whitespaces count as characters.

```
nchar(states)
```

```
##  [1]  7  6  7  8 10  8 11  8  7  7  6  5  8  7  4  6  8  9  5  8 13  8  9 11  8  7  8  6
## [29] 13 10 10  8 14 12  4  8  6 12 12 14 12  9  5  4  7  8 10 13  9  7
```

```
states[which(nchar(states) == 5)]
```

```
## [1] "Idaho" "Maine" "Texas"
```

```
library(stringr)

# trim leading and trailing white spaces at beginning and end of string
str_trim(" This is a test   .  ")
```

```
## [1] "This is a test   ."
```

```
# get a substring substr(x, start, stop)
substr(states[1], 3, nchar(states[1]))
```

```
## [1] "abama"
```

Splitting Strings

A simple word or sentence tokenizer works off splitting near white spaces or sentences.

```
link <- "http://stats.grok.se/json/en/201401/Donald_Trump"
str_split(link, "/")
```

```
## [[1]]
## [1] "http:"         ""              "stats.grok.se" "json"          "en"
## [6] "201401"        "Donald_Trump"
```

```
sentence <- "This is a sentence that is split by white spaces."
str_split(sentence, " ")
```

```
## [[1]]
##  [1] "This"     "is"       "a"        "sentence" "that"     "is"       "split"
##  [8] "by"       "white"    "spaces."
```

```
# str_split accepts regexes as split patterns
sentence <- "Two sentences example. The split occurs around full-stops followed by white space."
str_split(sentence, "\\. ")
```

```
## [[1]]
## [1] "Two sentences example"
## [2] "The split occurs around full-stops followed by white space."
```

Regular Expressions

- The first Information Retrieval pipelines working off textual data were making heavy use of regular expressions.
- Regular expressions, as the name indicates, is a pattern that describes a set of strings.
- If a string matches a regular expression, it indicates that the presented string follows the pattern of the regular expression.

Regex Character Classes

- `[a-z]` : lower case letters
- `[0-9]` or `[[:digit:]]`: Digits: 0 1 2 3 4 5 6 7 8 9.
- `[[:alnum:]]` Alphanumeric characters: `[[:alpha:]]` and `[[:digit:]]`.
- `[[:alpha:]]` Alphabetic characters: `[[:lower:]]` and `[[:upper:]]`.
- `[[:blank:]]` Blank characters: space and tab, and possibly other locale-dependent characters such as non-breaking space.
- `[[:lower:]]` or `[[:upper:]]` -case letters in the current locale.
- `[[:print:]]` Printable characters: `[[:alnum:]]`, `[[:punct:]]` and space.
- `[[:punct:]]` Punctuation characters:
- `[[:space:]]` Space characters: tab, newline, vertical tab, form feed, carriage return, space and possibly other locale-dependent characters.

Regex quantifiers and qualifiers

- ? The preceding item is optional and will be matched at most once.
- $*$ The preceding item will be matched zero or more times.
- + The preceding item will be matched one or more times.
- $\{n\}$ The preceding item is matched exactly n times.
- $\{n,\}$ The preceding item is matched n or more times.
- $\{n, m\}$ The preceding item is matched at least n times, but not more than m times.

Functions handeling regular expressions

Functions to handle/ deal with regular expressions

- `grep(pattern, string)` : find presence of pattern in string
- `gsub(pattern, replacement, string)` : replace pattern with replacement in string
- `str_extract(string, pattern)` : extract matching patterns from a string from `stringr` package.
- `str_match(string, pattern)`: extract matched groups from a string from `stringr` package.

Some Regex Examples

```
gsub("([A-z0-9]{3,})@([A-z0-9]{3,})\\.([A-z0-9]{3,})", "\\1 \\2 \\3", "test@devmag.net")

## [1] "test devmag net"

gsub("<a href=\"([^\"]*)\\>([^<]*)</a>", "\\1", "<a href=\"http://www.google.com\">Link to Google</a>")

## [1] "http://www.google.com"

# Often times, need to extract items of query string in URL, for example if you want to
# extract the doodle poll id from the URL http://doodle.com/poll/ga2thc6k5w9xa2z32kt452rz/
# a regex for this would be

library(stringr)
str_match("http://doodle.com/poll/ga2thc6k5w9xa2z32kt452rz/", "poll/([:alnum:]*)/$")[, 2]

## [1] "ga2thc6k5w9xa2z32kt452rz"
```

greping strings

```r
strings <- c(" 219 733 8965", "329-293-8753 ", "banana", "595 794 7569", "387 287 6718", "myweb@gmail.com",
    "233.398.9187  ", "482 952 3315", "239 923 8115 and 842 566 4692", "Work: 579-499-7527",
    "Email: president@whitehouse.gov", "Home: 543.355.3679")

## regex to match phone landlines
pattern <- "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"

strings[grep(pattern, strings)]
```

```
## [1] " 219 733 8965"                 "329-293-8753 "
## [3] "595 794 7569"                  "387 287 6718"
## [5] "233.398.9187  "                "482 952 3315"
## [7] "239 923 8115 and 842 566 4692" "Work: 579-499-7527"
## [9] "Home: 543.355.3679"
```

```r
# this returns the indices for found matches
grep(pattern, strings)
```

```
## [1]  1  2  4  5  7  8  9 10 12
```

```r
# this returns the underlying values
grep(pattern, strings, value = TRUE)
```

```
## [1] " 219 733 8965"                 "329-293-8753 "
## [3] "595 794 7569"                  "387 287 6718"
## [5] "233.398.9187  "                "482 952 3315"
## [7] "239 923 8115 and 842 566 4692" "Work: 579-499-7527"
## [9] "Home: 543.355.3679"
```

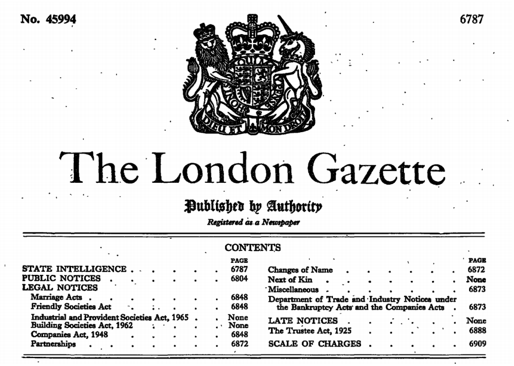
str_extracting, str_matching and gsub

```
# this actually extracts
str_extract(strings, pattern)

##  [1] "219 733 8965" "329-293-8753" NA             "595 794 7569" "387 287 6718"
##  [6] NA             "233.398.9187" "482 952 3315" "239 923 8115" "579-499-7527"
## [11] NA             "543.355.3679"

# this returns the underlying matching character classes (i.e. the area codes)
str_match(strings, pattern)

##       [,1]           [,2]  [,3]  [,4]
##  [1,] "219 733 8965" "219" "733" "8965"
##  [2,] "329-293-8753" "329" "293" "8753"
##  [3,] NA             NA    NA    NA
##  [4,] "595 794 7569" "595" "794" "7569"
##  [5,] "387 287 6718" "387" "287" "6718"
##  [6,] NA             NA    NA    NA
##  [7,] "233.398.9187" "233" "398" "9187"
##  [8,] "482 952 3315" "482" "952" "3315"
##  [9,] "239 923 8115" "239" "923" "8115"
## [10,] "579-499-7527" "579" "499" "7527"
## [11,] NA             NA    NA    NA
## [12,] "543.355.3679" "543" "355" "3679"

# lastly, can use to standardize phone number separating character - but match/ extract
# typically much more useful
gsub(pattern, "\\1-\\2-\\3", strings[grep(pattern, strings)])

## [1] " 219-733-8965"                 "329-293-8753 "
## [3] "595-794-7569"                  "387-287-6718"
## [5] "233-398-9187  "                "482-952-3315"
## [7] "239-923-8115 and 842-566-4692" "Work: 579-499-7527"
## [9] "Home: 543-355-3679"
```

⇒ extract email adresses, hash-tags in a tweet, URLs or handles?

Who becomes a British citizen?

One example from my research “Naturalizations of Aliens into the United Kingdom”.
Every person’s that adopted the British citizenship was listed with name, original nationality and full address in the British Official Government publication. In total, around 150 k naturalisations.



No. 45994 6787

The London Gazette

Published by Authority

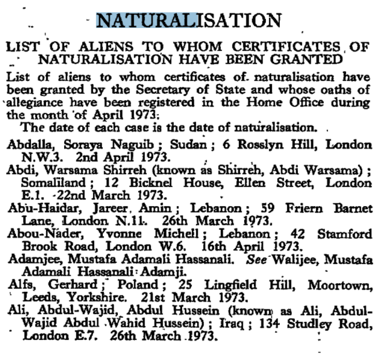
Registered as a Newspaper

CONTENTS

	PAGE		PAGE
STATE INTELLIGENCE	6787	Changes of Name	6872
PUBLIC NOTICES	6804	Next of Kin	None
LEGAL NOTICES		Miscellaneous	6873
Marriage Acts	6848	Department of Trade and Industry Notices under the Bankruptcy Acts and the Companies Acts	6873
Friendly Societies Act	6848		
Industrial and Provident Societies Act, 1965	None	LATE NOTICES	None
Building Societies Act, 1962	None	The Trustee Act, 1925	6888
Companies Act, 1948	6848		
Partnerships	6872	SCALE OF CHARGES	6909

Who becomes a British citizen?

One example from my research "Naturalizations of Aliens into the United Kingdom".
Every person's that adopted the British citizenship was listed with name, original nationality and full address in the British Official Government publication. In total, around 150 k naturalisations.



NATURALISATION

LIST OF ALIENS TO WHOM CERTIFICATES OF NATURALISATION HAVE BEEN GRANTED

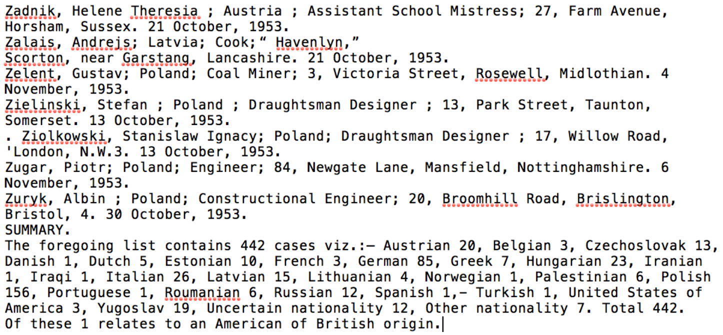
List of aliens to whom certificates of naturalisation have been granted by the Secretary of State and whose oaths of allegiance have been registered in the Home Office during the month of April 1973.

The date of each case is the date of naturalisation.

Abdalla, Soraya Naguib; Sudan; 6 Rosslyn Hill, London N.W.3. 2nd April 1973.
Abdi, Warsama Shirreh (known as Shirreh, Abdi Warsama); Somaliland; 12 Bicknel House, Ellen Street, London E.1. 22nd March 1973.
Abu-Haidar, Jareer Amin; Lebanon; 59 Friern Barnet Lane, London N.11. 26th March 1973.
Abou-Nader, Yvonne Michell; Lebanon; 42 Stamford Brook Road, London W.6. 16th April 1973.
Adamjee, Mustafa Adamali Hassanali. *See* Walijee, Mustafa Adamali Hassanali Adamji.
Alfs, Gerhard; Poland; 25 Lingfield Hill, Moortown, Leeds, Yorkshire. 21st March 1973.
Ali, Abdul-Wajid, Abdul Hussein (known as Ali, Abdul-Wajid Abdul Wahid Hussein); Iraq; 134 Studley Road, London E.7. 26th March 1973.

Who becomes a British citizen?

One example from my research "Naturalizations of Aliens into the United Kingdom".
Every person's that adopted the British citizenship was listed with name, original nationality and full address in the British Official Government publication. In total, around 150 k naturalisations.



```
Zadnik, Helene Theresia ; Austria ; Assistant School Mistress; 27, Farm Avenue,
Horsham, Sussex. 21 October, 1953.
Zalais, Andrejs; Latvia; Cook;" Havenlyn,"
Scorton, near Garstang, Lancashire. 21 October, 1953.
Zelent, Gustav; Poland; Coal Miner; 3, Victoria Street, Rosewell, Midlothian. 4
November, 1953.
Zielinski, Stefan ; Poland ; Draughtsman Designer ; 13, Park Street, Taunton,
Somerset. 13 October, 1953.
. Ziolkowski, Stanislaw Ignacy; Poland; Draughtsman Designer ; 17, Willow Road,
'London, N.W.3. 13 October, 1953.
Zugar, Piotr; Poland; Engineer; 84, Newgate Lane, Mansfield, Nottinghamshire. 6
November, 1953.
Zuryk, Albin ; Poland; Constructional Engineer; 20, Broomhill Road, Brislington,
Bristol, 4. 30 October, 1953.
SUMMARY.
The foregoing list contains 442 cases viz.:- Austrian 20, Belgian 3, Czechoslovak 13,
Danish 1, Dutch 5, Estonian 10, French 3, German 85, Greek 7, Hungarian 23, Iranian
1, Iraqi 1, Italian 26, Latvian 15, Lithuanian 4, Norwegian 1, Palestinian 6, Polish
156, Portuguese 1, Roumanian 6, Russian 12, Spanish 1,- Turkish 1, United States of
America 3, Yugoslav 19, Uncertain nationality 12, Other nationality 7. Total 442.
Of these 1 relates to an American of British origin.
```

Naturalisation Processing Steps

- Bulk download PDFs and bulk OCR processing using Abbyy Fine Reader (see further below)
- Cleaning lines: removing characters that are not A-z, 0-9, -, ., (,)

```
OUT <- gsub("([^A-z0-9,;\\-\\.\\(\\) ])", "", OUT)
```

- greping of start and end lines

```
START <- grep("Oaths of Allegiance|LIST OF ALIENS|CERTIFICATES OF NATURALISATION", TEMP, ignore.case =

END <- grep("foregoing|SUMMARY|The list contains", TEMP, ignore.case = TRUE)
END <- END[END > START][1]
```

- str_spliting by separation character ";" to separate different pieces;
- regex for date extraction, something like: `([0-9]+)(rd|th|st)? ([A-z]3,)`
  `,?  ([0-9]4)$`
- and a whole bunch of further refinements...

Here, information retrieval does not require a big statistical machinery: simple rules work well as the text data is broadly consistently formatted.

Plan

Accessing Textual Data

- Textual data can be stored in multiple different formats
  - JSON (JavaScript Object Notation) is a lightweight data-interchange format for structured data.
  - structured XML
  - Flat text files
  - (machine readable?) PDFs
  - Word
  - Data bases
- Parsing or reading text data into R can be achieved by a range of functions.

Accessing Textual Data

```
# con can be any connection, could be a URL or a path to a file
TEXT <- readLines(con = "https://www.dropbox.com/s/eynnvac4kurnjon/speaches-2016-election.json?dl=1",
    encoding = "UTF-8")

head(TEXT)
```

```
## [1] "{\"congress\":104,\"title\":\"JOIN THE SENATE AND PASS A CONTINUING RESOLUTION\",\"text\":\"Mr. Speake
## [2] "{\"congress\":104,\"title\":\"MEETING THE CHALLENGE\",\"text\":\"Mr. Speaker, a relationship, to work
## [3] "{\"congress\":104,\"title\":\"DISPOSING OF SENATE AMENDMENT TO H.R. 1643, EXTENSION OF MOST-FAVORED- I
## [4] "{\"congress\":104,\"title\":\"EXAMINING THE SPEAKER'S UPCOMING TRAVEL SCHEDULE\",\"text\":\"Mr. Speake
## [5] "{\"congress\":104,\"title\":\"FLOODING IN PENNSYLVANIA\",\"text\":\"Mr. President, I wanted to follow
## [6] "{\"congress\":104,\"title\":\"EMERGENCY RELIEF\",\"text\":\"Mr. President, I ask unanimous consent tha
```

`readLines` is the most basic function. It returns a character vector, where each element is a row in the underlying text document. Knowing and specifying the encoding correctly can make your life a lot easier. UTF-8 encoding is becoming more and more the standard...but

What to do in case we do not know the encoding?

There are lots of possible encoding standards, they tend to vary by geographic location and by operating system. For example: Latin1 is common in Europe on Microsoft based systems, while MacOS uses Mac-OS Roman.

```
# list of all system supported character encodings
iconvlist()[1:50]
```

```
##  [1] "437"             "850"             "852"             "855"
##  [5] "857"             "860"             "861"             "862"
##  [9] "863"             "865"             "866"             "869"
## [13] "ANSI_X3.4-1968"  "ANSI_X3.4-1986"  "ARABIC"          "ARMSCII-8"
## [17] "ASCII"           "ASMO-708"        "ATARI"           "ATARIST"
## [21] "BIG-5"           "BIG-FIVE"        "BIG5"            "BIG5-2003"
## [25] "BIG5-HKSCS"      "BIG5-HKSCS:1999" "BIG5-HKSCS:2001" "BIG5-HKSCS:2004"
## [29] "BIG5HKSCS"       "BIGFIVE"         "C99"             "CHINESE"
## [33] "CN"              "CN-BIG5"         "CN-GB"           "CN-GB-ISOIR165"
## [37] "CP-GR"           "CP-IS"           "CP1046"          "CP1124"
## [41] "CP1125"          "CP1129"          "CP1133"          "CP1161"
## [45] "CP1162"          "CP1163"          "CP1250"          "CP1251"
## [49] "CP1252"          "CP1253"
```

```
# total system supported encodings
length(iconvlist())
```

```
## [1] 419
```

```
# (older) word / excel documents are typically encoded as CP1251/CP1252
```

What to do in case we do not know the encoding?

After reading text data of unknwon encoding, a string is displayed in R internally as `D\xa0ZCE` - without knowing the proper encoding, string manipulations often throw the annyoing error `unknown multibyte string`.

```
# We can try out all possible source encodings...

TEMP <- unlist(lapply(iconvlist(), function(x) iconv("D\xa0ZCE", x, "UTF-8")))
sort(table(TEMP[!is.na(TEMP)]), decreasing = TRUE)

##
## DZCE DZCE DZCE DZCE  DZC DZCE DCE DZCE DZCE  DZCE DZCE DZCE DCE  DCE DZCE
##   175   37   11     9     6     5     5     4     4     4     3     3     2     2     1
## DZCE DZCE
##     1     1

# it seems that 'DZCE' is the right spelling (a town in Turkey)
iconvlist()[which(TEMP == "DZCE")]

## [1] "CSVISCII"    "VISCII"      "VISCII1.1-1"
```

Reading Tabular Data

- A lot of social program data is provided in some form of *SV format: Comma-, Colon-,Tabular- separated values.
- Writing a function that reads in all files of a specific type in a folder

```
# con can be any connection, could be a URL or a path to a file commonly used for tabular
# data formats
TEXT <- read.table(file = "file.txt", sep = "\t")
# may need to iterate over a whole folder of documents
TEST <- lapply(list.files("Path"), function(x) readLines(con = x))
```

Function that reads file types in folders

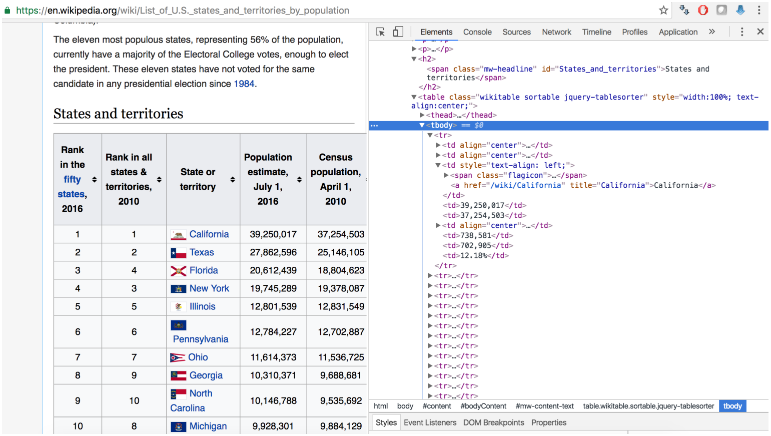
```r
readFiles <- function(folder, ftype = "csv", collate = "rbind", Encoding = "latin1", fname = TRUE) {

    ffs <- list.files(folder)
    ffs <- grep(paste(ftype, "$", sep = ""), ffs, value = TRUE)
    if (ftype == "dta") {
        library(foreign)
        DAT <- lapply(ffs, function(x) read.dta(file = paste(folder, x, sep = "")))
    } else if (ftype == "csv") {
        if (fname == TRUE) {
            DAT <- lapply(ffs, function(x) data.table(data.frame(fname = x, read.csv(file = paste(folder,
                x, sep = ""), fileEncoding = Encoding))))
        } else {
            DAT <- lapply(ffs, function(x) data.table(data.frame(read.csv(file = paste(folder,
                x, sep = ""), fileEncoding = Encoding))))
        }
    }
    if (collate == "rbind") {
        DAT <- rbindlist(DAT, fill = TRUE)
    }
    DAT
}

# reads csv files in folder Folder with file extension .csv
CALL <- readFiles("Folder")
```

Plan

Reading and Parsing HTML Data

- HTML is a markup language that is derived from the XML standard and is commonly interpreted in the browser.
- HTML is made up of tags that encapsulate information on how elements inside the tag are to be presented visually in the browser.
- Can define table-, image-, headline-, and paragraph environments, among many others.
- There exist many direct ways of parsing HTML using existing R packages
- HTML, just as any XML document has a tree structure with the whole content of the page being wrapped in a `body` environment.
- This tree structure allows each item in the document to be "addressed" by a path, each element has a unique `xpath`.

Example: Extracting data from a Wikipedia page table


https://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_population

The eleven most populous states, representing 56% of the population, currently have a majority of the Electoral College votes, enough to elect the president. These eleven states have not voted for the same candidate in any presidential election since 1984.

States and territories

Rank in the fifty states, 2016	Rank in all states & territories, 2010	State or territory	Population estimate, July 1, 2016	Census population, April 1, 2010
1	1	California	39,250,017	37,254,503
2	2	Texas	27,862,596	25,146,105
3	4	Florida	20,612,439	18,804,623
4	3	New York	19,745,289	19,378,087
5	5	Illinois	12,801,539	12,831,549
6	6	Pennsylvania	12,784,227	12,702,887
7	7	Ohio	11,614,373	11,536,725
8	9	Georgia	10,310,371	9,688,681
9	10	North Carolina	10,146,788	9,535,692
10	8	Michigan	9,928,301	9,884,129

```
▶<p>…</p>
▼<h2>
   <span class="mw-headline" id="States_and_territories">States and
   territories</span>
 </h2>
▼<table class="wikitable sortable jquery-tablesorter" style="width:100%; text-
align:center;">
  ▶<thead>…</thead>
  ▼<tbody> == $0
    ▼<tr>
      ▶<td align="center">…</td>
      ▶<td align="center">…</td>
      ▼<td style="text-align: left;">
        ▶<span class="flagicon">…</span>
         <a href="/wiki/California" title="California">California</a>
       </td>
       <td>39,250,017</td>
       <td>37,254,503</td>
      ▶<td align="center">…</td>
       <td>738,581</td>
       <td>702,905</td>
       <td>12.18%</td>
     </tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
    ▶<tr>…</tr>
```

html body #content #bodyContent #mw-content-text table.wikitable.sortable.jquery-tablesorter tbody
Styles Event Listeners DOM Breakpoints Properties

⇒ Chrome's inspect element feature is very helpful in identifying the correct xpath to an element to be extracted (and to learn HTML while you are at it).

Example: Extracting a HTML table the “hard” way

```r
HTML <- readLines(con = "https://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_population")
# subsetting
HTML <- HTML[grep("<span class=\"mw-headline\" id=\"States_and_territories\">", HTML):grep("<span class=\"mw
    HTML)]

head(HTML)
```

```
## [1] "<h2><span class=\"mw-headline\" id=\"States_and_territories\">States and territories</span></h2>"
## [2] "<table class=\"wikitable sortable\" style=\"width:100%; text-align:center;\">"
## [3] "<tr style=\"vertical-align: top;\">"
## [4] "<th style=\"vertical-align: middle\">Rank in the <a href=\"/wiki/Fifty_States\" class=\"mw-redirect\"
## [5] "<th style=\"vertical-align: middle\">Rank in all states &amp; territories, 2010</th>"
## [6] "<th style=\"width: 20%; vertical-align: middle\"><b>State or territory</b></th>"
```

```r
# tr is a table-row

# indices of start of new rows
grep("<tr>", HTML)
```

```
##  [1]  14  25  36  47  58  69  80  91 102 113 124 135 146 157 168 179 190 201 212 223 234
## [22] 245 256 267 278 289 300 311 322 344 355 366 377 388 399 410 421 432 443 454 465 476
## [43] 487 498 509 520 531 542 564 575
```

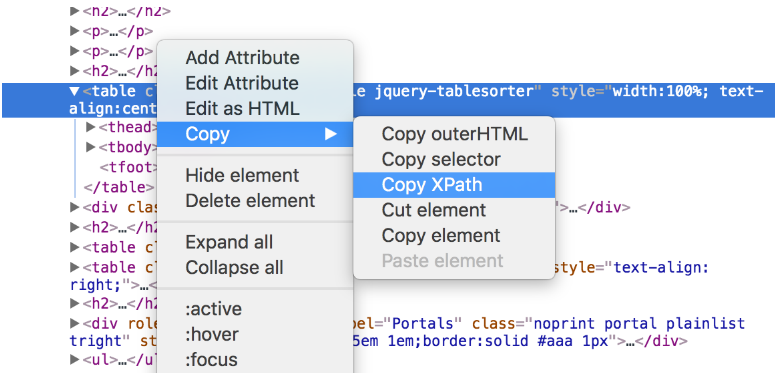
```r
population <- lapply(grep("<tr>", HTML), function(x) gsub("<.*?>", "", HTML[x:(x + 10)]))
population <- lapply(population, function(x) x[x != ""])

population <- do.call("rbind", population)
class(population)
```

```
## [1] "matrix"
```

```r
head(population)
```

```
##      [,1]                   [,2]                   [,3]                  [,4]
## [1,] "70001000000000000001" "70001000000000000001" "&#160;California"   "39,250,017"
## [2,] "70002000000000000002" "70002000000000000002" "&#160;Texas"        "27,862,596"
## [3,] "70003000000000000003" "70004000000000000004" "&#160;Florida"      "20,612,439"
## [4,] "70004000000000000004" "70003000000000000003" "&#160;New York"     "19,745,289"
## [5,] "70005000000000000005" "70006000000000000006" "&#160;Pennsylvania" "12,802,503"
## [6,] "70006000000000000006" "70005000000000000005" "&#160;Illinois"     "12,801,539"
```

Example: Extracting data from a Wikipedia page table



```
▶<h2>…</h2>
▶<p>…</p>
▶<p>…</p>
▶<h2>…</h
▼<table c                                  e jquery-tablesorter" style="width:100%; text-
align:cent
  ▶<thead>
  ▶<tbody>
    <tfoot>
  </table>
▶<div clas                                                     '>…</div>
▶<h2>…</h
▶<table c
▶<table c                                                      style="text-align:
right;">…<
▶<h2>…</h
▶<div role                                 el="Portals" class="noprint portal plainlist
tright" st                                 5em 1em;border:solid #aaa 1px">…</div>
▶<ul>…</u
```

⇒ Chrome's inspect element feature is very helpful in identifying the correct xpath to an element to be extracted (and to learn HTML while you are at it).

Example: Extracting a HTML table the "easy" way

```
# install.packages('rvest')
library("rvest")
url <- "http://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_population"
population <- url %>% read_html() %>% html_nodes(xpath = "//*[@id=\"mw-content-text\"]/table[1]") %>%
    html_table()

population <- population[[1]]

head(population)
```

```
##   Rank in the fifty states, 2016 Rank in all states & territories, 2010
## 1          7000100000000000000001                 7000100000000000000001
## 2          7000200000000000000002                 7000200000000000000002
## 3          7000300000000000000003                 7000400000000000000004
## 4          7000400000000000000004                 7000300000000000000003
## 5          7000500000000000000005                 7000600000000000000006
## 6          7000600000000000000006                 7000500000000000000005
##   State or territory Population estimate, July 1, 2016 Census population, April 1, 2010
## 1         California                        39,250,017                       37,254,503
## 2              Texas                        27,862,596                       25,146,105
## 3            Florida                        20,612,439                       18,804,623
## 4           New York                        19,745,289                       19,378,087
## 5       Pennsylvania                        12,802,503                       12,702,887
## 6           Illinois                        12,801,539                       12,831,549
##   Total seats in House of Representatives, 20132023 Estimated pop. per House seat, 2016
## 1                                     7001530000000000000053                        738,581
## 2                                     7001360000000000000036                        763,031
## 3                                     7001270000000000000027                        750,788
## 4                                     7001270000000000000027                        733,177
## 5                                     7001180000000000000018                        711,250
## 6                                     7001180000000000000018                        714,444
##   Census pop. per House seat, 2010 Percent of total U.S. pop., 2016[note 1]
## 1                          702,905                                   12.15%
## 2                          698,487                                    8.62%
## 3                          696,345                                    6.38%
## 4                          717,707                                    6.11%
```

Plan

APIs

When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

The practice of publishing APIs has allowed web communities to create an open architecture for sharing content and data between communities and applications. In this way, content that is created in one place can be dynamically posted and updated in multiple locations on the web -Wikipedia

$\Rightarrow$ many online servies and sites offer APIs, with some having dedicated R-packages (see Twitter and Facebook packages for R in a bit).

Lightweight JSON format to send and receive information

JSON (Java Script Object Notation) is a very lightweight format to send and receive data, which is why it is typically preferred to XML for sending structured data to APIs.

```
# download some wikipedia traffic statistics
var <- 201401

url <- paste("http://stats.grok.se/json/en/", var, "/Donald_Trump", sep = "")
raw.data <- readLines(url, warn = "F")

# raw JSON data
raw.data

## [1] "{\"daily_views\": {\"2014-01-15\": 3795, \"2014-01-14\": 3754, \"2014-01-17\": 3538, \"2014-01-16\":

# parsing JSON string into a list object
library(RJSONIO)
data <- fromJSON(raw.data)

class(data)

## [1] "list"

head(data[[1]])

## 2014-01-15 2014-01-14 2014-01-17 2014-01-16 2014-01-11 2014-01-10
##       3795       3754       3538       3593       4508       4160
```

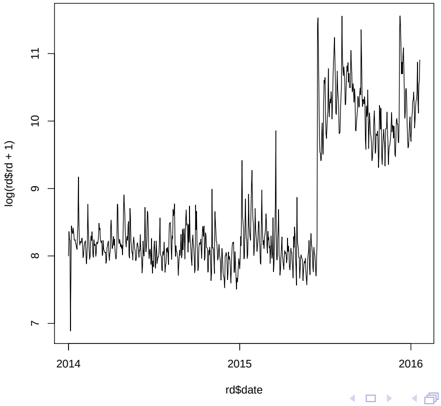
Reading and Parsing JSON Data

JSON is a very common format used by web services for sending and receiving data.

```
# download some wikipedia traffic statistics

datevector <- unlist(lapply(2008:2016, function(x) paste(x, c("01", "02", "03", "04", "05",
    "06", "07", "08", "09", "10", "11", "12"), sep = "")))

head(datevector)

## [1] "200801" "200802" "200803" "200804" "200805" "200806"

datevector <- seq(from = as.POSIXct("2008-01-01"), to = as.POSIXct("2016-12-31"), by = "months")
datevector <- gsub("-", "", substr(datevector, 0, 7))

head(datevector)

## [1] "200801" "200802" "200803" "200804" "200805" "200806"

# this is not treated as character but as numeric as we create a sequence using the :
# operator
datevector <- unlist(lapply(2014:2016, function(x) eval(paste(x, "01", sep = ""):paste(x, "12",
    sep = ""))))

head(datevector)

## [1] 201401 201402 201403 201404 201405 201406
```

Interest in Donald Trump on English Wikipedia over time

```r
# download some wikipedia traffic statistics

rd <- unlist(lapply(datevector, function(x) fromJSON(readLines(paste("http://stats.grok.se/json/en/",
    x, "/Donald_Trump", sep = "")))$daily_views))

rd <- data.frame(rd)
rd$date <- strptime(rownames(rd), "%Y-%m-%d")
rd <- rd[rd > 0, ]
rd <- rd[order(rd$date), ]
plot(rd$date, log(rd$rd + 1), type = "l")
```



Social Media API's

- All big social media sites - Facebook and Twitter - provide an API that allows access to their data.
- API's can either be anonymous or require authentication
- API's tend to communicate through JSON data objects.
- Calls to APIs are sent through HTTP, so you can just call them using your browser
- API calls are just "complicated URLs".
- For our purposes, we will just highlight existing packages implemented for *R* to access social media data from Facebook and Twitter.

Types of Requests

GET requests a representation of the specified resource. Note that GET should not be used for operations that cause side-effects, such as using it for taking actions in web applications. One reason for this is that GET may be used arbitrarily by robots or crawlers, which should not need to consider the side effects that a request should cause.

POST submits data to be processed (e.g., from an HTML form) to the identified resource. The data is included in the body of the request. This may result in the creation of a new resource or the updates of existing resources or both. we use get for scraping, post is more complicated. Use it to navigate logins, popups, etc.

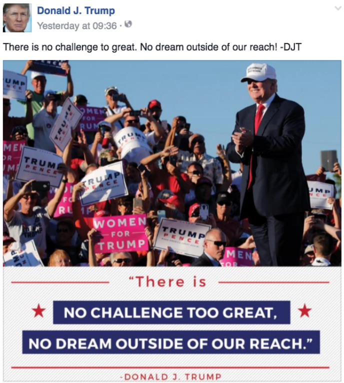
Named Entity Recognition via Web API

```
url <- "http://www.huffingtonpost.com/entry/house-republicans-ethics_us_586bdb14e4b0de3a08f99e66?6ztihpvi"
api <- "http://juicer.herokuapp.com/api/article?url="
target <- paste(api, url, sep = "")

raw.data <- readLines(target, warn = "F")

rd <- fromJSON(raw.data)
dat <- rd$article

ENTITIES <- data.frame(do.call("rbind", dat$entities))

ENTITIES[1:10, ]

##            type                                                    text frequency
## 1        Person                                                   Trump         1
## 2  Organization Campaign for Accountability , Citizens for Responsibility         1
## 3        Person                                            Nancy Pelosi         1
## 4  Organization                                                   House         3
## 5  Organization                                           People 's House         1
## 6        Person                                               Goodlatte         1
## 7      Location                                              Washington         2
## 8        Person                                              Paul Ryan         1
## 9      Location                                              WASHINGTON         1
## 10       Person                                                  Pelosi         1
```

⇒ this works off the Stanford NLP NER module, which we will also directly use in R.

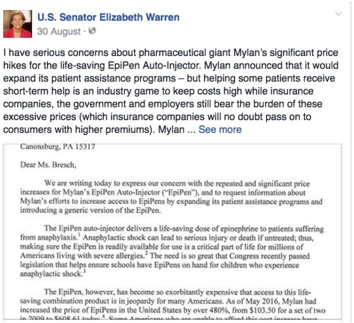
Social Media

Social Media


Donald J. Trump
Yesterday at 09:36 ·

There is no challenge to great. No dream outside of our reach! -DJT

"There is
NO CHALLENGE TOO GREAT,
NO DREAM OUTSIDE OF OUR REACH."
-DONALD J. TRUMP


U.S. Senator Elizabeth Warren
30 August ·

I have serious concerns about pharmaceutical giant Mylan's significant price hikes for the life-saving EpiPen Auto-Injector. Mylan announced that it would expand its patient assistance programs – but helping some patients receive short-term help is an industry game to keep costs high while insurance companies, the government and employers still bear the burden of these excessive prices (which insurance companies will no doubt pass on to consumers with higher premiums). Mylan ... See more

Canonsburg, PA 15317

Dear Ms. Bresch,

We are writing today to express our concern with the repeated and significant price increases for Mylan's EpiPen Auto-Injector ("EpiPen"), and to request information about Mylan's efforts to increase access to EpiPens by expanding its patient assistance programs and introducing a generic version of the EpiPen.

The EpiPen auto-injector delivers a life-saving dose of epinephrine to patients suffering from anaphylaxis.[1] Anaphylactic shock can lead to serious injury or death if untreated; thus, making sure the EpiPen is readily available for use is a critical part of life for millions of Americans living with severe allergies.[2] The need is so great that Congress recently passed legislation that helps ensure schools have EpiPens on hand for children who experience anaphylactic shock.[3]

The EpiPen, however, has become so exorbitantly expensive that access to this life-saving combination product is in jeopardy for many Americans. As of May 2016, Mylan had increased the price of EpiPens in the United States by over 480%, from $103.50 for a set of two in 2009 to $608.61 today.[4] Some Americans who are unable to afford this cost increase have

Why working with Social Media data?

Social media is changing the way politicians, (government) organizations and corporations are interacting with ther electorate, stakeholders and customers.

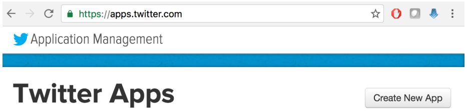
Questions that arise from reasearch are not constrained but may include...

- How do politicians engage with their electorate through social media?
- Does direct communication via social media replace reliance on other media sources?
- Does social media allow for a more effective monitoring of elected officials?

And there are likely a lot more... most social media data takes form of texts or tweets, so how can we get this data to work with?

TwitteR

- `TwitteR` package allows for basic Twitter scraping functionality.
- Need to create an access token to verify identity of requests (and to limit usage)
- `TwitteR` package handles authentification process and has core functionality to ...
  - Scrape tweets pertaining to individual hashtags
  - Scrape timelines of Twitter users
  - get data on follower network structure (who follows whom)



Sourcing Twitter data: Hash-tag level

```r
library(twitteR)
# setup_twitter_oauth(consumer_key, consumer_secret, access_token=NULL, access_secret=NULL)
set.seed(12122016)
tw <- searchTwitter("#Brexit", n = 500, since = "2016-12-12", lang = "en")
tw.df <- data.table(twListToDF(tw))
strwrap(head(tw.df$text))
```

```
##  [1] "RT @nia4_trump: #ThursdayThoughts Junker has lost his mind over #Brexit."
##  [2] "Threatens to promote OHIO &amp; TEXAS to secede from USA \xed\xa0\xbe\xed\xb4\xa3 https://t.c"
##  [3] "@empathyimpacts @nytimes Hear,hear!  It's the same this side of the"
##  [4] "Atlantic.Obsessed with #brexit and #indyref2 .B https://t.co/wNTL05Oujd"
##  [5] "RT @davesumnersmith: In reality, #Brexit negotiations could easily take a decade"
##  [6] "&amp; the UK could change its mind at any time https://t.co/"
##  [7] "RT @UKIPNFKN: How Brexit damaged Britains democracy via @theeconomist #Brexit"
##  [8] "https://t.co/zutJzVA25e"
##  [9] "RT @ka8895: So the #Indyref2 scores so far: Scots don't want to vote, would vote"
## [10] "No if forced &amp; don't want seperate #Brexit deal."
## [11] "RT @JamesCrisp6: Asked where free movement of pets ranked in EC priorities for"
## [12] "#brexit talks. Spokes: 'fate of cats &amp; dogs of utmost import"
```

```r
save(tw.df, file = "../../Data/brexittweets.March.rdata")
```

Sourcing Twitter data: Individual user level

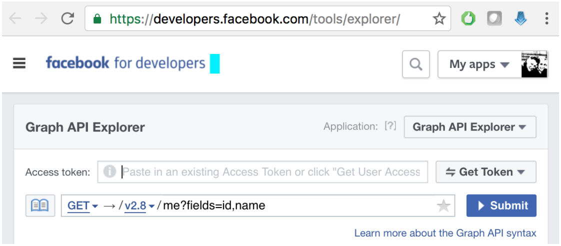
```
library(twitteR)
# setup_twitter_oauth(consumer_key, consumer_secret, access_token=NULL, access_secret=NULL)
set.seed(12122016)
tw.user <- userTimeline("Nigel_Farage", n = 500)
tw.user.df <- data.table(twListToDF(tw.user))
strwrap(head(tw.user.df$text))
```

```
##  [1] "Juncker has made a complete fool of himself again. The USA was formed by consent"
##  [2] "while the EU is being imposed. https://t.co/4XkgaQG56w"
##  [3] "There is no turning back now. We are getting our country back."
##  [4] "https://t.co/cKyTf2eqVw"
##  [5] "In the interests of bringing people together, I invited @theresa_may onto my"
##  [6] "@LBC show tonight. She didn't fancy it https://t.co/UgUchnPm87"
##  [7] "Article 50. LBC. 7pm. https://t.co/7fGnYNMtWX"
##  [8] "What started as an impossible dream 25 years ago has now come true. #BrexitDay"
##  [9] "https://t.co/pfKjPjGOY7"
## [10] "https://t.co/B2xsOJ7kMp"
```

```
save(tw.user.df, file = "../../Data/nigelstweets.March..rdata")
```

Rfacebook

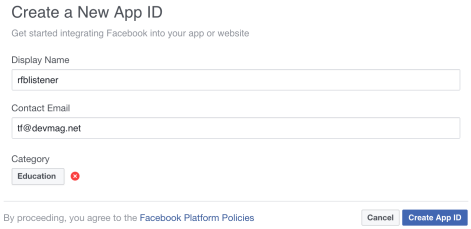
- Rfacebook package allows for very basic access to facebook post data on public profile pages.
- This can be useful to extract data on posting activity by politicians, in particular, regarding the messages sent.
- Authentification for rFacebook is more involved
- Simple access token can be generated, but its only valid for two hours


https://developers.facebook.com/tools/explorer/

facebook for developers
My apps
Graph API Explorer
Application: [?]
Graph API Explorer
Access token:
Paste in an existing Access Token or click "Get User Access
Get Token
GET → /v2.8 / me?fields=id,name
Submit
Learn more about the Graph API syntax

Creating a non-temporary authentification token for `Rfacebook`

- To create a token that is valid for around two month period, you need to follow two steps.
- Create a new app on `developers.facebook.com/apps/`, note down APP ID and APP SECRET and register the URL `http://localhost:1410/` under Settings - Basic - New platform


Create a New App ID

Get started integrating Facebook into your app or website

Display Name

rfblistener

Contact Email

tf@devmag.net

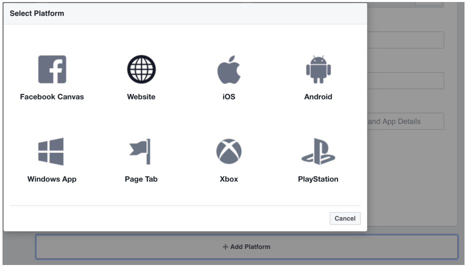
Category

Education

By proceeding, you agree to the Facebook Platform Policies

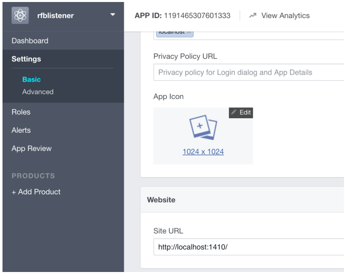
Creating a non-temporary authentification token for `Rfacebook`

- To create a token that is valid for around two month period, you need to follow two steps.
- Create a new app on `developers.facebook.com/apps/`, note down APP ID and APP SECRET and register the URL `http://localhost:1410/` under Settings - Basic - New platform



Creating a non-temporary authentification token for `Rfacebook`

- To create a token that is valid for around two month period, you need to follow two steps.
- Create a new app on `developers.facebook.com/apps/`, note down APP ID and APP SECRET and register the URL `http://localhost:1410/` under Settings - Basic - New platform



APP ID: 1191465307601333

Privacy Policy URL: Privacy policy for Login dialog and App Details

Site URL: http://localhost:1410/

Creating a non-temporary authentification token for Rfacebook

- In R, type the following lines

```
require("Rfacebook")
fb_oauth <- fbOAuth(app_id = "APPNUMBER", app_secret = "APPSECRET", extended_permissions = TRUE)
# should open a browser window and facebook page to grant accesss to the app.

# save the token for later reuse
save(fb_oauth, file = "fb_oauth.rdata")

load(file = "fb_oauth.rdata")
```

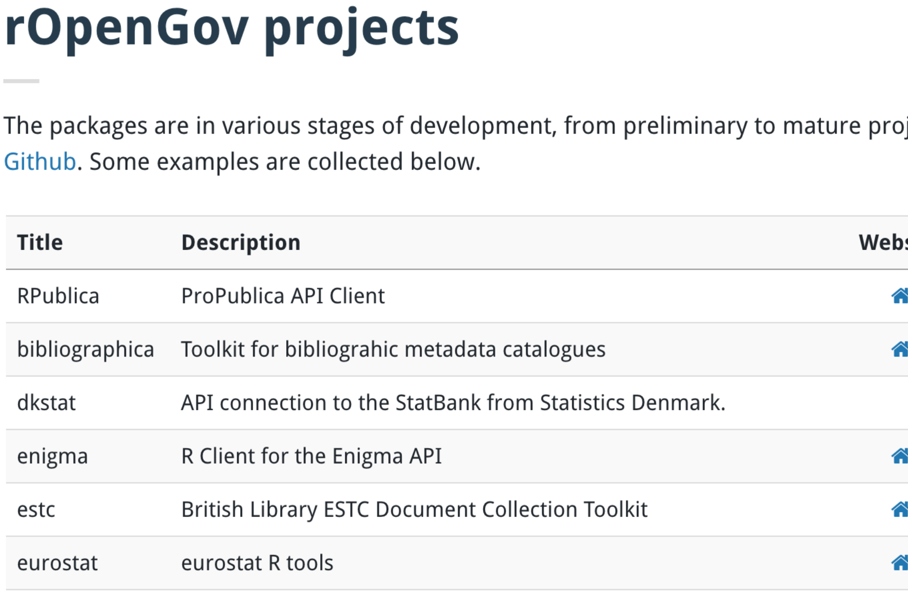
- This should open a browser window in which you are asked to grant permission to the App to login via your account.
- You should then save the access token and load it, so that you do not need to repeat this process each time you run a script.

Sourcing Facebook-page data

```r
require("Rfacebook")

me <- getUsers("me", fb_oauth, private_info=TRUE)
me$name # my name

## [1] "Thiemo Fetzer"

page <- getPage("barackobama", fb_oauth, n = 100)

## 25 posts 50 posts 75 posts 100 posts

strwrap(head(page$message))

##  [1] "\"Our goal wasn't just to make sure more people have coverageit was to make sure"
##  [2] "more people have better coverage.\" President Obama"
##  [3] "NA"
##  [4] "Today marks a crucial step forward in the fight against climate change, as the"
##  [5] "historic Paris Climate Agreement officially enters into force."
##  [6] ""
##  [7] "Let's keep pushing for progress."
##  [8] "The economic progress we're making is undeniableand it's up to all of us to"
##  [9] "keep building an economy that works for all Americans."
## [10] "Obamacare was designed on the principle that health care coverage that's"
## [11] "affordable, accessible to all, and free from discrimination should be a right,"
## [12] "not a privilege. We can't afford to let opponents roll that back."
## [13] "Millions of Americans are benefiting from Obamacare."
```

Plan

Some R-accessible open government type API's



rOpenGov projects

The packages are in various stages of development, from preliminary to mature proj
Github. Some examples are collected below.

Title	Description	Webs
RPublica	ProPublica API Client	🏠
bibliographica	Toolkit for bibliograhic metadata catalogues	🏠
dkstat	API connection to the StatBank from Statistics Denmark.	
enigma	R Client for the Enigma API	🏠
estc	British Library ESTC Document Collection Toolkit	🏠
eurostat	eurostat R tools	🏠

Political Party `manifestoR` database for R

> *ManifestoR is a free package for the open source statistical software R. It provides access to coded election programmes from the Manifesto Corpus and to the Manifesto Project's Main Dataset.*

Available via

```r
install.packages("manifestoR")
library(manifestoR)
```

Need to create an account and register for an API key on `https://manifestoproject.wzb.eu/information/documents/manifestoR`

$\Rightarrow$ may be useful to study raising Euro(pe)-skepticism...

rsunlight package for R

CURRENTLY NOT FUNCTIONAL DUE TO TRANSFER TO POLITICO

> *The Sunlight Foundation is an American 501 nonpartisan, nonprofit organization that advocates for open government.*

An example of the services - many of which have APIs

Capitol Words	Explore and compare what Congress says.
Email Congress	Contacting Congress is now as easy as email.
Foreign Lobbying Influence Tracker	Follow foreign influence on U.S. policy.
Hall of Justice	An inventory of criminal justice data.
House Staff Directory	Look up House staffers.
Influence Explorer	Uncover political activity.
Open States	Discover and follow all state legislatures.
Party Time	Tracking the political fundraising circuit.
Political Ad Sleuth	See the details of political ad purchases.
Politwoops	Archive of deleted tweets from U.S. politicians.