MPP-E1180 Lecture 7: Web Scraping + Transforms

Christopher Gandrud

14 March 2016

Objectives for the week

- Assignments
- Review
- Intro to web scraping
- Processing strings, including an intro to regular expressions
- Data and data set transformations with dplyr

Assignment 2

Proposal for your Collaborative Research Project.
Deadline: 25 March
Submit: A (max) 2,000 word proposal created with **R Markdown**.
The proposal will:

- Be written in R Markdown.
- State your research question. And justify why it is interesting.
- Provide a basic literature review (properly cited with BibTeX).
- Identify data sources and appropriate research methodologies for answering your question.

As always, submit the entire GitHub repo.

Assignment 3

Purpose: Gather, clean, and analyse data
Deadline: TBD
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, It also will use literate programming.

Assignment 3

This is ideally a **good first run** at the data gathering and analysis parts of 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 are the characteristics of tidy data?
Why are unique observation IDs so important for data cleaning?

Caveat to Web scraping

I don't expect you to master the tools of web scraping in this course. I just want you to know that these things are **possible**, so that you **know where to look** 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
- Information that requires you to navigate through web forms

To really master web scraping you need a good knowledge of HTML.

Key tools

The most basic tools for web scraping in R:

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

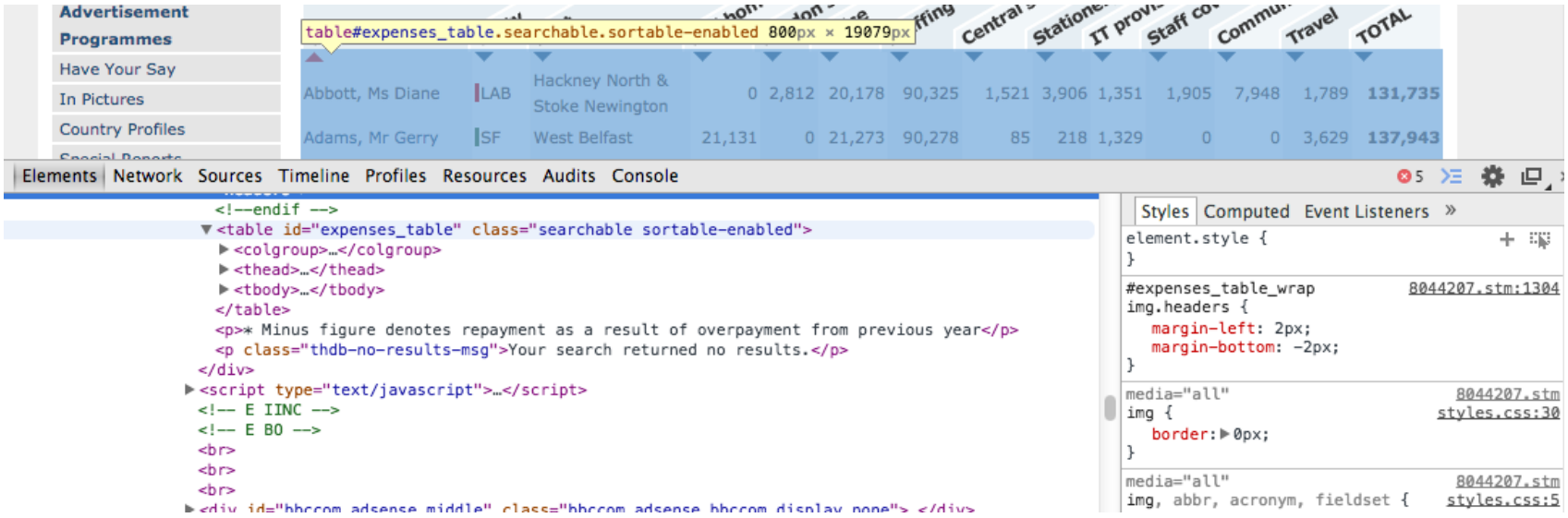
Key steps:

1. **Look at** the HTML for the webpage you want to scrape (e.g. use Inspect Element in Chrome).
2. **Request** a URL with `read_html` (rvest) or `GET` (httr).
3. **Extract** the specific content nodes from the request with `html_nodes`.
4. **Convert** the nodes to your desired R object type.
5. **Clean** 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 it into data frames.
In particular, we want the table tag with the **id** `expenses_table`.
This will be the *node* that we want to extract.

Viewing the web pages source



Web scraping example

```
library(rvest)
library(dplyr)

URL <- 'http://news.bbc.co.uk/2/hi/uk_news/politics/8044207

# Get and parse expenses_table from the webpage
ExpensesTable <- URL %>% read_html() %>%
                         html_nodes('#expenses_table') %>%
                         html_table() %>%
                         as.data.frame
```

Web scraping example

Now we need to clean the ExpensesTable data frame.

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

```
##                         MP Party
## 1        Abbott, Ms Diane   LAB
## 2         Adams, Mr Gerry    SF
## 3           Afriyie, Adam   CON
## 4            Ainger, Nick   LAB
## 5     Ainsworth, Mr Peter   CON
## 6 Ainsworth, Rt Hon Bob   LAB
```

Background on GET from httr

`GET` is probably the most common *RESTful* API **verb** you will use when webscraping.

- **RESTful API** (Representational State Transfer) an approach to creating APIs where resources are referenced (usually via URLs) and representations (documents in HTML, JSON, CSV, etc) are transfered.

Another important verb to consider is `POST`, which allows you to fill in web forms. httr has a `POST` function

Processing strings

A (frustratingly) large proportion of time web scraping and doing data cleaning generally is taken 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.
See also Wiki Books R Programming/Text Processing

`grep`, `gsub`, and related functions

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

Terminology

grep stands for: **G**lobally search a **R**egular **E**xpression and **P**rint

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"
```

agrep

You can do approximate (fuzzy) string matching with agrep.

```
agrep(pattern = "lasy", x = "1 lazy 2")
```

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

Manipulation

Use gsub to substitute strings.

```
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 beginning or end of a string
- find characters that follow or are preceded by a particular character
- find only the first or last occurrence 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

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

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

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

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

```
# 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 expressions cheatsheet

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

Simple regular expressions cheatsheet

You can also find the cheat-sheet at:
SyllabusAndLectures/Lecture7/README

String processing with stringr

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):

```
library(stringr)

str_trim(' hello   ')
```

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

stringr examples

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

```
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 data from **grouped observations**, e.g. countries, households.

dplyr

Set up for examples

```
# 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

```
head(data)
```

```
##              people year    income
## 1 Abeyta, Jinjian 2010  3554.355
## 2 Abeyta, Jinjian 2011 37718.092
## 3 Abeyta, Jinjian 2012 27663.996
## 4 Abeyta, Jinjian 2013 13216.650
## 5    Ablay, Devin 2010 21598.754
## 6    Ablay, Devin 2011 42645.906
```

Simple dplyr

Select rows

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

head(higher_income)
```

```
##                                people year   income
## 1     Aguilar Castellanos, Cindy 2011 62800.33
## 2               Aldawoodi, Houdini 2010 63697.56
## 3                     Allen, Raul 2012 71044.02
## 4 Ammerman Cusack, Marquise-Dion 2010 68264.28
## 5                     Aochi, Brian 2012 62354.67
## 6           Apodaca-Anaya, James 2011 75898.96
```

Simple dplyr

Select columns

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

# OR

people_income <- select(data, -year)

head(people_income)
```

```
##              people    income
## 1 Abeyta, Jinjian  3554.355
## 2 Abeyta, Jinjian 37718.092
## 3 Abeyta, Jinjian 27663.996
## 4 Abeyta, Jinjian 13216.650
## 5    Ablay, Devin 21598.754
## 6    Ablay, Devin 42645.906
```

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 [2]
##
##              people  year    income
##              (fctr) (int)     (dbl)
## 1 Abeyta, Jinjian  2010  3554.355
## 2 Abeyta, Jinjian  2011 37718.092
## 3 Abeyta, Jinjian  2012 27663.996
## 4 Abeyta, Jinjian  2013 13216.650
## 5    Ablay, Devin  2010 21598.754
```

Note: 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:

```
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
##              (fctr)      (dbl)      (dbl)
## 1  Abeyta, Jinjian   3554.355   37718.09
## 2     Ablay, Devin  12409.953   47961.12
## 3 Abzari, Mitchell  10794.974   46711.40
```

dplyr with grouped data

We can sort the data using arrange.

```
# 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
##                (fctr)      (dbl)      (dbl)
## 1       Cloud, Nikhil      0.000   14224.52
## 2        Truitt, John   4131.568   17226.50
## 3 Carrington, Kolter      0.000   18730.91
```

dplyr with grouped data

Add `desc` to sort in descending order

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

```
## Source: local data frame [3 x 3]
##
##                people min_income max_income
##                (fctr)      (dbl)      (dbl)
## 1   Holmes, Nishika   27095.45  100228.15
## 2    Bradshaw, Alex   26236.68   95337.12
## 3 Hildreth, William   27958.38   90074.69
```

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 [1]
##
##              people  year    income min_income max_income
##              (fctr) (int)     (dbl)      (dbl)      (dbl)
## 1 Abeyta, Jinjian  2010  3554.355   3554.355   37718.09
## 2 Abeyta, Jinjian  2011 37718.092   3554.355   37718.09
## 3 Abeyta, Jinjian  2012 27663.996   3554.355   37718.09
```

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**.