Module 6 Lesson A

Clint Valentine

February 18, 2017

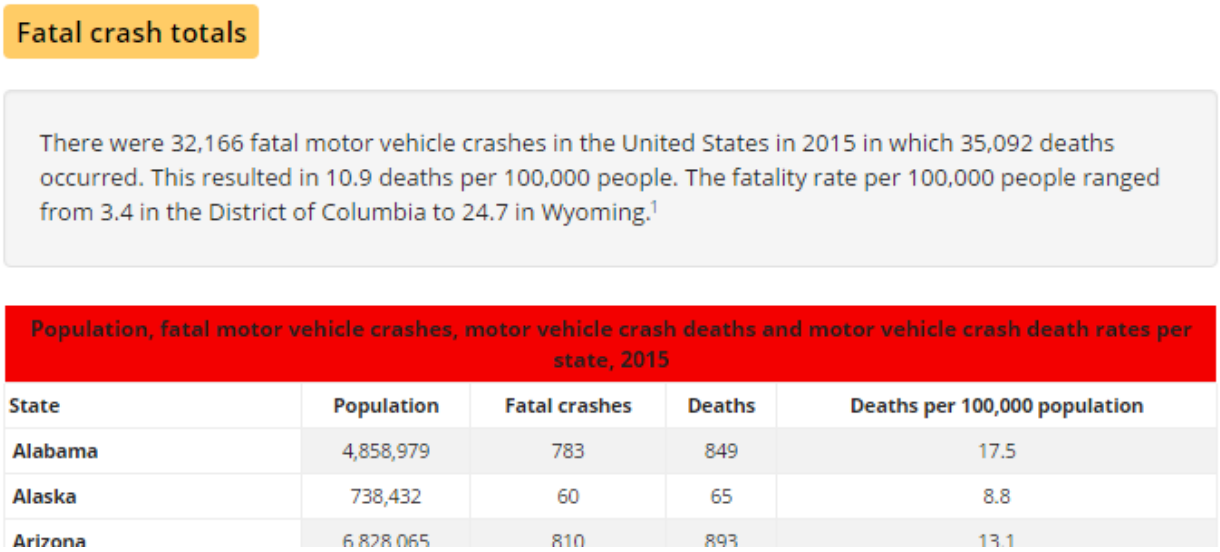
```
rm(list = ls())
library(XML)
library(rvest)
```

Introduction

For this analysis we will be comparing three HTML scraping toolkits. The first and most simple is the standard `XML` style `HTMLTableParser` provided in base `R`. This is the least extensible option but provides easy to use functionality for simple data table scraping. The second being a widely used toolkit for tight integration with `R`, the package `rvest`. The final scraping toolkit will be a package in Python named `BeautifulSoup`.

All three toolkits will be put to the test in scraping the first table on the following URL which summarizes fatalities due to car crashes *per* state in 2015.

http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview



Fatal crash totals

There were 32,166 fatal motor vehicle crashes in the United States in 2015 in which 35,092 deaths occurred. This resulted in 10.9 deaths per 100,000 people. The fatality rate per 100,000 people ranged from 3.4 in the District of Columbia to 24.7 in Wyoming.[1]

Population, fatal motor vehicle crashes, motor vehicle crash deaths and motor vehicle crash death rates per state, 2015

State	Population	Fatal crashes	Deaths	Deaths per 100,000 population
Alabama	4,858,979	783	849	17.5
Alaska	738,432	60	65	8.8
Arizona	6,828,065	810	893	13.1

Figure 1: optional caption text

Figure 1. Slice of the table to be extracted. Note that the first row in the header of the table is merged across all column cells (colored red for this example only). This should give all web scraping approaches summarized here a challenge as this table cell is not needed for catagorical labelling of the data.

R package XML

For simple HTML tables embedded at static URLs the `XML` package simplifies scraping! One command to grab the first table on the page and we have a `data.frame` suitable for downstream analysis. Nothing more to note!.

```
url <- 'http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview'

fatalities <- readHTMLTable(url)[[1]]

head(fatalities)
```

```
##        State Population Fatal crashes Deaths Deaths per 100,000 population
## 1    Alabama  4,858,979           783    849                           17.5
## 2     Alaska    738,432            60     65                            8.8
## 3    Arizona  6,828,065           810    893                           13.1
## 4   Arkansas  2,978,204           472    531                           17.8
## 5 California 39,144,818         2,925  3,176                            8.1
## 6   Colorado  5,456,574           506    546                           10.0
```

R package rvest

The package `rvest` which uses a piping syntax to make data selection quick and easy. This function reads in the HTML document and pipes it to the function `html_node` which selects the first attribute `table`. We then pipe this to a the function `html_table()` which parses the data selected as a table and returns structured data which we can then directly convert to a `data.frame`.

The methods for cleaning up a split row for `rvest` are slightly more complicated than just making the change in the `data.frame` so that is the approach we took.

```
url <- 'http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview'

# Parse first table on webpage with rvest and remove multi-column name.
fatalities <- unname(as.data.frame(read_html(url) %>% html_node('table') %>% html_table()))

# Rename columns with entries of first row then delete first row.
colnames(fatalities) <- fatalities[1,]
fatalities <- fatalities[-1,]

head(fatalities)
```

```
##        State Population Fatal crashes Deaths Deaths per 100,000 population
## 2    Alabama  4,858,979           783    849                           17.5
## 3     Alaska    738,432            60     65                            8.8
## 4    Arizona  6,828,065           810    893                           13.1
## 5   Arkansas  2,978,204           472    531                           17.8
## 6 California 39,144,818         2,925  3,176                            8.1
## 7   Colorado  5,456,574           506    546                           10.0
```

Python package BeautifulSoup

Next up is the Python package `BeautifulSoup`. Note you will need both Python and the packages `bs4` and `pandas` installed to run this code block. First we download the html as text and then instantiate a `BeautifulSoup` class which interprets the HTML structure as `lxml`. Then we iterate over the rows (`tr`) of the first `table` (remember Python has 0-based indexing). The first row is skipped since it has the multicell row with no information. Then we iterate over all of the cells of the row and append them to a list of lists `rows`. We also strip whitespace from all entries and remove commas (used as delimeters of thousands places in numbers). Finally we save as a `.csv` so we can load the data structure into `R`.

This approach, although far more verbose that the `rvest` method has the benefit of manipulating the data in the its native format before coercing it into a `data.frame`. This is seen as a benefit when skipping hard to parse table properties such as multicell rows.

```
import requests
import pandas as pd
from itertools import chain
from bs4 import BeautifulSoup

url = 'http://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview'

rows = []

# Skip the first row as its a multicell row.
skiprows = 1

# Instantiate the parser class.
soup = BeautifulSoup(requests.get(url).text, 'lxml')

for i, row in enumerate(soup.find_all('table')[0].find_all('tr')):
    # Skip ahead skiprows.
    if i < skiprows: continue

    # If there is a row here append an empty list for storing columns.
    rows.append([])

    # Loop through any header and cell fields in this row.
    for column in chain(row.find_all('th'), row.find_all('td')):
        # Get plain text, strip padding, and replace commas for csv export.
        rows[i - skiprows].append(column.get_text().strip().replace(',', ''))

# Save as a csv from a DataFrame.
pd.DataFrame(rows, columns=rows.pop(0)).to_csv('fatalities.csv', index=False)
```

```
fatalities <- read.csv2('fatalities.csv', sep=',', header=T)

head(fatalities)
```

```
##        State Population Fatal.crashes Deaths Deaths.per.100000.population
## 1    Alabama    4858979           783    849                         17.5
## 2     Alaska     738432            60     65                          8.8
## 3    Arizona    6828065           810    893                         13.1
## 4   Arkansas    2978204           472    531                         17.8
## 5 California   39144818          2925   3176                          8.1
## 6   Colorado    5456574           506    546                         10.0
```