Web scraping

STA 032: Gateway to data science Lecture 25

Jingwei Xiong

June 5, 2023

Web scraping in R

- What is web scraping:
  - Web scraping is the automated process of extracting data from websites.
  - It involves fetching and parsing HTML content to extract specific information.
  - It Enables gathering data for analysis, research, or automation purposes.
- Why web scraping:
  - Access to vast amounts of data available on websites.
  - Extract structured data from unstructured web pages.
  - Automate data collection and save time.

What can web scraping data do?

Feature	Description
Tables	Fetch tables like from Wikipedia
Forms	You can submit forms and fetch the results
CSS	You can access parts of a website using style or CSS selectors
Tweets	Process tweets including emojis
Web Sites	User forums have lots of content
Instagram	Yes, you can "scrape" photos also

- Common Applications of Web Scraping:
    - Market research and competitive analysis.
    - Price comparison and monitoring.
    - News aggregation and sentiment analysis.
    - Data collection for machine learning models.
- Before we do web scraping, we need to know how website is generated.

Crush course of website

- Clients and Servers
  - The Internet is a vast computer network that allows computers to send messages to each other.
  - Networking techniques enable communication between computers, with one computer acting as a server and others as clients.
  - The World Wide Web uses the Hyper Text Transfer Protocol (HTTP) to retrieve web pages and associated files, with the client (e.g., web browser) making HTTP requests and the server responding in a request-response cycle.

The Internet is, basically, just a computer network spanning most of the world. Computer networks make it possible for computers to send each other messages. Typically, one computer, which we will call the server, is waiting for other computers to start talking to it. Once another computer, the client, opens communications with this server, they will exchange whatever it is that needs to be exchanged using some specific language, a protocol.

The protocol that is of interest to us is that used by the World Wide Web. It is called HTTP, which stands for Hyper Text Transfer Protocol, and is used to retrieve web-pages and the files associated with them.

In HTTP communication, the server is the computer on which the web-page is stored. The client is the computer, such as yours, which asks the server for a page, so that it can display it. Asking for a page like this is called an HTTP request and the exchange of messages between the client (usually a web browser) and the server is called the request-response-cycle.

Crush course of website

- URLs

Web-pages and other files that are accessible though the Internet are identified by URLs, which is an abbreviation of **Universal Resource Locators**. A URL looks like this:

http://acc6.its.brooklyn.cuny.edu/~phalsall/texts/taote-v3.html

It is composed of three parts. The start, http://, indicates that this URL uses the HTTP protocol.

The next part, acc6.its.brooklyn.cuny.edu, names the server on which this page can be found.

The end of the URL, /~phalsal/texts/taote-v3.html, names a specific file on this server.

Crush course of website

The Web has its own languages: HTML, CSS, Javascript

- HTML stands for **HyperText Mark-up Language**. An HTML document is all text with HTML tags to control information flows.
- CSS stands for **Cascading Style Sheets**. CSS is a styling language designed to describe the look and formatting (the presentation semantics) of web pages.
- JavaScript is the last of the three languages that can be natively consumed by web browsers.

More introduction:
https://www.openbookproject.net/books/mi2pwjs/ch05.html

Web scraping in R using package: rvest

- **rvest** is a package inside **tidyverse**. It helps you scrape (or harvest) data from web pages, easy to express common web scraping tasks.
- Installation:

```
# The easiest way to get rvest is to install the whole tidyverse:
install.packages("tidyverse")

# Alternatively, install just rvest:
install.packages("rvest")
```

- There are several steps involved in using **rvest** which are conceptually quite straightforward:

1. Identify a URL to be examined for content
2. Use **Selector Gadet, xPath, or Google Insepct** to identify the “selector” This will be a paragraph, table, hyper links, images
3. Load rvest
4. Use **read_html** to “read” the URL
5. Pass the result to **html_nodes** to get the selectors identified in step number 2
6. Get the text or table content

Example: Parsing A Table From Wikipedia

Look at the Wikipedia Page for world population:
(https://en.wikipedia.org/wiki/World_population)

Rank	Country / Territory	Population	Date	% of world population	Source
1	China[note 4]	1,394,840,000	October 27, 2018	18.2%	[82]
2	India	1,338,780,000	October 27, 2018	17.5%	[83]
3	United States	328,071,000	October 27, 2018	4.28%	[84]
4	Indonesia	265,015,300	July 1, 2018	3.46%	[85]
5	Pakistan	202,466,000	October 27, 2018	2.64%	[86]
6	Brazil	209,768,000	October 27, 2018	2.74%	[87]
7	Nigeria	188,500,000	October 31, 2016	2.46%	[88]
8	Bangladesh	165,430,000	October 27, 2018	2.16%	[89]
9	Russia[note 5]	146,877,088	January 1, 2018	1.92%	[90]
10	Japan	126,440,000	October 1, 2018	1.65%	[91]

How to get this table?

Get the desired table

First we will load packages **rvest** that will help us throughout this session. Then set the **url** and fetch the webpage by **read_html**.

```
library(rvest)
url <- "https://en.wikipedia.org/wiki/World_population"
webpage <- read_html(url)
```

In this case we'll need to figure out what number table it is we want. We could fetch all the tables and then experiment to find the precise one.

```
webpage %>% html_nodes("table")
```

```
{xml_nodeset (29)}
 [1] <table class="wikitable" style="text-align:center; float:right; clear:ri ...
 [2] <table class="wikitable">\n<caption>Current world population and latest  ...
 [3] <table class="box-Notice plainlinks metadata ambox ambox-notice" role="p ...
 [4] <table class="wikitable sortable">\n<caption>Population by region (2020  ...
 [5] <table class="wikitable sortable"><tbody>\n<tr>\n<th scope="col">Rank\n< ...
 [6] <table class="wikitable" style="font-size:90%">\n<caption>World populati ...
 [7] <table class="box-Notice plainlinks metadata ambox ambox-notice" role="p ...
 [8] <table class="wikitable sortable" style="text-align:right">\n<caption>10 ...
 [9] <table class="wikitable sortable" style="text-align:right">\n<caption>Co ...
[10] <table class="box-Notice plainlinks metadata ambox ambox-notice" role="p ...
[11] <table class="box-Notice plainlinks metadata ambox ambox-notice" role="p ...
[12] <table class="wikitable sortable">\n<caption>Global annual population gr ...
[13] <table class="wikitable sortable" style="font-size:97%; text-align:right ...
[14] <table class="wikitable sortable" style="font-size:97%; text-align:right ...
[15] <table class="wikitable" style="text-align:right;"><tbody>\n<tr>\n<th>Ye ...
[16] <table class="box-More_citations_needed_section plainlinks metadata ambo ...
[17] <table class="wikitable" style="text-align:center; margin-top:0.5em; mar ...
[18] <table class="wikitable" style="text-align:right; margin-top:2.6em; font ...
```

```
length(webpage %>% html_nodes("table"))
```

```
[1] 29
```

It suggests there are 19 tables. If we want the specific one in the picture, which one is the correct one?

We need **inspect** the webpage.

Listed below are the steps to inspect element in the Chrome browser:

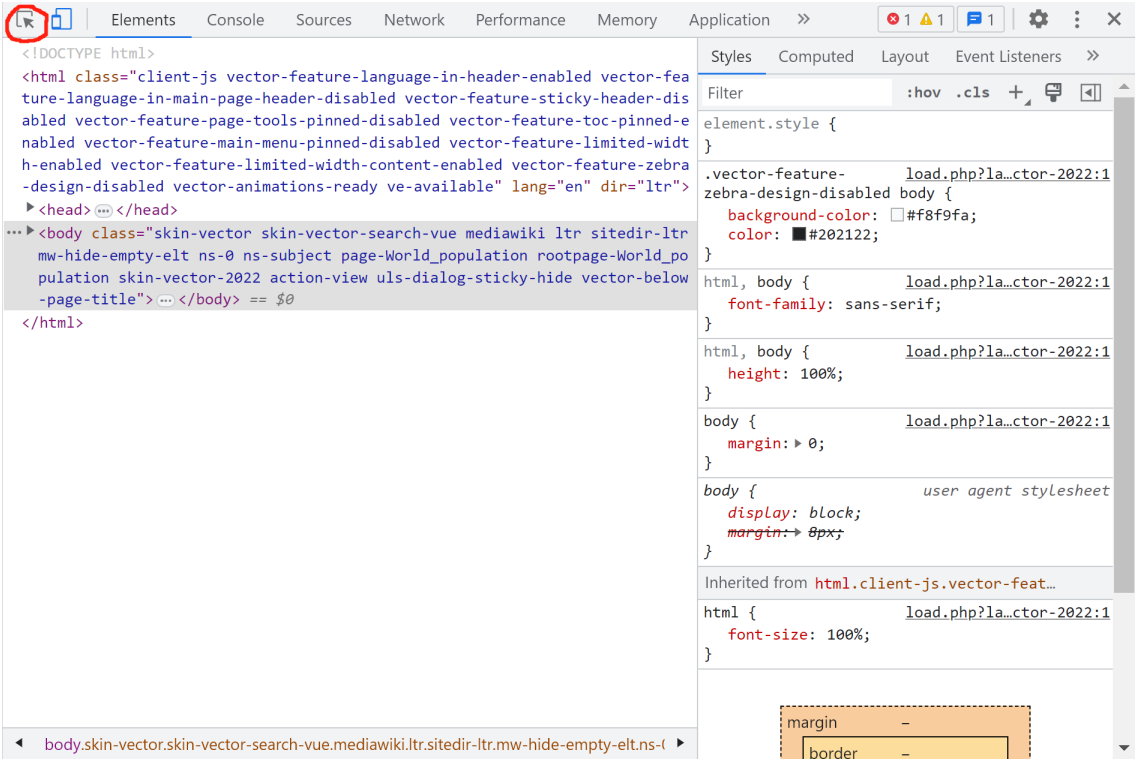
1. Launch Chrome and navigate to the desired web page that needs to be inspected.
2. At the top right corner, click on three vertical dots
3. From the drop-down menu, click on More tools -> Developer Tools
4. Alternatively, you can use the Chrome inspect element shortcut Key.

MacOS – Command + Option + C

Windows – Control + Shift + C.

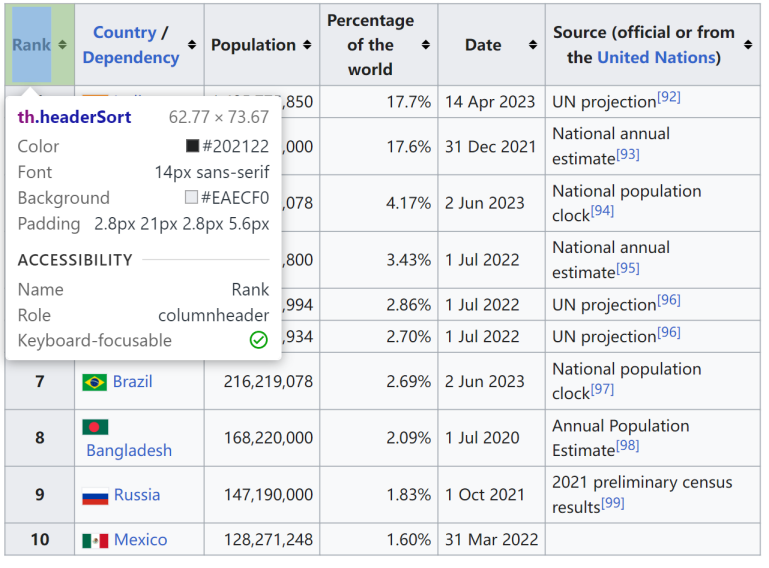
After we open the https://en.wikipedia.org/wiki/World_population webpage, and open the **inspect** mode, we need first click the red circle icon (select element), activate it. (When activated it will become blue)

```
knitr::include_graphics("1.png", dpi = 320)
```



Then we scroll down to the table, move your mouse on the element of the table, click it.

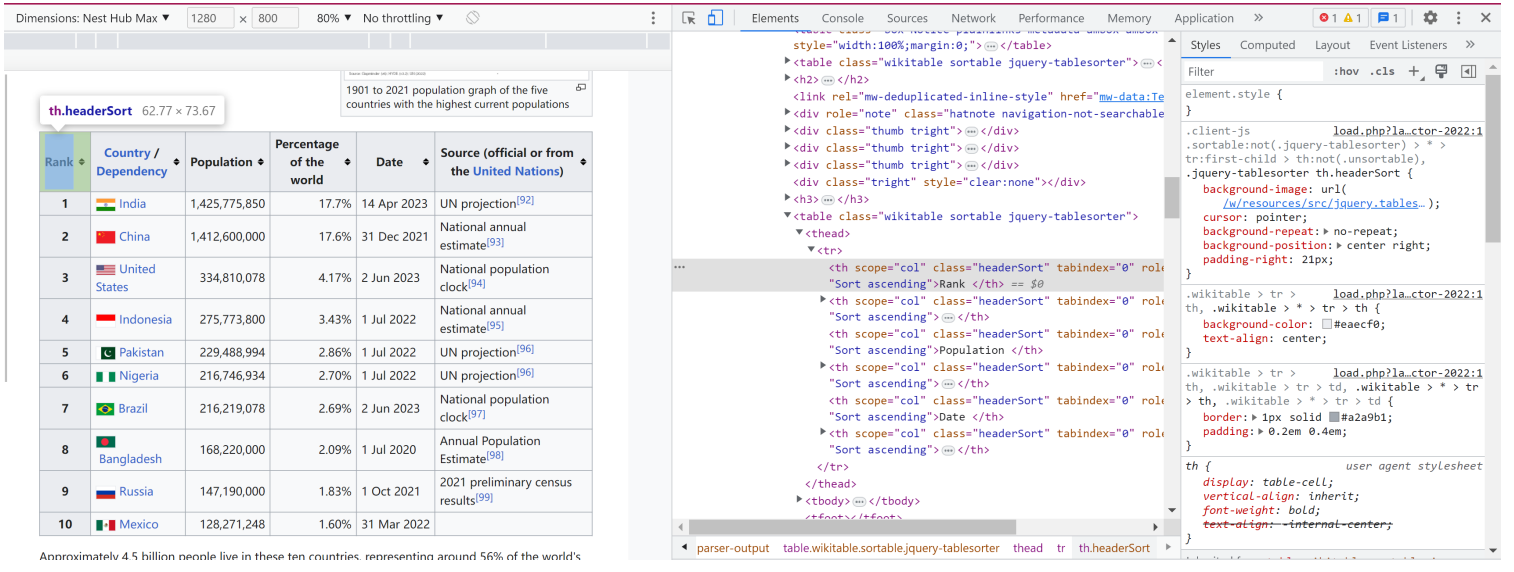
```
knitr::include_graphics("2.png", dpi = 320)
```



Rank	Country / Dependency	Population	Percentage of the world	Date	Source (official or from the United Nations)
		,850	17.7%	14 Apr 2023	UN projection[92]
		,000	17.6%	31 Dec 2021	National annual estimate[93]
		,078	4.17%	2 Jun 2023	National population clock[94]
		,800	3.43%	1 Jul 2022	National annual estimate[95]
		,994	2.86%	1 Jul 2022	UN projection[96]
		,934	2.70%	1 Jul 2022	UN projection[96]
7	Brazil	216,219,078	2.69%	2 Jun 2023	National population clock[97]
8	Bangladesh	168,220,000	2.09%	1 Jul 2020	Annual Population Estimate[98]
9	Russia	147,190,000	1.83%	1 Oct 2021	2021 preliminary census results[99]
10	Mexico	128,271,248	1.60%	31 Mar 2022	

After we click it (select the element), we will find the **inspect** penal automatically navigated to the source code of the table. That grey row is the code of that element inside the table.

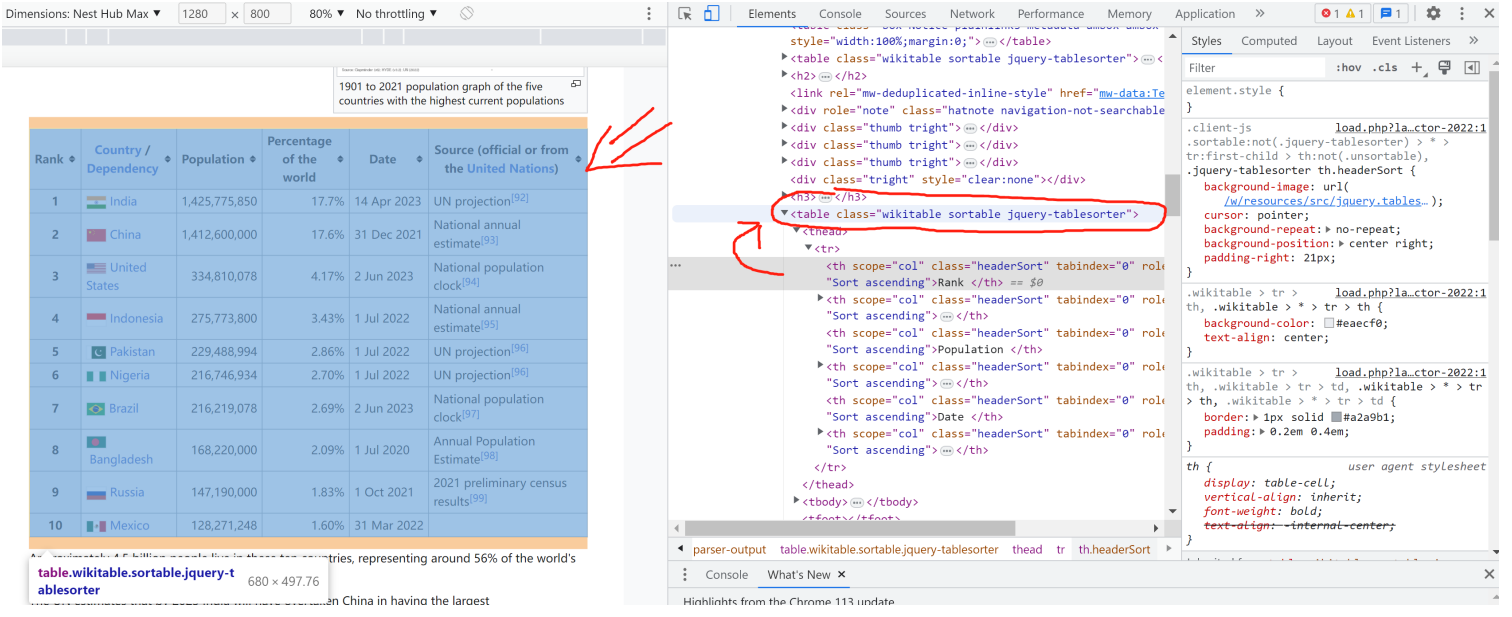
```
knitr::include_graphics("3.png", dpi = 320)
```



> Remark: Once you select the element, the element select mode will be deactivated. If you want to select another table, you need to activate it again.

Then we hover the mouse to find the table element. While hovering, the corresponding element will be marked colored.

```
knitr::include_graphics("4.png", dpi = 320)
```



Now congratulation, you find the table! it's class name is **wikitable sortable jquery-tablesorter**. Double click the class name you can copy that.

Now with this class name **wikitable sortable jquery-tablesorter**, we can narrow down our search range.

Basically, we need to replace space by ., and then apply the class name into the search inside **html_elements**.

```
webpage %>% html_elements(".wikitable.sortable.jquery-tablesorter")
```

```
{xml_nodeset (0)}
```

But this will give you no results. This is because when we use browser, data is loaded dynamically while in **rvest::read_html** in webpage <- read_html(url), it is not. Now when we use this search:

```
webpage %>% html_elements(".wikitable.sortable") # You need to replace space by .
```

```
{xml_nodeset (7)}
[1] <table class="wikitable sortable">\n<caption>Population by region (2020 e ...
[2] <table class="wikitable sortable"><tbody>\n<tr>\n<th scope="col">Rank\n</ ...
[3] <table class="wikitable sortable" style="text-align:right">\n<caption>10  ...
[4] <table class="wikitable sortable" style="text-align:right">\n<caption>Cou ...
[5] <table class="wikitable sortable">\n<caption>Global annual population gro ...
[6] <table class="wikitable sortable" style="font-size:97%; text-align:right; ...
[7] <table class="wikitable sortable" style="font-size:97%; text-align:right; ...
```

There are 7 tables now, we need to use trail and error to find the correct one.

To access the **i** th table, we can use

html_elements(search) %>% [[(i) %>% html_table()

to obtain the table result.

```
webpage %>% html_elements(".wikitable.sortable") %>% [[(2) %>% html_
```

```
# A tibble: 10 x 6
    Rank Country / Dependency Population    Percentage  of the world Date
   <int> <chr>                <chr>         <chr>                        <chr
 1     1 India                1,425,775,850 17.7%                        14 A
 2     2 China                1,412,600,000 17.6%                        31 D
 3     3 United States        334,810,078   4.17%                        2 Ju
 4     4 Indonesia            275,773,800   3.43%                        1 Ju
 5     5 Pakistan             229,488,994   2.86%                        1 Ju
 6     6 Nigeria              216,746,934   2.70%                        1 Ju
 7     7 Brazil               216,219,078   2.69%                        2 Ju
 8     8 Bangladesh           168,220,000   2.09%                        1 Ju
 9     9 Russia               147,190,000   1.83%                        1 Oc
10    10 Mexico               128,271,248   1.60%                        31 M
# i 1 more variable: Source (official or from the United Nations) <chr>
```

We find the desired table by trial and error of the **i**!

Summary of Wikipedia web scraping

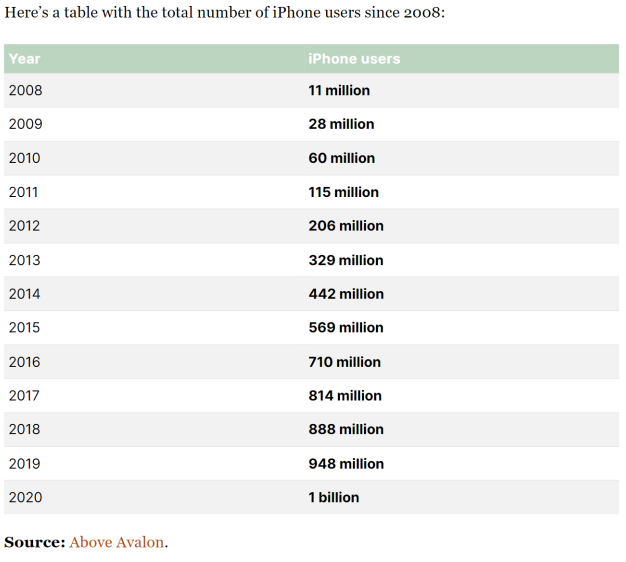
Now summary all of the procedure:

1. obtain the **url** (website link)
2. obtain the full webpage.
3. use **inspector** to get the search **".wikitable.sortable"** and then trial and error find the table **2**.

```
library(rvest)
url <- "https://en.wikipedia.org/wiki/World_population"
webpage <- read_html(url)
table = webpage %>%
  html_elements(".wikitable.sortable") %>%
  [[(2) %>% html_table()
```

Now let's try another website: https://backlinko.com/iphone-users

```
knitr::include_graphics("5.png", dpi = 400)
```



Here's a table with the total number of iPhone users since 2008:

Year	iPhone users
2008	**11 million**
2009	**28 million**
2010	**60 million**
2011	**115 million**
2012	**206 million**
2013	**329 million**
2014	**442 million**
2015	**569 million**
2016	**710 million**
2017	**814 million**
2018	**888 million**
2019	**948 million**
2020	**1 billion**

Source: Above Avalon.

```
url <- "https://backlinko.com/iphone-users"
webpage <- read_html(url)
table = webpage %>%
  html_elements(???) %>%
  html_table()  %>%
  [[(???)
head(table)
```

```r
url <- "https://backlinko.com/iphone-users"
webpage <- read_html(url)
table = webpage %>%
  html_elements(".table.table-primary.table-striped.table-hover") %>%
  html_table()  %>%
  [[(1)
head(table)
```

```
# A tibble: 6 x 2
   Year iPhone users
  <int> <chr>
1  2008 11 million
2  2009 28 million
3  2010 60 million
4  2011 115 million
5  2012 206 million
6  2013 329 million
```

Please try it yourself for more website tables!

> Remarks: Sometime the website will deny our access because they don't want robots to get the data! You will get HTTP error 403 when you use **read_html(url)**

Process the data and generate plots

The data you scrap may need further processing: Look at the wikipedia table, what type are the Population, Percentage and Date?

```
url <- "https://en.wikipedia.org/wiki/World_population"
webpage <- read_html(url)
table = webpage %>%
  html_elements(".wikitable.sortable") %>%
  [[(2) %>% html_table()
glimpse(table)
```

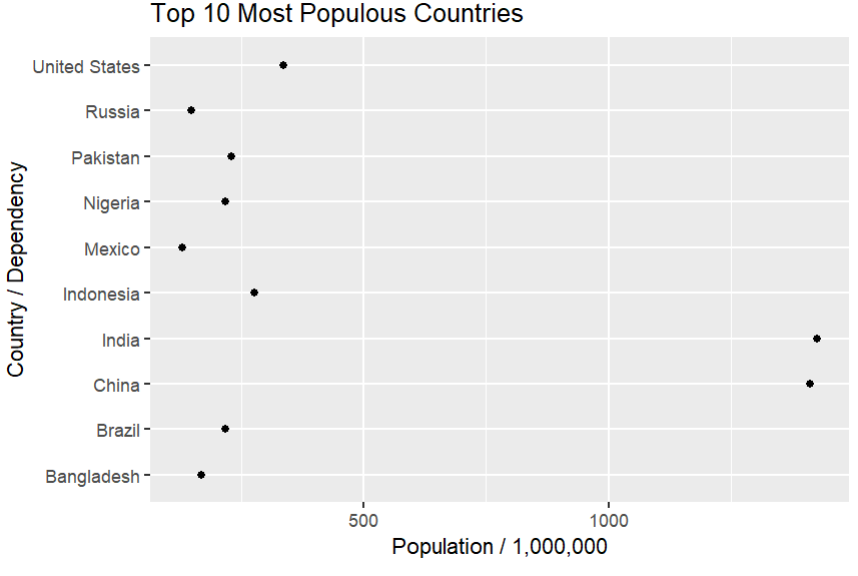
```
Rows: 10
Columns: 6
$ Rank                                          <int> 1, 2, 3, 4, 5, 6, 7, 8
$ Country / Dependency                          <chr> "India", "China", "Unit
$ Population                                    <chr> "1,425,775,850", "1,41
$ Percentage  of the world                      <chr> "17.7%", "17.6%", "4.17
$ Date                                          <chr> "14 Apr 2023", "31 Dec
$ Source (official or from the United Nations)  <chr> "UN projection[92]", "N
```

They are all Chr (Strings), not the desired data format.

So we need to reformat the data: Here **gsub(",","",Population)** means change all , into nothing in the column Population.

You need to use **Country / Dependency** to choose the column including spaces. (GRAVE ACCENT) is the key of similar sign. (The code here cannot show it, check it in source code!)

```
table %>%
  mutate(Population=gsub(",","",Population)) %>%
  mutate(Population=round(as.numeric(Population)/1e+06))  %>%
  ggplot(aes(x= Country / Dependency ,y=Population)) +
  geom_point() +
  labs(y = "Population / 1,000,000") + coord_flip() +
  ggtitle("Top 10 Most Populous Countries")
```



How to do with the **Percentage of the world**? We need to replace **%** into nothing, and then call **as.numeric**, and then divide by 100.

> Percentage of the world "17.7%", "17.6%", "4.17%", "3.43%", "2.86%", "2.70%", "2.69%", "

```
t1 = table %>%
  mutate(Percentage=gsub("%","",Percentage  of the world)) %>%
  mutate(Percentage=as.numeric(Percentage)/1e+02) %>%
  dplyr::select(Country / Dependency,Percentage )
head(t1)
```

```
# A tibble: 6 x 2
  Country / Dependency Percentage
  <chr>                       <dbl>
1 India                       0.177
2 China                       0.176
3 United States               0.0417
4 Indonesia                   0.0343
5 Pakistan                    0.0286
6 Nigeria                     0.027
```

```
#Now add others and remaining percentage
t2 = t1 %>% add_row(Country / Dependency = "others",
                    Percentage = 1 - sum(t1$Percentage))
```

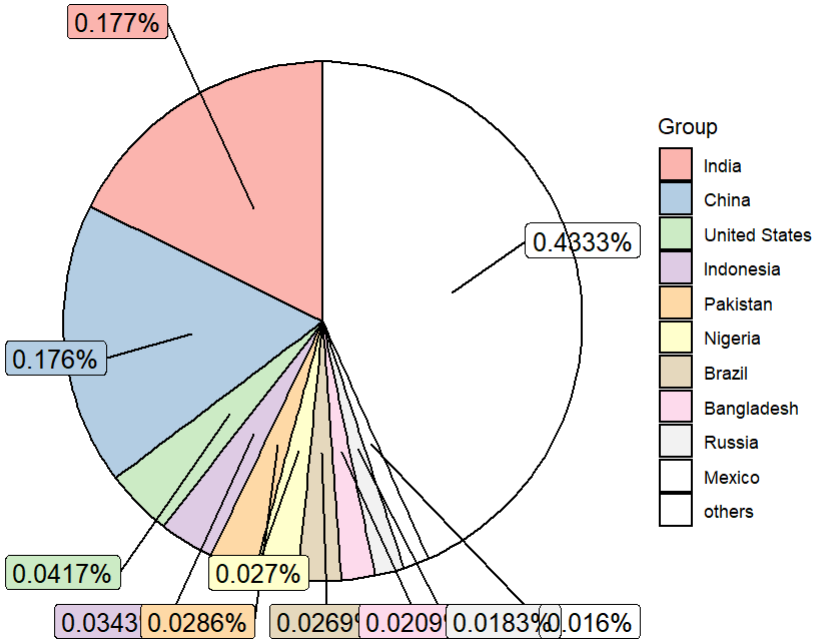
And then we can generate the pie chart using the **t2**: pie chart reference

code plot

```
library(ggplot2)
library(ggrepel)
library(tidyverse)
# Get the positions
t2 <- t2 %>%
  mutate(csum = rev(cumsum(rev(Percentage))),
         pos = Percentage/2 + lead(csum, 1),
         pos = if_else(is.na(pos), Percentage/2, pos))

ggplot(t2, aes(x = "" , y = Percentage, fill = fct(Country / Depende
  geom_col(width = 1, color = 1) +
  coord_polar(theta = "y") +
  scale_fill_brewer(palette = "Pastel1") +
  geom_label_repel(data = t2,
                   aes(y = pos, label = paste0(Percentage, "%")),
                   size = 4.5, nudge_x = 1, show.legend = FALSE) +
  guides(fill = guide_legend(title = "Group")) +
  theme_void()
```

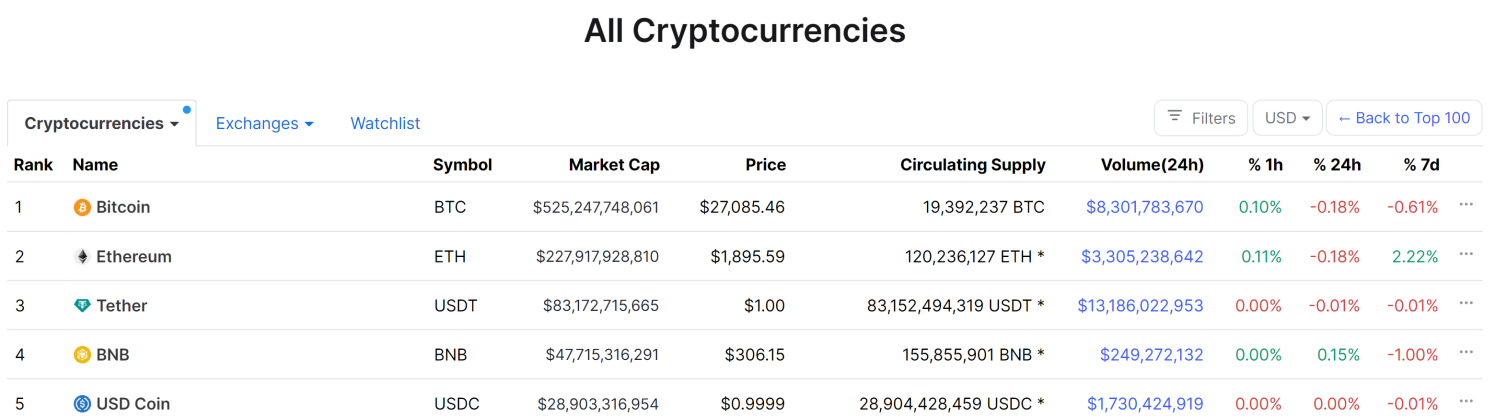
And then we can generate the pie chart using the **t2**: pie chart reference

code | plot



Some other examples: BitCoin Prices

```
knitr::include_graphics("6.png", dpi = 400)
```



All Cryptocurrencies

Cryptocurrencies | Exchanges | Watchlist | Filters | USD | ← Back to Top 100

Rank	Name	Symbol	Market Cap	Price	Circulating Supply	Volume(24h)	% 1h	% 24h	% 7d
1	Bitcoin	BTC	$525,247,748,061	$27,085.46	19,392,237 BTC	$8,301,783,670	0.10%	-0.18%	-0.61%
2	Ethereum	ETH	$227,917,928,810	$1,895.59	120,236,127 ETH *	$3,305,238,642	0.11%	-0.18%	2.22%
3	Tether	USDT	$83,172,715,665	$1.00	83,152,494,319 USDT *	$13,186,022,953	0.00%	-0.01%	-0.01%
4	BNB	BNB	$47,715,316,291	$306.15	155,855,901 BNB *	$249,272,132	0.00%	0.15%	-1.00%
5	USD Coin	USDC	$28,903,316,954	$0.9999	28,904,428,459 USDC *	$1,730,424,919	0.00%	0.00%	-0.01%

The challenge here is that it's all one big table and it's not clear how to address it.

```
url <- "https://coinmarketcap.com/all/views/all/"
bc <- read_html(url)

bc_table <- bc %>%
  html_nodes('.cmc-table__table-wrapper-outer') %>%
  html_table() %>% .[[3]]
bc_table = bc_table[,1:10]
 str(bc_table)
```

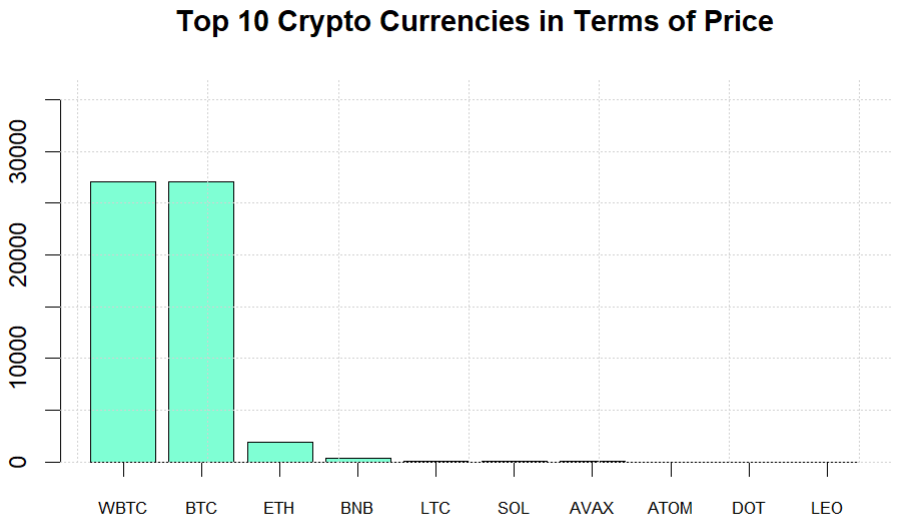
Everything is a character at this point so we have to go in an do some surgery on the data frame to turn the Price into an actual numeric.

```
# The data is "dirty" and has characers in it that need cleaning
bc_table <- bc_table %>% mutate(Price=gsub("\\$","",Price))
bc_table <- bc_table %>% mutate(Price=gsub(",","",Price))
bc_table <- bc_table %>% mutate(Price=round(as.numeric(Price),2))

# There are four rows wherein the Price is missing NA
bc_table <- bc_table %>% filter(complete.cases(bc_table))

# Let's get the Crypto currencies with the Top 10 highest prices
top_10 <- bc_table %>% arrange(desc(Price)) %>% head(10)
top_10
```

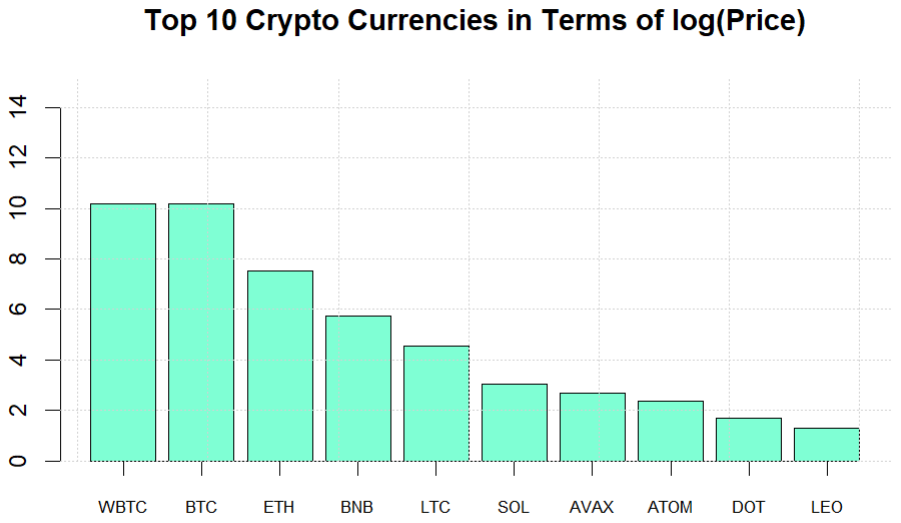
```
# A tibble: 10 x 10
    Rank Name       Symbol Market Cap   Price Circulating Supply Volume(24h)
   <int> <chr>      <chr>  <chr>          <dbl> <chr>                 <chr>
 1    18 WBTCWrap~ WBTC   $4.24B$4,24~ 2.71e4 156,628 WBTC *        $79,469,25
 2     1 BTCBitco~ BTC    $524.75B$52~ 2.71e4 19,392,237 BTC        $8,319,155
 3     2 ETHEther~ ETH    $227.84B$22~ 1.89e3 120,236,127 ETH *     $3,296,557
 4     4 BNBBNB    BNB    $47.71B$47,~ 3.06e2 155,855,901 BNB *     $247,579,8
 5    12 LTCLitec~ LTC    $6.95B$6,95~ 9.51e1 73,068,964 LTC        $501,407,5
 6     9 SOLSolana SOL    $8.46B$8,46~ 2.13e1 396,969,183 SOL *     $149,702,7
 7    16 AVAXAval~ AVAX   $5.04B$5,03~ 1.46e1 344,209,869 AVAX *    $89,801,03
 8    19 ATOMCosm~ ATOM   $3.73B$3,73~ 1.08e1 346,608,690 ATOM *    $47,682,73
 9    13 DOTPolka~ DOT    $6.36B$6,35~ 5.34e0 1,190,859,197 DOT *   $81,883,83
10    20 LEOUNUS ~ LEO    $3.34B$3,33~ 3.59e0 930,208,882 LEO *     $425,000
# i 3 more variables: % 1h <chr>, % 24h <chr>, % 7d <chr>
```

Let's make a barplot of the top 10 crypto currencies.

```
# Next we want to make a barplot of the Top 10
ylim=c(0,max(top_10$Price)+10000)
main="Top 10 Crypto Currencies in Terms of Price"
bp <- barplot(top_10$Price,col="aquamarine",
              ylim=ylim,main=main)
axis(1, at=bp, labels=top_10$Symbol,  cex.axis = 0.7)
grid()
```



```r
# Let's take the log of the price
ylim=c(0,max(log(top_10$Price))+5)
main="Top 10 Crypto Currencies in Terms of log(Price)"
bp <- barplot(log(top_10$Price),col="aquamarine",
              ylim=ylim,main=main)
axis(1, at=bp, labels=top_10$Symbol,  cex.axis = 0.7)
grid()
```

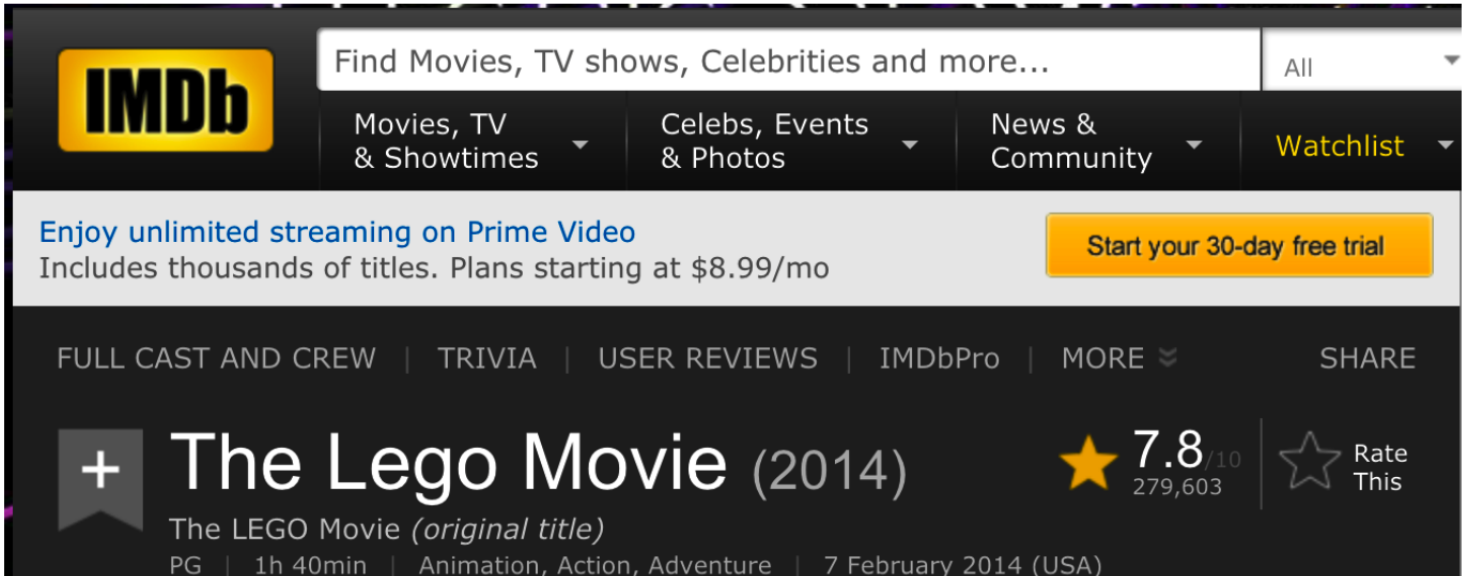


Scraping Via Xpath example: IMDB

Look at this example from IMDb (Internet Movie Database). According to Wikipedia: IMDb (Internet Movie Database) is an online database of information related to films, television programs, home videos, video games, and streaming content online – including cast, production crew and personal biographies, plot summaries, trivia, fan and critical reviews, and ratings.

We can search or refer to specific movies by URL if we wanted.

For example, consider the following link to the “Lego Movie”:
http://www.imdb.com/title/tt1490017/

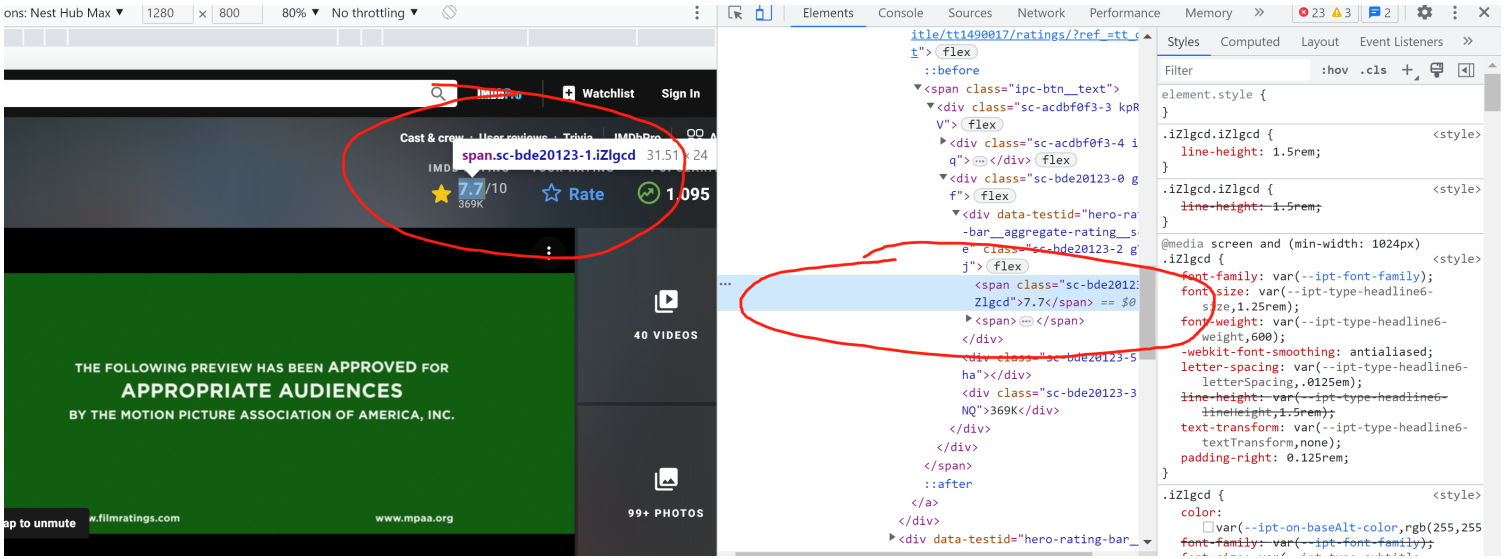


Let’s say that we wanted to capture the rating information.

```
url <- "http://www.imdb.com/title/tt1490017/"
lego_movie <- read_html(url)
```

Now let's go to the inspector and find the element:

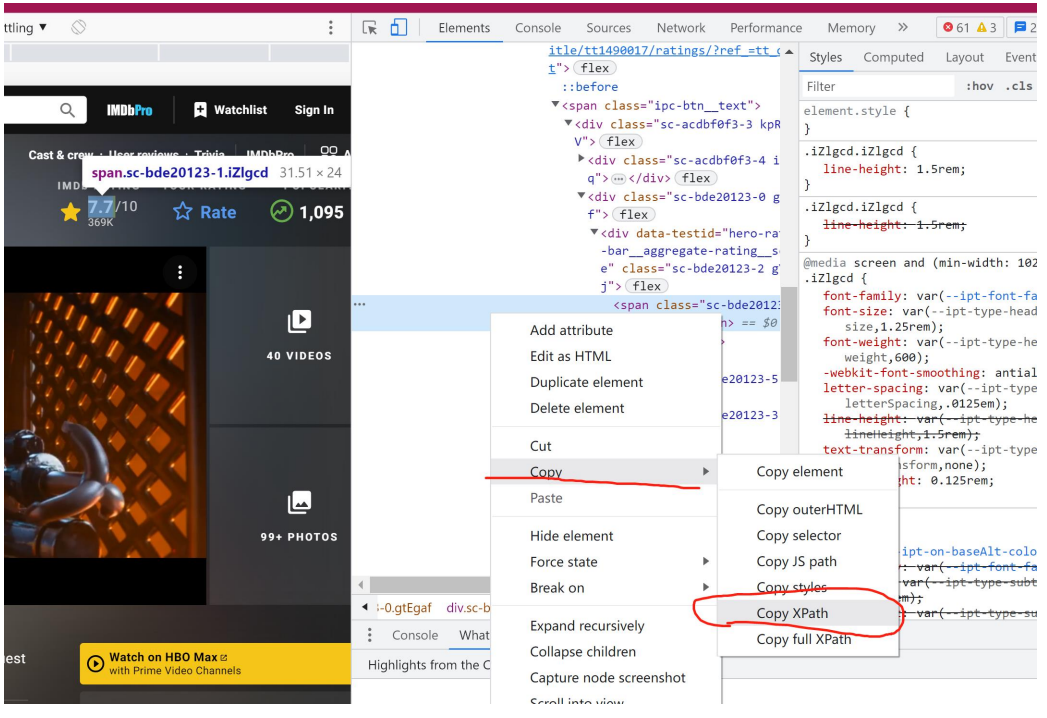
```
knitr::include_graphics("7.png", dpi = 400)
```



Now we need to obtain the **Xpath**.

Now we need to obtain the **Xpath**. To do so, right click on the selected line, and choose Copy->Copy Xpath.

```
knitr::include_graphics("9.jpg", dpi = 400)
```



Then you will have it.

Now copy that Xpath into your R code: Remember use " (single quotation) to include the Xpath (it will be all blue), and then assign it to xpath.

```
xpath='//*[@id="__next"]/main/div/section[1]/section/div[3]/section/s

rating <- lego_movie %>%
   html_nodes(xpath = xpath)  %>%
  html_text()
rating
```

```
[1] "7.7"
```

Now you are done! Then let's access the summary section of the link.

```
xpath='//*[@id="__next"]/main/div/section[1]/section/div[3]/section/s
mov_summary <- lego_movie %>%
  html_nodes(xpath=xpath) %>%
  html_text()

cat(mov_summary)
```

```
An ordinary LEGO construction worker, thought to be the prophesied as "specia
```