Live Session 2: Getting Data

Zak Varty

Week in Review

Tabular data

- Reading with base R and {readr}
- Tibbles
- Tidy data, wide data and tall data

Web Scraping

- Intro to HTML and CSS
- {rvest} for scraping webpages and extracting content
- Amazon task (review to come)

APIs

- Ways of sharing and sourcing data
- HTTP requests and responses
- Use wrappers where you can

Discussion

Question 1: RDS files

---

1. Roger Peng states that files can be imported and exported using readRDS() and saveRDS() for fast and space efficient data storage. What is the downside to doing so?

---

2. What data types have you come across (that we have not discussed already) and in what context are they used?

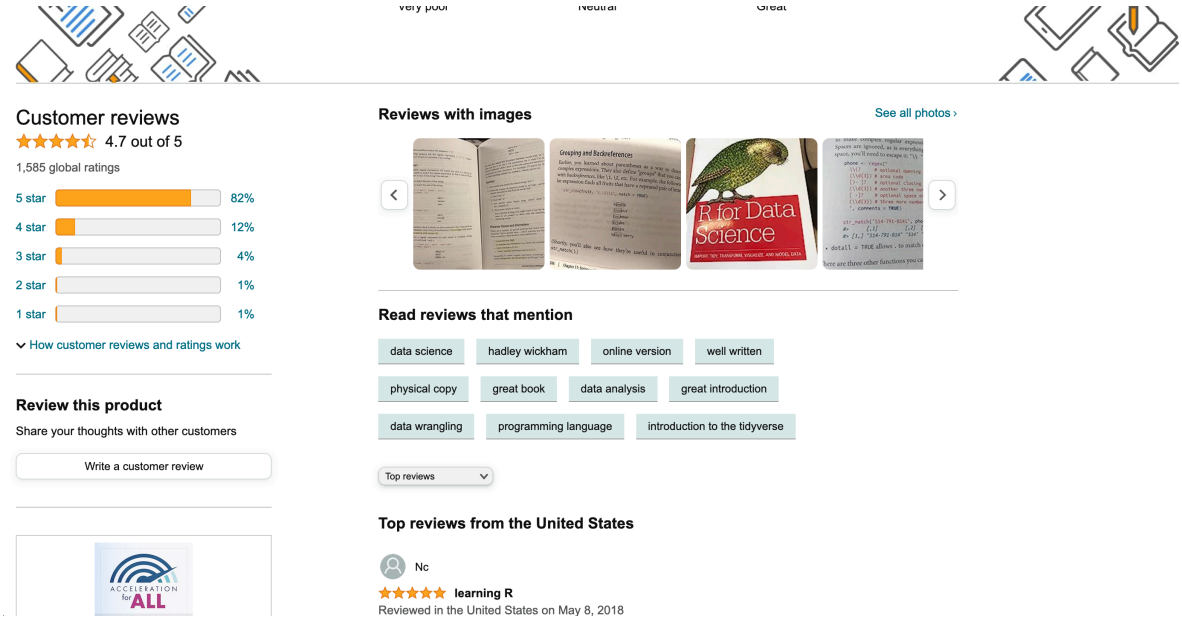
3. What do you have to give greater consideration to when scraping data than when using an API?

Scraping Book Reviews

Scrape R4DS Star Rating Percentages

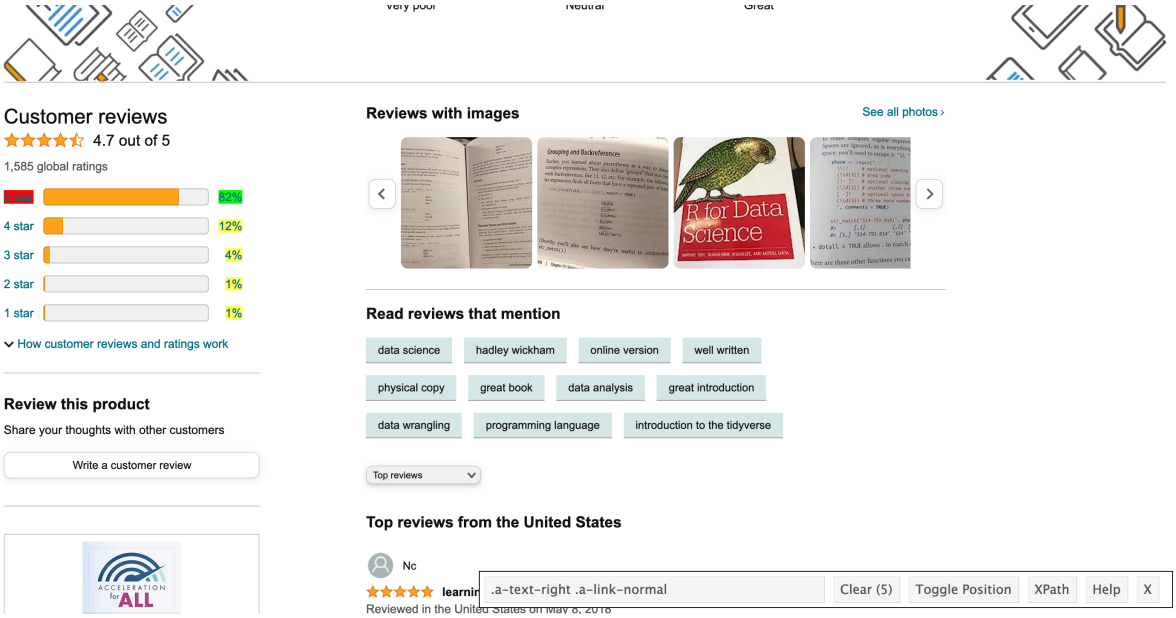
```
library("rvest")
library("httr")
library("magrittr")
```

Visiting the R for Data Science webpage and scrolling down we find the review summaries giving the percentage of reviewers in each category.



Using the httr selector gadget we can identify that the elements we want to scrape are given by

```
.a-text-right .a-link-normal
```



We first scrape the entire page.

```
r4ds_url <- "https://www.amazon.com/dp/1491910399/"
r4ds_html <- rvest::read_html(r4ds_url)
```

Then we can use Rvest functions to extract the elements that we care about and convert these to strings.

```
data_strings <- r4ds_html %>%
  #rvest::html_elements(".a-nowrap .a-link-normal") %>%
  rvest::html_elements(".a-text-right .a-link-normal") %>%
  rvest::html_text2()

data_strings
```

```
[1] "82%" "12%" "4%"  "1%"  "1%"
```

Finally, we want to drop the percentage sign from each element of the vector and convert this to a vector of integers, rather than strings.

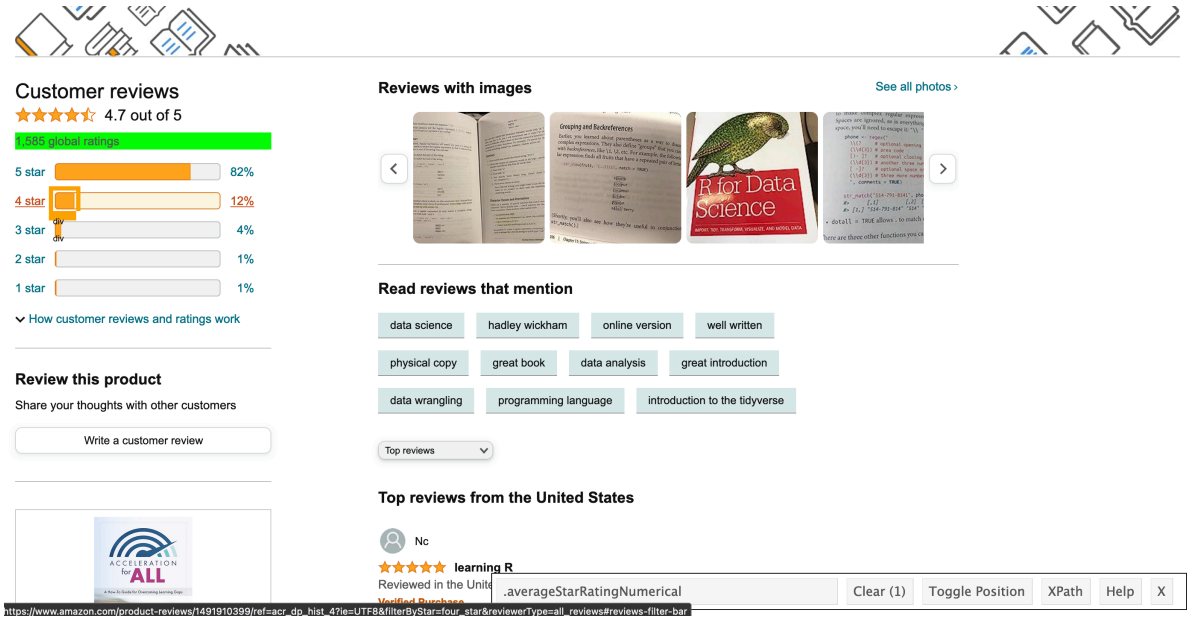
```
data_values_as_character <- stringr::str_sub(data_strings, start = 1, end = -2)
data_values <- as.integer(data_values_as_character)
data_values
```

```
[1] 82 12  4  1  1
```

Scrape R4DS Number of Ratings

Similarly, we can scrape the number of reviews using the selectors

`.averageStarRatingNumerical .a-color-secondary`



We extract the text element in the same way as before.

```
r4ds_review_count <- r4ds_html %>%
  rvest::html_elements(".averageStarRatingNumerical") %>%
  rvest::html_text2()

r4ds_review_count
```

```
[1] "1,586 global ratings"
```

To convert this to an integer we can work with, we first drop the 15 characters " global ratings" from the end.

```
r4ds_review_count <- r4ds_html %>%
  rvest::html_elements(".averageStarRatingNumerical .a-color-secondary") %>%
  rvest::html_text2() %>%
  stringr::str_sub(start = 1, end = -16)

r4ds_review_count
```

```
[1] "1,586"
```

The last things we need to do is get rid of the comma and convert this to an integer.

```
r4ds_review_count <- r4ds_html %>%
  rvest::html_elements(".averageStarRatingNumerical .a-color-secondary") %>%
  rvest::html_text2() %>%
  stringr::str_sub(start = 1, end = -16) %>%
  stringr::str_split_1(",") %>%
  stringr::str_flatten() %>%
  as.integer()

r4ds_review_count
```

```
[1] 1586
```

Summary table R4DS reviews

```
r4ds_data <- tibble::tibble(
  product = "R4DS",
  n_reviews = r4ds_review_count,
  percent_5_star = data_values[1],
  percent_4_star = data_values[2],
  percent_3_star = data_values[3],
  percent_2_star = data_values[4],
  percent_1_star = data_values[5],
  url = r4ds_url)

r4ds_data
```

```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 R4DS         1586             82             12              4              1
# i 2 more variables: percent_1_star <int>, url <chr>
```

Making this a function

Let's abstract out the URL and product name to turn this into a function.

```
get_amazon_reviews <- function(product_name, url){

  # Scrape Amazon page of product
  product_html <- rvest::read_html(url)

  # Extract percentage receiving each number of stars
  review_percentages <- product_html %>%
  rvest::html_elements(".a-text-right .a-link-normal") %>%  # extract information
  rvest::html_text2() %>%                                   # convert to text
  stringr::str_sub(start = 1, end = -2) %>%                 # remove "%" from string
  as.integer()                                              # convert to integer

  # Extract total number of reviews
  review_count <- product_html %>%
  rvest::html_elements(".averageStarRatingNumerical .a-color-secondary") %>%
  rvest::html_text2() %>%
  stringr::str_sub(start = 1, end = -16) %>%
  stringr::str_split_1(",") %>%
  stringr::str_flatten() %>%
  as.integer()

  # Construct Tibble
  product_data <- tibble::tibble(
  product = product_name,
  n_reviews = review_count,
  percent_5_star = review_percentages[1],
  percent_4_star = review_percentages[2],
  percent_3_star = review_percentages[3],
  percent_2_star = review_percentages[4],
  percent_1_star = review_percentages[5],
  url = url)
```

```
  product_data
  }
```

We can test that this works for R4DS.

```
  get_amazon_reviews("R4DS", url = r4ds_url)
```

```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 R4DS         1586             82             12              4              1
# i 2 more variables: percent_1_star <int>, url <chr>
```

This function is doing a lot, let's move some of the stages out to helper functions. This will make life easier for us if the structure of the webpages change over time and also if we need to debug the function.

We will have one function to extract the review percentages from the scraped html.

```
  extract_review_percentages <- function(scraped_html, css_selector = ".a-text-right .a-link
    scraped_html %>%
    rvest::html_elements(css_selector) %>%                # extract information
    rvest::html_text2() %>%                               # convert to text
    stringr::str_sub(start = 1, end = -2) %>%             # remove "%" from string
    as.integer()
  }
```

A second function to extract the review count from the scraped html.

```
  extract_review_count <- function(scraped_html, css_selector = ".averageStarRatingNumerical
    scraped_html %>%
    rvest::html_elements(".averageStarRatingNumerical .a-color-secondary") %>%
    rvest::html_text2() %>%
    stringr::str_sub(start = 1, end = -16) %>%
    stringr::str_split_1(",") %>%
    stringr::str_flatten() %>%
    as.integer()
  }
```

And a third function to assemble this information into a tibble.

```
construct_product_review_tibble <- function(product_name, url, review_count, review_percen
  tibble::tibble(
  product = product_name,
  n_reviews = review_count,
  percent_5_star = review_percentages[1],
  percent_4_star = review_percentages[2],
  percent_3_star = review_percentages[3],
  percent_2_star = review_percentages[4],
  percent_1_star = review_percentages[5],
  url = url)
}
```

Each of these can then be called from within an updated version of `get_amazon_reviews()`.

```
get_amazon_reviews <- function(product_name, url){

  # Scrape Amazon page of product
  product_html <- rvest::read_html(url)

  # Extract percentage receiving each number of stars
  review_percentages <- extract_review_percentages(product_html)

  # Extract total number of reviews
  review_count <- extract_review_count(product_html)

  # Construct Tibble
  construct_product_review_tibble(product_name, url, review_count, review_percentages)
}
```

Again, we should test that this still works.

```
get_amazon_reviews("R4DS", url = r4ds_url)
```

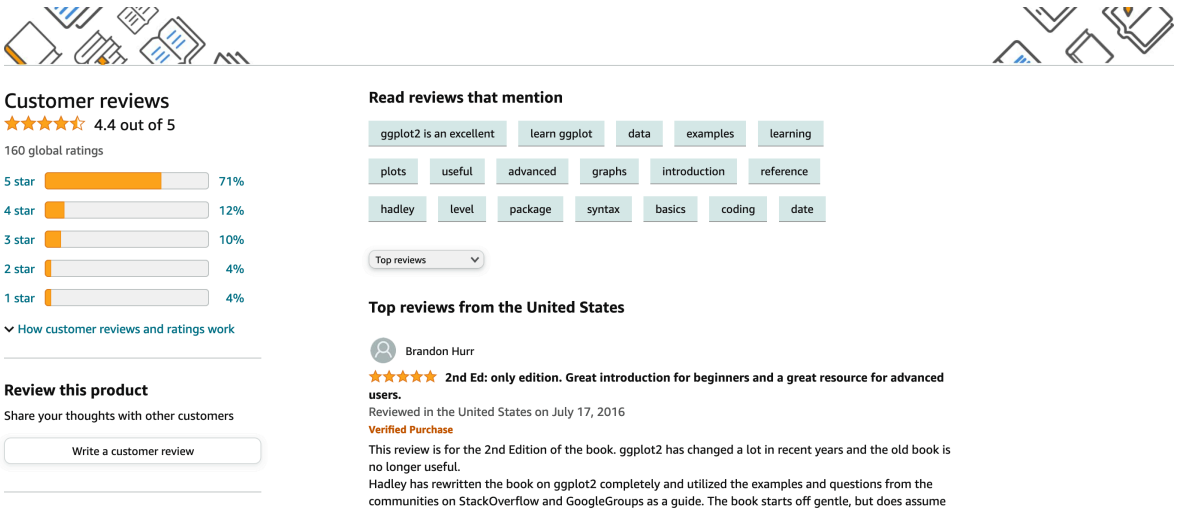
```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 R4DS         1586             82             12              4              1
# i 2 more variables: percent_1_star <int>, url <chr>
```

We can also try it with the ggplot2 book

```
ggplot2_url <- "https://www.amazon.com/dp/331924275X"
get_amazon_reviews("ggplot2", url = ggplot2_url)
```

```
Warning in scraped_html %>% rvest::html_elements(css_selector) %>%
rvest::html_text2() %>% : NAs introduced by coercion
```

```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 ggplot2       160             NA             71             12             10
# i 2 more variables: percent_1_star <int>, url <chr>
```



Hooray! It works! How about the R packages?

```
r_packages_url <- "https://www.amazon.com/dp/1491910593/"
get_amazon_reviews("R packages", url = r_packages_url)
```

```
# A tibble: 1 x 8
  product  n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>        <int>          <int>          <int>          <int>          <int>
1 R packa~       107             81             15              4             NA
# i 2 more variables: percent_1_star <int>, url <chr>
```

Once again, this has worked out.

But those `NA` values worry me. Let's take a look at where they are coming from.

```
r_packages_html <- rvest::read_html(r_packages_url)
extract_review_percentages(r_packages_html)
```

```
[1] 81 15  4
```

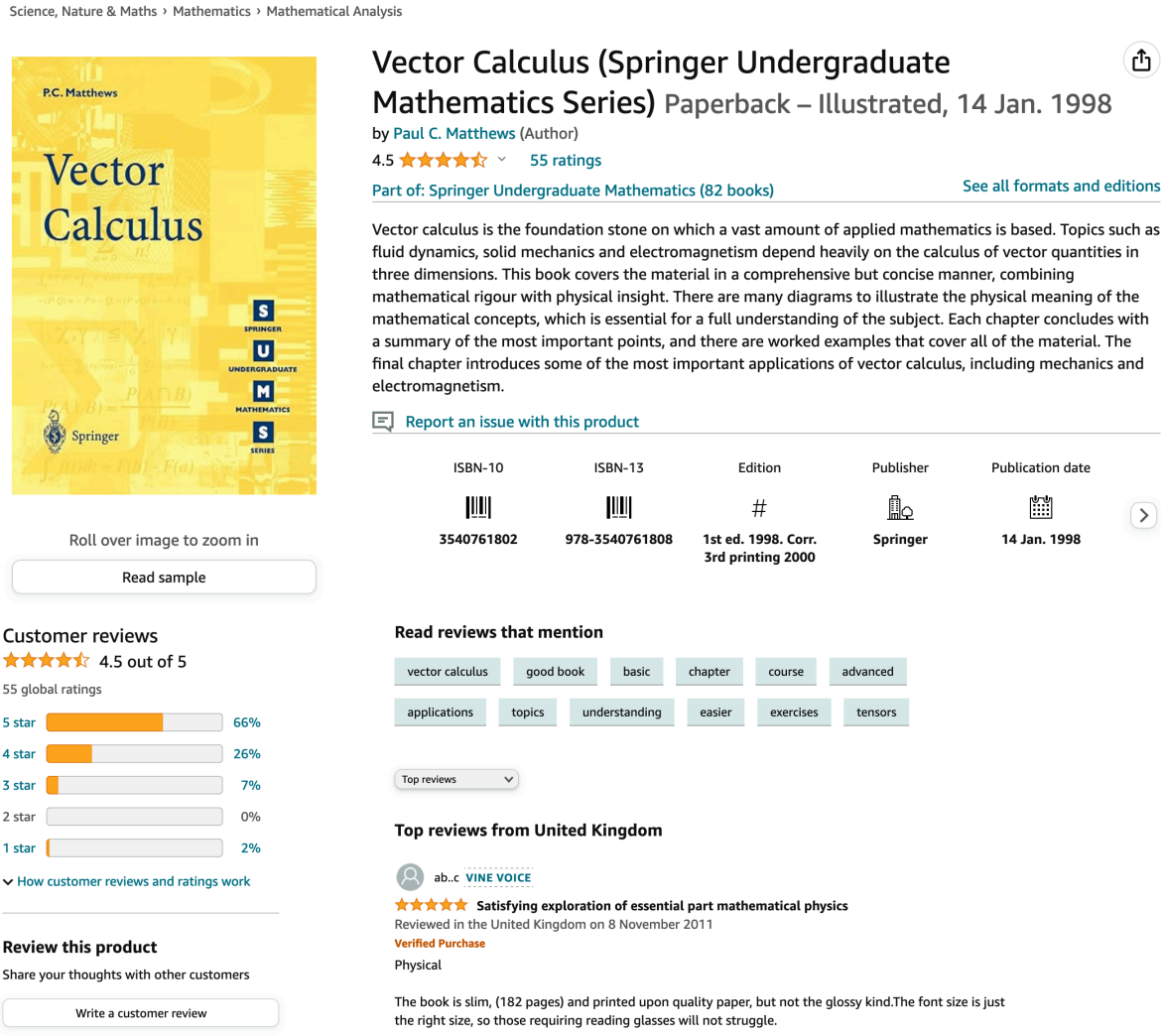
We only have three values being extracted. This is likely because only the non-zero values were click-able on the webpage. It seems we got lucky and those happened to be the first three, but what would have happened if that were not the case?

To find out, we need to identify a product which satisfies:

- (at least) one star category $x \in \{2, 3, 4, 5\}$ that has zero percent
- a second star category $y \in \{1, 2, 3, 4\}$ such that $y < x$ and y has non-zero percentage of reviews.

To get an empty star category, we can maximise our chances by looking at product with a low total number of reviews. Staying on topic, I decided to look at mathematics textbooks.

It took a bit of digging (lots of books received only 5-star and 4-star reviews) to find Vector Calculus which, *at the time of writing* has no 2-star reviews.



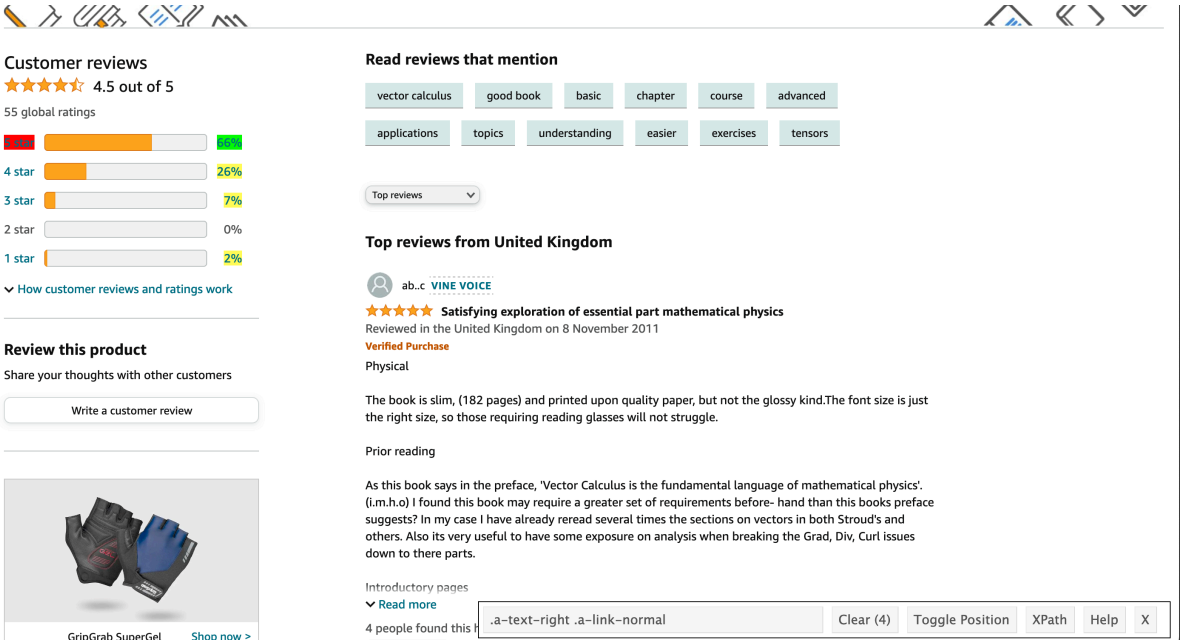

```
vector_calc_url <- "https://www.amazon.co.uk/dp/3540761802"
get_amazon_reviews(product_name = "vector calculus", url = vector_calc_url)
```

```
# A tibble: 1 x 8
  product  n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>        <int>          <int>          <int>          <int>          <int>
1 vector ~        55             66             26              7              2
# i 2 more variables: percent_1_star <int>, url <chr>
```

As we suspected - the one star reviews are misplaced.

I spent a long time trying to get workarounds, but missing values are tricky to deal with. I got some code working, but it was very clunky and involved using `try()` within a `for` loop.

A much simpler solution is to return to the Selector gadget and update our CSS selectors within the extraction function.



This more careful selection gives the following CSS selector:

`#histogramTable .a-text-right .a-size-base`

We can use this to update the default value in `extract_review_percentages()`

```
extract_review_percentages <- function(scraped_html, css_selector = "#histogramTable .a-te
  scraped_html %>%
  rvest::html_elements(css_selector) %>%              # extract information
  rvest::html_text2() %>%                             # convert to text
  stringr::str_sub(start = 1, end = -2) %>%           # remove "%" from string
  as.integer()
}
```

This works for our vector calculus example.

```
get_amazon_reviews(product_name = "vector calculus", url = vector_calc_url)
```

```
# A tibble: 1 x 8
  product  n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
```

```
  <chr>        <int>          <int>          <int>          <int>          <int>
1 vector ~        55             66             26              7              0
# i 2 more variables: percent_1_star <int>, url <chr>
```

It corrects also corrects our output for the R packages example to be `0` rather than `NA`,

```
get_amazon_reviews(product_name = "R packages", url = r_packages_url)
```

```
# A tibble: 1 x 8
  product  n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>        <int>          <int>          <int>          <int>          <int>
1 R packa~       107             81             15              4              0
# i 2 more variables: percent_1_star <int>, url <chr>
```

---

and it has not broken any of our complete examples

```
get_amazon_reviews(product_name = "R4DS", url = r4ds_url)
```

```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 R4DS         1586             82             12              4              1
# i 2 more variables: percent_1_star <int>, url <chr>
```

```
get_amazon_reviews(product_name = "ggplot2", url = ggplot2_url)
```

```
# A tibble: 1 x 8
  product n_reviews percent_5_star percent_4_star percent_3_star percent_2_star
  <chr>       <int>          <int>          <int>          <int>          <int>
1 ggplot2       160             71             12             10              4
# i 2 more variables: percent_1_star <int>, url <chr>
```

Discussion

- What did you do differently to me?
- What was easy, what was difficult?
- How could we formalise and automate this testing workflow? What might be make this difficult?