# Anweisungen zur Verwendung des Codes

Please follow the instructions below to use the R script successfully on your laptop…

1. Open the **moevenpick\_wine\_scraping.Rmd** file
2. The script consists of **3 parts**
   * **Part 1:** Loading the required libraries (**rvest**, **httr**, **dplyr**, and **stringr**)
     + **“rvest”** and **“httr”** are used to send GET requests to the website and crawl the HTML content
       - Crawling data requires the use of **selectors**. Use these two cheat sheets for an overview of [CSS](https://www.scraperapi.com/blog/css-selectors-cheat-sheet/) and [XPath](https://www.lambdatest.com/blog/most-exhaustive-xpath-locators-cheat-sheet/) selectors
     + **“dplyr”** is used to manipulate the crawled data and put it in a data frame
     + **“stringr”** is used to handle the clean and format the crawled data using **regular expressions (regex)**
       - To try out regex patterns and see how they extract specific parts of a string, use this [website](https://regex101.com/)
       - The second page of this [PDF](https://evoldyn.gitlab.io/evomics-2018/ref-sheets/R_strings.pdf) contains a handy cheat sheet for regex
   * **Part 2:** Creating the scraping function and defining the CSS/XPath selectors of the data points we want to extract
   * **Part 3:** Defining a function that randomly generates time values between 2 and 4 seconds. These values will be used to throttle the GET requests to prevent our scraper from getting blocked
   * **Part 4:** Executing the scraping function defined in step 2, looping over all **91 pages**
3. Run each chunk of code. You should **not** get any errors. Please contact me if you face any issues
4. After the 4th step is executed, a data frame similar to the one shown below will be produced

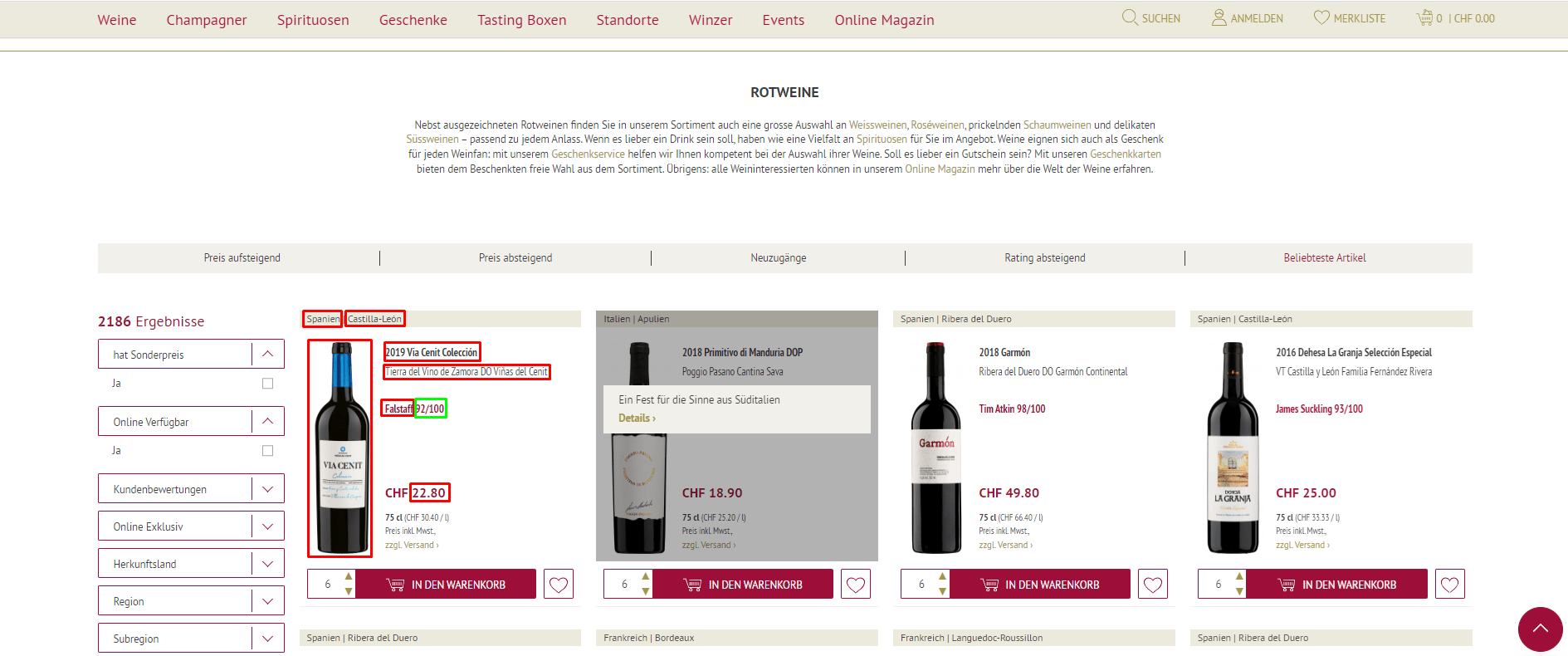
# 

# Die Beschreibung des Projekts

## The Goal of the Project

The goal of the project is to crawl this [website](https://www.moevenpick-wein.com/de/rotweine?p=1) and extract the data points shown below.

* product\_title
* product\_name
* product\_url
* rating\_score (out of 100 or 20)
* reviewer (could be a person, a magazine, or simple a displayed "Score")
* country
* city
* price (in CHF)
* image\_url



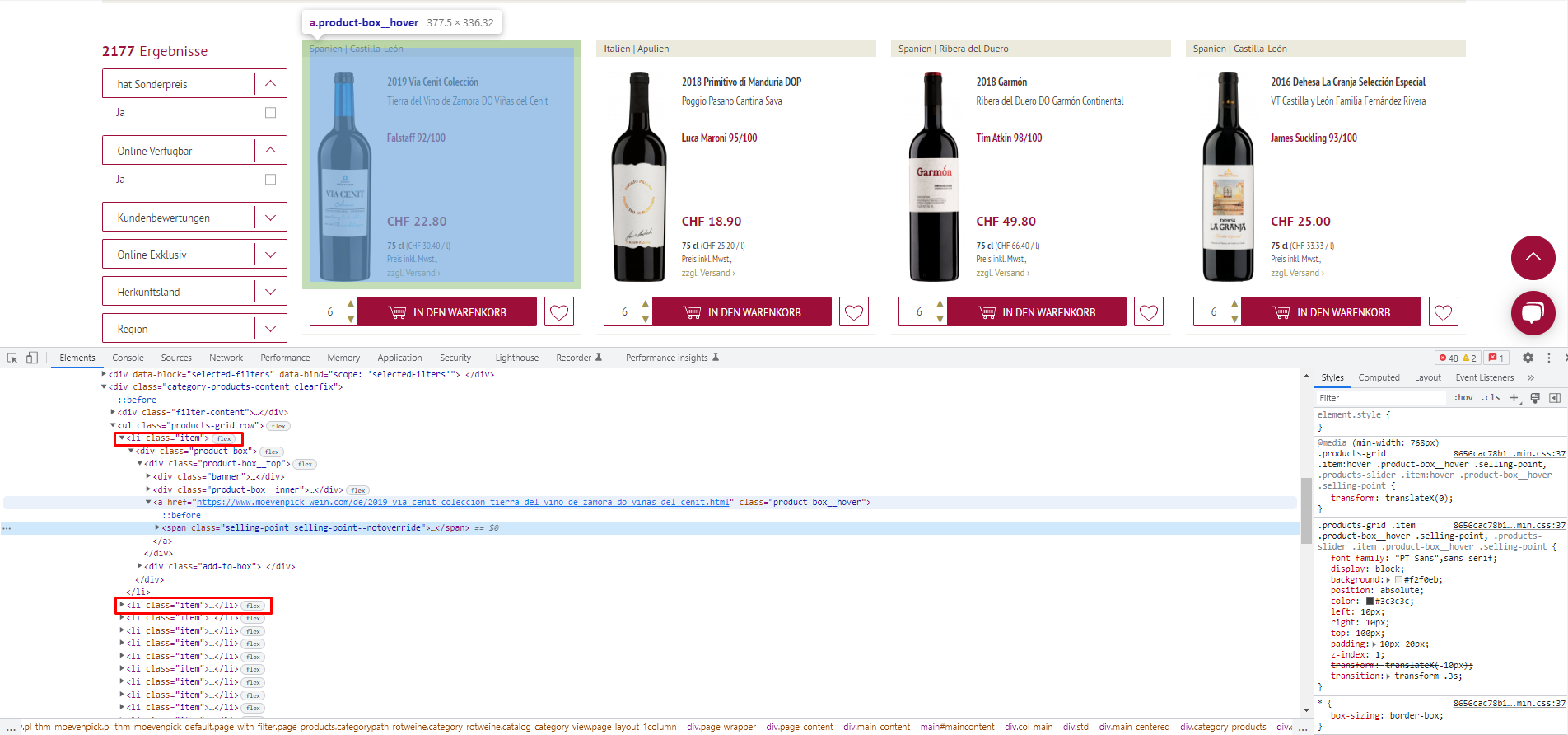
## Analysis of the Website

The website is simple to crawl as it does **not** use Javascript to render its content and does not employ sophisticated anti-bot mechanisms. Therefore, The **“rvest”** package in R is sufficient to crawl the content of the website. In addition to the **“rvest”** package, we use the **“dplyr”** and **“stringr”** libraries to **wrangle** and **clean** the crawled data.

The URL `https://www.moevenpick-wein.com/de/rotweine` returns **2177 results**. There are **24 results** on each page, which means there are ~ **91 pages to scrape**. The website is \*\*paginated\*\* `https://www.moevenpick-wein.com/de/rotweine**?p=1**`, meaning we can create a \*\*for\*\* loop to scrape the results from every page by changing the numeric parameter at the end of the URL.

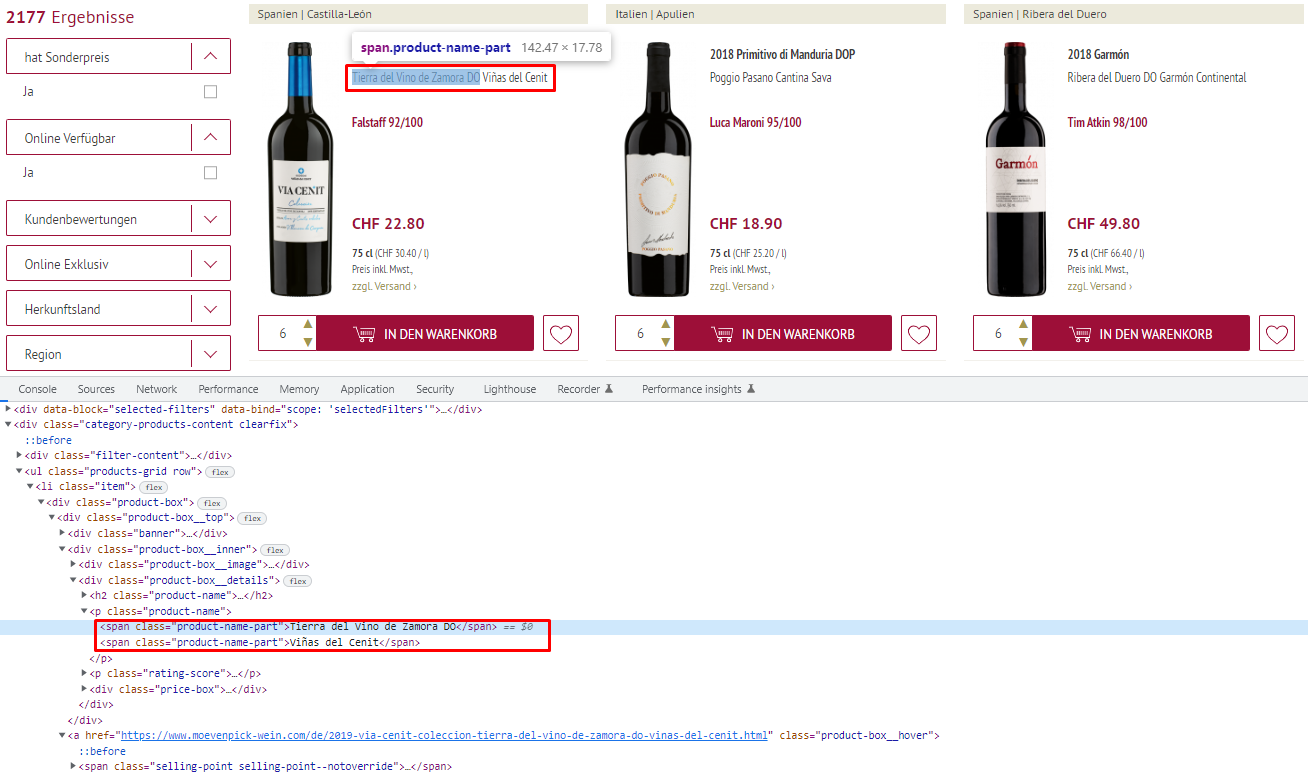
## Crawling Process

We start by defining the CSS/XPath selectors of **each data point** we want to crawl. The website is well-structured, so every item is stored in a **“list”** with a class **“item,”** as shown below.



The CSS/XPath selectors of the nine data points are as follows:

* **product\_title**
  + **span.product-name-1 → “**span” with a class “product-name-1”
* **product\_name**
  + The product name is composed of **two parts**, so we extract them separately and combine them into one variable afterward

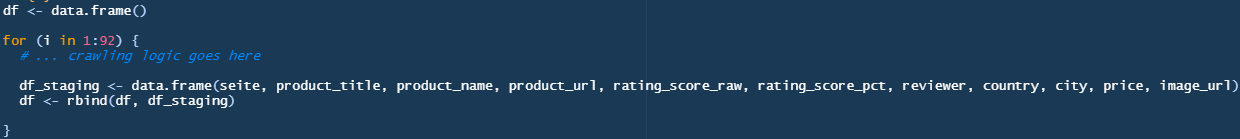
****

* + **p.product-name > span.product-name-part:first-child →** The CSS selector of the first part of the product\_name. “p” with a class “product-name” AND a **first child** of “span” with class “product-name”
  + **p.product-name > span.product-name-part:nth-child(2) →** The CSS selector of the second part of the product\_name. “p” with a class “product-name” AND a **second child** of “span” with class “product-name”
  + After extracting both parts separately, we combine them into one variable using the **paste0** function → **paste0(product\_name\_p1, " ", product\_name\_p2)**
* **product\_url**
  + **h2.product-name > a %>% html\_attr(“href”) →** “h2” with a class “product-name” AND a child “a”. Since this is a URL, we extract the “href” attribute using the **“html\_attr”** method in R
* **rating\_score and reviewer**
  + The rating score is a composite string that consists of **two parts**
  + **Part 1** is the person/magazine that reviewed the wine bottle
  + **Part 2** is the score **out of 100** or **20**
  + In the example below, the person who reviewed the wine is “Tim Atkin”. He gave the bottle a score of 98/100
  + Since the score can be out of 100 or 20, we crawl the raw score in **string format** and also the **individual components** to calculate a percentage (e.g., 98/100 = 0.98 or 15/20 = 0.75)

****

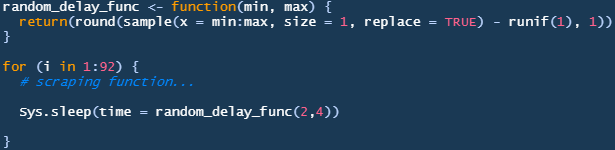
* + The CSS selector of the **composite review (reviewer + score)** is **p.rating-score →** “p” with class “rating-score”
  + **Part one** can be extracted using this regex → **"[a-zA-Z]+"**. This regex extracts any characters from A-Z in the string (lowercase or uppercase)
  + **Part two** as a whole can be extracted using this regex → **"\\d+\\/\\d+"**. This regex extracts any part of the string that matches this pattern **"number/number"**
  + The **left part of the raw score** (before the division sign) can be extracted using this regex → **"\\d+(?=\\/\\d+)"**. This regex extracts **any digit** \*\*before\*\* a division sign that is **followed by numbers**
  + The **right part of the raw score** (after the division sign) can be extracted using this regex → **"(?<=\\/)\\d+"**. This regex extracts **any digit** \*\*after\*\* the division sign
  + To check how these regular expressions work, one can use this [website](https://regex101.com/). Please note that in R, a double backslash is required. On this website, only one backslash is used
* **country & city**
  + The country and city are displayed together and separated by pipe “|”
  + **p.cellar-name** is the CSS selector of this composite string → **“p”** of class **“cellar-name”**
  + To extract the **country**, one can use this regex → **"\\w+(?=\\s\\|)"**. It extracts any word characters \*\*before\*\* **" |"**
  + To extract the **city**, one can use this regex → **"(?<=\\|\\s)\\w+"**. It extracts any word characters \*\*after\*\* **"| "**
* **price**
  + The XPATH selector of price is **//span[@data-price-type = 'finalPrice']/span** → “span” with attribute **data-price-type = ‘finalPrice’** AND a “span” child
  + price is displayed as a composite string with the currency symbol (e.g., CHF 950.00)
  + In addition, wines that have 4-digit prices are displayed with an **apostrophe** (e.g., CHF 1,150.00)
  + To handle the first case, we can use this regex → **"(?<=CHF\\s).\*"**. It extracts any alphanumeric character \*\*after\*\* (CHF )
  + To handle the second case, we can use the **“str\_replace”** method to replace the apostrophe with a blank character
* **image\_url**
  + The CSS selector of image\_url is **img.product-image-photo →** “img” with class “product-image-photo”
  + The URL of the image is stored under an attribute called “src”. We can extract it with the **“html\_attr”** method

After extracting these data points, we can place them in a data frame using the **data.frame** method in R. Since we loop over each page, we can define an **empty data frame** before the “for” loop. This data frame will grow over time because we will append the results of each crawling iteration to it. We can use a **temporary data frame within the “for” loop** to store the results of iteration “i’s”, and then bind the results to the data frame we created before the “for” loop. In code form, this looks something like this…



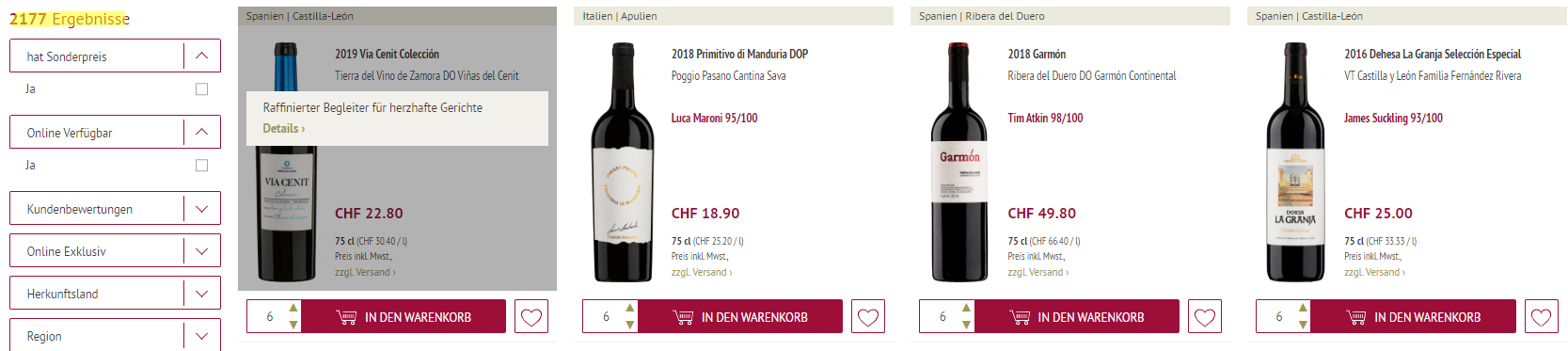
## Throttling Our GET Requests

To **stay polite** to the server, we need to throttle our GET requests to prevent our scraper from getting blocked. We can create a function to produce **random time values** and use the **Sys.sleep()** function to **slow down the scraper** in a random fashion. In code form, this looks something like this…



## Further Automating the Scraper

To fully automate the scraper, we can extract the **last page number** from the first page and set it as the end of the “for” loop’s range. The first page does not display the last page number, but rather the **total number of products,** as shown below.



Since there are 24 results on each page, we can calculate the total number of pages using this formula → ceiling(last\_page “2177” / 24) = 91

The CSS selector of this number is **“div.filter-results-count > strong”** → “div” with class “filter-results-count” AND child “strong”

## Bonus Part

The dataset contains the “price” and “country” fields, we can use these two fields to construct a price histogram per country, as shown below. The code is included in the last section of the .Rmd file

