# Web Scraping for Zillow Sacramento

Yidong Zhou

1/27/2021

Web scraping is the process of collecting structured web data in an automated fashion. It's also called web data extraction. Some of the main use cases of web scraping include price monitoring, price intelligence, news monitoring, lead generation and market research among many others.

We'll going to scrap the Zillow website for Sacramento (https://www.zillow.com/sacramento-ca/) using CSS selectors. CSS selectors are particularly useful in conjunction with http://selectorgadget.com/: it makes it easy to find exactly which selector you should be using. If you haven't used CSS selectors before, work your way through the fun tutorial at http://flukeout.github.io/ (class selector and descendant selector).

```
library(tidyverse)
library(rvest)
```

The `tidyverse` is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures, e.g., `ggplot2` for nice plot, `dplyr` for data manipulation, and so on. The `tidyverse` also includes many other packages with more specialised usage. They are not loaded automatically with `library(tidyverse)`, so you'll need to load each one with its own call to `library()`. The one we are going to use is `rvest`, a new package that makes it easy to scrape (or harvest) data from html web pages, inspired by libraries like beautiful soup. Another choice is `ZillowR` package, which serve as an R Inteface to Zillow Real Estate and Mortgage.

It seems the version of `rvest` integrated in `tidyverse` is out of date. To use function `html_text2()`, you need to install `rvest` directly from GitHub (https://github.com/tidyverse/rvest) using command `devtools::install_github("https://github.com/tidyverse/rvest")`.

```
links <- sprintf("https://www.zillow.com/sacramento-ca/%d_p", 1:11)
```

`sprintf()` ia a wrapper for the C function `sprintf()`, that returns a character vector containing a formatted combination of text and variable values. Here, it's equivalent to `paste0("https://www.zillow.com/sacramento-ca/", 1:11, "_p")`.

```
results <- map(links, ~ {
  # select houses
  houses <- read_html(.x) %>%
    html_nodes(".photo-cards li article")
  z_id <- houses %>%
    html_attr("id")
  # address
  address <- houses %>%
    html_node(".list-card-addr") %>%
    html_text()
  # price
  price <- houses %>%
    html_node(".list-card-price") %>%
    html_text() %>%
    readr::parse_number()
```

```
  # info
  params <- houses %>%
    html_node(".list-card-info") %>%
    html_text2()
  # number of bedrooms
  beds <- params %>%
    str_extract("\\d+(?=\\s*bds)") %>%
    as.numeric()
  # number of bathrooms
  baths <- params %>%
    str_extract("\\d+(?=\\s*ba)") %>%
    as.numeric()
  # total square footage
  house_a <- params %>%
    str_extract("[0-9,]+(?=\\s*sqft)") %>%
    str_replace(",", "") %>%
    as.numeric()

  tibble(price = price, beds= beds, baths=baths, house_area = house_a)

}
) %>%
  bind_rows(.id = 'page_no')
```

`map()` apply a function or a formula to each element of a list or atomic vector (the first argument). `.x` represents one element in the first argument.

When applied to a list of nodes, `html_nodes()` returns all matching nodes beneath any of the elements, flattening results into a new nodelist. `html_node()` returns the first matching node. If there are no matching nodes, it returns a "missing" node.

`html_text()`

`parse_number()` drops any non-numeric characters before or after the first number.

`strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)` split the elements of a character vector `x` into substrings according to the matches to substring `split` within them.

`str_extract(string, pattern)` is used to extract matching patterns from a string.

### Basic concepts of regular expressions

- `\\d`: matches a single character that is a digit
- `\\s`: matches a whitespace character (includes tabs and line breaks)
- `\`: there are 12 characters with special meanings: the backslash `\`, the caret `^`, the dollar sign `$`, the period or dot `.`, the vertical bar or pipe symbol `|`, the question mark `?`, the asterisk or star `*`, the plus sign `+`, the opening parenthesis `(`, the closing parenthesis `)`, the opening square bracket `[`, and the opening curly brace `{`. If you want to use any of these characters as a literal in a regex, you need to escape them with a backslash `\`. If you want to match 1+1=2, the correct regex is `1\+1=2`. Otherwise, the plus sign has a special meaning. Similarly, a literal `\` should be `\\`.
- `?`: The question mark indicates zero or one occurrences of the preceding element. For example, `colou?r` matches both "color" and "colour".
- `*`: The asterisk indicates zero or more occurrences of the preceding element. For example, `ab*c` matches "ac", "abc", "abbc", "abbbc", and so on.
- `+`: The plus sign indicates one or more occurrences of the preceding element. For example, `ab+c` matches "abc", "abbc", "abbbc", and so on, but not "ac".

- `()`: Parentheses are used to define the scope and precedence of the operators (among other uses). For example, `gray|grey` and `gr(a|e)y` are equivalent patterns which both describe the set of "gray" or "grey".
- `[]`: A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches "a", "b", or "c". `[a-z]` specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: `[abcx-z]` matches "a", "b", "c", "x", "y", or "z", as does `[a-cx-z]`. The `-` character is treated as a literal character if it is the last or the first character within the brackets: `[abc-]`, `[-abc]`. Note that backslash escapes are not allowed. The `]` character can be included in a bracket expression if it is the first character: `[]abc]`.
- `d(?=r)`: matches a `d` only if is followed by `r`, but `r` will not be part of the overall regex match.

Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

## Code Appendix

```
library(tidyverse)
library(rvest)
links <- sprintf("https://www.zillow.com/sacramento-ca/%d_p", 1:11)
results <- map(links, ~ {
  # select houses
  houses <- read_html(.x) %>%
    html_nodes(".photo-cards li article")
  z_id <- houses %>%
    html_attr("id")
  # address
  address <- houses %>%
    html_node(".list-card-addr") %>%
    html_text()
  # price
  price <- houses %>%
    html_node(".list-card-price") %>%
    html_text() %>%
    readr::parse_number()
  # info
  params <- houses %>%
    html_node(".list-card-info") %>%
    html_text2()
  # number of bedrooms
  beds <- params %>%
    str_extract("\\d+(?=\\s*bds)") %>%
    as.numeric()
  # number of bathrooms
  baths <- params %>%
    str_extract("\\d+(?=\\s*ba)") %>%
    as.numeric()
  # total square footage
  house_a <- params %>%
    str_extract("[0-9,]+(?=\\s*sqft)") %>%
    str_replace(",", "") %>%
    as.numeric()

  tibble(price = price, beds= beds, baths=baths, house_area = house_a)
```

```
}
) %>%
  bind_rows(.id = 'page_no')
```