# Introduction to Webscraping

Theresa Gessler

July 2019

# Plan of this afternoon

- general introduction to webscraping
- we will focus on *static pages* without interactivity
- we work with different exercise files, depending on your level

# What is webscraping?

- extracting data from webpages
  - anything from the summer school page to social media
  - lots of different techniques
- why should you scrape?
  - masses of data
  - reproducible and renewable data collection
  - once you learned it: simpler

# Experiences with webscraping?

Do you have. . .

- experience with HTML?
- experience with scraping?
- any ideas that involve scraping?

# How To

## Three packages

- rvest
  - our tool for scraping
- learnr
  - our tool for practicing what we learned
  - not needed for webscraping but for generating interactive HTML pages
- learn2scrape
  - package for this course that contains all exercises
  - we will install this package together

# learnr & learn2scrape

- everything we do today, including content of most of my slides, included as a tutorial
- if you do not want to code along, the final file is also included

## Steps

- install learnr package (done!?)
- install learn2scrape

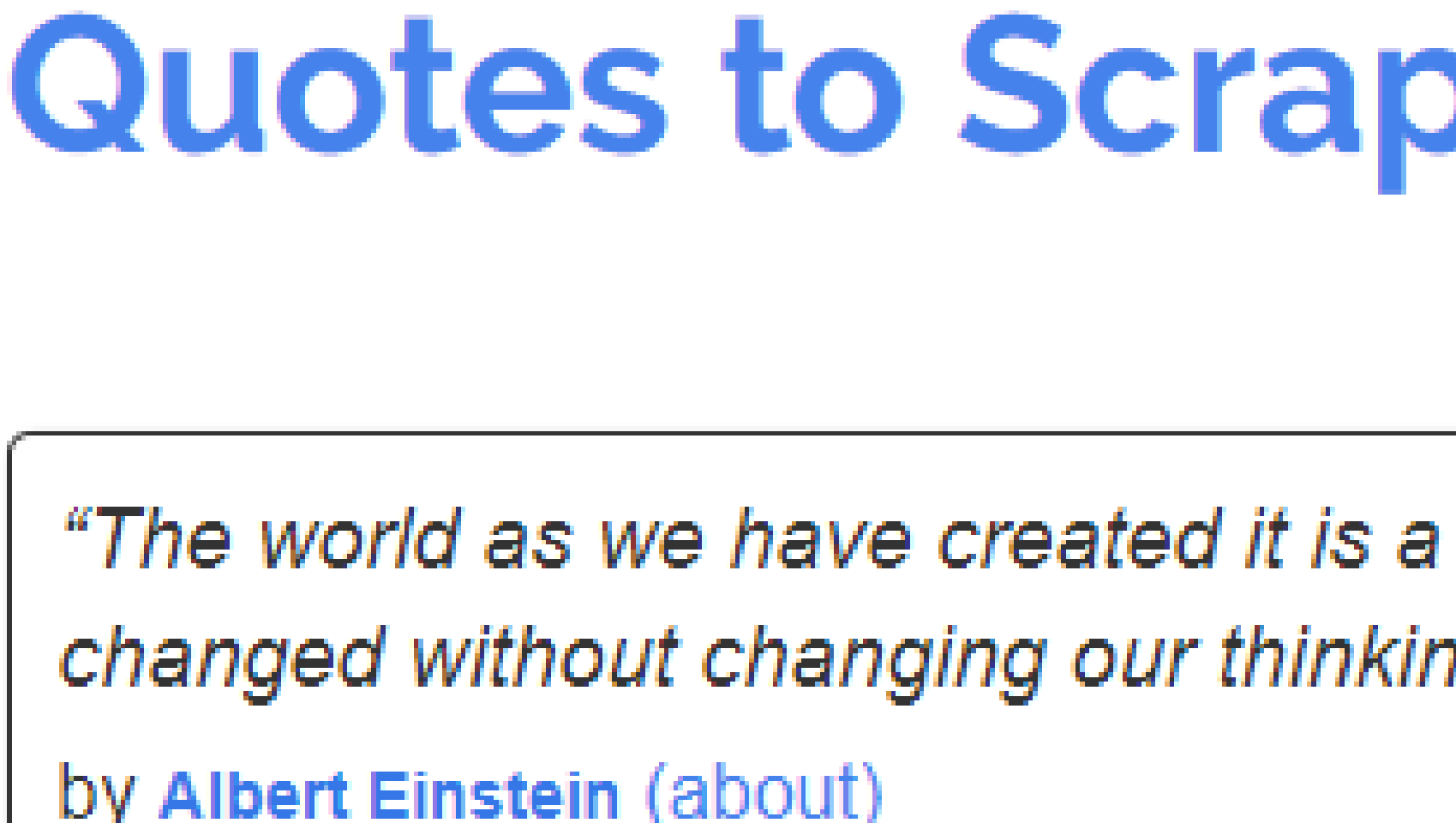
```
install.packages("path/to/tar/file", source = TRUE, repos=N
```

- run tutorial through learnr package

```
library(learnr)
run_tutorial("introduction",package="learn2scrape")
```

# HTML: The basics

- **H**yper **T**ext **M**arkup **L**anguage
  - *markup*:
- use of HTML tags to specify behaviour of text
- example page



# Example: Quotes to scrape Webpage

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Quotes to Scrape</title>
    <link rel="stylesheet" href="/static/bootst
    <link rel="stylesheet" href="/static/main.c
</head>
<body>
    <div class="container">
        <div class="row header-box">
            <div class="col-md-8">
                <h1>
                    <a href="/" style="text-dec
                </h1>
            </div>
```

# Browsing vs. scraping

- browsing
  - you click on something
  - browser sends request to server that hosts webpage
  - server returns resource (e.g. HTML document)
  - browser interprets HTML and renders it in a nice fashion
- scraping with R
  - you manually specify a resource
  - R sends request to server that hosts website
  - server returns resource
  - R parses HTML (i.e., interprets the structure), but does not render it in a nice fashion
  - you tell R which parts of the structure to focus on and what to extract

# Basics of webscraping

**Web scraping** is the process of extracting data from webpages. Normally, we do that intuitively - we scan a webpages with our eyes and read only those parts that are important to us. This is more complicated when we want R to read pages for us - R does not know what is important. So we have to tell R exactly where to look!

But don't worry, we will go step by step. Let's start with something simple. Imagine, we want to scrape this simple webpage full of quotes. Its address is http://quotes.toscrape.com/ Just have a look at the webpage.

**First**, we need to load the package we'll use for most of our scraping. It is called *rvest*. Please load it with the library() command.

**Next**, to read the page into R, we need to tell R the address - please create a character vector named url that contains the URL http://quotes.toscrape.com/

# Exercise

Now, we start by reading (sometimes called parsing) the webpage. To tell R to read the webpage, we can use the function read_html. Apply the function to the url object we just created!

The output does not look like you expected? You would be surprised to see that the original webpage looks quite similar. Have a look at the webpage source code. Depending on your browser, you can probably select 'view source' after a right mouse click. If you have trouble, google has an up-to-date explanation.

Just if you were wondering, of course, we could have also skipped creating a variable with the URL. Try to apply the read_html() function directly to the URL http://quotes.toscrape.com/ and save the result to an object called page.

The function read_html parses the html code, similar to what our browser does. Still, it gives us the entire document including the HTML commands. You can use the function html_text() to extract the Webpage text.