MPP-E1180 Lecture 7: Web Scraping + Transforms

Christopher Gandrud

27 October 2014

Objectives for the week

- Assignment 3
- Review
- Intro to web scraping
- Processing strings, including an intro to Regular Expressions
- Data and data set transformations with dplyr

Assignment 3

Purpose: Gather, clean, and analyse data

Deadline: 14 November 2015

You will submit a GitHub repo that:

- Gathers web-based data from at least **two sources**. Cleans and merges the data so that it is ready for statistical analyses.
- Conducts basic descriptive and inferential statistics with the data to address a relevant research question.
- Briefly describes the results including with dynamically generated tables and figures.
- Has a write up of 1,500 words maximum that describes the data gathering and analysis and uses literate programming.

Assignment 3

This is ideally a **good first run** at the data gathering and analysis for your final project.

Review

What is open public data?

- Name one challenge and one opportunity presented by open public data.

What is a data API?

What is tidy data?

Why are unique observation IDs so important for data cleaning?

Caveat to Web scraping

Caveat: I don't expect you to master or even use the tools of web scraping.

I just want you to know that these things are possible, so that you know where to turn in future work.

Web scraping

Web scraping simply means gathering data from websites.

Last class we learned a particular form of web scraping: downloading explicitly structured data files/data APIs.

You can also download information that is not as well structured for statistical analysis:

- HTML tables
- Text on websites
- Navigate through web forms

To really master you need a good knowledge of HTML.

Key tools

The most basic tools for web scraping in R:

- httr: gather data + simple parsing
- XML: more advanced parsing
  - Parsing: the analysis of HTML (and other) markup so that each element is syntactically related in a **parse tree**.

Key steps:

1. **Look at** the HTML for the webpage you want to scrape (use Inspect Element in Chrome).
2. **Request** a URL with `GET`.
3. **Extract** the content from the request with `content`.
   - You can extract either the raw text with `as = 'text'` or parse the content with `as = 'parsed'`.
4. **Clean** parsed content (there are many tools for this suited to a variety of problems).

Web scraping example

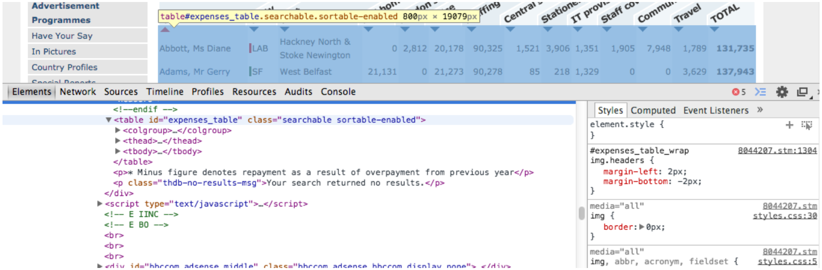
Scrape BBC's MP's Expenses table.

HTML markup marks tables using `<table>` tags.

We can use these to extract tabular information and convert them into data frames.

In particular we want the table tag with the id `expenses_table`.

Viewing the web pages source



Advertisement Programmes
Have Your Say
In Pictures
Country Profiles

`table#expenses_table.searchable.sortable-enabled 800px × 19079px`

						ffing	Central	Statione	IT provi	Staff co	Commu	Travel	TOTAL
Abbott, Ms Diane	LAB	Hackney North & Stoke Newington	0	2,812	20,178	90,325	1,521	3,906	1,351	1,905	7,948	1,789	**131,735**
Adams, Mr Gerry	SF	West Belfast	21,131	0	21,273	90,278	85	218	1,329	0	0	3,629	**137,943**

```
    <!--endif -->
  ▼<table id="expenses_table" class="searchable sortable-enabled">
    ▶<colgroup>…</colgroup>
    ▶<thead>…</thead>
    ▶<tbody>…</tbody>
    </table>
    <p>* Minus figure denotes repayment as a result of overpayment from previous year</p>
    <p class="thdb-no-results-msg">Your search returned no results.</p>
  </div>
▶<script type="text/javascript">…</script>
  <!-- E IINC -->
  <!-- E BO -->
  <br>
  <br>
  <br>
▶<div id="bbccom_adsense_middle" class="bbccom_adsense bbccom_display_none"> </div>
```

```
element.style {
}
#expenses_table_wrap                8044207.stm:1304
img.headers {
    margin-left: 2px;
    margin-bottom: -2px;
}
media="all"                             8044207.stm
img {                                 styles.css:30
    border:▶0px;
}
media="all"                             8044207.stm
img, abbr, acronym, fieldset {         styles.css:5
```

Web scraping example

```
library(httr)
library(dplyr)
library(XML)

URL <- 'http://news.bbc.co.uk/2/hi/uk_news/politics/804420

# Get and parse all tables on the webpage
tables <- URL %>% GET() %>%
            content(as = 'parsed') %>%
            readHTMLTable()

names(tables)
```

```
## [1] "NULL"            "NULL"            "NULL"
## [5] "expenses_table"  "NULL"            "NULL"
```

Web scraping example

Now we just need to subset the *tables* list for the expenses_table data frame.

```
ExpensesTable <- tables[[5]]

head(ExpensesTable)[, 1:3]
```

```
##                            MP Party
## 1       Abbott, Ms Diane   LAB       Hackney North & Stol
## 2         Adams, Mr Gerry    SF                         V
## 3           Afriyie, Adam   CON
## 4            Ainger, Nick   LAB Carmarthen West & Pembroke
## 5    Ainsworth, Mr Peter   CON
## 6 Ainsworth, Rt Hon Bob   LAB                  Coventry
```

Processing strings

A (frustratingly) large proportion of time spent web scraping and doing data cleaning generally is take up with **processing strings**.

Key tools for processing strings:

- knowing your encoding and `iconv` function in base R
- `grep`, `gsub`, and related functions in base R
- Regular expressions
- stringr package

Character encoding: Motivation

Sometimes when you load text into R you will get weird symbols like (the replacement character) or other strange things will happen to the text.

NOTE: remember to always check your data when you import it!

This often happens when R is using the **wrong character encoding**.

Character encoding

All characters in a computer are **encoded** using some standardised system.

R can recognise latin1 and UTF-8.

- latin1 is fairly limited (mostly to the latin alphabet)
- UTF-8 covers a much wider range of characters in many languages

You may need to use the `iconv` function to convert a text to UTF-8 before trying to process it.

grep, gsub, and related functions

R (and many programing languages) have functions for **identifying** and **manipulating** strings.

Matching

You can use grep and grepl to find patterns in a vector.

```
pets <- c('cats', 'dogs', 'a big snake')

grep(pattern = 'cat', x = pets)
```

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

```
grepl(pattern = 'cat', pets)
```

```
## [1]  TRUE FALSE FALSE
```

```
# Subset vector
pets[grep('cats', pets)]
```

```
## [1] "cats"
```

Manipulation

Use gsub to substitute strings.

```r
gsub(pattern = 'big', replacement = 'small', x = pets)
```

```
## [1] "cats"             "dogs"             "a small snake"
```

Regular expressions

Regular expressions are a powerful tool for finding and manipulating strings.

They are special characters that can be used to search for text.

For example:

- find characters at only the begining or end of a string
- find characters that follow or are preceeded by a particular character
- find only the first or last occurance of a character in a string

Many more possibilities.

Regular expressions examples

Examples modified from Robin Lovelace.

```
base <- c("cat16_24", "25_34cat", "35_44catch",
          "45_54Cat", "55_4fat$", 'colour', 'color')

## Find only all 'cat' regardles of case
grep('cat', base, ignore.case = T)
```

```
## [1] 1 2 3 4
```

Regular expressions examples

```
# Find only 'cat' at the end of the string with $
grep('cat$', base)
```

```
## [1] 2
```

```
# Find only 'cat' at the begining of the string with ^
grep('^cat', base)
```

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

Regular expressions examples

```r
# Find zero or one of the preceeding character with ?
grep('colou?r', base)
```

```
## [1] 6 7
```

```r
# Find one or more of the preceeding character with +
grep('colou+r', base)
```

```
## [1] 6
```

```r
# Find '$' with the escape character \
grep('\\$', base)
```

```
## [1] 5
```

Regular expressions examples

```
# Find string with any single character between 'c' and 'l
grep('c.l', base)
```

```
## [1] 6 7
```

```
# Find a range of numbers with [ - ]
grep('[1-3]', base)
```

```
## [1] 1 2 3
```

```
# Find capital letters
grep('[A-Z]', base)
```

```
## [1] 4
```

Simple regular expression cheapsheet

Character	Use
`$`	characters at the end of the string
`^`	characters at the begining of the string
`?`	zero or more of the preceeding character
`*`	zero or more of the preceeding character
`+`	one or more of the preceeding character
`\`	escape character use to find strings that are expressions
`.`	any single character
`[ - ]`	a range of characters

String processing with stringer

The stringr package has many helpful functions that make dealing with strings a bit **easier**.

stringr examples

Remove leading and trailing **whitespace** (this can be a real problem when creating consistent variable values):

```r
library(stringr)

str_trim(' hello   ')
```

```
## [1] "hello"
```

stringr examples

Split strings (really useful for turning 1 variable into 2):

```r
trees <- c('Jomon Sugi', 'Huon Pine')

str_split_fixed(trees, pattern = ' ', n = 2)
```

```
##      [,1]    [,2]
## [1,] "Jomon" "Sugi"
## [2,] "Huon"  "Pine"
```

More data transformations with dplyr

The **dplyr** package has powerful capabilities to manipulate data frames quickly (many of the functions are written in the compiled language C++).

It is also useful for transforming **grouped observations**, e.g. countries, households.

dplyr

Set up for examples

```r
# Create fake grouped data
library(randomNames)
library(dplyr)
library(tidyr)

people <- randomNames(n = 1000)
people <- sort(rep(people, 4))
year <- rep(2010:2013, 1000)
trend_income <- c(30000, 31000, 32000, 33000)
income <-  replicate(trend_income + rnorm(4, sd = 20000),
                     n = 1000) %>%
            data.frame() %>%
            gather(obs, value, X1:X1000)
income$value[income$value < 0] <- 0
data <- data.frame(people, year, income = income$value)
```

dplyr

```r
head(data)
```

```
##                people year   income
## 1 Abdalla, Jackson 2010 29971.70
## 2 Abdalla, Jackson 2011 33136.72
## 3 Abdalla, Jackson 2012 22250.69
## 4 Abdalla, Jackson 2013 53392.71
## 5    Abdi, Allegra 2010 13584.76
## 6    Abdi, Allegra 2011 20510.07
```

Simple dplyr

Select rows

```r
higher_income <- filter(data, income > 60000)

head(higher_income)
```

```
##                people year   income
## 1   Abril, Destiney 2012 62105.38
## 2 Acevedo, Victoria 2012 76499.51
## 3 Acevedo, Victoria 2013 64942.73
## 4     Adkins, Neema 2013 72515.12
## 5       Ahmed, Joel 2013 61999.80
## 6     Allen, Sierra 2011 69108.46
```

Simple dplyr

Select columns

```
people_income <- select(data, people, income)

# OR

people_income <- select(data, -year)

head(people_income)
```

```
##                  people   income
## 1 Abdalla, Jackson 29971.70
## 2 Abdalla, Jackson 33136.72
## 3 Abdalla, Jackson 22250.69
## 4 Abdalla, Jackson 53392.71
## 5    Abdi, Allegra 13584.76
## 6    Abdi, Allegra 20510.07
```

dplyr with grouped data

Tell dplyr what the groups are in the data with group_by.

```
group_data <- group_by(data, people)
head(group_data)[1:5, ]
```

```
## Source: local data frame [5 x 3]
## Groups: people
##
##             people year   income
## 1 Abdalla, Jackson 2010 29971.70
## 2 Abdalla, Jackson 2011 33136.72
## 3 Abdalla, Jackson 2012 22250.69
## 4 Abdalla, Jackson 2013 53392.71
## 5    Abdi, Allegra 2010 13584.76
```

Note: all of the following functions work on **non-grouped data** as well.

dplyr with grouped data

Now that we have declared the data as grouped, we can do operations on each group.

For example, we can extract the highest and lowest income years for each person:

```r
min_max_income <- summarize(group_data,
                            min_income = min(income),
                            max_income = max(income))
head(min_max_income)[1:3, ]
```

```
## Source: local data frame [3 x 3]
##
##             people min_income max_income
## 1 Abdalla, Jackson  22250.692   53392.71
## 2    Abdi, Allegra  13584.764   34150.15
## 3  Abril, Destiney   7637.069   62105.38
```

dplyr with grouped data

We can sort the data using `arrange`.

```r
# Sort highest income for each person in ascending order
ascending <- arrange(min_max_income, max_income)
head(ascending)[1:3, ]
```

```
## Source: local data frame [3 x 3]
##
##            people min_income max_income
## 1 Juniel, Trevor   916.3455   12527.69
## 2  Lao, T'Shinae     0.0000   13460.24
## 3    Kreul, Tony     0.0000   16445.93
```

dplyr with grouped data

Add desc to sort in descending order

```r
descending <- arrange(min_max_income, desc(max_income))
head(descending)[1:3, ]
```

```
## Source: local data frame [3 x 3]
##
##                       people min_income max_income
## 1            Isaac, Jordahn   14208.26   99642.71
## 2        Colquitt, Matthew       0.00   96647.76
## 3 Kanamu-Hauanio, Hector       0.00   93675.34
```

dplyr with grouped data

summarize creates a new data frame with the summarised data.

We can use mutate to add new columns to the original data frame.

```
data <- mutate(group_data,
               min_income = min(income),
               max_income = max(income))
head(data)[1:3, ]
```

```
## Source: local data frame [3 x 5]
## Groups: people
##
##             people year   income min_income max_income
## 1 Abdalla, Jackson 2010 29971.70   22250.69   53392.71
## 2 Abdalla, Jackson 2011 33136.72   22250.69   53392.71
## 3 Abdalla, Jackson 2012 22250.69   22250.69   53392.71
```

Seminar: Web scraping and data transformations

Scrape and **clean** the Medal Table from `http://www.bbc.com/sport/winter-olympics/2014/medals/countries`.

- Also, sort by total medals in **descending order**.

Work on **gathering data and cleaning** for **Assignment 3**.