

Campus Luzern

A Primer to Web Scraping with R

Introduction

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Welcome

Welcome!

Thank you for your interest in the course!

I'm glad that I will have the opportunity to give you a gentle introduction to web scraping with R.

Just a few words on my personal background:

- Lecturer in Political Data Science at the Hertie School of Governance
- political scientist by training
- working with web-based data since about 2010
- what I do with web data: measure public awareness, news consumption, political behavior

Goals and outline

Goals

After attending this course, ...

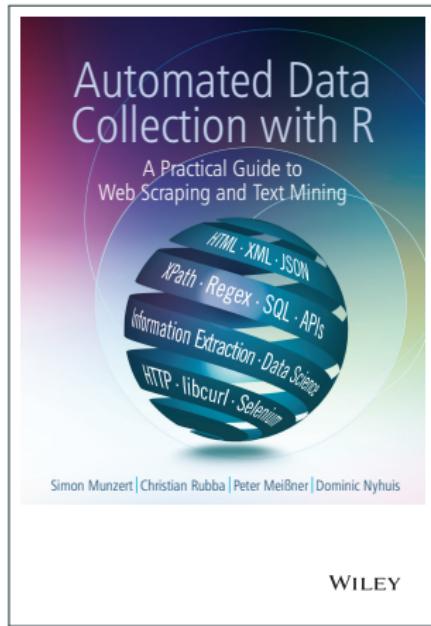
- you have acquired basic knowledge of web technologies
- you are able to scrape information from static and dynamic websites using R
- you are able to access web services (APIs) with R
- you can build up and maintain your own original sets of web-based data

Course outline

Unit	Topic
1	Introduction – setup and first steps
2	Regular expressions
3	Scraping static webpages
4	Advanced scraping of static webpages
5	Scraping dynamic webpages
6	Tapping APIs
7	Legal and ethical issues in web scraping
8	Scraping workflow and tricks of the trade

The accompanying book

- contains most of which I tell you during the course (but much more, and at times more accurate)
- written between 2012 and 2014 → not entirely up-to-date anymore, but I will provide updated material during the course
- homepage with materials: www.r-datacollection.com
- if you find any errors in the book, please let me know!



Campus Luzern

A Primer to Web Scraping with R

Overview

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Web scraping in social science research

Assessing internet censorship in China

How Censorship in China Allows Government Criticism but Silences Collective Expression

GARY KING *Harvard University*

JENNIFER PAN *Harvard University*

MARGARET E. ROBERTS *Harvard University*

We offer the first large scale, multiple source analysis of the outcome of what may be the most extensive effort to selectively censor human expression ever implemented. To do this, we have devised a system to locate, download, and analyze the content of millions of social media posts originating from nearly 1,400 different social media services all over China before the Chinese government is able to find, evaluate, and censor (i.e., remove from the Internet) the subset they deem objectionable. Using modern computer-assisted text analytic methods that we adapt to and validate in the Chinese language, we compare the substantive content of posts censored to those not censored over time in each of 85 topic areas. Contrary to previous understandings, posts with negative, even vitriolic, criticism of the state, its leaders, and its policies are not more likely to be censored. Instead, we show that the censorship program is aimed at curtailing collective action by silencing comments that represent, reinforce, or spur social mobilization, regardless of content. Censorship is oriented toward attempting to forestall collective activities that are occurring now or may occur in the future—and, as such, seem to clearly expose government intent.

Measuring political ideology with Twitter networks

Birds of the Same Feather Tweet Together: Bayesian Ideal Point Estimation Using Twitter Data

Pablo Barberá

*Wilf Family Department of Politics, New York University, 19 W 4th Street, 2nd Floor,
New York, NY 10012.*

e-mail: pablo.barbera@nyu.edu

Edited by R. Michael Alvarez

Politicians and citizens increasingly engage in political conversations on social media outlets such as Twitter. In this article, I show that the structure of the social networks in which they are embedded can be a source of information about their ideological positions. Under the assumption that social networks are homophilic, I develop a Bayesian Spatial Following model that considers ideology as a latent variable, whose value can be inferred by examining which politics actors each user is following. This method allows us to estimate ideology for more actors than any existing alternative, at any point in time and across many polities. I apply this method to estimate ideal points for a large sample of both elite and mass public Twitter users in the United States and five European countries. The estimated positions of legislators and political parties replicate conventional measures of ideology. The method is also able to successfully classify individuals who state their political preferences publicly and a sample of users matched with their party registration records. To illustrate the potential contribution of these estimates, I examine the extent to which online behavior during the 2012 US presidential election campaign is clustered along ideological lines.

Measuring inflation using online prices

The Billion Prices Project: Using Online Prices for Measurement and Research

Alberto Cavallo and Roberto Rigobon

NBER Working Paper No. 22111

March 2016, Revised April 2016

JEL No. E31,F3,F4

ABSTRACT

New data-gathering techniques, often referred to as “Big Data” have the potential to improve statistics and empirical research in economics. In this paper we describe our work with online data at the Billion Prices Project at MIT and discuss key lessons for both inflation measurement and some fundamental research questions in macro and international economics. In particular, we show how online prices can be used to construct daily price indexes in multiple countries and to avoid measurement biases that distort evidence of price stickiness and international relative prices. We emphasize how Big Data technologies are providing macro and international economists with opportunities to stop treating the data as “given” and to get directly involved with data collection.

Measuring the Importance of Political Elites^{*}

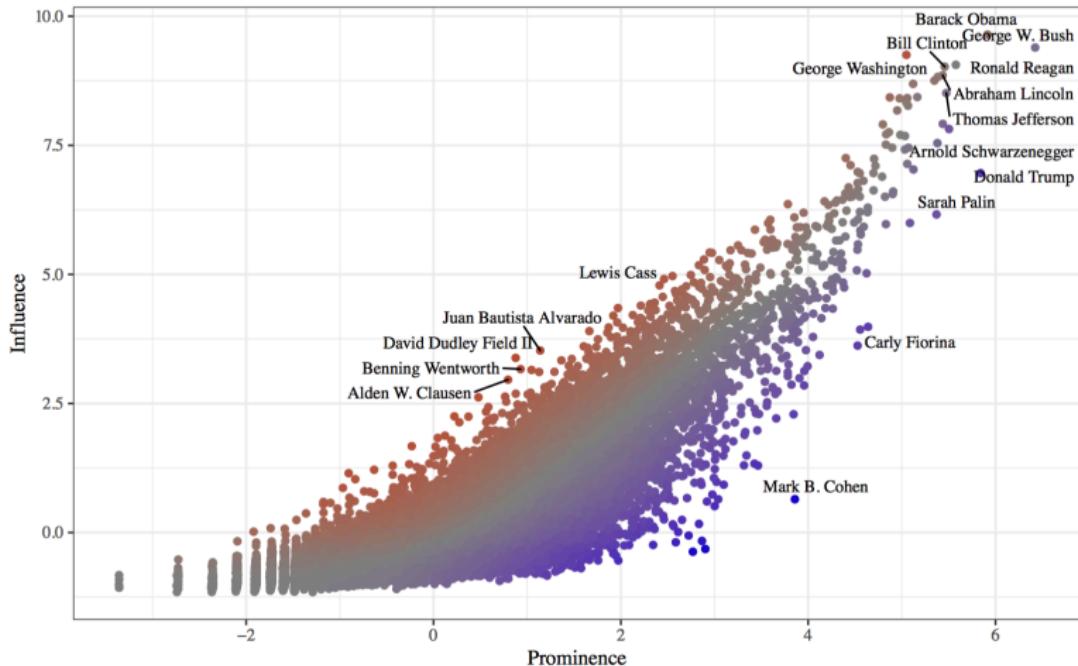
Simon Munzert (Hertie School of Governance)[†]

This version: December 14, 2018

Abstract. Political influence and prominence are both important drivers and outcomes of political careers, however are hard to quantify. I develop a method to measure these characteristics for a large set of political elites using information from the largest database of politicians—the English Wikipedia. I argue that metadata from the encyclopaedia, such as article editing, readership metrics, and linkage structures are reflective of the latent traits. A Bayesian latent variable model is employed to identify prominence and influence scores for a set of 45,000 U.S. politicians. Several validation exercises corroborate the superiority of the measures over existing alternatives. To illustrate their usefulness, I present an analysis of media appearances of political elites. The ability to measure the importance of political elites on a large scale has various implications for the understanding of political career paths, electoral performance, and legislative politics.

Measuring political importance using Wikipedia data

Figure 2: Estimated person values from the Bayesian latent variable model.



Web scraping with R

Web scraping. What? Why?

Web scraping

A.k.a. screen scraping, is the business of

- getting (unstructured) data from the web and
- bringing it into shape (e.g., clean, make tabular format)

A data analyst's view

- data abundance online
- social interaction online
- services track social behavior

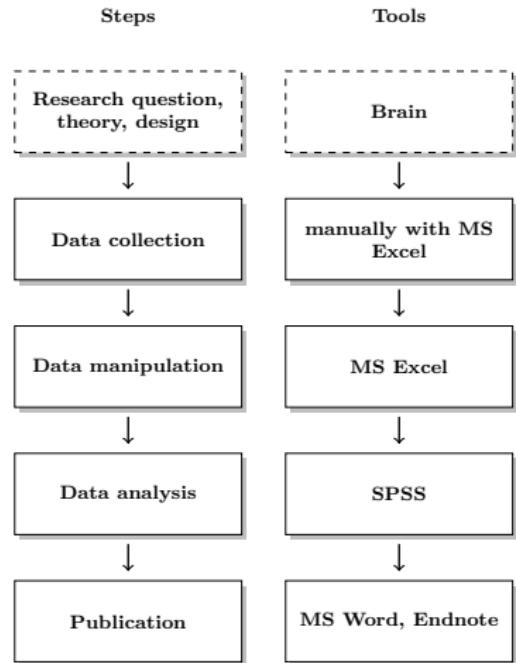
A pragmatist's view

- financial resources
- time resources
- reproducibility
- updateability

Why R?

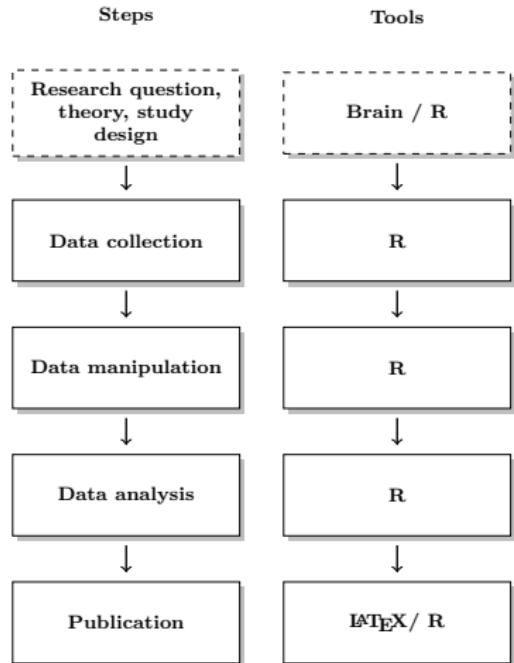
Why R?

- free
- open source
- large community
- powerful tools for statistical analysis
- powerful tools for visualization
- flexible in processing all kinds of data/languages
- useful in every step of the workflow



Why R?

- free
- open source
- large community
- powerful tools for statistical analysis
- powerful tools for visualization
- flexible in processing all kinds of data/languages
- useful in every step of the workflow

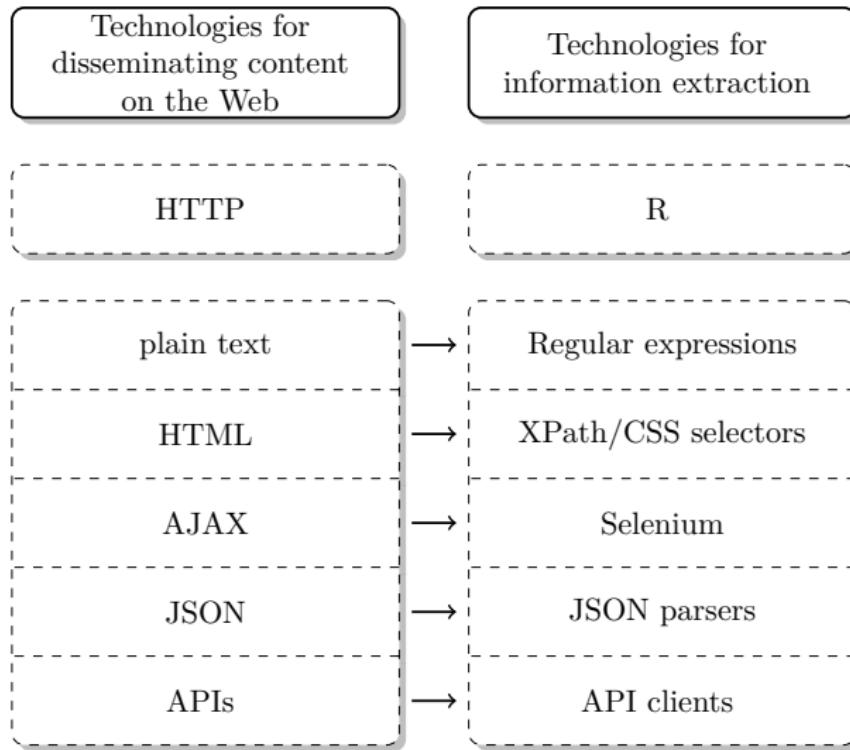


The philosophy behind web data collection with R

- no point-and-click procedure
- script the entire process from start to finish
- automation of
 - downloading
 - classical screen scraping
 - tapping APIs
 - parsing
 - data tidying, text data processing
- scaling up scraping procedures
- scheduling of scraping tasks

Technologies of the World Wide Web

Technologies of the World Wide Web



Technical setup

1. make sure that the newest version of R is installed on your computer (available here: <https://cran.r-project.org/>)
2. install the newest stable version of *RStudio Desktop* (available here: <https://www.rstudio.com/products/rstudio/download>)

Campus Luzern

A Primer to Web Scraping with R

An Introductory Case Study

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Data on the web

W Berlin - Wikipedia

Sicher https://en.wikipedia.org/wiki/Berlin

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

Berlin

From Wikipedia, the free encyclopedia

This article is about the capital of Germany. For other uses, see Berlin (disambiguation).

Berlin (berlín, German: [bɛɐ̯lɪn] (listen)) is the capital and the largest city of Germany as well as one of its constituent 16 states. With a population of approximately 3.5 million people,^[4] Berlin is the second most populous city proper and the seventh most populous urban area in the European Union.^[5] Located in northeastern Germany on the banks of rivers Spree and Havel, it is the centre of the Berlin-Brandenburg Metropolitan Region, which has about 6 million residents from more than 180 nations.^{[6][7][8][9]} Due to its location in the European Plain, Berlin is influenced by a temperate seasonal climate. Around one-third of the city's area is composed of forests, parks, gardens, rivers and lakes.^[10]

First documented in the 13th century and situated at the crossing of two important historic trade routes,^[11] Berlin became the capital of the Margravate of Brandenburg (1417–1701), the Kingdom of Prussia (1701–1918), the German Empire (1871–1918), the Weimar Republic (1919–1933) and the Third Reich (1933–1945).^[12] Berlin in the 1920s was the third largest municipality in the world.^[13] After World War II and its consequent occupation by the victorious countries, the city was divided; East Berlin became the capital of East Germany while West Berlin became a de facto West German exclave, surrounded by the Berlin Wall (1961–1989). Following German reunification in 1990,^[14] Berlin once again became the capital of Germany.

Berlin
State of Germany



https://en.wikipedia.org/wik/File:Alte_Nationalgalerie_Berlin_2011.jpg

Data on the web

W Berlin - Wikipedia

Sicher https://en.wikipedia.org/wiki/Berlin

Foreign's mayors make up the Council of mayors (*rat der burgermeister*), which is led by the city's Governing Mayor and advises the Senate. The neighborhoods have no local government bodies.

Twin towns – sister cities [edit]

See also: *List of twin towns and sister cities in Germany*

Berlin maintains official partnerships with 17 cities.^[100] Town twinning between Berlin and other cities began with its sister city Los Angeles in 1967. East Berlin's partnerships were canceled at the time of German reunification but later partially reestablished. West Berlin's partnerships had previously been restricted to the borough level. During the Cold War era, the partnerships had reflected the different power blocs, with West Berlin partnering with capitals in the Western World, and East Berlin mostly partnering with cities from the Warsaw Pact and its allies.

There are several joint projects with many other cities, such as Beirut, Belgrade, São Paulo, Copenhagen, Helsinki, Johannesburg, Mumbai, Oslo, Shanghai, Seoul, Sofia, Sydney, New York City and Vienna. Berlin participates in international city associations such as the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin's official sister cities are:^[100]

- 1967 Los Angeles, United States
- 1987 Paris, France
- 1988 Madrid, Spain
- 1989 Istanbul, Turkey
- 1991 Warsaw, Poland^[101]
- 1991 Moscow, Russia
- 1992 Brussels, Belgium
- 1992 Budapest, Hungary^[102]
- 1993 Tashkent, Uzbekistan
- 1993 Mexico City, Mexico
- 1993 Jakarta, Indonesia
- 1994 Beijing, China
- 1994 Tokyo, Japan
- 1994 Buenos Aires, Argentina
- 1995 Prague, Czech Republic^[103]
- 2000 Windhoek, Namibia
- 2000 London, United Kingdom

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The [President of Germany](#), whose functions are mainly ceremonial under the [German constitution](#), has his official residence in [Schloss Bellevue](#).^[104] Berlin is the seat of the [German executive](#), housed in the [Chancellery](#), the [Bundeskanzleramt](#). Facing the Chancellery is the [Bundestag](#), the German Parliament, housed in the renovated Reichstag building since the government relocated to Berlin in 1998. The [Bundesrat](#) ("federal council", performing the function of an upper house) is the representation of the Federal States (*Bundesländer*) of Germany and has its

Data on the web

W Berlin - Wikipedia

Sicher https://en.wikipedia.org/wiki/Berlin

Foreign mayors make up the Council of mayors (*rat der burgermeister*), which is led by the city's Governing Mayor and advises the Senate. The neighborhoods have no local government bodies.

Twin towns – sister cities [edit]

See also: *List of twin towns and sister cities in Germany*

Berlin maintains official partnerships with 17 cities.^[100] Town twinning between Berlin and other cities began with its sister city Los Angeles in 1967. East Berlin's partnerships were canceled at the time of German reunification but later partially reestablished. West Berlin's partnerships had previously been restricted to the borough level. During the Cold War era, the partnerships had reflected the different power blocs, with West Berlin partnering with capitals in the Western World, and East Berlin mostly partnering with cities from the Warsaw Pact and its allies.

There are several joint projects with many other cities, such as Beirut, Belgrade, São Paulo, Copenhagen, Helsinki, Johannesburg, Mumbai, Oslo, Shanghai, Seoul, Sofia, Sydney, New York City and Vienna. Berlin participates in international city associations such as the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin's official sister cities are:^[100]

- 1967 Los Angeles, United States
- 1987 Paris, France
- 1988 Madrid, Spain
- 1989 Istanbul, Turkey
- 1991 Warsaw, Poland^[101]
- 1991 Moscow, Russia
- 1992 Brussels, Belgium
- 1992 Budapest, Hungary^[102]
- 1993 Tashkent, Uzbekistan
- 1993 Mexico City, Mexico
- 1993 Jakarta, Indonesia
- 1994 Beijing, China
- 1994 Tokyo, Japan
- 1994 Buenos Aires, Argentina
- 1995 Prague, Czech Republic^[103]
- 2000 Windhoek, Namibia
- 2000 London, United Kingdom

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The [President of Germany](#), whose functions are mainly ceremonial under the [German constitution](#), has his official residence in [Schloss Bellevue](#).^[104] Berlin is the seat of the [German executive](#), housed in the [Chancellery](#), the [Bundeskanzleramt](#). Facing the Chancellery is the [Bundestag](#), the German Parliament, housed in the renovated Reichstag building since the government relocated to Berlin in 1998. The [Bundesrat](#) ("federal council", performing the function of an upper house) is the representation of the Federal States (*Bundesländer*) of Germany and has its

Let's grab these data!

Step 1: Load packages

R code —

```
1 library(rvest)  
2 library(stringr)
```

— end

Let's grab these data!

Step 2: Parse page source

R code —

```
3 library(rvest)
4 library(stringr)
5 parsed_url <- read_html("https://en.wikipedia.org/wiki/Berlin")
```

— end

Let's grab these data!

Step 3: Extract information

R code —

```
6 library(rvest)
7 library(stringr)
8 parsed_url <- read_html("https://en.wikipedia.org/wiki/Berlin")
9 parsed_nodes <- html_nodes(parsed_url, xpath = "//div[contains(@class,
  'column-count-3')]/li")
10 cities <- html_text(parsed_nodes)
11 cities[1:10]
[1] "1967 Los Angeles, United States" "1987 Paris, France"
[3] "1988 Madrid, Spain"                  "1989 Istanbul, Turkey"
[5] "1991 Warsaw, Poland[103]"           "1991 Moscow, Russia"
[7] "1992 Brussels, Belgium"              "1992 Budapest, Hungary[104]"
[9] "1993 Tashkent, Uzbekistan"          "1993 Mexico City, Mexico"
```

end

Why was this so easy?

The screenshot shows a browser window displaying the Wikipedia page for Berlin. The developer tools are open, specifically the Elements tab, which is showing the HTML structure of a list of capital cities. The list is contained within a `div` element with the class `mw-content-ltr`. Each item in the list is a `li` element representing a capital city with its name, year, flag, and country. The developer tools also show the CSS styles applied to these elements, including the margin, border, and padding properties for the list items.

Year	Capital City	Country
1967	Los Angeles, United States	United States
1987	Paris, France	France
1988	Madrid, Spain	Spain
1989	Istanbul, Turkey	Turkey
1991	Warsaw, Poland ^[101]	Poland
1991	Moscow, Russia	Russia
1992	Brussels, Belgium	Belgium
1992	Budapest, Hungary ^[102]	Hungary
1993	Tashkent, Uzbekistan	Uzbekistan
1993	Mexico City, Mexico	Mexico
1993	Jakarta, Indonesia	Indonesia
1994	Beijing, China	China
1994	Tokyo, Japan	Japan
1994	Buenos Aires, Argentina	Argentina
1995	Prague, Czech Republic ^[103]	Czech Republic
2000	Windhoek, Namibia	Namibia
2000	London, United Kingdom	United Kingdom

Why was this so easy?

The screenshot shows a web browser window with two tabs: "Berlin - Wikipedia" and "view-source:https://en.wikipedia.org/wik...". The main content is the Wikipedia article for Berlin, specifically the section on "Capital city". A red arrow points from the text "right-click, 'inspect element...' " to the "Los Angeles, United States" entry in the list of capital cities.

right-click, „inspect element...“

the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin [1] | 82, 94 + 88 cities are [100]

- 1967 Los Angeles, United States
- 1992 Brussels, Belgium
- 1994 ● Tokyo, Japan
- 1987 Paris, France
- 1993 Buenos Aires, Argentina
- 1995 Prague, Czech Republic [103]
- 1988 Tashkent, Uzbekistan
- 1993 Mexico City, Mexico
- 2000 Windhoek, Namibia
- 1991 Jakarta, Indonesia
- 1993 London, United Kingdom
- 1994 Beijing, China
- 1991 Warsaw, Poland [101]

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The President of Germany, whose functions are mainly ceremonial under the German constitution,

The developer tools (Elements tab) are open over the page content. The inspected element is a list item for "Los Angeles, United States". The DOM tree shows the structure of the list items, and the CSS panel displays the styles applied to the elements, including the class ".mw-content-ltr ul li". The right-hand sidebar shows the element's bounding box and its position relative to the page.

Why was this so easy?

The screenshot shows a browser window with two tabs: "Berlin - Wikipedia" and "view HTML source code". The main content area displays a list of capital cities from 1967 to 2000, with a red arrow pointing to the first item, "1967 Los Angeles, United States". A red box highlights the "view HTML source code" tab, which is active. The source code pane shows the HTML structure for the list, including spans with class="flagicon" and href attributes linking to Wikipedia pages for each city.

right-click,
„inspect
element...“

the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin [1] | 82, 94 + 85 cities are [100]

- 1967 Los Angeles, United States
- 1992 Brussels, Belgium
- 1994 ● Tokyo, Japan
- 1987 Paris, France
- 1993 Budapest, Hungary^[102]
- 1995 Tashkent, Uzbekistan
- 1999 Mexico City, Mexico
- 2000 Windhoek, Namibia
- 1991 Jakarta, Indonesia
- 1993 London, United Kingdom
- 1994 Beijing, China

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The President of Germany, whose functions are mainly ceremonial under the German constitution,

view HTML source code

Elements Console Sources Network Timeline Profiles ▾ A 2 : x

```
<div>=div.col column-column column-count="3" style="column-count: 3; -moz-column-count: 3; -webkit-column-count: 3; column-count: 3;">
  <ul>=ul
    <li>=li
      "1967 "
      <span>=span class="flagicon">=</span>
      <a href="/wiki/Los_Angeles" title="Los Angeles">Los Angeles</a>
      ", United States"
    </li>
    <li>=li
      "1992 "
      <span>=span class="flagicon">=</span>
      <a href="/wiki/Paris" title="Paris">Paris</a>
      ", France"
    </li>
    <li>=li
    <li>=li
    <li>=li
    <li>=li
    <li>=li
  </ul>
</div>
```

Filter: show .cls + element.style { }

div.columns dl, .load.php?debug=1 .mw-content-ltr ul, .mw-content-rtl .mw-content-ltr ul { margin-top: 0; }

.mw-content-ltr ul, .load.php?debug=1 .terrlanguage .i1 .mw-content-rtl .mw-content-ltr ul { margin: 0.3em 0 1.6em; padding: 0; }

ul { .load.php?debug=1 .terrlanguage .i1 list-style-type: disc; list-style-image: url("data:image/svg+xml,%3C%3Fxml%3Aversion%3D%22.229.5422528%22r305222,%222%20f11%3D%225238852 List-style-image"); }

color direction display font-family font-size

rgb(3, ltr block sans-serif 14px

Why was this so easy?

right-click,
„inspect
element...“

the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin [1] | 82, 94 + 85 cities are [100]

- 1967 Los Angeles, United States
- 1992 Brussels, Belgium
- 1994 ● Tokyo, Japan
- 1987 Paris, France
- 1992 Budapest, Hungary^[102]
- 1993 Tashkent, Uzbekistan
- 1995 Prague, Czech Republic^[103]
- 1999 Mexico City, Mexico
- 2000 Windhoek, Namibia
- 1991 Warsaw, Poland^[101]
- 1993 Jakarta, Indonesia
- 1994 Beijing, China

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The President of Germany, whose functions are mainly ceremonial under the German constitution,

view HTML source code

Elements Console Sources Network Timeline Profiles ▾ A 2 : Simon

div class="div-col column-column-count column-count-3" style="column-count: 3; -webkit-column-count: 3; column-count: 3;" id="mw-content-ltr" data-ltr="1" data-rtl="0">

- 1967 "Los Angeles", United States
- 1992 "Brussels", Belgium
- 1994 "Tokyo, Japan", United States
- 1987 "Paris", France
- 1992 "Budapest", Hungary
- 1993 "Tashkent", Uzbekistan
- 1995 "Prague", Czech Republic
- 1999 "Mexico City", Mexico
- 2000 "Windhoek", Namibia
- 1991 "Warsaw", Poland
- 1993 "Jakarta", Indonesia
- 1994 "Beijing", China

Identify elements that store information of interest

margin:
border:
padding:
width:
height:
font-size:
color:
direction:
display:
font-family:
font-weight:

Let's clean up these data!

R code

```
12 cities[1:10]
[1] "1967 Los Angeles, United States" "1987 Paris, France"
[3] "1988 Madrid, Spain"                 "1989 Istanbul, Turkey"
[5] "1991 Warsaw, Poland[103]"          "1991 Moscow, Russia"
[7] "1992 Brussels, Belgium"            "1992 Budapest, Hungary[104]"
[9] "1993 Tashkent, Uzbekistan"         "1993 Mexico City, Mexico"
```

end

Step 1: Remove footnotes with a regular expression

R code

```
13 cities <- str_replace(cities, "\\[\\d+\\]", "")
14 cities[1:10]
[1] "1967 Los Angeles, United States" "1987 Paris, France"
[3] "1988 Madrid, Spain"                 "1989 Istanbul, Turkey"
[5] "1991 Warsaw, Poland"              "1991 Moscow, Russia"
[7] "1992 Brussels, Belgium"            "1992 Budapest, Hungary"
[9] "1993 Tashkent, Uzbekistan"         "1993 Mexico City, Mexico"
```

end

Let's clean up these data!

Step 2: Extract data with regular expressions

R code —

```
15 year <- str_extract(cities, "\\d{4}")  
16 city <- str_extract(cities, "[[:alpha:] ]+") %>% str_trim  
17 country <- str_extract(cities, "[[:alpha:] ]+$") %>% str_trim  
18 year[1:10]  
[1] "1967" "1987" "1988" "1989" "1991" "1991" "1992" "1992" "1993" "  
1993"  
19 city[1:10]  
[1] "Los Angeles" "Paris"          "Madrid"        "Istanbul"      "Warsaw"  
[6] "Moscow"       "Brussels"       "Budapest"     "Tashkent"      "Mexico  
City"  
20 country[1:10]  
[1] "United States" "France"       "Spain"        "Turkey"  
[5] "Poland"        "Russia"       "Belgium"     "Hungary"  
[9] "Uzbekistan"    "Mexico"
```

end

Let's clean up these data!

Step 3: Put everything into data frame

R code —

```
21 cities_df <- data.frame(year, city, country)  
22 head(cities_df)
```

	year	city	country
1	1967	Los Angeles	United States
2	1987	Paris	France
3	1988	Madrid	Spain
4	1989	Istanbul	Turkey
5	1991	Warsaw	Poland
6	1991	Moscow	Russia

end

Let's map these data!

Step 1: Load necessary packages

R code —

```
23 library(ggmap)  
24 library(maps)
```

end

Let's map these data!

Step 2: Geocode cities with the Google Maps API

R code —

```
25 library(ggmap)
26 library(maps)
27 cities_coords <- geocode(paste0(cities_df$city, ", ", cities_df$country)
  )
28 cities_df$lon <- cities_coords$lon
29 cities_df$lat <- cities_coords$lat
30 cities_df$lon[1:10]
  [1] -118.243685  2.352222 -3.703790 28.978359      NA
  [6]  37.617300       NA 19.040235      NA -99.133208
31 cities_df$lat[1:10]
  [1] 34.05223 48.85661 40.41678 41.00824      NA 55.75583      NA
  [8] 47.49791       NA 19.43261
```

end

Let's map these data!

Step 3: Plot world map, add coordinates

R code

```
32 map_world <- borders("world", colour = "gray50", fill = "white")
33 ggplot() + map_world + geom_point(aes(x = cities_df$lon, y = cities_df$lat), color = "red", size = 1) + theme_void()
```

end



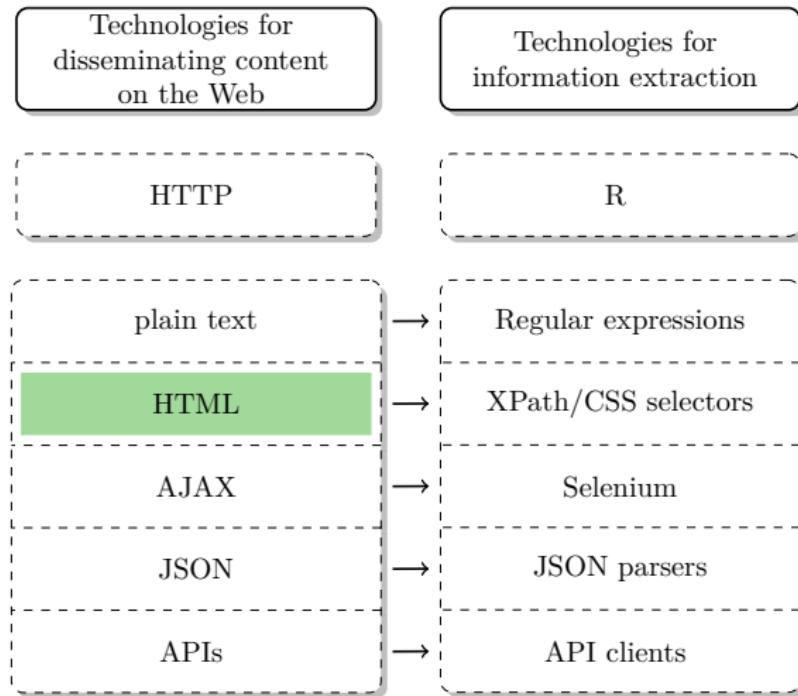
Campus Luzern

A Primer to Web Scraping with R

HTML

Simon Munzert
Hertie School of Governance, Berlin
May 20-21, 2019

Technologies of the World Wide Web



HTML basics

HTML – a quick primer

What's HTML?

- HyperText Markup Language
- markup language = plain text + markups
- W3C standard for the construction of websites
- lies underneath of what you see in your browser

Why is this important to us?

- it determines where and how information is stored
- a basic understanding of HTML helps us locate the information we want to retrieve
- relax. A passive understanding of HTML is sufficient

HTML in the wild

Sicher <https://en.wikipedia.org/wiki/Berlin>

Not logged in Talk Contributions Create account Log in

Berlin

From Wikipedia, the free encyclopedia

This article is about the capital of Germany. For other uses, see Berlin (disambiguation).

Berlin (/bərˈlɪn/, German: [bɛɐ̯ˈliːn] (listen)) is the capital and the largest city of Germany as well as one of its constituent 16 states. With a population of approximately 3.5 million people,^[4] Berlin is the second most populous city proper and the seventh most populous urban area in the European Union.^[5] Located in northeastern Germany on the banks of rivers Spree and Havel, it is the centre of the Berlin-Brandenburg Metropolitan Region, which has about 6 million residents from more than 180 nations.^{[6][7][8][9]} Due to its location in the European Plain, Berlin is influenced by a temperate seasonal climate. Around one-third of the city's area is composed of forests, parks, gardens, rivers and lakes.^[10]

First documented in the 13th century and situated at the crossing of two important historic trade routes,^[11] Berlin became the capital of the Margravate of Brandenburg (1417–1701), the Kingdom of Prussia (1701–1918), the German Empire (1871–1918), the Weimar Republic (1919–1933) and the Third Reich (1933–1945).^[12] Berlin in the 1920s was the third largest municipality in the world.^[13] After World War II and its consequent occupation by the victorious countries, the city was divided; East Berlin became the capital of East Germany while West Berlin became a de facto West German exclave, surrounded by the Berlin Wall (1961–1989).^[14] Following German reunification in 1990, Berlin once again became the capital of Germany.



HTML in the wild

Berlin's mayors make up the council of mayors (*rat der Bürgermeister*), which is led by the city's Governing Mayor and advises the Senate. The neighborhoods have no local government bodies.

Twin towns – sister cities [edit]

See also: List of twin towns and sister cities in Germany

Berlin maintains official partnerships with 17 cities.^[100] Town twinning between Berlin and other cities began with its sister city Los Angeles in 1967. East Berlin's partnerships were canceled at the time of German reunification but later partially reestablished. West Berlin's partnerships had previously been restricted to the borough level. During the Cold War era, the partnerships had reflected the different power blocs, with West Berlin partnering with capitals in the Western World, and East Berlin mostly partnering with cities from the Warsaw Pact and its allies.

There are several joint projects with many other cities, such as Beirut, Belgrade, São Paulo, Copenhagen, Helsinki, Johannesburg, Mumbai, Oslo, Shanghai, Seoul, Sofia, Sydney, New York City and Vienna. Berlin participates in international city associations such as the Union of the Capitals of the European Union, Eurocities, Network of European Cities of Culture, Metropolis, Summit Conference of the World's Major Cities, and Conference of the World's Capital Cities. Berlin's official sister cities are:^[100]

- 1967 Los Angeles, United States
- 1987 Paris, France
- 1988 Madrid, Spain
- 1989 Istanbul, Turkey
- 1991 Warsaw, Poland^[101]
- 1991 Moscow, Russia
- 1992 Brussels, Belgium
- 1992 Budapest, Hungary^[102]
- 1993 Tashkent, Uzbekistan
- 1993 Mexico City, Mexico
- 1993 Jakarta, Indonesia
- 1994 Beijing, China
- 1994 Tokyo, Japan
- 1994 Buenos Aires, Argentina
- 1995 Prague, Czech Republic^[103]
- 2000 Windhoek, Namibia
- 2000 London, United Kingdom

Capital city [edit]

Berlin is the capital of the Federal Republic of Germany. The President of Germany, whose functions are mainly ceremonial under the German constitution, has his official residence in Schloss Bellevue.^[104] Berlin is the seat of the German executive, housed in the Chancellery, the *Bundeskanzleramt*. Facing the Chancellery is the *Bundestag*, the German Parliament, housed in the renovated Reichstag building since the government relocated to Berlin in 1998. The *Bundesrat* ("federal council", performing the function of an upper house) is the representation of the Federal States (*Bundesländer*) of Germany and has its

HTML in the wild

The screenshot shows a web browser window with the URL <https://en.wikipedia.org/wiki/Berlin>. The developer tools are open, specifically the Elements and Styles tabs.

Elements Tab: Shows the HTML structure of the 'Capital city' section. The list items are wrapped in `` tags, each containing a flag icon (`flagicon` class) and a link to the city's Wikipedia page.

Styles Tab: Shows the CSS styles applied to the list items. Key styles include:

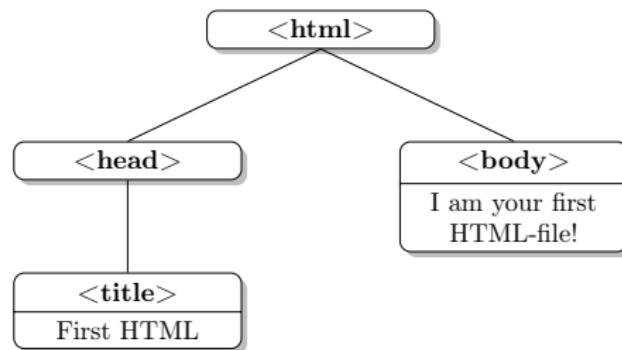
- `list-style-type: none;`
- `margin-left: 2em;`
- `padding-left: 0.3em;`
- `font-size: 14px;`

DOM Tree: Shows the full DOM structure of the page, including the main content area and various sidebar components.

Year	City	Country
1967	Los Angeles	United States
1992	Brussels	Belgium
1992	Buenos Aires	Argentina
1993	Tashkent	Uzbekistan
1993	Mexico City	Mexico
1993	Jakarta	Indonesia
1994	Beijing	China
1994	Tokyo	Japan
1994	Budapest	Hungary
1995	Prague	Czech Republic
2000	Windhoek	Namibia
2000	London	United Kingdom

Tree structure

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title id=1>First HTML</title>
5   </head>
6   <body>
7     I am your first HTML file!
8   </body>
9 </html>
```



Elements and attributes

Elements and attributes

Elements

Elements are a combination of *start tags*, content, and *end tags*.

Example:

```
1 <title>First HTML</title>
```

Syntax

element title	title
start tag	<title>
end tag	</title>
value	First HTML

Elements and attributes

Attributes

Attributes describe elements and are stored in the start tag. In HTML, there are specific attributes for specific elements.

Example:

```
1 <a href="http://www.r-datacollection.com/">Link to Homepage</a>
```

Syntax

- name-value pairs: `name="value"`
- simple and double quotation marks possible
- several attributes per element possible

Why tags and attributes are important

- tags structure HTML documents
- everything that structures a document can be used to extract information
- in the following, we get to know some important tags which are useful when scraping information from the Web

Important tags and attributes

Anchor tag <a>

- links to other pages or resources
- classical links are always formatted with an anchor tag
- the **href** attribute determines the target location
- the value is the name of the link

Link to another resource:

```
1 <a href="en.wikipedia.org/wiki/List_of_lists_of_lists">Link with absolute path</a>
```

Reference in a document:

```
1 <a id="top">Reference Point</a>
```

Link to a reference:

```
1 <a href="#top">Link to Reference Point</a>
```

Important tags and attributes

Heading tags `<h1>`, `<h2>`, ..., and paragraph tag `<p>`

- structure text and paragraphs
- heading tags range from level 1 to 6
- paragraph tag induces line break

Examples:

```
1 <p>This text is going to be a paragraph one day and separated from other  
2 text by line breaks.</p>
```

```
1 <h1>heading of level 1 -- this will be BIG</h1>  
2 ...  
3 <h6>heading of level 6 -- the smallest heading</h6>
```

Important tags and attributes

Listing tags , and <dl>

- the `` tag creates a numeric list, `` an unnumbered list, `<dl>` a definition list
- list elements are indicated with the `` tag

Example:

```
1 <ul>
2   ^^I<li>Dogs</li>
3   ^^I<li>Cats</li>
4   ^^I<li>Fish</li>
5 </ul>
```

Important tags and attributes

Organizational tags `<div>` and ``

- grouping of content over lines (`<div>`) or within lines (``)
- do not change the layout themselves but work together with CSS

Example of CSS definition

```
1  div.happy { color:pink;  
2      font-family:"Comic Sans MS";  
3      font-size:120% }  
4  span.happy { color:pink;  
5      font-family:"Comic Sans MS";  
6      font-size:120% }
```

In the HTML document

```
1  <div class="happy"><p>I am a happy styled paragraph</p></div>  
2  non-happy text with <span class="happy">some happiness</span>
```

Important tags and attributes

Form tag <form>

- allows to incorporate HTML forms
- client can send information to the HTTP server via forms
- whenever you type something into a field or click on radio buttons in your browser, you are interacting with forms

Example:

```
1 <form name="submitPW" action="Passed.html" method="get">
2   password:
3   <input name="pw" type="text" value="">
4   <input type="submit" value="SubmitButtonText">
5 </form>
```

Important tags and attributes

Table tags `<table>`, `<tr>`, `<td>`, and `<th>`

- standard HTML tables always follow a standard architecture
- the different tags allow to define the table as a whole, individual rows (including the heading), and cells
- if the data is hidden in tables, scraping will be straightforward

Example:

```
1 <table>
2   <tr> <th>Rank</th> <th>Nominal GDP</th> <th>Name</th> </tr>
3   <tr> <th></th> <th>(per capita, USD)</th> <th></th> </tr>
4   <tr> <td>1</td> <td>170,373</td> <td>Lichtenstein</td> </tr>
5   <tr> <td>2</td> <td>167,021</td> <td>Monaco</td> </tr>
6   <tr> <td>3</td> <td>115,377</td> <td>Luxembourg</td> </tr>
7   <tr> <td>4</td> <td>98,565</td> <td>Norway</td> </tr>
8   <tr> <td>5</td> <td>92,682</td> <td>Qatar</td> </tr>
9 </table>
```

Summary

Summary

- HTML is the *lingua franca* on the web
- content on webpages is structured by HTML tags that are nested in a tree structure
- to break open information, we will have to locate it in the HTML tree
- for web scraping purposes, a mostly passive knowledge of HTML is sufficient

```
<DIV>Q: HOW DO YOU ANNOY A WEB DEVELOPER?</SPAN>
```

Source: <https://xkcd.com/1144/> (Randall Munroe)

Campus Luzern

A Primer to Web Scraping with R

Regular Expressions Basics

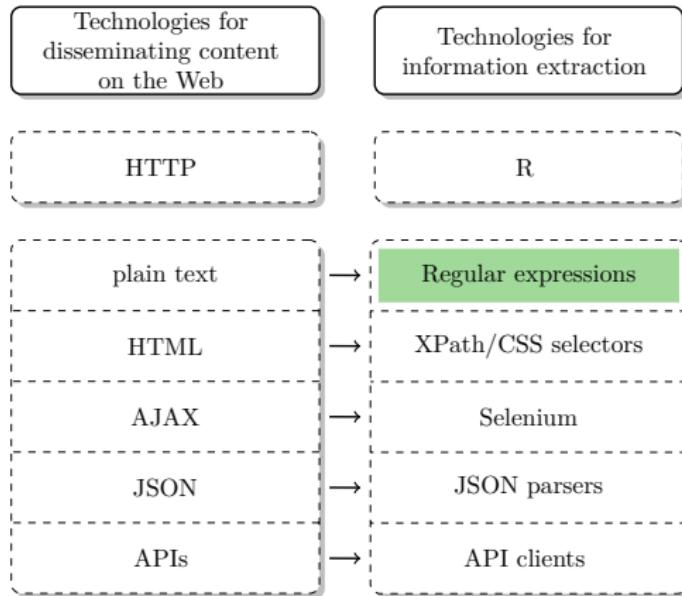
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Regular expressions

Technologies of the World Wide Web



What are regular expressions?

Definition

- a.k.a. *regex* or *RegExp*
- origins in formal language theory
- sequences of characters that describe patterns in text
- implemented in many programming languages, including R

Why are regular expressions useful for web scraping?

- information on the web can often be described by patterns (think email addresses, numbers, cells in HTML tables, ...)
- if the data of interest follow specific patterns, we can match and extract them—regardless of page layout and HTML overhead
- whenever the information of interest is (stored in) text, regular expressions are useful for extraction and tidying purposes

Introductory example

Introductory example

R code

```
1 raw.data <- "555-1239Moe Szyslak(636) 555-0113Burns, C. Montgomery  
2 555-6542Rev. Timothy Lovejoy555 8904Ned Flanders636-555-3226  
3 Simpson,Homer5553642Dr. Julius Hibbert"
```

end

- vector `raw.data` contains unstructured phonebook entries
- goal: extraction of entries
- problem: find a pattern that matches names and numbers
- solution: regex!

Introductory example

R code

```
4 raw.data <- "555-1239Moe Szyslak(636) 555-0113Burns, C. Montgomery  
5 555-6542Rev. Timothy Lovejoy555 8904Ned Flanders636-555-3226  
6 Simpson,Homer5553642Dr. Julius Hibbert"
```

end

Solution:

- load package **stringr** (more on that later)
- a detective's work: construct regex for names
- apply regex on raw vector

R code

```
7 library(stringr)  
8 name <- unlist(str_extract_all(raw.data, "[[:alpha:].[[:alpha:]]{2,}"))  
9 name  
[1] "Moe Szyslak"           "Burns, C. Montgomery" "Rev. Timothy Lovejoy"  
[4] "Ned Flanders"          "Simpson,Homer"        "Dr. Julius Hibbert"
```

end

Introductory example

Solution, continued:

- construct and apply regex for phone numbers
- combine both vectors

R code

```
10 phone <- unlist(str_extract_all(raw.data,
11                               "\\\((?(\\\d{3})?\\)?)?(-| )?\\d{3}(-| )?\\d{4}\")")
12 phone
13 [1] "555-1239"          "(636) 555-0113" "555-6542"           "555 8904"
14 [5] "636-555-3226"      "5553642"
15
16 data.frame(name = name, phone = phone)
17
18   name       phone
19   1 Moe Szyslak 555-1239
20   2 Burns, C. Montgomery (636) 555-0113
21   3 Rev. Timothy Lovejoy 555-6542
22   4 Ned Flanders 555 8904
23   5 Simpson, Homer 636-555-3226
24   6 Dr. Julius Hibbert 5553642
```

end

Summary



Source: <https://xkcd.com/208/> (Randall Munroe)

Campus Luzern

A Primer to Web Scraping with R

Regular Expressions in R

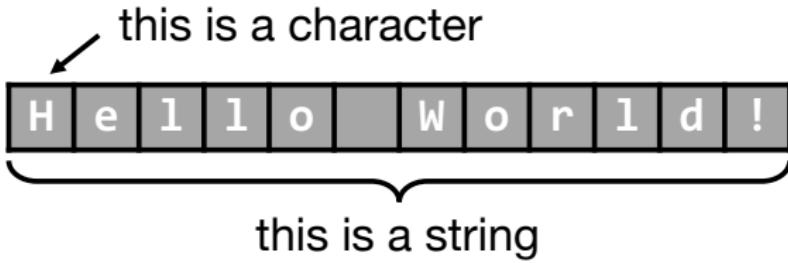
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Regular expressions in R

Regular expressions in R



Regular expressions in R

An example string:

R code

```
1 example.obj <- "1. A small sentence. - 2. Another tiny sentence."  
-----  
end
```

We are going to use the `str_extract()` function and the `str_extract_all()` function from the `stringr` package to apply regular expressions in strings. The generic syntax is:

```
str_extract(string, pattern)  
str_extract_all(string, pattern)
```

`str_extract()` returns the first match, `str_extract_all()` returns all matches.

Strings match themselves

R code

```
2 str_extract(example.obj, "small")
[1] "small"
3 str_extract(example.obj, "banana")
[1] NA
```

end

Multiple matches are returned as a list

R code

```
4 (out <- str_extract_all(c("text", "manipulation", "basics"), "a"))
[[1]]
character(0)

[[2]]
[1] "a" "a"

[[3]]
[1] "a"
```

end

Regular expressions in R

"1. A small sentence. - 2. Another tiny sentence."

Character matching is case sensitive

R code

```
5 str_extract(example.obj, "small")
[1] "small"
6 str_extract(example.obj, "SMALL")
[1] NA
7 str_extract(example.obj, regex("SMALL", ignore_case = TRUE))
[1] "small"
```

end

We can match arbitrary combinations of characters

R code

```
8 str_extract(example.obj, "mall sent")
[1] "mall sent"
```

end

Regular expressions in R

"1. A small sentence. - 2. Another tiny sentence."

Matching the beginning of a string

R code —

```
9 str_extract(example.obj, "^1")
[1] "1"
10 str_extract(example.obj, "^2")
[1] NA
```

— end

Matching the ending of a string

R code —

```
11 str_extract(example.obj, "sentence$")
[1] NA
12 str_extract(example.obj, "sentence.$")
[1] "sentence."
```

— end

Regular expressions in R

"1. A small sentence. - 2. Another tiny sentence."

Express an "or" with the pipe operator

R code

```
13 unlist(str_extract_all(example.obj, "tiny|sentence"))
[1] "sentence" "tiny"      "sentence"
```

end

The dot: the ultimate wildcard

R code

```
14 str_extract(example.obj, "sm.11")
[1] "small"
```

end

Matching of meta characters

- some symbols have a special meaning in the regex syntax: `.`, `|`, `(`, `)`, `[`, `]`, `{`, `}`, `^`, `$`, `*`, `+`, `?` and `-`.
- if we want to match them literally, we have to use an escape sequence: `\symbol`
- as `\` is a meta character itself, we have to escape it with `\`, so we always write `\\symbol` (weird, isn't it?!)
- alternatively, use `fixed("symbols")` to let the parser interpret a chain of symbols literally

R code

```
15 unlist(str_extract_all(example.obj, "\\\."))  
[1] "." "." "." ".."  
16 unlist(str_extract_all(example.obj, fixed(".")))  
[1] "." "." "." ".."
```

Character classes

Square brackets define character classes

Character classes help define special wild cards. The idea is that any of the characters within the brackets can be matched.

R code

```
17 str_extract(example.obj, "sm[abc]ll")  
[1] "small"
```

end

The hyphen defines a range of characters

R code

```
18 str_extract(example.obj, "sm[a-p]ll")  
[1] "small"
```

end

Predefined character classes

<code>[:digit:]</code>	Digits: 0 1 2 3 4 5 6 7 8 9
<code>[:lower:]</code>	Lower-case characters: a–z
<code>[:upper:]</code>	Upper-case characters: A–Z
<code>[:alpha:]</code>	Alphabetic characters: a–z and A–Z
<code>[:alnum:]</code>	Digits and alphabetic characters
<code>[:punct:]</code>	Punctuation characters: ‘,’ ‘;’ etc.
<code>[:graph:]</code>	Graphical characters: <code>[:alnum:]</code> and <code>[:punct:]</code>
<code>[:blank:]</code>	Blank characters: Space and tab
<code>[:space:]</code>	Space characters: Space, tab, newline, etc.
<code>[:print:]</code>	Printable characters: <code>[:alnum:]</code> , <code>[:punct:]</code> , <code>[:space:]</code>

Predefined character classes in action

R code

```
19 unlist(str_extract_all(example.obj, "[[:punct:]])")
[1] "." "." "-" "." "."
20 unlist(str_extract_all(example.obj, "[[:alpha:]])")
[1] "A" "s" "m" "a" "l" "l" "s" "e" "n" "t" "e" "n" "c" "e" "A" "n" "o"
[18] "t" "h" "e" "r" "t" "i" "n" "y" "s" "e" "n" "t" "e" "n" "c" "e"
```

end

Predefined character classes are useful because they are efficient

- combine different kinds of characters
- facilitate reading of an expression
- include special characters, e.g., ß, ö, ...
- can be extended

R code

```
21 unlist(str_extract_all(example.obj, "[[:punct:]ABC]"))
[1] "." "A" "." "-" "." "A" "."
22 unlist(str_extract_all(example.obj, "[^[:alnum:]]"))
[1] " " " " " " " " " " " " " " " " " " " " " "
```

end

Alternative character classes

\w	Word characters: <code>[:alnum:]_]</code>
\W	No word characters: <code>^[:alnum:]_]</code>
\s	Space characters: <code>[:blank:]</code>
\S	No space characters: <code>^[:blank:]</code>
\d	Digits: <code>[:digit:]</code>
\D	No digits: <code>^[:digit:]</code>
\b	Word edge
\B	No word edge
\<	Word beginning
\>	Word end

Regular expressions in R

"1. A small sentence. - 2. Another tiny sentence."

Alternative character classes in action

R code

```
23 unlist(str_extract_all(example.obj, "\\w+"))
[1] "1"          "A"          "small"       "sentence"    "2"          "Another"
[7] "tiny"        "sentence"

24 unlist(str_extract_all(example.obj, "e\\b"))
[1] "e" "e"
```

end

Use of quantifiers

?	The preceding item is optional and will be matched at most once
*	The preceding item will be matched zero or more times
+	The preceding item will be matched one or more times
{n}	The preceding item is matched exactly n times
{n,}	The preceding item is matched n or more times
{n,m}	The preceding item is matched between n and m times

R code

```
25 str_extract(example.obj, "s[[[:alpha:]][[:alpha:]][[:alpha:]]1")  
[1] "small"  
26 str_extract(example.obj, "s[[[:alpha:]]{3}1")  
[1] "small"  
27 str_extract(example.obj, "A.+sentence")  
[1] "A small sentence. - 2. Another tiny sentence"
```

end

Greedy quantification

- the use of `'.+'` results in 'greedy' matching, i.e. the parser tries to match as many characters as possible
- not always desired, but `'.+?'` helps avoid greedy quantification

R code

```
28 str_extract(example.obj, "A.+sentence")
[1] "A small sentence. - 2. Another tiny sentence"
29 str_extract(example.obj, "A.+?sentence")
[1] "A small sentence"
```

end

Meta symbols in character classes

- within a *character class*, most meta symbols lose their special meaning
- exceptions: `^` and `-`
- `-` at the beginning or end matches the hyphen

R code

```
30 unlist(str_extract_all(example.obj, "[1-2]"))
[1] "1" "2"
31 unlist(str_extract_all(example.obj, "[12-]"))
[1] "1" "--" "2"
```

end

Backreferencing

Positive and negative lookahead and lookbehind assertions

- match if a certain pattern before (or after) the actual match is found (or not), but are not returned themselves
- positive and negative look**ahead** assertions:
`(?=...)` and `(?!...)`
- positive and negative look**behind** assertions:
`(?<=...)` and `(?<!...)`

R code

```
32 unlist(str_extract_all(example.obj, "(?<=2. ).+"))
[1] "Another tiny sentence."
33 unlist(str_extract_all(example.obj, ".+(?=2)"))
[1] "1. A small sentence. - "
```

end

Backreferencing

- regular expression ‘with memory’
- repeated match of previously matched pattern
- we refer to the first match (defined with round brackets) using \1, to the second match with \2 etc. (up to 9)

R code

```
34 str_extract(example.obj, "([[:alpha:]]) .+?\\1")  
[1] "A small sentence. - 2. A"
```

end

Logic: Match the first letter, then anything until you find the first letter again (not greedy)

Regular expressions in R

"1. A small sentence. - 2. Another tiny sentence."

Backreferencing: a bit more complicated

Goal: match a word that does not include 'a' until the word appears the second time

Solution:

R code

```
35 str_extract(example.obj, "(\\b[b-z]+\\b).+?\\1")  
[1] "sentence. - 2. Another tiny sentence"
```

end

Mechanic:

1. match all letters without 'a': [b-z]+
2. match complete words (have beginning and end): \\b
3. refer to the word: ()
4. match anything in between: .+?\\1

Summary

Summary

- regular expressions are magic
- regular expressions are non-trivial
- regular expressions are unreadable
- the good news: for scraping purposes, we can (and should!) rely on other, simpler and more robust tools
- still, regex can prove incredibly powerful under certain circumstances

CODING HORROR



Campus Luzern

A Primer to Web Scraping with R

String Manipulation

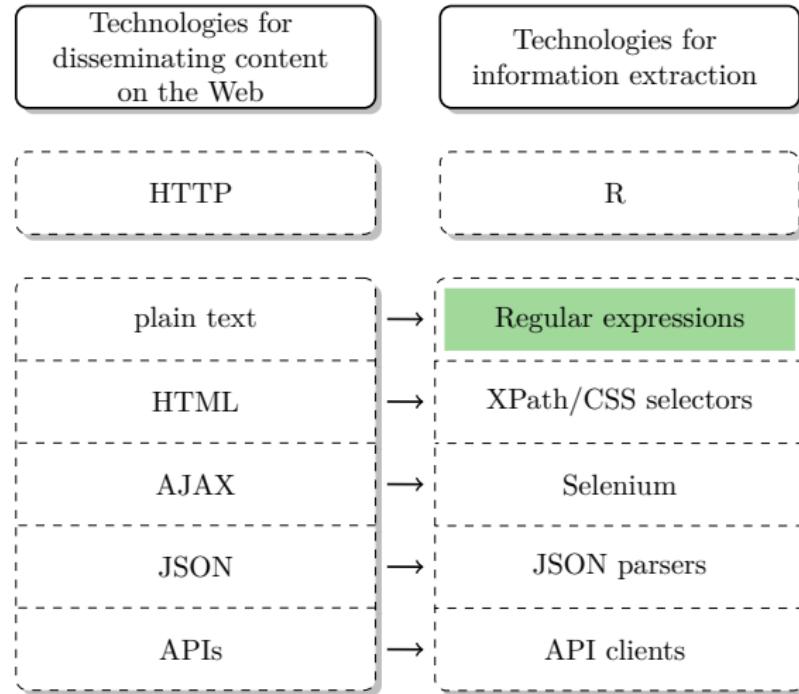
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

String manipulation in R

Technologies of the World Wide Web



String manipulation in R

What is string manipulation?

- processing of string (character, text) data
- important operations: extraction of text patterns, data tidying, preparation of text corpora for statistical text processing
- web data often is text data!

String manipulation with R

- base R provides basic string manipulation functionality but not a very consistent syntax
- comfortable string processing with Hadley Wickham's **stringr** package

String manipulation in R with the `stringr` package

Function	Description	Output
<i>Functions using regular expressions</i>		
<code>str_extract()</code>	Extracts first string that matches pattern	Character vector
<code>str_extract_all()</code>	Extracts all strings that match pattern	List of character vectors
<code>str_locate()</code>	Return position of first pattern match	Matrix of start/end positions
<code>str_locate_all()</code>	Return positions of all pattern matches	List of matrices
<code>str_replace()</code>	Replaces first pattern match	Character vector
<code>str_replace_all()</code>	Replaces all pattern matches	Character vector
<code>str_split()</code>	Split string at pattern	List of character vectors
<code>str_split_fixed()</code>	Split string into fixed number of pieces	Matrix of character vectors
<code>str_detect()</code>	Detect pattern in string	Boolean vector
<code>str_count()</code>	Count number of patterns in string	Numeric vector
<i>Further useful functions</i>		
<code>str_sub()</code>	Extract strings by position	Character vector
<code>str_subset()</code>	Extract strings for which condition applies	Character vector
<code>str_length()</code>	Length of string	Numeric vector
<code>str_trim()</code>	Discard string padding	Character vector

Useful functions

Regular expressions in R

Again, our example string:

R code —

```
1 example.obj <- "1. A small sentence. - 2. Another tiny sentence."
```

end

String manipulation in R

"1. A small sentence. - 2. Another tiny sentence."

String localization

R code

```
2 str_locate(example.obj, "small")
```

	start	end
[1,]	6	10

end

Substring extraction

R code

```
3 str_sub(example.obj, start = 6, end = 10)
```

[1] "small"

end

String manipulation in R

"1. A small sentence. - 2. Another tiny sentence."

String replacement

R code

```
4 str_replace(example.obj, pattern = "tiny", replacement = "huge")
[1] "1. A small sentence. - 2. Another huge sentence."
```

end

String splitting

R code

```
5 unlist(str_split(example.obj, "-"))
[1] "1. A small sentence. "      " 2. Another tiny sentence."
```

end

String manipulation in R

Manipulation of several elements

- until this point we applied functions to vectors of length one
- now: apply functions to multi-element vectors

Example object with several elements:

R code

```
6 (char.vec <- c("this", "and this", "and that"))
[1] "this"      "and this"   "and that"
```

end

String detection

R code

```
7 str_detect(char.vec, "this")
[1] TRUE  TRUE FALSE
```

end

String counting

R code

```
8 str_count(char.vec, "this")
[1] 1 1 0
9 str_count(char.vec, "\\w+")
[1] 1 2 2
10 str_length(char.vec)
[1] 4 8 8
```

end

String subsetting

R code

```
11 str_subset(char.vec, "this")
[1] "this"      "and this"
```

end

String joining

R code

```
12 str_c("text", "manipulation", sep = " ")
[1] "text manipulation"
13 cat(str_c(char.vec, collapse = "\n"))
this
and this
and that
14 str_c("text", c("manipulation", "basics"), sep = " ")
[1] "text manipulation" "text basics"
```

end

... but you might want prefer to stick to good old `paste()` and `paste0()`.

Approximate matching

- Matching of approximately equal strings, (e.g., “Beyoncé” and “Beyonce”)
- we could program naïve matching algorithms using regex
- better: use of more powerful algorithms, e.g. Levenshtein distance
- `agrep()` function in base R
- for more functionality see `stringdist` package

R code

```
15  agrep("Barack Obama", "Barack H. Obama", max.distance = list(all = 3))  
[1] 1  
16  agrep("Barack Obama", "Michelle Obama", max.distance = list(all = 3))  
integer(0)
```

end

Summary

Summary

- being able to manipulate string data in R is very useful when you want to automate processing web data
- there are base R commands for string manipulation, but the **stringr** package provides many useful functions with a consistent syntax
- if you need more, check out the even more powerful **stringi** package



Source:

<http://kevingleong.blogspot.de/2011/01/string-manipulation-exercises.html>

Campus Luzern

A Primer to Web Scraping with R

Inspecting the HTML Tree

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Browsing vs. scraping

Browsing vs. scraping the web

Using your browser to access webpages

1. you click on a link, enter a URL, etc.
2. browser/your machine sends request to server that hosts website
3. server returns resource (often an HTML document)
4. browser interprets HTML and renders it nicely



Using R to access webpages

1. you manually specify a resource
2. R/your machine sends request to server
3. server returns resource
4. R parses HTML but does not render it
5. it's up to you to tell R what content to extract



Interacting with your browser

On web browsers

- modern browsers are complex pieces of software that take care of multiple operations while you browse the web
- common operations: retrieve resources, render and display information, provide interface for user-webpage interaction
- although your goal is to automate web data retrieval, your browser is an important tool in web scraping workflow



The use of browsers for web scraping

- give you an intuitive impression of the architecture of a webpage
- allow you to inspect the source code
- let you construct XPath/CSS selector expressions with plugins
- render dynamic web content (JavaScript interpreter)

A note on browser differences

- inspecting the source code (as shown on the following slides) works more or less identically in **Chrome** and **Firefox**
- in **Safari**, go to → **Preferences**, then → **Advanced** and select "Show Develop menu in menu bar". This unlocks the "Show Page Source" option and the Web Developer Tools

Inspecting the HTML source code

Inspecting the HTML source code

Example

- browser: Google Chrome
- source: https://en.wikipedia.org/wiki/List_of_tallest_buildings

The screenshot shows a web browser window with the URL https://en.wikipedia.org/wiki/List_of_tallest_buildings. The page title is "List of tallest buildings". The left sidebar contains a navigation menu with links like "Main page", "Contents", "Featured content", "Current events", "Random article", "Wikipedia store", "Interaction", "Help", "About Wikipedia", "Community portal", "Recent changes", "Contact page", "Tools", "What links here", "Related changes", "Upload file", "Special pages", "Permanent link", "Page information", "Wikidata item", "Cite this page", "Print/export", and "Create a book". The main content area starts with a section titled "List of tallest buildings" from Wikipedia, followed by a note about ranking criteria and alternative measurements. A sidebar on the right features a large image of the Burj Khalifa and a caption describing it as the world's tallest building.

Click to go forward, hold to see history | Wikipedia

Secure | https://en.wikipedia.org/wiki/List_of_tallest_buildings

Not logged in | Talk | Contributions | Create account | Log in

Article | Talk

Read | View source | View history | Search Wikipedia

List of tallest buildings

From Wikipedia, the free encyclopedia

This list of tallest buildings in the world ranks skyscrapers by height. Only buildings with continuously occupiable floors are included, thus non-building structures, including towers, are not included. (See List of tallest buildings and structures.)

Contents [hide]

- 1 Ranking criteria and alternatives
- 2 Tallest buildings in the world (350 m+)
- 3 Photo gallery
- 4 Alternative measurements
 - 4.1 Height to pinnacle (highest point)
- 5 Buildings under construction
- 6 List by continent
- 7 See also
- 8 Notes
- 9 References
- 10 External links

Ranking criteria and alternatives

The International non-profit organization Council on Tall Buildings and Urban Habitat (CTBUH) was formed in 1969 and announces the title of "The World's Tallest Building" and sets the standards by which buildings are measured. It maintains a list of the 100 tallest completed buildings in the world.^[3] The organization currently ranks Burj Khalifa in Dubai as the tallest at 828 m (2,717 ft).^[3] The CTBUH only recognizes buildings that are complete, however, and some buildings listed within these list articles are not considered complete by the CTBUH.

The 828-metre (2,717 ft) tall Burj Khalifa in Dubai has been the world's tallest building since 2008.^[1] The Burj Khalifa has been classified as Megatall.^[2]

Inspecting the HTML source code

Example

- browser: Google Chrome
- source: https://en.wikipedia.org/wiki/List_of_tallest_buildings
- right-click on page

The screenshot shows a web browser window displaying the Wikipedia article "List of tallest buildings". The URL in the address bar is https://en.wikipedia.org/wiki/List_of_tallest_buildings. The browser interface includes a top navigation bar with tabs for "Article" and "Talk", and buttons for "Read", "View source", "View history", and "Search Wikipedia". On the left, there's a sidebar with links like "Main page", "Contents", "Featured content", etc. The main content area features the title "List of tallest buildings" and a brief introduction about the list ranking skyscrapers by height. A sidebar on the right contains "Contents" and other article details. A context menu is open over the page content, with the "Inspect" option highlighted in red.

Inspecting the HTML source code

Example

- browser: Google Chrome
- source: https://en.wikipedia.org/wiki/List_of_tallest_buildings
- right-click on page
- select "View Page Source"

The screenshot shows a web browser window displaying the Wikipedia article 'List of tallest buildings'. The URL in the address bar is https://en.wikipedia.org/wiki/List_of_tallest_buildings. The page content includes a sidebar with navigation links like 'Main page', 'Contents', and 'Interaction', and a main article section titled 'List of tallest buildings' with a photo of the Burj Khalifa.

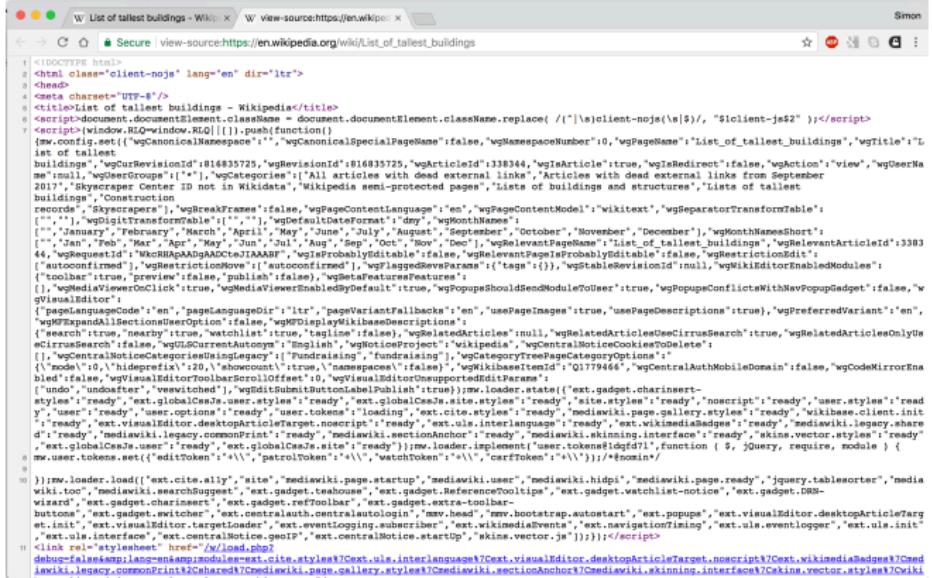
A context menu is open, showing options such as 'Back', 'Forward', 'Reload', 'Save As...', 'Print...', 'Cast...', 'Translate to English', 'View Page Source' (which is highlighted with a red arrow), and 'Inspect'.

The main article text discusses the ranking of skyscrapers by height, mentioning the Council on Tall Buildings and Urban Habitat (CTBUH) and the Burj Khalifa as the tallest building at 828 m (2,717 ft).

Inspecting the HTML source code

Example

- HTML (and JavaScript) code can be ugly...



Inspecting the HTML source code

Example

- HTML (and JavaScript) code can be ugly...
- but looking more closely, we can find the displayed information

```
</head>
<body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject page-List_of_tallest_buildings rootpage-List_of_tallest_buildings skin-vector action-view">
  <div id="mw-page-base" class="noprint"></div>
  <div id="mw-headertop" class="noprint"></div>
  <div id="content" class="mw-body" role="main">
    <a id="top"></a>
    <div id="siteNotice" class="mw-body-content"><!-- CentralNotice --></div>
    <div id="mw-indicator-pp-default" class="mw-indicator"><a href="/wiki/Wikipedia:Protection_policy#semi" title="This article is semi-protected until March 17, 2019."></a></div>
  </div>
  <div id="firstHeading" class="firstHeading" lang="en">List of tallest buildings</h1>
  <div id="bodyContent" class="mw-body-content">
    <div id="siteSub" class="noprint">From the free encyclopedia</div>
    <div id="contentSub"></div>
    <div id="Jump_to" style="float: right;">navigation <a href="#p-search">search</a>
```

Inspecting the live HTML tree

Inspecting individual elements and the live HTML tree

Example

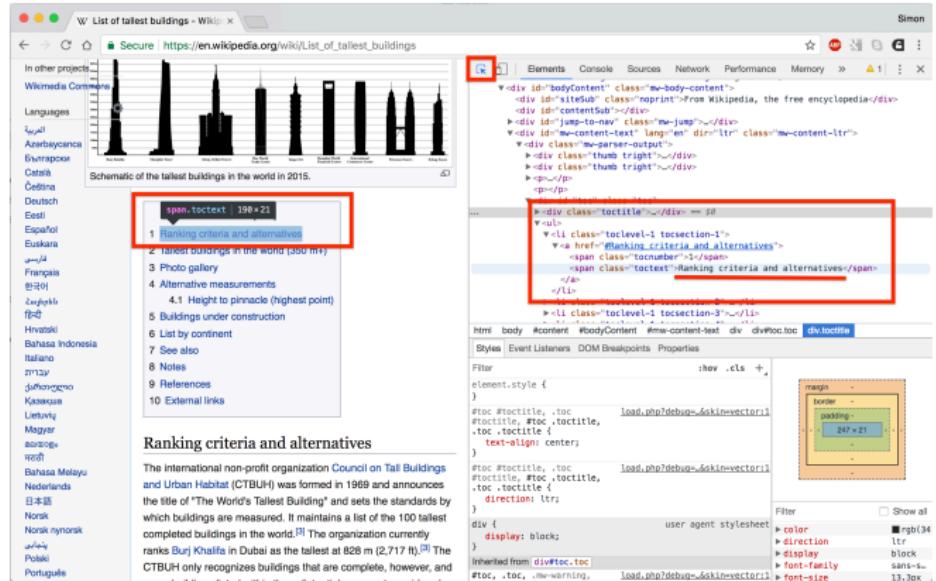
- another way of inspecting the HTML code is to use the **Web Developer Tools**
- to that end, do:
 - right-click on element of interest
 - select "Inspect"

The screenshot shows a web browser window displaying the Wikipedia page for the 'List of tallest buildings'. The page content includes a sidebar with navigation links like 'Main page', 'Contents', and 'Tools', and a main article section about ranking criteria and the Burj Khalifa. A context menu is open over the page content, with the 'Inspect' option highlighted by a red arrow. The URL in the address bar is https://en.wikipedia.org/wiki/List_of_tallest_buildings.

Inspecting individual elements and the live HTML tree

Example

- the Web Developer Tools window pops up
- corresponding part in the HTML tree is highlighted
- you can hover over other parts of the code to get an instant view of the page at that position
- click on arrows to expand/hide tags



Summary

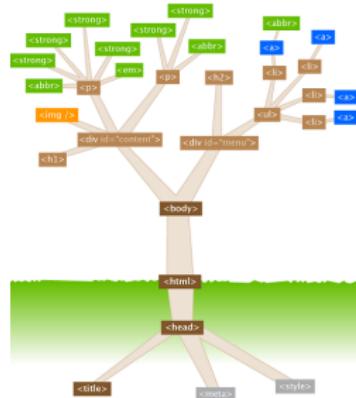
Summary

When to inspect the complete page source

- check whether data is in static source code
- for small HTML files: understand structure
- count tables

When to inspect individual elements using the Web Inspector Tools

- almost always
- particularly useful to construct XPath/CSS selector expressions
- to monitor dynamic changes in the DOM tree (see later)



Source: <http://watershedcreative.com/naked/html-tree.html>

Campus Luzern

A Primer to Web Scraping with R

XPath, Part I

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

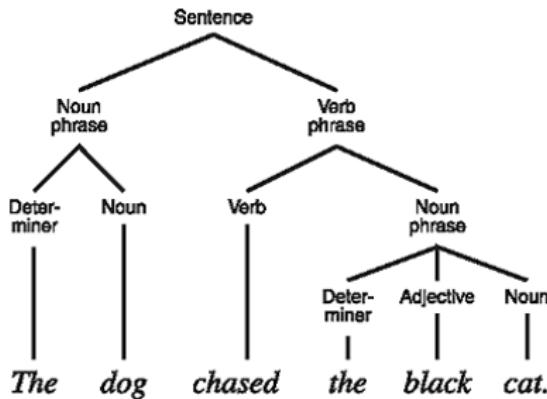
Accessing the HTML tree with R

Accessing the HTML tree with R

- HTML documents are human-readable
- HTML tags structure the document
- **web user perspective:** the browser interprets the code and renders the page
- **web scraper perspective:** use the tags to locate information; document has to be parsed first

Parsing

Parsing originally describes the syntactic analysis of text according to grammatical rules; analysis of the relationship between single parts of text. In programming, the input has to be interpreted (e.g., by R) to process the command.



Tools

- the `xml2` package allows us to parse XML-style documents
- the `rvest` package, which we will mainly use for scraping, wraps the `xml2` package, so we rarely have to load it manually
- HTML is a "flavor" of XML, so we can use the package to parse HTML
- one high-level function: `read_html()`
- `read_html()` represents the HTML in a list-style fashion
- we could also import HTML files via `readLines()`, but this is not parsing—the document's structure is not retained

HTML parsing with R

Parsing a website is straightforward

R code

```
1 library(rvest)  
2 parsed_doc <- read_html("https://google.com")
```

end

Functions to inspect the parsed document - better use the browser instead

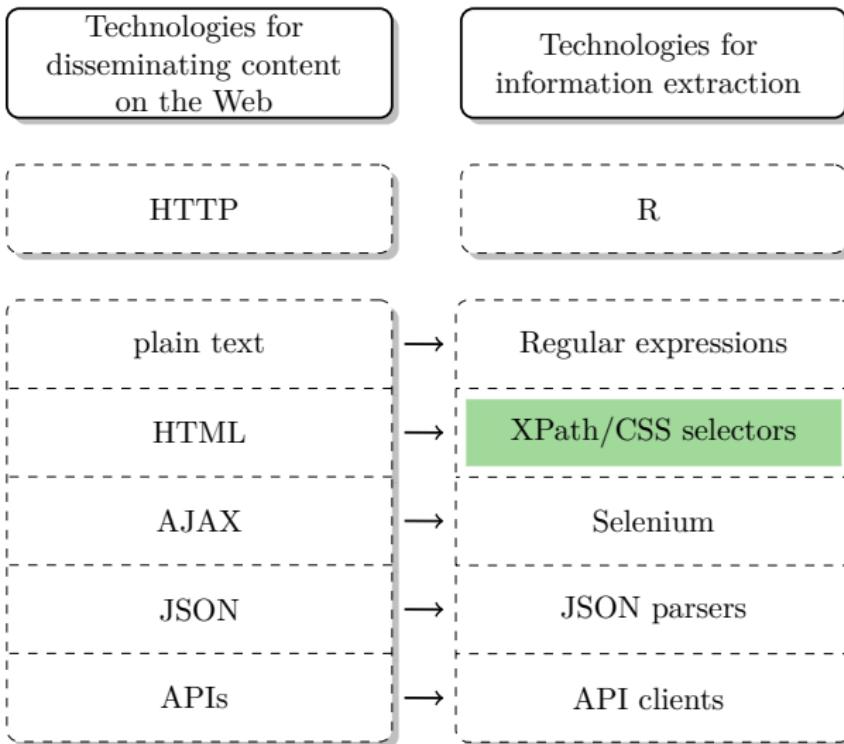
R code

```
3 html_structure(parsed_doc)  
4 as_list(parsed_doc)
```

end

XPath

Technologies of the World Wide Web



What's XPath?

Definition

- XML Path language, a W3C standard
- query language for XML-based documents (→ HTML)
- access node sets and extract content

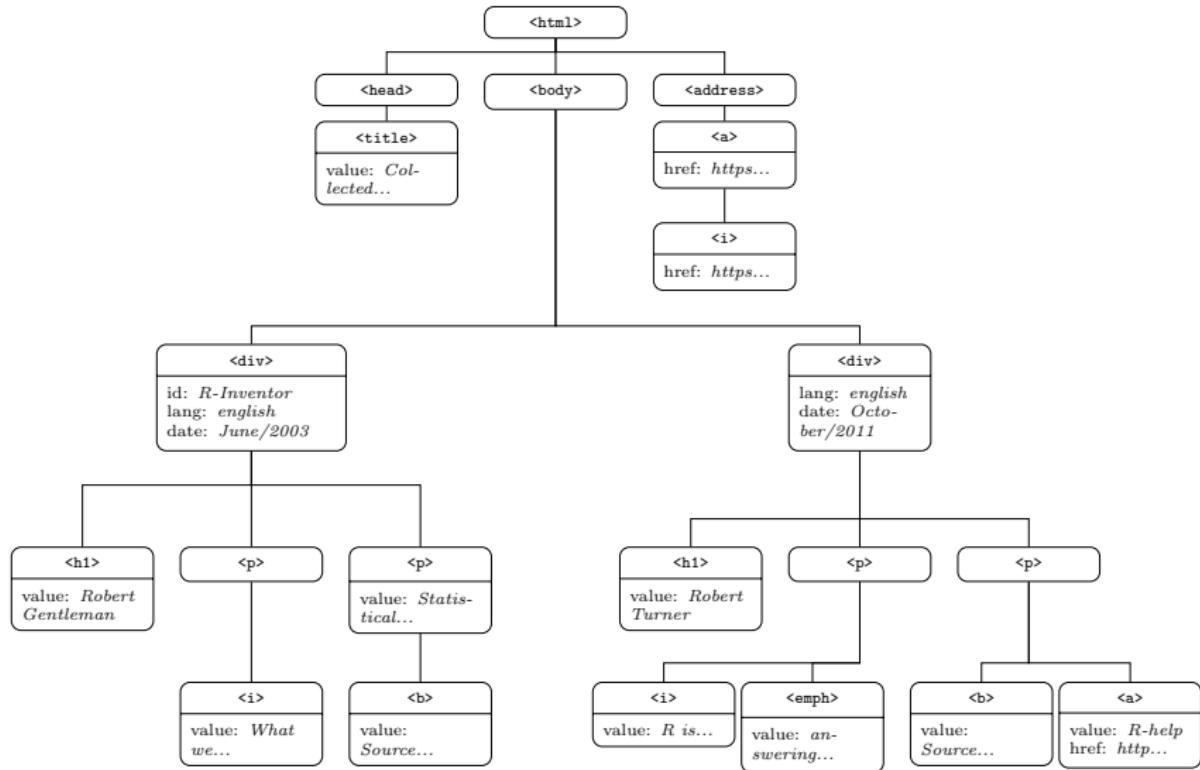
Why XPath for web scraping?

- source code of webpages (HTML) structures both layout and content
- not only content, but context matters!
- enables us to extract content based on its location in the document and (usually) regardless of its shape

Example

```
1 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
2 <html> <head>
3 <title>Collected R wisdoms</title>
4 </head>
5 <body>
6 <div id="R Inventor" lang="english" date="June/2003">
7   <h1>Robert Gentleman</h1>
8   <p><i>'What we have is nice, but we need something very different'</i></p>
9   <p><b>Source: </b>Statistical Computing 2003, Reisenburg</p>
10 </div>
11 <div lang="english" date="October/2011">
12   <h1>Rolf Turner</h1>
13   <p><i>'R is wonderful, but it cannot work magic'</i> <br><emph>answering a
14     request for automatic generation of 'data from a known mean and 95% CI'</
15     emph></p>
16   <p><b>Source: </b><a href="https://stat.ethz.ch/mailman/listinfo/r-help">R-help</
17     a></p>
18 </div>
19 </body>
20 <address><a href="http://www.r-datacollection.com"><i>The book homepage</i><a/></
21   address>
22 </html>
```

Example



Applying an XPath expression in R

- load package `rvest`
- parse document with `read_html()`
- query document with XPath expression using `html_nodes()`
- `rvest` can process XPath queries as well as CSS selectors
- in this course, we'll focus on XPath

R code

```
5 library(rvest)
6 parsed_doc <- read_html("../materials/fortunes.html")
7 html_nodes(parsed_doc, xpath = "//div[last()]/p/i")
{xml_nodeset (1)}
[1] <i>'R is wonderful, but it cannot work magic'</i>
```

end

The grammar of XPath

Grammar of XPath

Basic rules

1. we access nodes by writing down the hierarchical structure in the DOM that locates the node set of interest
2. a sequence of nodes is separated by slash symbols
3. the easiest localization of a node is given by the absolute path (but often not the most efficient one!)
4. apply XPath on document in R with the `html_nodes()` function

R code

```
8 html_nodes(parsed_doc, xpath = "//div[last()]/p/i")
{xml_nodeset (1)}
[1] <i>'R is wonderful, but it cannot work magic'</i>
```

end

Absolute vs. relative paths

- absolute paths start at the root node and follow the whole way down to the target node (with simple slashes, '/')
- relative paths skip nodes (with double slashes, '//')

R code

```
9 html_nodes(parsed_doc, xpath = "/html/body/div/p/i")
{xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>

10 html_nodes(parsed_doc, xpath = "//body//p/i")
{xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
```

end

When to use absolute, when relative paths?

- relative paths faster to write
- relative paths often more comprehensive (but less robust)
- relative paths consume more computing time, as the whole tree has to be parsed, but this is usually of less relevance for reasonably small documents

R code

```
11 html_nodes(parsed_doc, xpath = "//i")
{xml_nodeset (3)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
[3] <i>The book homepage</i>
```

end

Wildcard operator

- meta symbol *
- matches any node
- works only for one arbitrary node
- far less important than wildcards in regular expressions

R code

```
12 html_nodes(parsed_doc, xpath = "/html/body/div/*/i")
  {xml_nodeset (2)}
[1] <i>'What we have is nice, but we need something very different'</i>
[2] <i>'R is wonderful, but it cannot work magic'</i>
13 # this does not work:
14 html_nodes(parsed_doc, xpath = "/html/body/*/i")
  {xml_nodeset (0)}
```

end

Navigational operators ‘.’ and ‘..’

- . accesses nodes at the same level ('self axis')
- useful when working with predicates
- .. accesses nodes at a higher hierarchical level

R code

```
15 html_nodes(parsed_doc, xpath = "//title/..")
{xml_nodeset (1)}
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=
...

```

end

Pipe operator

- combines several paths

R code —

```
16 html_nodes(parsed_doc, xpath = "//address | //title")
{xml_nodeset (2)}
[1] <title>Collected R wisdoms</title>
[2] <address>\n<a href="http://www.r-datacollection.com"><i>The book hom
...
————— end
```

Summary

Summary

- XPath is a little language that lets you query specific parts of an XML-style document
- it has its own grammar (logic) and vocabulary
- in this session, you learned the basics of XPath
- in the next session, you will learn more advanced, powerful XPath expressions



Source: https://commons.wikimedia.org/wiki/File:Mozie_Law_path_junction_-_geograph.org.uk_-_1131.jpg (Andy Stephenson)

Campus Luzern

A Primer to Web Scraping with R

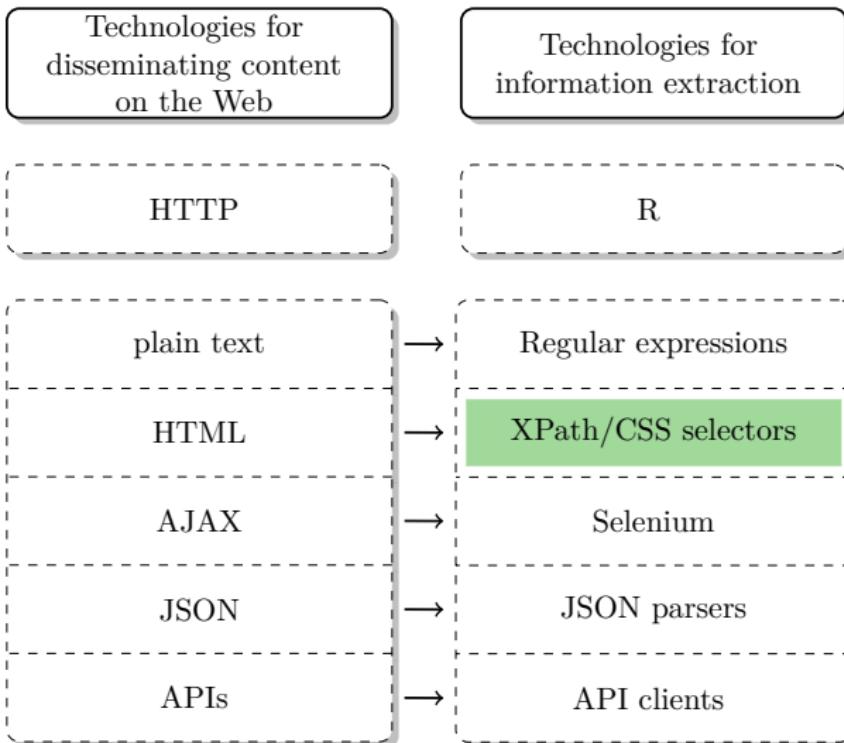
XPath, Part II

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

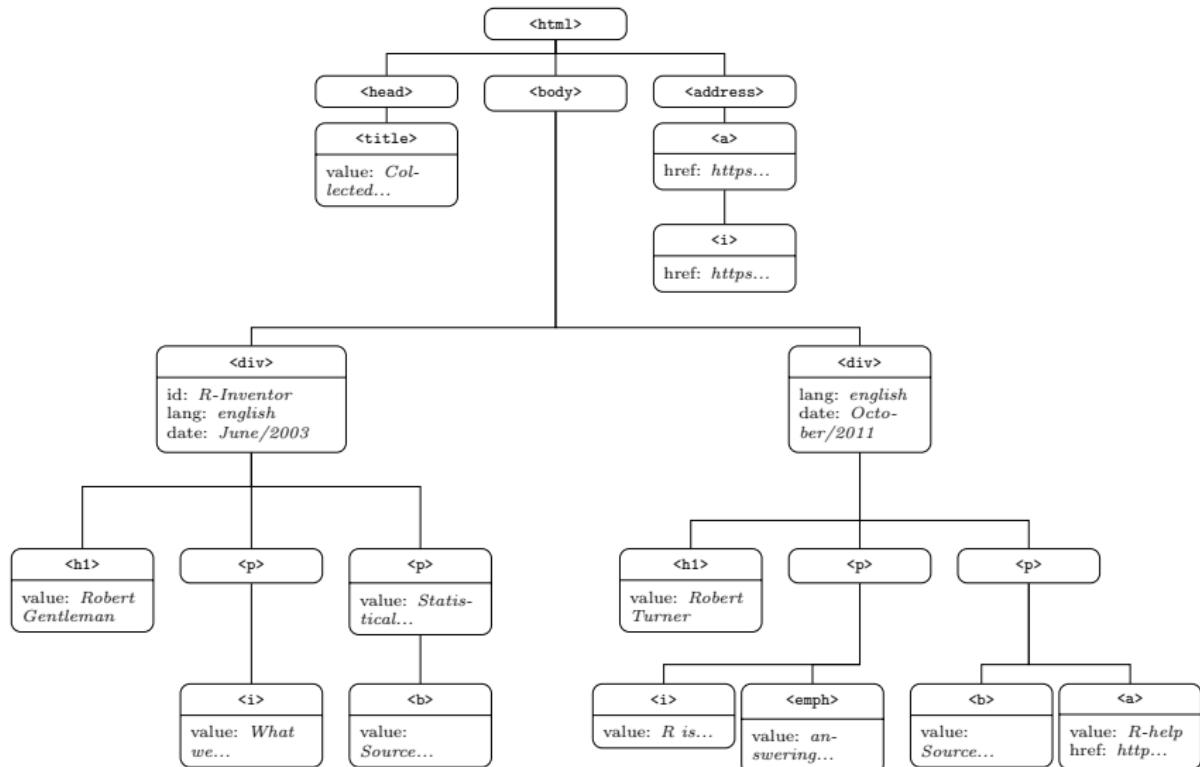
Technologies of the World Wide Web



Example

```
1 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
2 <html> <head>
3 <title>Collected R wisdoms</title>
4 </head>
5 <body>
6 <div id="R Inventor" lang="english" date="June/2003">
7   <h1>Robert Gentleman</h1>
8   <p><i>'What we have is nice, but we need something very different'</i></p>
9   <p><b>Source: </b>Statistical Computing 2003, Reisenburg</p>
10 </div>
11 <div lang="english" date="October/2011">
12   <h1>Rolf Turner</h1>
13   <p><i>'R is wonderful, but it cannot work magic'</i> <br><emph>answering a
14     request for automatic generation of 'data from a known mean and 95% CI'</
15     emph></p>
16   <p><b>Source: </b><a href="https://stat.ethz.ch/mailman/listinfo/r-help">R-help</
17     a></p>
18 </div>
19 </body>
20 <address><a href="http://www.r-datacollection.com"><i>The book homepage</i><a/></
21   address>
22 </html>
```

Example

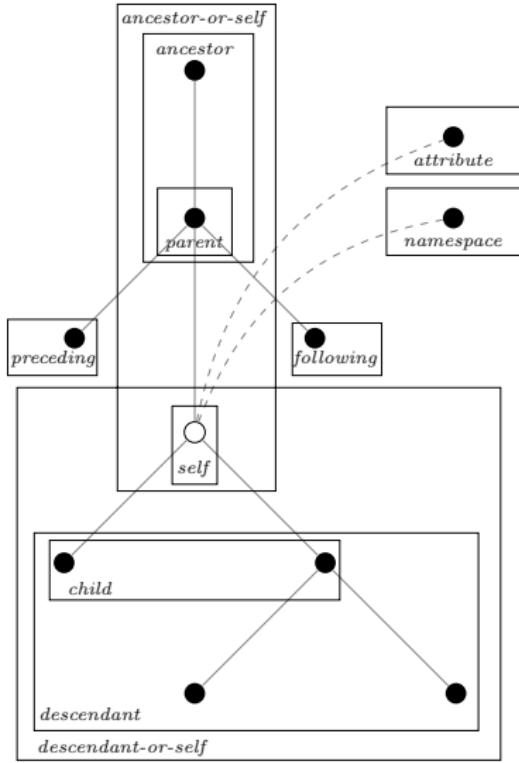


Node relations in XPath

'Family relations' between nodes

- the tools learned so far are sometimes not sufficient to access specific nodes without accessing other, undesired nodes as well
- relationship statuses are useful to establish unambiguity
- can be combined with other elements of the grammar
- basic syntax: `node1/relation::node2`
- we describe *relation* of `node2` to `node1`
- `node2` is to be extracted—we **always** extract the node at the end

Node relations in XPath



Node relations in XPath

Axis name	Result
ancestor	all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	all ancestors of the current node and the current node itself
attribute	all attributes of the current node
child	all children of the current node
descendant	all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	all descendants of the current node and the current node itself
following	everything in the document after the closing tag of the current node
following-sibling	all siblings after the current node
namespace	all namespace nodes of the current node
parent	the parent of the current node
preceding	all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	all siblings before the current node
self	the current node

Node relations in XPath

Example: access the `<div>` nodes that are ancestors to an `<a>` node:

R code

```
1 html_nodes(parsed_doc, xpath = "//a/ancestor::div")
{xml_nodeset (1)}
[1] <div lang="english" date="October/2011">\n    <h1>Rolf Turner</h1>\n    ...

```

end

Another example: Select all `<h1>` nodes that precede a `<p>` node:

R code

```
2 html_nodes(parsed_doc, xpath = "//p/preceding-sibling::h1")
{xml_nodeset (2)}
[1] <h1>Robert Gentleman</h1>
[2] <h1>Rolf Turner</h1>

```

end

Predicates

Predicates

- Conditions based on a node's features (`true/false`)
- applicable to a variety of features: name, value, attribute
- basic syntax:

node[predicate]

Commands in predicates

Function	Returns...
<code>name(<node>)</code>	name of <code><node></code> or the first node in a node set
<code>text(<node>)</code>	value of <code><node></code> or the first node in a node set
<code>@attribute</code>	value of a node's <i>attribute</i>
<code>string-length(str1)</code>	length of <code>str1</code> . If there is no string argument, it length of the string value of the current node
<code>translate(str1, str2, str3)</code>	<code>str1</code> by replacing the characters in <code>str2</code> with the characters in <code>str3</code>
<code>contains(str1,str2)</code>	TRUE if <code>str1</code> contains <code>str2</code> , otherwise FALSE
<code>starts-with(str1,str2)</code>	TRUE if <code>str1</code> starts with <code>str2</code> , otherwise FALSE
<code>substring-before(str1,str2)</code>	start of <code>str1</code> before <code>str2</code> occurs in it
<code>substring-after(str1,str2)</code>	remainder of <code>str1</code> after <code>str2</code> occurs in it
<code>not(arg)</code>	TRUE if the Boolean value is FALSE, and FALSE if the boolean value is TRUE
<code>local-name(<node>)</code>	name of the current <code><node></code> or the first node in a node set – without the namespace prefix
<code>count(<node>)</code>	count of a nodeset <code><node></code>
<code>position(<node>)</code>	index position of <code><node></code> that is processed
<code>last()</code>	number of items in the processed node list <code><node></code>

Predicates

Numeric predicates indicate positions, counts, etc.

Example: Select all first `<p>` nodes that are children of a `<div>` node:

R code

```
3 html_nodes(parsed_doc, xpath = "//div/p[position()=1]")
{xml_nodeset (2)}
[1] <p><i>'What we have is nice, but we need something very different'</i>
...
[2] <p><i>'R is wonderful, but it cannot work magic'</i> <br><emph>answe
...

```

end

Further examples:

R code

```
4 html_nodes(parsed_doc, xpath = "//div/p[last()-1]")
5 html_nodes(parsed_doc, xpath = "//div[count(./*)>2]")
6 html_nodes(parsed_doc, xpath = "//*[string-length(text())>50]")

```

end

Textual predicates

- describe text features
- applicable on: node name, content, attributes, attribute values
- XPath 2.0 supports (simplified) regex, however, R (the **rvest** package and the underlying **xml2** package) does not support it

Example: Select all `<div>` nodes that contain an attribute named '`October/2011`':

R code

```
7 html_nodes(parsed_doc, xpath = "//div[@date='October/2011'])  
{xml_nodeset (1)}  
[1] <div lang="english" date="October/2011">\n    <h1>Rolf Turner</h1>\n    ...
```

end

Partial matching

- rudimentary string matching
- 'content contains' (`contains()`), 'content begins with' (`starts-with()`), 'content ends with' (`ends-with()`), 'content contains after split' (`substring-after()`)

R code

```
8 html_nodes(parsed_doc, xpath = "//*[contains(text(), 'magic')]")
{xml_nodeset (1)}
[1] <i>'R is wonderful, but it cannot work magic'</i>
```

end

Further examples:

R code

```
9 html_nodes(parsed_doc, xpath = "//div[starts-with(./@id, 'R')]")
10 html_nodes(parsed_doc, xpath = "//div[substring-after(./@date, '/') 
='2003']//i")
```

end

Content extraction

Extraction of text and attributes

Extraction

- until now: query of complete nodes
- common scenario: only parts of the node are interesting, e.g., content (value)
- additional extraction operations from a selected node set possible with additional extractor functions

Function	Argument	Return value
<code>html_text</code>		node value
<code>html_attr</code>	<code>name</code>	node attribute
<code>html_attrs</code>		(all) node attributes
<code>html_name</code>	<code>trim</code>	node name
<code>html_children</code>		node children

Extraction of node elements

Values/text

R code

```
11 html_nodes(parsed_doc, xpath = "//title") %>% html_text()  
[1] "Collected R wisdoms"
```

end

Attributes

R code

```
12 html_nodes(parsed_doc, xpath = "//div") %>% htmlAttrs()  
[[1]]  
      id          lang         date  
"R Inventor"    "english"   "June/2003"
```

[[2]]

lang	date
"english"	"October/2011"

end

Extraction of node elements

Attribute values

R code —

```
13 html_nodes(parsed_doc, xpath = "//div") %>% html_attr("lang")
[1] "english" "english"
```

— end

A silver lining

Do I really have to construct XPath expressions all by my own?

No!



XPath creator tools

- *Selectorgadget*: <http://selectorgadget.com/>. Browser plugin that constructs XPath statements via a point-and-click approach. The generated expressions are not always efficient though
- *Web Developer Tools*: internal browser functionality which return XPath statements for selected nodes
- you will learn how to use these tools in upcoming sessions

Campus Luzern

A Primer to Web Scraping with R

Scraping HTML Tables

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

HTML tables are everywhere

Purchased Equipments (June, 2006)			
Item Num#	Item Picture	Item Description	Price
1.		Shipping, Handling, Installation, etc.	Expense
		IBM Clone Computer	\$ 400.00
2.		Shipping, Handling, Installation, etc.	\$ 20.00
		1GB RAM Module for Computer	\$ 50.00
Purchased Equipments (June, 2006)			

DATES	POLLSTER	GRADE	SAMPLE	WEIGHT	APPROVE/DISAPPROVE		ADJUSTED
					Approve	Disapprove	
* DEC. 28-30	Gallup	B-	1,500 A	cell	1.03	40%	55%
* DEC. 26-28	Rasmussen Reports/Pulse Opinion Research	C+	1,500 LV	cell	0.85	45%	53%
* DEC. 24-28	Ipsos	A-	1,510 A	cell	2.01	37%	58%
* DEC. 23-27	Gallup	B-	1,500 A	cell	0.58	38%	56%
* DEC. 24-26	YouGov	B	1,500 A	cell	1.13	38%	52%

Built	Building	City	Country	Roof	Floors	Pinnacle	Current status
1870	Equitable Life Building	New York City		043 m	142 ft	8	Destroyed by fire in 1912
1889	Auditorium Building	Chicago		062 m	269 ft	17	Standing
1890	New York World Building	New York City		094 m	309 ft	20	Demolished in 1955
1894	Philadelphia City Hall	Philadelphia		155.8 m	511 ft	9	Standing
1908	Singer Building	United States		187 m	612 ft	47	Demolished in 1968
1909	Met Life Tower			213 m	700 ft	50	Standing
1913	Woolworth Building			241 m	792 ft	57	Standing
1930	40 Wall Street			70	283 m	927 ft	Standing
1930	Chrysler Building			282.9 m	927 ft	77	Standing
1931	Empire State Building			381 m	1,250 ft	102	Standing
1972	World Trade Center (North Tower)			417 m	1,368 ft	110	Destroyed in 2001 in the September 11 attacks
1974	Willis Tower (formerly Sears Tower)	Chicago		442 m	1,450 ft	108	Standing
1996	Petronas Towers	Kuala Lumpur	Malaysia	379 m	1,242 ft	88	Standing
2004	Taipei 101	Taipei	Taiwan	449 m	1,474 ft	101	Standing
2010	Burj Khalifa	Dubai	United Arab Emirates	828 m	2,717 ft	163	Standing

Recall: HTML tables share the same basic syntax

Syntax

- HTML tables are easy to spot in the wild
- just look for `<table>` tags

Example

```
1 <table>
2   <tr> <th>Rank</th> <th>Nominal GDP</th> <th>Name</th> </tr>
3   <tr> <th></th> <th>(per capita, USD)</th> <th></th> </tr>
4   <tr> <td>1</td> <td>170,373</td> <td>Lichtenstein</td> </tr>
5   <tr> <td>2</td> <td>167,021</td> <td>Monaco</td> </tr>
6   <tr> <td>3</td> <td>115,377</td> <td>Luxembourg</td> </tr>
7   <tr> <td>4</td> <td>98,565</td> <td>Norway</td> </tr>
8   <tr> <td>5</td> <td>92,682</td> <td>Qatar</td> </tr>
9 </table>
```

Scraping HTML tables in R

Scraping HTML tables in R

A neat `rvest` solution

- exactly because scraping tables is an easy and repetitive task, there is a dedicated function for it (literally from the documentation):

```
html_table(x, header = NA, trim = TRUE, fill = FALSE, dec  
          = ".")
```

<code>x</code>	A node, node set or document
<code>header</code>	Use first row as header? If <code>NA</code> , will use first row if it consists of <code><th></code> tags
<code>trim</code>	Remove leading and trailing whitespace within each cell?
<code>fill</code>	If <code>TRUE</code> , automatically fill rows with fewer than the maximum number of columns with <code>NAs</code>
<code>dec</code>	The character used as decimal mark

Example

Example

- source: https://en.wikipedia.org/wiki/List_of_human_spaceflights

Wikipedia - List of human spaceflights - W | Sicher https://en.wikipedia.org/wiki/List_of_human_spaceflights

List of human spaceflights

From Wikipedia, the free encyclopedia

For a list of spaceflights with human crews organised by program, see [List of human spaceflights by program](#). For a list of spaceports with achieved launches of humans to space, see [Spaceport](#).

These chronological lists include all crewed spaceflights that reached an altitude of at least 100 km (the FAI definition of spaceflight, see [Kármán line](#)), or were launched with that intention but failed. The USA has adopted a slightly different definition of spaceflight, requiring an altitude of only 50 miles (80 km). During the 1960s, 13 flights of the US X-15 rocket plane met the USA criteria, but only two met the FAI's. These lists include only the later two flights; see the list of [highest X-15 flights](#) for all 13. As of 29 January 2017, there have been 314 manned spaceflights that reached 100 km or more in altitude (316 including two failed attempts), 8 of which were [sub-orbital spaceflights](#).

To date, there have been four [fatal missions](#) in which 18(19 if you count the US Air Force Definition) astronauts died.

Contents [hide]

- 1 Summary
- 2 Detailed lists
- 3 Timeline
- 4 See also

Summary [edit]

	Russia USSR	United States	China	Total
1961–1970	16	25	41	
1971–1980	30	8	38	
1981–1990	*25	*38	*63	
1991–2000	20	63	83	
2001–2010	24	34	3	61
2011–2020	24	3	3	30
Total	*139	*171	6	*316

Chart of all humans launched into space as of December the 31st, 2016. Including unsuccessful launches (STS-51-L and Soyuz T-10-1).

*Includes the two failed launches of STS-51-L and Soyuz T-10-1.

Apollo 7 heads into orbit with its crew of three, 1968



Example

Example

- source: https://en.wikipedia.org/wiki/List_of_human_spaceflights
- not entirely clean table: some cells empty, images, links

Wikipedia - List of human spaceflights - W | Sicher https://en.wikipedia.org/wiki/List_of_human_spaceflights

List of human spaceflights

From Wikipedia, the free encyclopedia

For a list of spaceflights with human crews organised by program, see [List of human spaceflights by program](#). For a list of spaceports with achieved launches of humans to space, see [Spaceport](#).

This chronological lists include all crewed spaceflights that reached an altitude of at least 100 km (the FAI definition of spaceflight, see [Kármán line](#)), or were launched with that intention but failed. The USA has adopted a slightly different definition of spaceflight, requiring an altitude of only 50 miles (80 km). During the 1960s, 13 flights of the US X-15 rocket plane met the US criteria, but only two met the FAI's. These lists include only the later two flights; see the list of [highest X-15 flights](#) for all 13. As of 29 January 2017, there have been 314 manned spaceflights that reached 100 km or more in altitude (316 including two failed attempts), 8 of which were [sub-orbital spaceflights](#).

To date, there have been four [fatal missions](#) in which 18(19 if you count the US Air Force Definition) astronauts died.

Contents [hide]

- 1 Summary
- 2 Detailed lists
- 3 Timeline
- 4 See also

Summary [edit]

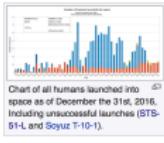
	Russia USSR	United States	China	Total
1961–1970	16	25	41	
1971–1980	30	8	38	
1981–1990	*25	*38	*63	
1991–2000	20	63	83	
2001–2010	24	34	3	61
2011–2020	24	3	3	30
Total	*139	*171	6	*316

*Includes the two failed launches of STS-51-L and Soyuz T-10-1.

Apollo 7 heads into orbit with its crew of three, 1968



Chart of all humans launched into space as of December the 31st, 2016. Including unsuccessful launches (STS-51-L and Soyuz T-10-1).



Example

Example

- source: https://en.wikipedia.org/w/index.php?title=List_of_human_spaceflights&oldid=9600000
- not entirely clean table: some cells empty, images, links
- HTML code is straightforward

The screenshot shows a browser window displaying the Wikipedia page for "List of human spaceflights". The page includes a table of launches from 1961 to 2010, with a note about failed launches. Below the table is a section titled "Detailed lists". The developer tools are open, specifically the Elements tab, which shows the HTML structure of the table. The table has a class of "wikitable" and a style of "text-align:right". It consists of a thead section with four columns (Russia/USSR, United States, China, Total) and a tbody section with 10 rows of data. The tbody also contains a note about failed launches. The developer tools also show the CSS for the table and its child elements.

Russia USSR	United States	China	Total
1961–1970	16	25	41
1971–1980	30	8	38
1981–1990	*25	*38	*63
1991–2000	20	63	83
2001–2010	24	34	3 61
2011–2020	24	3	3 30
Total	139	171	6 *216

Including unsuccessful launches (STS-51-L and Soyuz T-10-1).

Includes the two failed launches of STS-51-L and Soyuz T-10-1.

Detailed lists [edit]

The [Salyut](#) series, [Skylab](#), [Mir](#), [ISS](#), and [Tiangong](#) series space stations, with various of these flights docked in orbit, are not listed separately here. See the detailed lists (links above) for information.

Missions which were intended to reach space but which failed to do so are listed in italics, and fatal missions are marked with asterisk.

1961	Vostok 1 — Mercury-Redstone 3 — Mercury-Redstone 4 — Vostok 2
1962	Mercury-Atlas 6 — Mercury-Atlas 7 — Vostok 3 — Vostok 4 — Mercury-Atlas 8
1963	Mercury-Atlas 9 — Vostok 5 — Vostok 6 — X-15 Flight 90 — X-15 Flight 91

Example

Example

- source: https://en.wikipedia.org/wiki/List_of_human_spaceflights
- not entirely clean table: some cells empty, images, links
- HTML code is straightforward

```
▼<table class="wikitable" style="text-align:right;">
  ▼<tbody>
    ▶<tr>...</tr>
    ▶<tr>...</tr>
    ▼<tr>
      ▶<td>...</td>
      <td>30</td>
      <td>8</td>
      <td></td>
      <td>38</td>
    </tr>
    ▶<tr>...</tr>
    ▶<tr>...</tr>
    ▶<tr>...</tr>
    ▶<tr>...</tr>
    ▶<tr>...</tr>
  </tbody>
</table>
```

Example

Scraping the table with R

R code

```
1 library(rvest)
2 url_p <- read_html("https://en.wikipedia.org/wiki/List_of_human_
spaceflights")
3 tables <- html_table(url_p, header = TRUE, fill = TRUE)
4 spaceflights <- tables[[1]]
5 spaceflights
```

	Russia	Soviet Union	United States	China	Total	
1	1961-1970		16	25	NA	41
2	1971-1980		30	8	NA	38
3	1981-1990		*25	*38	NA	*63
4	1991-2000		20	63	NA	83
5	2001-2010		24	34	3	61
6	2011-2020		28	3	3	34
7	Total		*143	*171	6	*320

end

Summary

Summary

- scraping HTML tables with R is straightforward
- the high-level `html_table()` function is easy to use and generally robust
- sometimes, HTML tables are a bit more complex. You might encounter:
 - tables where cells span multiple rows
 - tables where headers are not in the first row
- in such cases, you might want to check out the `htmltab` package, which is more complex to use but also provides more flexibility



Campus Luzern

A Primer to Web Scraping with R

Using SelectorGadget

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

SelectorGadget

Scraping webpages

- the most cumbersome part of web scraping (data tidying aside) is the construction of XPath expressions that match the components of a page you want to extract
- it will take a couple of scraping projects until you'll truly have mastered XPath

A much-appreciated helper

- **SelectorGadget** is a JavaScript browser plugin that constructs XPath statements (or CSS selectors) via a point-and-click approach
- it is available here: <http://selectorgadget.com/> (there's also a Chrome extension)
- the tool is magic and you will love it

What does SelectorGadget do?

What does SelectorGadget do?

On the surface

1. you activate the tool on any webpage you want to scrape
2. based on your selection of components, the tool learns about your desired components and generates an XPath expression (or CSS selector) for you

Under the hood

- based on your selection(s), the tool looks for similar elements on the page
- the underlying algorithm, which draws on Google's diff-match-patch libraries, focuses on CSS characteristics, such as tag names and `<div>` and `` attributes

Installation and use

Installation

Go to <http://selectorgadget.com/> and drag the link provided at the end of the page ("Or drag this link...") to your bookmark bar.

Use

1. Open the webpage you want to scrape
2. Click on the SelectorGadget bookmarklet
3. Click on the elements you want to extract. Manually selected elements will turn **green**. Elements that match the selector will be highlighted **yellow**. De-select the elements you do not want to extract - they will turn **red**.
4. Repeat step 3 until the marked set of elements matches the set you want to extract
5. Click on the "XPath" button in the console
6. Copy the generated XPath expression and process it with **rvest**

Example

Example

Example

- source: <https://nytimes.com>

The screenshot shows the homepage of The New York Times on Tuesday, March 21, 2017. At the top, there's a navigation bar with links for 'SUBSCRIBE NOW' and 'LOG IN'. Below it is the main masthead with the title 'The New York Times'. A banner headline reads 'Fresh Worries on Russia for Trump's Weary Defenders' by Glenn Thrush and Maggie Haberman. To the right is a map of the United States with a color scale from light orange ('LESS') to dark orange ('MORE') representing support for CO2 limits on power plants. Below the map, a section titled 'How Americans Think About Climate Change' discusses public opinion on global warming. On the right side, there are columns for 'The Opinion Pages' featuring articles like 'Comey's Haunting News on Trump and Russia' and 'Gorsuch Faces the Senate'. There are also sections for 'OPINION CONTRIBUTORS' and 'THE CROSSWORD'.

Example

Example

- source: <https://nytimes.com>
- goal: scrape all headlines

The screenshot shows the homepage of The New York Times on Tuesday, March 21, 2017. The main headline is "Fresh Worries on Russia for Trump's Weary Defenders" by Glenn Thrush and Maggie Haberman. Below it, a map of the United States titled "Support CO2 limits on power plants" shows a gradient from green (less support) to red (more support). A sub-headline "How Americans Think About Climate Change" is also present. The right sidebar features various opinion pieces and puzzles.

The New York Times

Tuesday, March 21, 2017 | [Today's Paper](#) | [Video](#) | [54°F](#) | [DAX +0.02%](#)

World U.S. Politics N.Y. Business Opinion Tech Science Health Sports Arts Style Food Travel Magazine T Magazine Real Estate ALL

THE 45TH PRESIDENT

Fresh Worries on Russia for Trump's Weary Defenders

By GLENN THRUSH and MAGGIE HABERMAN 9:24 AM ET

The obsessiveness and ferocity of President Trump's pushback against the Russia allegations, often untempered from fact, are making an uncertain situation worse.

"The tweets make it much more difficult for us as we try to build a case against these leakers," said Representative Peter T. King, a New York Republican who sits on the

Support CO2 limits on power plants

LESS → MORE

How Americans Think About Climate Change

Americans overwhelmingly believe that global warming is happening, and that carbon emissions should be scaled back. But fewer are sure that the changes will harm them.

OPINION

Cormer's Haunting News on Trump and Russia

Possible election collusion makes a special prosecutor and a select congressional committee crucial.

Gorsuch Faces the Senate

Here's a good question for Judge Gorsuch: Why are you here?

U.N. Accepts Blame but Dodges the Bill in Haiti

THE OPINION PAGES

How to Fix Health Care

The plan to replace Obamacare has won few fans on the right or left. Here's a proposal to unite these catastrophic coverage for all.

Brooks: The Unifying American Story

- Leonhardt: All the President's Lies
- "W" and the Art of Redemption

THE CROSSWORD » Play Today's Puzzle

Example

Example

1. click on SelectorGadget bookmarklet (not shown)
2. click on first headline

The screenshot shows the homepage of The New York Times. At the top, there's a navigation bar with links for ENGLISH, 中文 (CHINESE), ESPAÑOL, SUBSCRIBE NOW, and LOG IN. Below the header, the main news area features a large headline "Fresh Worries on Russia for Trump's Weary Defenders" by BRIAN KARLSSON, dated MARCH 21, 2017, 8:24 AM ET. The article discusses the Trump administration's pushback against Russia. To the right of the article is a map titled "How Americans Think About Climate Change", which uses color coding (purple, green) across the US states to represent different views on global warming. Below the map, a sub-headline reads "Americans overwhelmingly believe that global warming is happening". On the far right, there are several sidebar columns: "EDITORIALS", "GORSUCH FACES THE SENATE", "TIMES PODCAST", "THE CROSSWORD", and "The Opinion Pages". At the bottom of the page, there are buttons for "Clear", "Toggle Position", "XPath", and "Help".

Example

Example

1. click on SelectorGadget bookmarklet (not shown)
2. click on first headline

The screenshot shows a browser window displaying The New York Times homepage. A red arrow points from the word "Selection" in the top left corner to the "Selection" button in the SelectorGadget toolbar at the bottom of the page. Another red arrow points from the headline "Fresh Worries on Russia for Trump's Weary Defenders" to the "SelectorGadget console" text area. A third red arrow points from the "Selector" button in the toolbar to the "No valid path found." message in the console. A fourth red arrow points from the "X" button in the toolbar to the "Exit" link in the top right corner of the console. The SelectorGadget toolbar also includes buttons for "Console", "Reset", "Return XPaths", "Clear", "Toggle Position", "XPath", "Help", and "X".

Example

Example

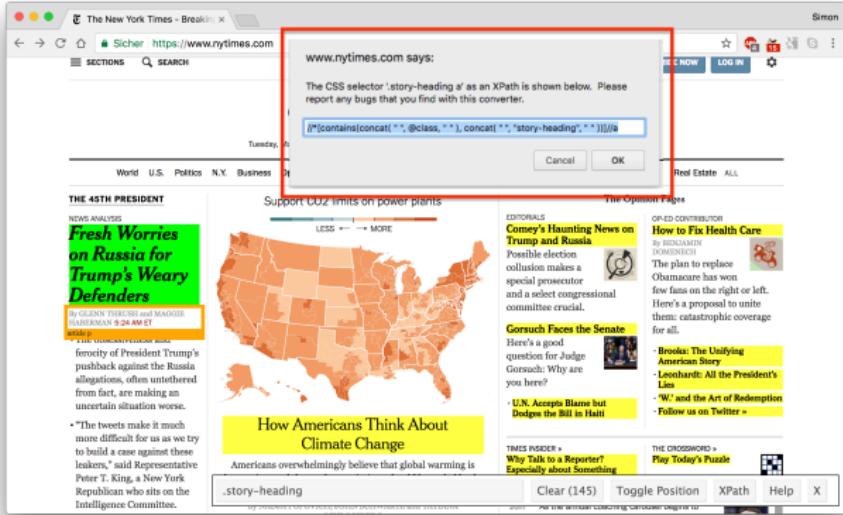
1. click on SelectorGadget bookmarklet (not shown)
2. click on first headline
3. all relevant headlines selected

The screenshot shows the homepage of The New York Times. At the top, there's a navigation bar with links for ENGLISH, 中文 (CHINESE), ESPAÑOL, SUBSCRIBE NOW, and LOG IN. Below the header, the main news area features a large map of the United States where states are colored according to support for CO2 limits on power plants, with a slider from LESS to MORE. To the left of the map is a headline: "Fresh Worries on Russia for Trump's Weary Defenders" by GLENAN STUCKEY and MAGGIE HABERMAN, dated 9:24 AM ET. Below this is a snippet: "The tweets make it much more difficult for us as we try to build a case against these leakers," said Representative Peter T. King, a New York Republican who sits on the Intelligence Committee. To the right of the map, there's a section titled "How Americans Think About Climate Change" with a sub-headline: "Americans overwhelmingly believe that global warming is". Other visible headlines include "Gorsuch Faces the Senate", "U.N. Accepts Blame but Dodges the Bill in Haiti", and "Brooks: The Unifying American Story". On the far right, there are links for "How to Fix Health Care", "Editorials", "Op-Ed Contributors", "Times Insider", "The Crossword", and "Play Today's Puzzle". At the bottom, there are buttons for "story-heading", "Clear (145)", "Toggle Position", "XPath", "Help", and "X".

Example

Example

1. click on SelectorGadget bookmarklet (not shown)
2. click on first headline
3. all relevant headlines selected
4. click on "XPath"
5. copy proposed expression



Example

Now, we switch to R

R code

```
1 library(rvest)
2 url_p <- read_html("https://www.nytimes.com")
3 headlines <- html_nodes(url_p, xpath = '//*[@contains(concat( " ", @class
 , " " ), concat( " ", "story-heading", " " ))]//a') # we use single
quotation marks here to wrap around the expression!
4 headlines_raw <- html_text(headlines)
5 head(headlines_raw)
[1] "Fresh Worries on Russia for Trump's Weary Defenders"
[2] "F.B.I. Confirms Inquiry on Trump Team's Russia Ties"
[3] "Takeaways From the Hearing "
[4] "\n      Trump and the Russians: Links? No Links?"
[5] "G.O.P. Responds by Changing Subject"
[6] "U.S. Limits Devices on Foreign Airlines From 8 Countries"
```

end

Example

Let's clean the data

R code —

```
6 headlines_clean <- headlines_raw %>% str_replace_all("\n", "") %>% str_
trim()

7 length(headlines_clean)
[1] "Fresh Worries on Russia for Trump's Weary Defenders"
[2] "F.B.I. Confirms Inquiry on Trump Team's Russia Ties"
[3] "Takeaways From the Hearing"
[4] "Trump and the Russians: Links? No Links?"
[5] "G.O.P. Responds by Changing Subject"
[6] "U.S. Limits Devices on Foreign Airlines From 8 Countries"

8 str_detect(headlines_clean, "Trump") %>% table()
FALSE    TRUE
 133      12
```

end

Summary

So why did I bother you with learning XPath at all?

Caveats

- SelectorGadget is not perfect. Sometimes, the algorithm will fail
- starting from a different element sometimes (but not always!) helps
- often the generated expressions are unnecessarily complex and therefore difficult to debug
- in my experience, SelectorGadget works 70-80% of the times when scraping from static webpages
- you are also prepared for the remaining 20-30%!



Source: http://inspectorgadget.wikia.com/wiki/File:Inspector_Gadget_Thinking.png
(DANTHEMAN123)

Campus Luzern

A Primer to Web Scraping with R

The Scraping Workflow

Simon Munzert

Hertie School of Governance, Berlin

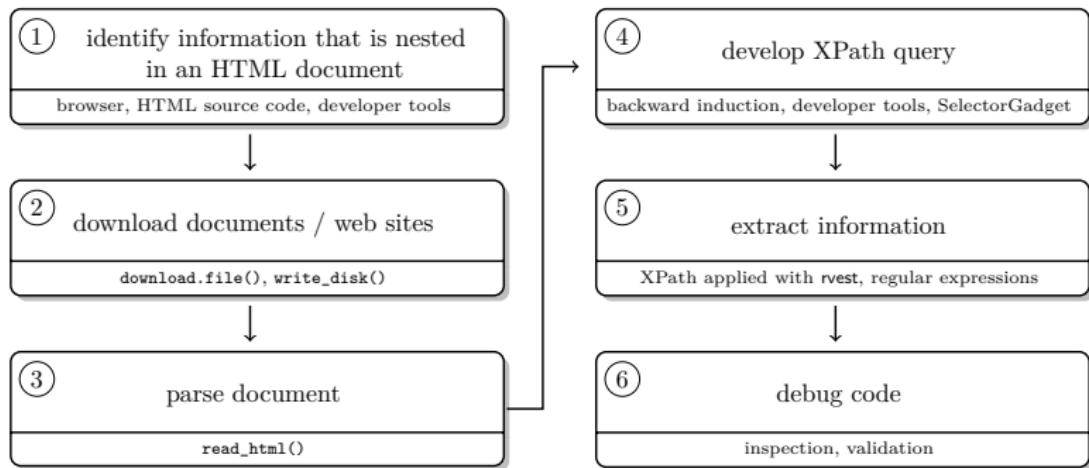
May 20-21, 2019

You have learned the main tools necessary to scrape static webpages with R

1. you are able to inspect HTML pages in your browser using the web developer tools
2. you are able to parse HTML into R with `rvest`
3. you are able to speak XPath
4. you are able to apply XPath expressions with `rvest`
5. you are able to tidy web data with your R skills and regular expressions

The scraping workflow

The scraping workflow



Downloading HTML files

Stay modest when accessing lots of data

- content on the web is publicly available, ...
- but accessing the data causes server traffic
- stay polite by querying resources as sparsely as possible

Two easy-to-implement practices

1. do not bombard the server with requests—and if you have to, do at a reasonable speed
2. download HTML files first, then parse

Downloading HTML files

R code

```
1 for (i in 1:length(list_of_urls)) {  
2   if (!file.exists(paste0(folder, file_names[i]))) {  
3     download.file(list_of_urls[i], destfile = paste0(folder, file_names[  
i]))  
4     Sys.sleep(runif(1, 1, 2))  
5   }  
6 }
```

end

Looping over a list of URLs

- `!file.exists()` checks whether a file does not yet exist in the local folder
- `download.file()` downloads the file to a folder; file name has to be specified
- `Sys.sleep()` suspends the execution of R code for a given time interval. Here: random interval between 1 and 2 seconds

Don't be a phantom

- downloading massive amounts of data may arouse attention from server administrators
- assuming that you've got nothing to hide, you should stay identifiable beyond your IP address

Two easy-to-implement practices

1. personally get in touch with website owners
2. use HTTP header fields `From` and `User-Agent` to provide information about yourself

Staying identifiable

R code —

```
7 url <- "http://a-totally-random-website.com"
8 session <- html_session(url, add_headers(From = "my@email.com", `User-
Agent` = R.Version()$version.string)))
9 headlines <- session %>% html_nodes(xpath = "p//a") %>% html_text()

```

end

The code snippet explained

- `rvest`'s `html_session()` creates a session object that responds to HTTP and HTML methods
- here, we provide our email address and the current R version as User-Agent information
- this will pop up in the server logs—the webpage administrator has the chance to easily get in touch with you

Summary

Summary

- the basic scraping workflow with R is straightforward
- with great power comes great responsibility: stay polite on the web when scraping lots of data!
- more complexity is added when you want to gather data from multiple websites, or when dynamic elements such as forms or JavaScript content is involved
- we will consider such cases in upcoming sessions



Campus Luzern

A Primer to Web Scraping with R

Scraping Multiple Pages

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

An advanced scraping scenario

Motivation

- until now, the toy examples were limited to single HTML pages
- often, we want to scrape data from multiple pages
- in such scenarios, automating the scraping process becomes **really** powerful
- also, the principles of polite scraping are more relevant

The scenario

Goal: examine download statistics of articles of the Journal of Statistical Software

- download HTML pages
- extract bibliometrical information

Tasks:

- identify relevant resources on <http://www.jstatsoft.org/>
- download HTML pages
- import them into R
- extract information via XPath



Scraping multiple pages with R

Step 1: Inspect the source

Procedure

- source:
<http://www.jstatsoft.org/>
- go to "Archives"



The screenshot shows the homepage of the Journal of Statistical Software. At the top right, there is a red box highlighting the "Archives" link in the navigation bar. The main content area features a large image of a person wearing a hard hat and safety glasses, possibly a statistician or engineer, working at a computer. Below the image, there is a "Recent Publications" section listing several academic papers with their PDF links. To the right of this section is a sidebar with various links and settings for users.

Recent Publications

- Enhancing Reproducibility and Collaboration via Management of R Package Cohorts
Gabriel Becker, Cory Bern, Robert Gentleman, Michael Lawrence
- A Recipe for Inference: Start with Causal Inference, Add Interference, Mix Well with R,
Bradley C. Saul, Michael G. Hudgens
- Robust Standard Error Estimators for Penalized Models: A Unifying Approach
Giovanni Millo
- ThresholdROC: Optimum Threshold Estimation Tools for Continuous Diagnostic Tests in R
Sara Perez-Jaime, Konstantinos Skaltsas, Natalia Pallantis, Josep L. Carrasco
- tacount: An R Package for Analysis of Count Time Series
Following Generalized Linear Models
Tobias Liboschik, Konstantinos Fokanas, Roland Fried
- vinMR: Generating Web-Based Visual Data Mining Tools with R
Toshiyuki Fujio
- trackerR: Infrastructure for Running and Cycling Data from GPS-Enabled Tracking Devices in R
Hannah Frick, Ismaela Kosmidis

Archives

Simon

Published by the Foundation for Open Access Statistics

Editor-in-chief: Bettina Grün, Torsten Hothorn, Edzer Pebesma, Achim Zeileis

ISBN 1548-7860; CODEN JSSOBK

Journal of Statistical Software

Home | Mission | Instructions for Authors | Style Guide | Editorial Board | Contact | Register

Current Volume | Archives | Search

OPEN JOURNAL SYSTEMS

Journal Help

USER

Username: Password:
 Remember me

FEEDS

RSS Atom

NOTIFICATIONS

View Subscribe

JOURNAL CONTENT

Search...
Search Scope: All

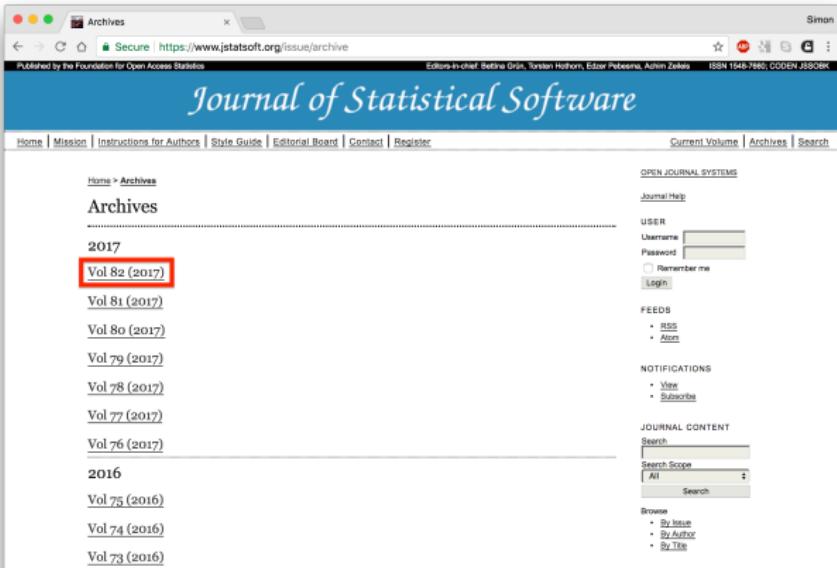
Browse

- By Issue
- By Author
- By Title

Step 1: Inspect the source

Procedure

- source:
<http://www.jstatsoft.org/>
- go to "Archives"
- inspect the most recent volume



The screenshot shows a web browser window for the Journal of Statistical Software. The URL is https://www.jstatsoft.org/issue/archive. The page title is "Journal of Statistical Software". The header includes "Published by the Foundation for Open Access Statistics", "Editor-in-chief: Bettina Grün, Torsten Hothorn, Edgar Pebesma, Achim Zeileis", "ISBN 1548-7660; CODEN JSSOBK", and a "Simon" user profile. The main content area is titled "Archives" and lists volumes from 2017 down to 2016. The 2017 section includes links for "Vol 82 (2017)" (which is highlighted with a red box), "Vol 81 (2017)", "Vol 80 (2017)", "Vol 79 (2017)", "Vol 78 (2017)", "Vol 77 (2017)", and "Vol 76 (2017)". The 2016 section includes links for "Vol 75 (2016)", "Vol 74 (2016)", and "Vol 73 (2016)". The right sidebar contains sections for "OPEN JOURNAL SYSTEMS", "USER" (with fields for Username, Password, Remember me, and Login), "FEEDS" (RSS and Atom), "NOTIFICATIONS" (View and Subscribe), "JOURNAL CONTENT" (Search, Search Scope: All, and a "Search" button), and "Browse" (By Issue, By Author, By Title).

Step 1: Inspect the source

Procedure

- source:
<http://www.jstatsoft.org/>
- go to "Archives"
- inspect the most recent volume
- inspect the first article



The screenshot shows a web browser displaying the *Journal of Statistical Software* website. The URL is <https://www.jstatsoft.org/issue/view/v082>. The page title is "Vol 82 (2017)". A red box highlights the first article in the table of contents:

Enhancing Reproducibility and Collaboration via Management of R Package Cohorts
Gerrit Becker, Cory Ball, Robert Gentleman, Michael Lawrence

Below the table of contents, there are several sidebar sections:

- OPEN JOURNAL SYSTEMS**
- Journal Help**
- USER**: Username: [redacted], Password: [redacted], Remember me, Login
- PDF**: RSS, Atom
- FEEDS**: PDF, RSS, Atom
- NOTIFICATIONS**: PDF, RSS, Atom
- JOURNAL CONTENT**: Search, Search Scope (All), Search
- Browse**: By Issue, By Author, By Title

Step 1: Inspect the source

Procedure

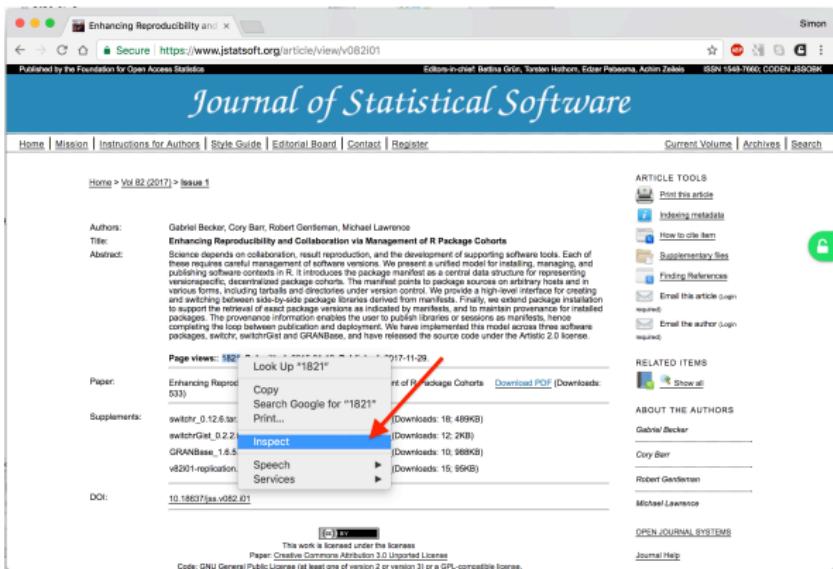
- source:
<http://www.jstatsoft.org/>
- go to "Archives"
- inspect the most recent volume
- inspect the first article

The screenshot shows a web browser displaying the *Journal of Statistical Software* website. The page title is "Enhancing Reproducibility and Collaboration via Management of R Package Cohorts" by Gabriel Becker, Cory Barr, Robert Gentleman, Michael Lawrence. The article was submitted on 2015-01-12 and published on 2017-11-29. It has 533 page views. The page includes links to the paper PDF (48KB), switchr_0.12.6.tar.gz (49KB), switchr_0.12.6.tar.gz.R (12KB), GRANBase_1.6.5.tar.gz (98KB), v8201-replication.zip (95KB), and DOI 10.18637/jss.v062.i01. There are also sections for Article Tools (Print this article, Indexing metadata, How to cite item, Supplementary files, Finding References, Email the article, Email the author) and Related Items (Show all). The page footer includes information about the Creative Commons Attribution 3.0 Unported License and Open Journal Systems.

Step 1: Inspect the source

Procedure

- source:
<http://www.jstatsoft.org/>
- go to "Archives"
- inspect the most recent volume
- inspect the first article
- inspect the page views element

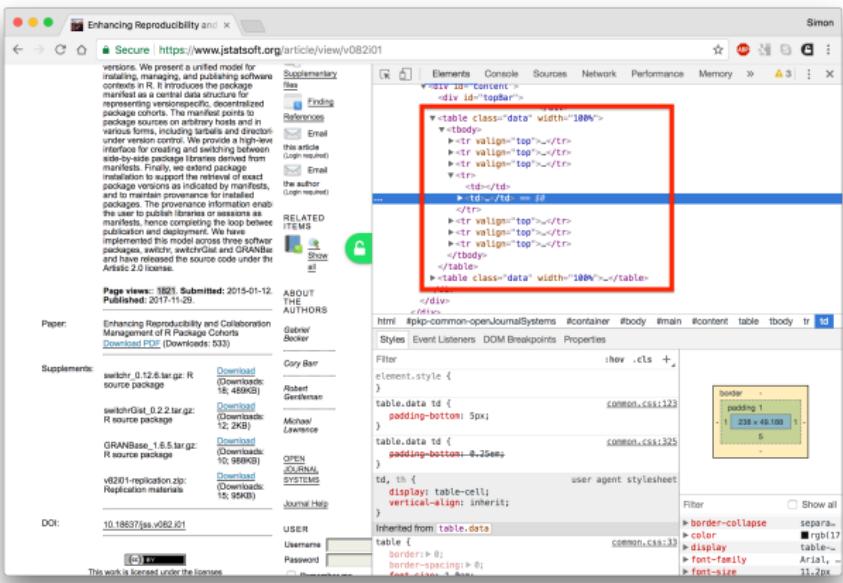


The screenshot shows a web browser displaying the *Journal of Statistical Software* website. The URL is <https://www.jstatsoft.org/article/view/v082i01>. The page title is "Enhancing Reproducibility and Collaboration via Management of R Package Cohorts". A context menu is open over the "Page views: 182" link, with the "Inspect" option highlighted by a red arrow. The menu also includes "Look Up '182!'" and "Print...". Other options like "Search Google for '182!'" and "Supplementary files" are visible but not highlighted.

Step 1: Inspect the source

Procedure

- source:
[http://www.jstatsoft.org/](http://www.jstatsoft.org)
- go to "Archives"
- inspect the most recent volume
- inspect the first article
- inspect the page views element
- it's in a table!



The screenshot shows a web browser displaying a JSTOR article page. The URL is https://www.jstatsoft.org/article/view/v082i01. The page content includes a sidebar with links like 'Supplemental files', 'References', 'Email this article', and 'Log in'. Below the sidebar, there's a section for 'RELATED ITEMS' with a 'Show all' link. The main content area contains a table with several rows of data. The browser's developer tools are open, showing the DOM structure. A red box highlights the table element in the DOM tree. A green box highlights the same table element in the main content area. The developer tools also show the CSS styles applied to the table.

Step 2: Develop a scraping strategy

Observations

- getting the information out of the table will be straightforward
- this applies to all articles (check other articles on a sample basis)
- what we need is the set of **URLs leading to all articles**

Inspecting the URLs

- the URL of the selected article looks as follows:
<https://www.jstatsoft.org/article/view/v082i01>
- we find out that the final part, `v082i01`, always follows the same pattern:
`v<volume number>i<issue number>`

Step 2: Develop a scraping strategy

Let's try to construct the list of URLs from scratch

R code

```
1 baseurl <- "http://www.jstatsoft.org/article/view/v"
2 volurl <- paste0("0", seq(1,78,1))
3 volurl[1:9] <- paste0("00", seq(1, 9, 1))
4 brurl <- paste0("0", seq(1,9,1))
5 urls_list <- paste0(baseurl, volurl)
6 urls_list <- paste0(rep(urls_list, each = 9), "i", brurl)
7 urls_list[1:5]
[1] "http://www.jstatsoft.org/article/view/v001i01"
[2] "http://www.jstatsoft.org/article/view/v001i02"
[3] "http://www.jstatsoft.org/article/view/v001i03"
[4] "http://www.jstatsoft.org/article/view/v001i04"
[5] "http://www.jstatsoft.org/article/view/v001i05"
8 names <- paste0(rep(volurl, each = 9), "_", brurl, ".html")
9 names[1:5]
[1] "001_01.html" "001_02.html" "001_03.html" "001_04.html" "001_05.html"
"
```

Step 3: Download the files

Set working directory

R code

```
10 tempwd <- ("data/jstatsoftStats")
11 dir.create(tempwd)
12 setwd(tempwd)
```

end

Download pages

R code

```
13 folder <- "html_articles/"
14 dir.create(folder)
15 for (i in 1:length(urls_list)) {
16   if (!file.exists(paste0(folder, names[i]))) {
17     download.file(urls_list[i], destfile = paste0(folder, names[i]))
18     Sys.sleep(runif(1, 0, 1))
19   }
20 }
```

Step 3: Download the files

Check success

R code

```
21 list_files <- list.files(folder, pattern = "0.*")
22 list_files_path <- list.files(folder, pattern = "0.*", full.names =
TRUE)
23 length(list_files)

[1] 666
```

end

Step 4: Import files and parse out information

Build loop

R code

```
24 authors <- character()
25 title <- character()
26 statistics <- character()
27 numViews <- numeric()
28 datePublish <- character()
29 for (i in 1:length(list_files_path)) {
30   html_out <- read_html(list_files_path[i])
31   table_out <- html_table(html_out, fill = TRUE)[[6]]
32   authors[i] <- table_out[1,2]
33   title[i] <- table_out[2,2]
34   statistics[i] <- table_out[4,2]
35   numViews[i] <- statistics[i] %>% str_extract("[[:digit:]]+")
36   %>% as.numeric()
37   datePublish[i] <- statistics[i] %>% str_extract("[[:digit:]]{4}-[[:digit:]]{2}-[[:digit:]]{2}.\$")
38   %>% str_replace("\\"., "")
```

end

Step 4: Import files and parse out information

Inspect parsed data

R code

```
38 authors[1:3]
[1] "Ronald Barry"    "Jason Bond, George Michailides"    "Thomas Lumley"
39 title[1:2]
[1] "A Diagnostic to Assess the Fit of a Variogram Model to Spatial Data
"
[2] "Homogeneity Analysis in Xlisp-Stat"
40 numViews[1:3]
[1] 5835 3939 4379
```

end

Construct data frame

R code

```
41 dat <- data.frame(authors = authors, title = title, numViews = numViews,
  datePublish = datePublish)
42 dim(dat)
```

Step 5: Visualize data

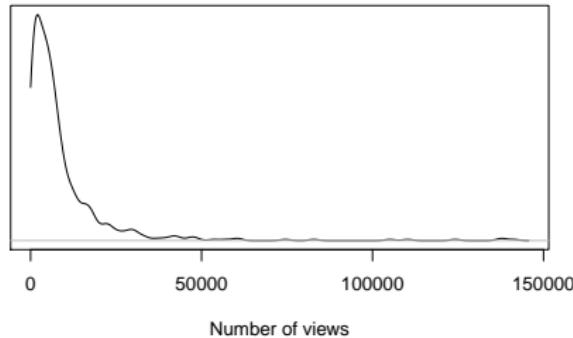
Density plot of download statistics

R code —

```
43 plot(density(dat$numViews, from = 0), yaxt="n", ylab="", xlab="Number of  
views", main="Distribution of article page views in JStatSoft")
```

— end

Distribution of article page views in JStatSoft



Summary

Summary

- scraping data from multiple pages is no problem in R
- most of the brain work often goes into developing a scraping strategy and tidying the data, not into the actual downloading/scraping part
- scraping is also possible in even more complex scenarios, e.g., when HTML forms are involved or you have to take care of cookies or authentication
- this is beyond the scope of this course → check out the book for more applications



Source: Horia Varlan

Campus Luzern

A Primer to Web Scraping with R

Dynamic Webpages

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Static webpages

The simple world of static webpages

Static webpages

- every user who visits the site gets the same content (unless the developer edits the source code)
- example: <https://www.jstatsoft.org>
- can contain "dynamic features", such as video or sound, but the page itself is not dynamic



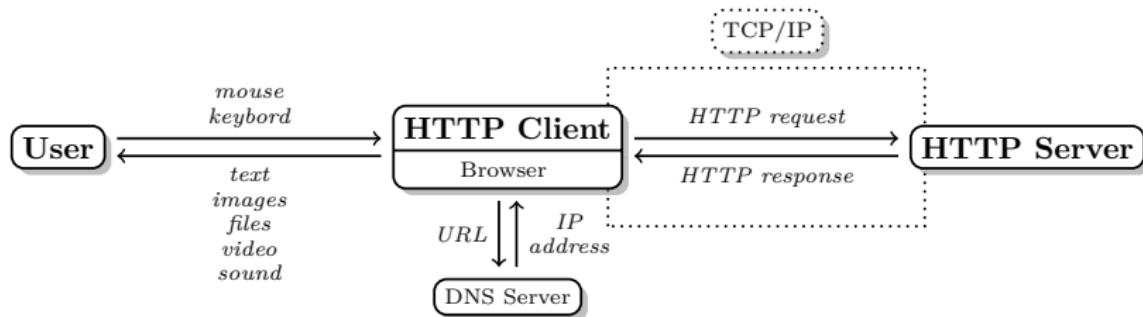
The screenshot shows the Amazon.com homepage from 1997. At the top, it says "WELCOME TO EARTH'S BIGGEST BOOKSTORE" and "Amazon.com". Below that, it displays "1.5 Million Books in Print" and "1 Million Out-of-Print Books". A large "A" logo is on the left. The main navigation bar includes "Text Only", "SEARCH BY", "Books", "CDs", "VHS", "Advanced Search", "BUY BOOKS", "Software", "Movies", "Superior Room", "Computer & Internet", "MILES OF ASIES", "LISTS", "Books & Media", "Computers & Technology", "Business & Money", "Science & Nature", "ARTICLES & REVIEWS", "Books & Media", "Computers & Technology", "Business & Money", "Science & Nature", "SIGN UP", "Books", "Email Editions", "BIG FUN", "Books", "Media", "Computers & Technology", "PARKERS", "Books", "Media", "Computers & Technology". On the right, there's a "First-Time Visitors Please Click Here" link and a "Book of the Day" section featuring "The Great Hunter S. Thompson Book". The central part of the page shows a book cover for "The Great Hunter S. Thompson Book" and some promotional text.

Screenshot of Amazon.com in 1997

A tiny bit of HTTP

Client-server communication with HTTP

- HTTP, the **Hypertext Transfer Protocol**, is a stateless protocol
- no information is retained by either sender or receiver
- makes interaction with websites straightforward, but not very exciting



Client-server communication with HTTP

1. Establishing connection

```
1 About to connect() to www.r-datacollection.com port 80 (#0)
2 Trying 173.236.186.125... connected
3 Connected to www.r-datacollection.com (173.236.186.125) port 80 (#0)
4 Connection #0 to host www.r-datacollection.com left intact
```

2. HTTP request

```
1 GET /index.html HTTP/1.1
2 Host: www.r-datacollection.com
3 Accept: */*
```

A tiny bit of HTTP

Client-server communication with HTTP

3. HTTP response

```
1  HTTP/1.1 200 OK
2  Date: Thu, 27 Feb 2014 09:40:35 GMT
3  Server: Apache
4  Vary: Accept-Encoding
5  Content-Length: 131
6  ...
7
8  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
9  <html> <head>
10 <title></title>
11 </head>
12 ...
```

4. Closing connection

```
1  Closing connection #0
```

The "problem" of static webpages

Static webpages reconsidered

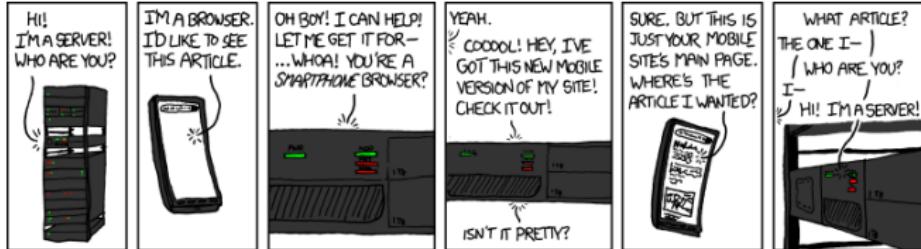
- HTML/HTTP are used for static display of content → same content for every visitor
- in order to display dynamic content, they lack
 1. mechanisms to detect user behavior in the browser (and not only on the server)
 2. a scripting engine that reacts on this behavior
 3. a mechanism for asynchronous queries

Dynamic webpages

The not so simple world of dynamic webpages

Dynamic webpages

- with dynamic webpages, the displayed content can differ between users, even if the source code is the same
- things that can cause the display of different content:
 - operating system, browser, device
 - user actions on the page (mouse movements, scrolling, clicks, keyboard strokes)
 - conditions on the server-side



Source: <https://xkcd.com/869/> (Randall Munroe)

The not so simple world of dynamic webpages

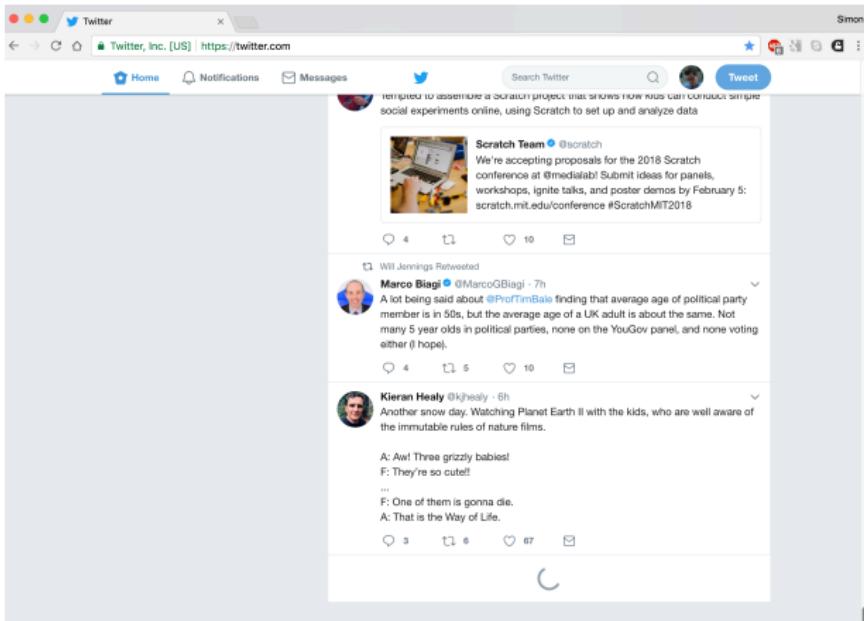
Dynamic webpages

- massively used in modern webpage design and architecture
- (are thought to) enhance the user experience
- allow for much more ways to interact with webpage content

Technologies

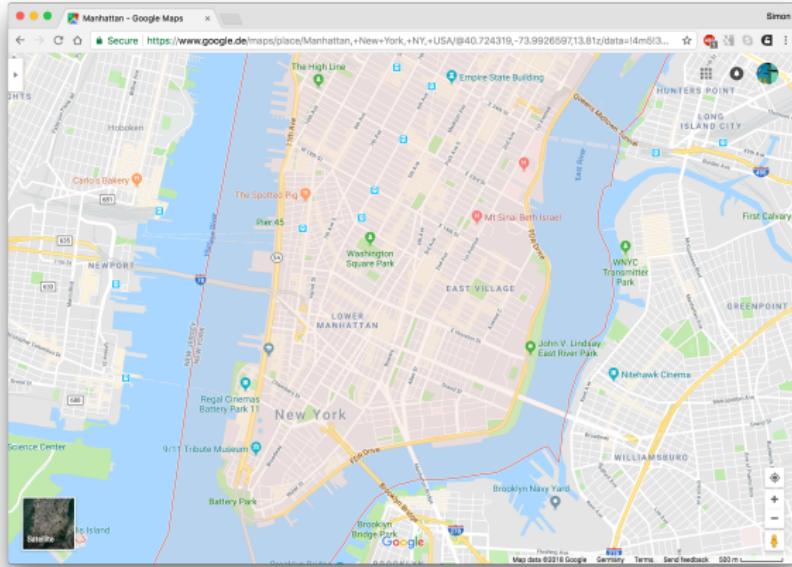
- in the case of **server-side** dynamic webpages, scripts on the server control webpage construction (e.g., PHP script reacts to user input to a form and returns subsets of a database)
- in the case of **client-side** dynamic webpages, (typically) JavaScript embedded in HTML determine what is displayed and how the DOM is changed
- a combination of these technologies is **Asynchronous JavaScript and XML (AJAX)**

Examples of dynamic webpages



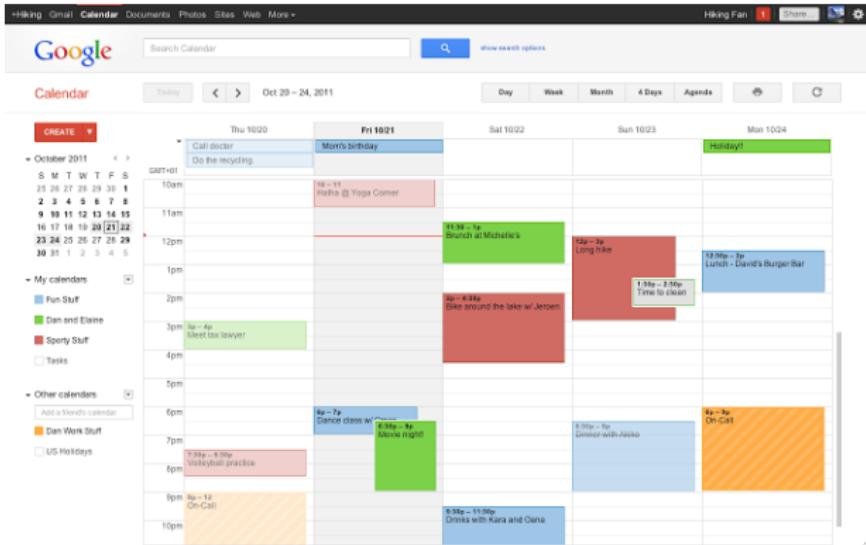
Your Twitter or Facebook feed that automatically gets updated when you scroll down

Examples of dynamic webpages



The map application that automatically loads new content when you zoom in

Examples of dynamic webpages



The calendar app that displays personal information and lets you interact a lot

The problem of dynamic webpages

The problem of dynamic webpages

- the tools you have encountered so far operated on the static source code: you access a page, download it, parse it
- but what if content on the page changes, but the source code does not?
- dynamic webpages make classical screen scraping more difficult if not impossible
- and: dynamically rendered webpages become more and more common
- before we learn about tools that can help us extract data in such scenarios, we will briefly consider the underlying processes, in particular AJAX technologies

Campus Luzern

A Primer to Web Scraping with R

AJAX Technologies

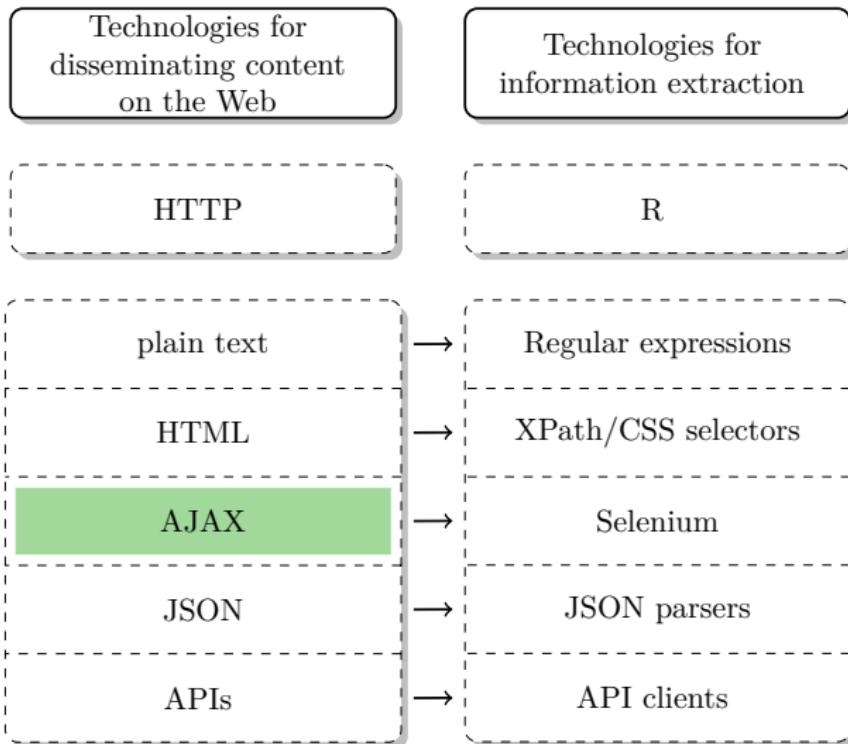
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

AJAX

Technologies of the World Wide Web



What's AJAX?

Purpose

- recall: HTML/HTTP lack
 1. mechanisms to detect user behavior in the browser
 2. a scripting engine that reacts on this behavior
 3. a mechanism for asynchronous queries
- **A**synchronous **J**ava**S**cript **a**nd **X**ML is a set of technologies that serve these purposes

Components

- HTML/CSS for presentation
- Document Object Model (DOM; “tree structure”) to interact with data
- JSON/XML for data interchange
- XMLHttpRequest for asynchronous communication
- JavaScript as a scripting language

JavaScript

What's JavaScript? I

- Programming language that connects well to web technologies
- W3C web standard
- native browser support (built-in JS engine)
- nowadays employed on the majority of websites
- extensible by libraries, e.g. *jQuery*



What's JavaScript? II

- technology to make webpages interactive ('dynamic HTML')
- allows to...
 - animate page elements
 - offer interactive content (games, video, ...)
 - manipulate page content and communication with the server without reloading the page

How's JavaScript code embedded in HTML?

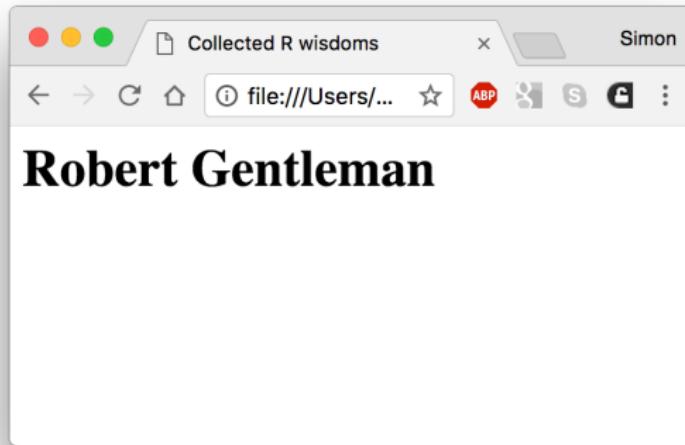
- between `<script>` tags
- as an external reference in the `src` attribute of a `<script>` element
- directly in certain HTML attributes ('event handler')

DOM manipulation with JavaScript

- adding/removing HTML elements
- changing attributes
- modification of CSS styles
- ...

Example:

```
1 <script type="text/javascript" src="jquery-1.8.0.min.js"></script>
2 <script type="text/javascript" src="script1.js"></script>
```



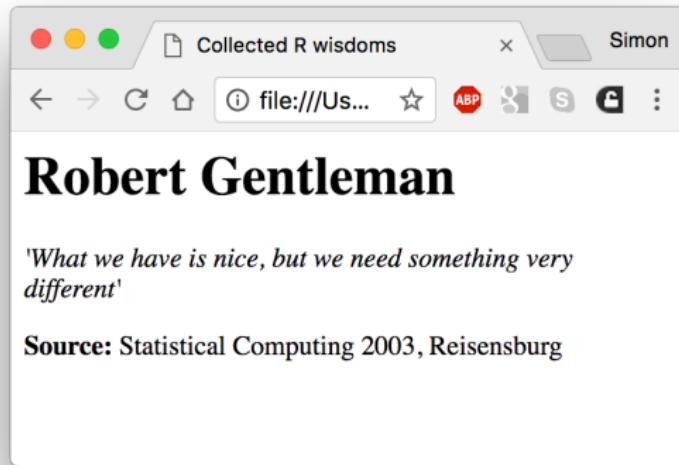
JavaScript on the Web

```
1 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
2 <html>
3
4 <script type="text/javascript" src="jquery-1.8.0.min.js"></script>
5 <script type="text/javascript" src="script1.js"></script>
6
7 <head>
8 <title>Collected R wisdoms</title>
9 </head>
10
11 <body>
12 <div id="R Inventor" lang="english" date="June/2003">
13   <h1>Robert Gentleman</h1>
14   <p><i>'What we have is nice, but we need something very different'</i>
15     </p>
16   <p><b>Source: </b>Statistical Computing 2003, Reisenburg</p>
17 </div>
18 </body>
19 </html>
```

A JavaScript code snippet

```
1 $(document).ready(function() {  
2     $("p").hide();  
3     $("h1").click(function(){  
4         $(this).nextAll().slideToggle(300);  
5     });  
6 });
```

- `$()` operator: addresses DOM elements
- `ready()`: JavaScript execution starts when the complete DOM is ready, i.e. fetched from the server
- `hide()`: element is hidden at first place
- `click` event: identifies mouse click and executes a certain action
- `nextAll()`: all subsequent elements in the DOM are addressed
- `slideToggle()`: Toggle effect, 300 milli-seconds



Summary

Summary

- AJAX technologies allow for a dynamic manipulation of the HTML tree
- what the user sees in the browser is not necessarily what's in the static HTML source code
- elements can be added, removed, or changed
- the "live" DOM tree that you saw in the Web Developer Tools takes track of such changes
- but you cannot simply access this live HTML with R and **rvest**
- we need another technology that helps us mimic a session in the browser to render all dynamic content



Campus Luzern

A Primer to Web Scraping with R

Selenium: Basics

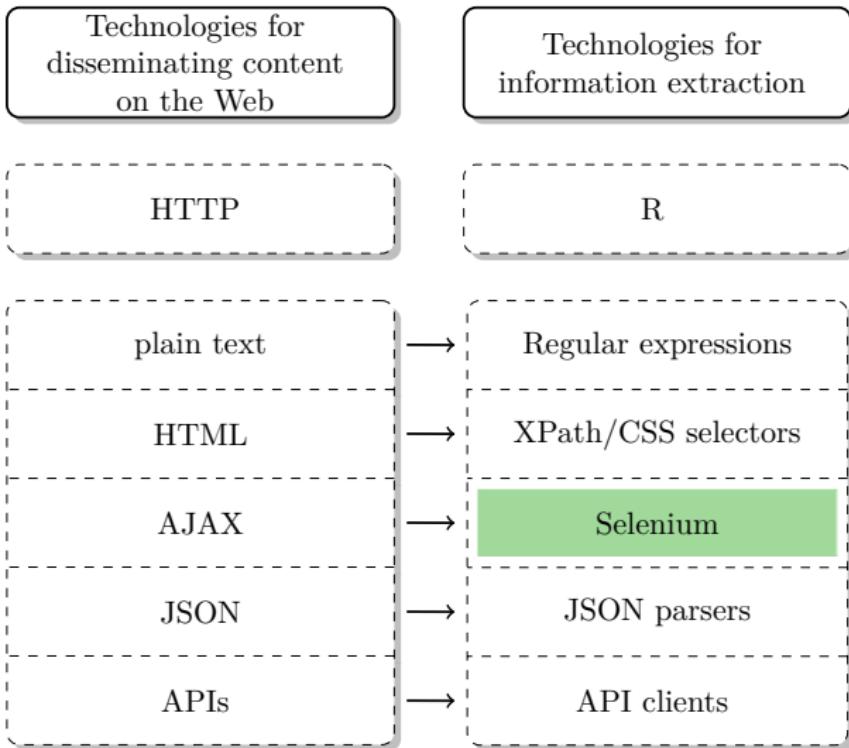
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Selenium

Technologies of the World Wide Web



Scraping dynamic webpages

The problem reconsidered

- dynamic data requests are not stored in the static HTML page
- therefore, we cannot access them with classical methods and packages (`httr`, `rvest`, `download.file()`, etc.)
- R is not a browser with a JavaScript rendering engine

The solution

- initiate and control a web browser session with R
- let the browser do the JavaScript interpretation work and the manipulations in the live DOM tree
- access information from the web browser session

What's Selenium?

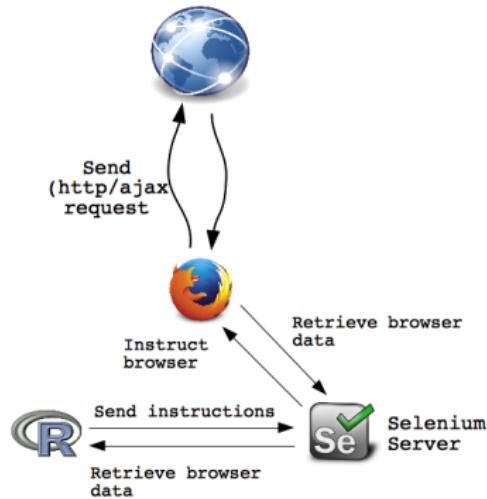
- free software environment for automated web application testing
- webpage: <http://www.seleniumhq.org>
- several modules for different tasks; most important for our purposes: Selenium WebDriver
- enables automated browsing via scripts



Selenium and R

The scraping workflow with Selenium and R

1. Selenium—an external, Java-based program—is launched
2. via the R package **RSelenium**, we remote-control Selenium and let it start a virtual server
3. we let Selenium start a browser session (e.g., Chrome)
4. everything we do in the browser—open a page, click on links or buttons, enter information—is described in R and sent to the server via the “remote-control” Selenium
5. we can gather the live HTML tree at any instance



Software requirements

- Java, <https://www.java.com/de/download/>
- Selenium Standalone Server, newest version available here:
<http://www.seleniumhq.org/download/> or via RSelenium and
`rsDriver()`
- browser (on most systems, both Chrome and Firefox seem to work reliably)
- **RSelenium** package

Campus Luzern

A Primer to Web Scraping with R

Selenium: Case Study

Simon Munzert

Hertie School of Governance, Berlin

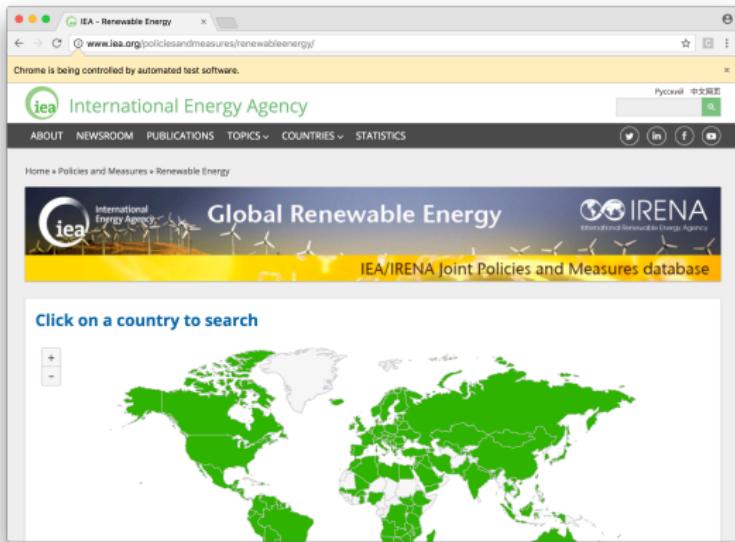
May 20-21, 2019

Selenium: Case Study

Example

Overview

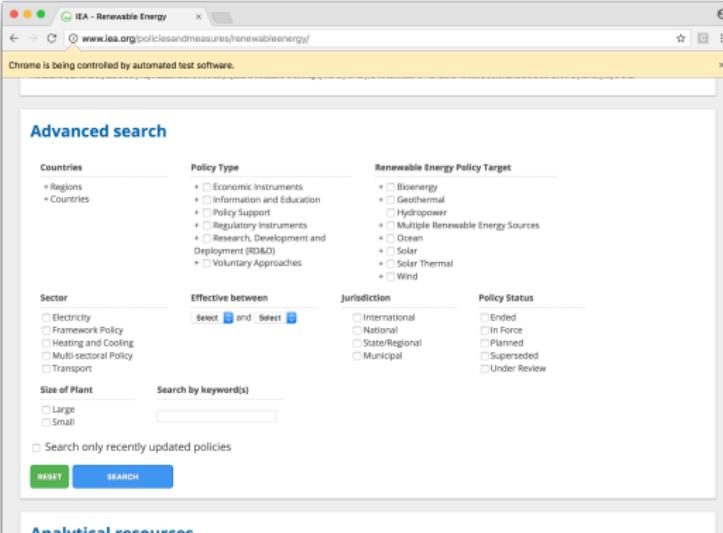
- **goal:** scrape data from the IEA/IRENA database
- URL:
<http://www.iea.org/policiesandmeasures/renewableenergy/>
- the query form has multiple parameters
- the output comes in form of an HTML table, but content is injected into DOM



Example

Overview

- **goal:** scrape data from the IEA/IRENA database
- URL:
<http://www.iea.org/policiesandmeasures/renewableenergy/>
- the query form has multiple parameters
- the output comes in form of an HTML table, but content is injected into DOM

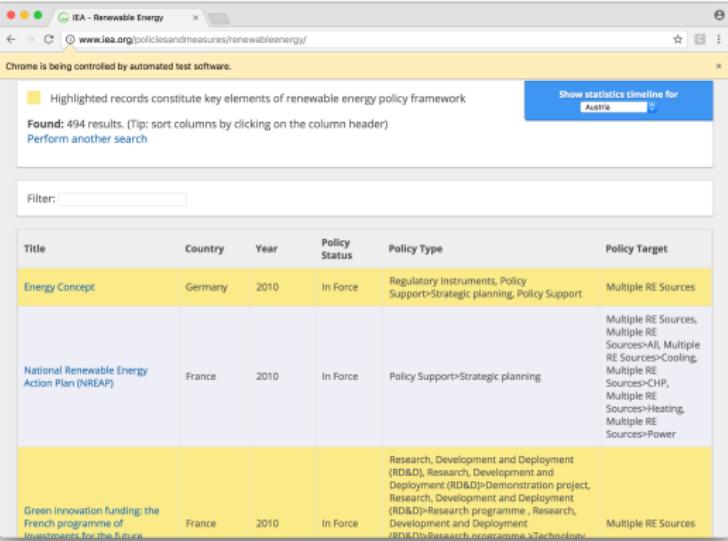


The screenshot shows a web browser window titled "IEA - Renewable Energy" with the URL "www.iea.org/policiesandmeasures/renewableenergy/". A status bar at the top indicates "Chrome is being controlled by automated test software". The main content is titled "Advanced search". The search form includes sections for "Countries" (with "+Regions" and "+Countries" buttons), "Policy Type" (with checkboxes for Economic Instruments, Information and Education, Policy Support, Regulatory Instruments, Research, Development and Deployment (R&D), and Voluntary Approaches), "Renewable Energy Policy Target" (with checkboxes for Bioenergy, Geothermal, Hydropower, Multiple Renewable Energy Sources, Ocean, Solar, Solar Thermal, and Wind), "Sector" (with checkboxes for Electricity, Framework Policy, Heating and Cooling, Multi-sectoral Policy, and Transport), "Effective between" (with "Select" buttons for start and end dates), "Jurisdiction" (with checkboxes for International, National, State/Regional, and Municipal), "Policy Status" (with checkboxes for Ended, In Force, Planned, Superseded, and Under Review), "Size of Plant" (with checkboxes for Large and Small), and a "Search by keyword(s)" input field. Below the form is a section titled "Analytical resources".

Example

Overview

- **goal:** scrape data from the IEA/IRENA database
- URL:
<http://www.iea.org/policiesandmeasures/renewableenergy/>
- the query form has multiple parameters
- the output comes in form of an HTML table, but content is injected into DOM



Title	Country	Year	Policy Status	Policy Type	Policy Target
Energy Concept	Germany	2010	In Force	Regulatory Instruments, Policy Support>Strategic planning, Policy Support	Multiple RE Sources
National Renewable Energy Action Plan (NREAP)	France	2010	In Force	Policy Support>Strategic planning	Multiple RE Sources, Multiple RE Sources>All, Multiple RE Sources>Cooling, Multiple RE Sources>CHP, Multiple RE Sources>Heating, Multiple RE Sources>Power
Green innovation funding: the French programme of investments for the future	France	2010	In Force	Research, Development and Deployment (RD&D), Research, Development and Deployment (RD&D)>Demonstration project, Research, Development and Deployment (RD&D)>Research Programme , Research, Development and Deployment (RD&D)>Research, Development and Deployment (RD&D)>Technology	Multiple RE Sources

Example

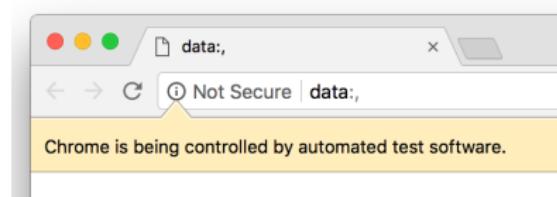
Setup

- load **RSelenium**
- check installed Java version (helpful for debugging if Selenium fails to launch)
- initiate SeleniumDriver and browser

R code —————

```
1 library(RSelenium)
2 system("java -version")
  java version "9"
  Java(TM) SE Runtime
  Environment (build 9+181)
  Java HotSpot(TM) 64-Bit
  Server VM (build 9+181,
  mixed mode)
3 rD <- rsDriver()
4 remDr <- rD[["client"]]
```

————— end



Example

R calls

- `navigate()` to URL
- page opens
"automatically"

R code —————

```
5 url <- "http://www.iea.org/  
policiesandmeasures/  
renewableenergy/"  
6 remDr$navigate(url)
```

————— end



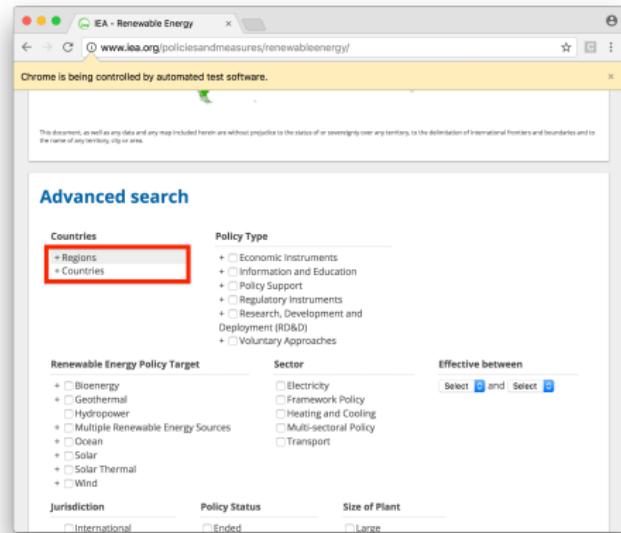
Example

R calls

- click on the "Regions" menu to unfold it
- extract XPath expression from Web Developer Tools
- pass XPath to `findElement()`, then click on it with `clickElement()`

R code

```
xpath <- '//*[@id="main"]/div/form/div[1]/ul/li[1]/span'  
regionsElem <- remDr$findElement(using = 'xpath', value = xpath)  
openRegions <- regionsElem$clickElement()  
  
end
```



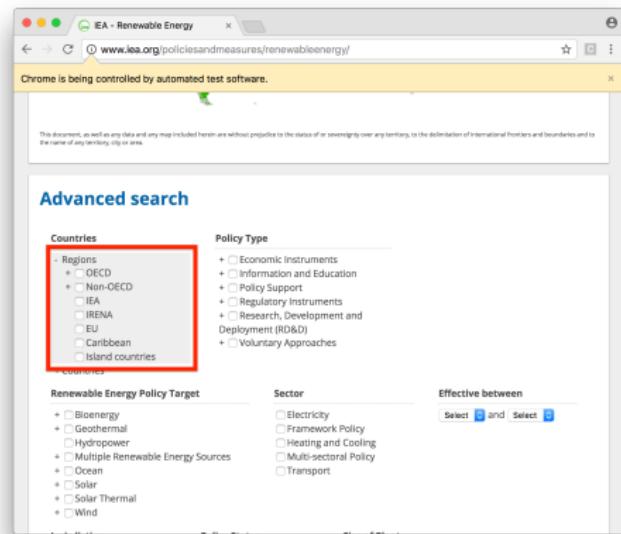
Example

R calls

- click on the "Regions" menu to unfold it
- extract XPath expression from Web Developer Tools
- pass XPath to `findElement()`, then click on it with `clickElement()`

R code

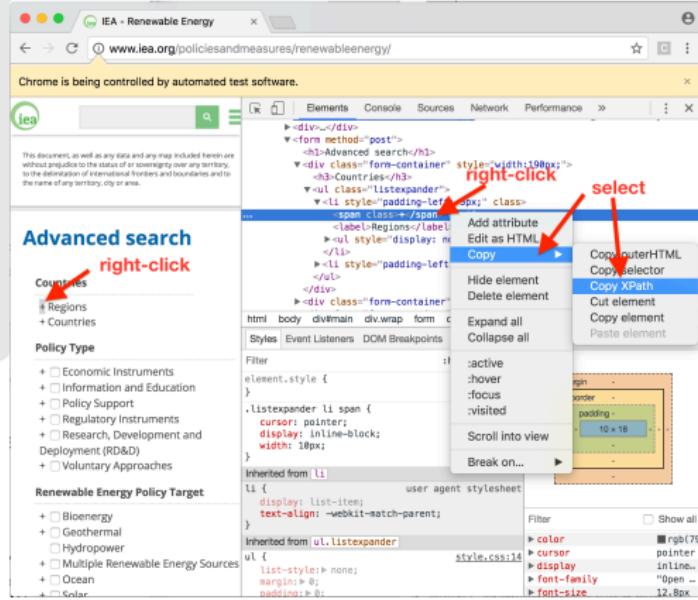
```
xpath <- '//*[@id="main"]/div/form/div[1]/ul/li[1]/span'  
regionsElem <- remDr$findElement(using = 'xpath', value = xpath)  
openRegions <- regionsElem$clickElement()  
  
end
```



Example

R calls

- click on the "Regions" menu to unfold it
- extract XPath expression from Web Developer Tools
- pass XPath to `findElement()`, then click on it with `clickElement()`



R code

```
xpath <- '//*[@id="main"]/div/form/div[1]/ul/li[1]/span'  
regionsElem <- remDr$findElement(using = 'xpath', value = xpath)  
openRegions <- regionsElem$clickElement()  
(  
)
```

end

Example

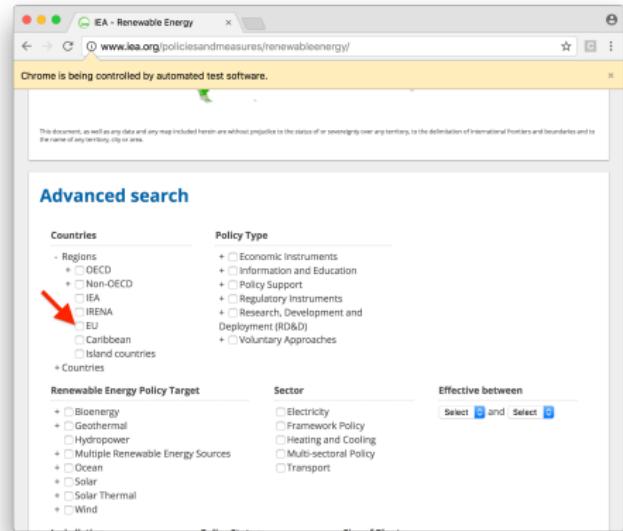
R calls

- select "EU" option
- extract XPath expression from Web Developer Tools
- pass XPath to `findElement()`, then click on it with `clickElement()`

R code

```
xpath <- '//*[@id="main"]//div/form/div[1]/ul/li[1]/ul/li[5]/label/input'  
euElem <- remDr$findElement(using = 'xpath', value = xpath)  
selectEU <- euElem$clickElement()
```

end



Example

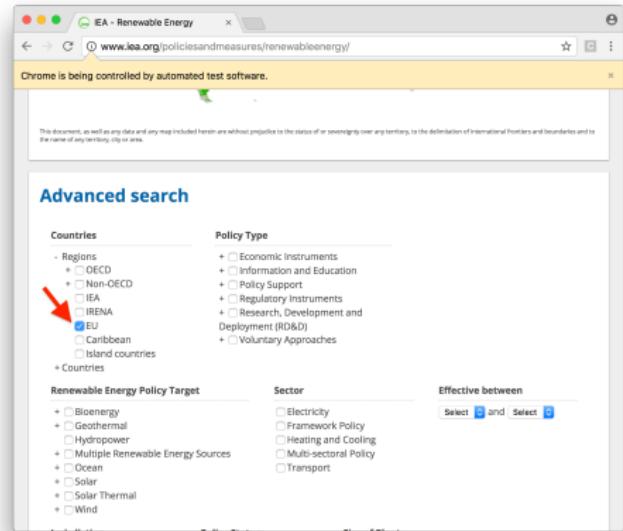
R calls

- select "EU" option
- extract XPath expression from Web Developer Tools
- pass XPath to `findElement()`, then click on it with `clickElement()`

R code

```
xpath <- '//*[@id="main"]//div/form/div[1]/ul/li[1]/ul/li[5]/label/input'  
euElem <- remDr$findElement(using = 'xpath', value = xpath)  
selectEU <- euElem$clickElement()
```

end



Example

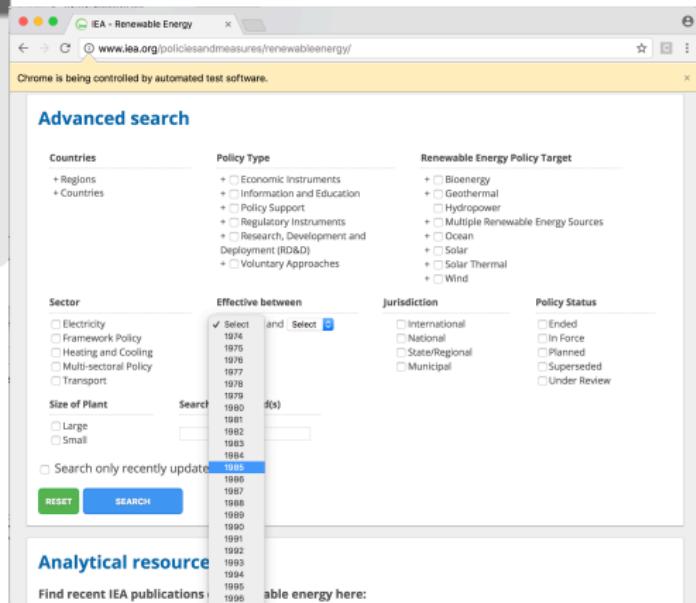
R calls

- set time frame
- pass XPath to `findElement()`,
then click on it, then enter text
with `sendKeysToElement()`

R code —

```
xpath <- '//*[@id="main"]/div/form/div[5]/select[1]'  
  
fromDrop <- remDr$findElement(using = 'xpath', value = xpath)  
clickFrom <- fromDrop$clickElement()  
writeFrom <- fromDrop$sendKeysToElement(list("2000"))
```

— end



The screenshot shows a web browser window for the IEA Renewable Energy website at www.iea.org/policiesandmeasures/renewableenergy/. The page title is "IEA - Renewable Energy". A yellow bar at the top indicates "Chrome is being controlled by automated test software". The main content is titled "Advanced search". It features several filter sections: "Countries" (dropdown for Regions and Countries), "Policy Type" (checkboxes for Economic Instruments, Information and Education, Policy Support, Regulatory Instruments, Research, Development and Deployment (R&D), and Voluntary Approaches), "Renewable Energy Policy Target" (checkboxes for Bioenergy, Geothermal, Hydropower, Multiple Renewable Energy Sources, Ocean, Solar, Solar Thermal, and Wind), "Sector" (checkboxes for Electricity, Framework Policy, Heating and Cooling, Multi-sectoral Policy, and Transport), "Effectiveness between" (dropdown menu with "Select" and "Select" buttons, showing years from 1990 to 1998, with 1995 selected), "Jurisdiction" (checkboxes for International, National, State/Regional, and Municipal), and "Policy Status" (checkboxes for Ended, In Force, Planned, Superseded, and Under Review). At the bottom, there are buttons for "RESET", "SEARCH", and "Analytical resource" which says "Find recent IEA publications".

Example

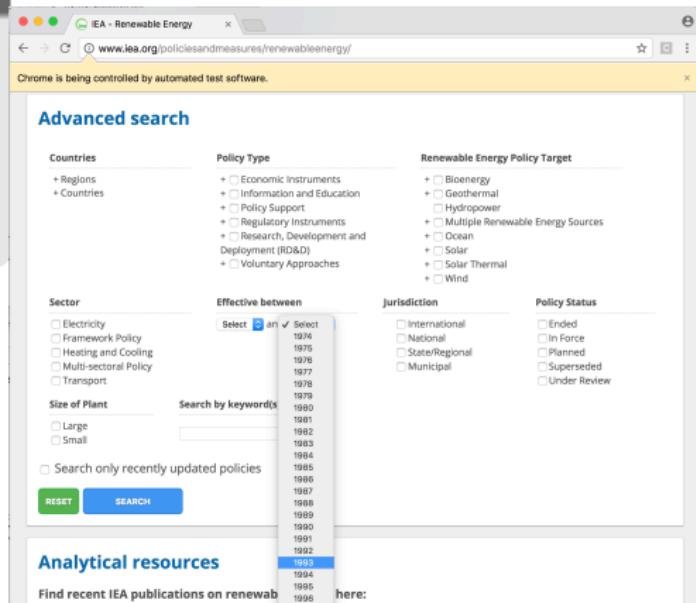
R calls

- set time frame
- pass XPath to `findElement()`,
then click on it, then enter text
with `sendKeysToElement()`

R code —

```
xpath <- '//*[@id="main"]/div/form/div[5]/select[2]'  
  
fromDrop <- remDr$findElement(using = 'xpath', value = xpath)  
clickFrom <- fromDrop$clickElement()  
writeFrom <- fromDrop$sendKeysToElement(list("2010"))
```

— end



The screenshot shows a web browser window for the IEA Renewable Energy website at www.iea.org/policiesandmeasures/renewableenergy/. The page title is "IEA - Renewable Energy". A yellow bar at the top indicates "Chrome is being controlled by automated test software". The main content is an "Advanced search" form. It has four main sections: "Countries" (with "+ Regions" and "+ Countries" buttons), "Policy Type" (with various policy instruments like Economic Instruments, Information and Education, Policy Support, etc.), "Renewable Energy Policy Target" (with Bioenergy, Geothermal, Hydropower, etc.), and "Policy Status" (with Ended, In Force, Planned, Superseded, Under Review). Below these are sections for "Sector" (Electricity, Framework Policy, Heating and Cooling, Multi-sectoral Policy, Transport), "Effective between" (a date range selector from 1994 to 1998), "Jurisdiction" (International, National, State/Regional, Municipal), and "Size of Plant" (Large, Small). There is also a "Search by keyword(s)" input field and a "RESET" button. At the bottom, there is a section titled "Analytical resources" with a link "Find recent IEA publications on renewable energy here".

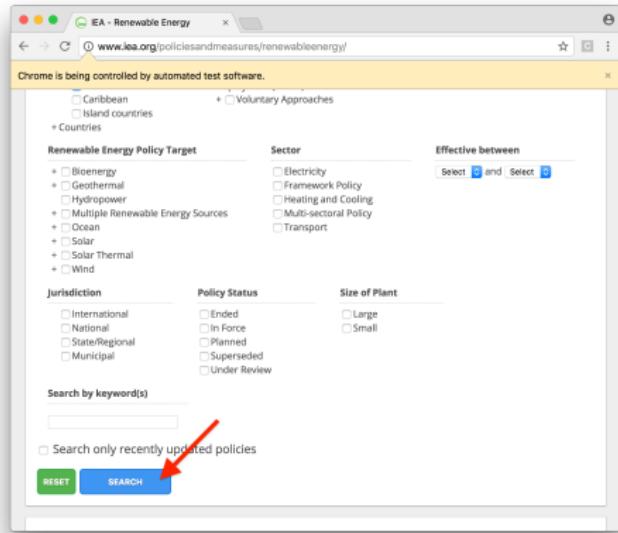
Example

R calls

- click on search button
- extract XPath expression of element from Web Developer Tools
- pass XPath to `findElement()`, then click on it

R code —

```
xpath <- '//*[@id="main"]/div/form/  
button[2]'  
  
searchElem <- remDr$findElement(using =  
  'xpath', value = xpath)  
  
resultsPage <- searchElem$clickElement  
  
————— end
```



Example

R calls

- finally, we're there!
- now take snapshot of the DOM with `getPageSource()` and store it as HTML with `write()`

R code

```
output <- remDr$getPageSource(header =  
TRUE)  
write(output[[1]], file = "iea-  
renewables.html")
```

end

The screenshot shows a web browser window titled 'IEA - Renewable Energy' with the URL 'www.iea.org/policiesandmeasures/renewableenergy/'. A message at the top says 'Chrome is being controlled by automated test software.' Below it, a search bar shows 'Found: 494 results. (Tip: sort columns by clicking on the column header)' and a button 'Perform another search'. A 'Filter:' input field is present. The main content is a table with the following columns: Title, Country, Year, Policy Status, Policy Type, and Policy Target. Three rows are highlighted in yellow:

Title	Country	Year	Policy Status	Policy Type	Policy Target
Energy Concept	Germany	2010	In Force	Regulatory Instruments, Policy Support>Strategic planning, Policy Support	Multiple RE Sources
National Renewable Energy Action Plan (NREAP)	France	2010	In Force	Policy Support>Strategic planning	Multiple RE Sources, Multiple RE Sources>AI, Multiple RE Sources>Cooling, Multiple RE Sources>CHP, Multiple RE Sources>Heating, Multiple RE Sources>Power
Green Innovation funding: the French programme of investments for the future	France	2010	In Force	Research, Development and Deployment (R&D), Research, Development and Deployment (R&D)>Innovation project, Research, Development and Deployment (R&D)>Research programme, Research, Development and Deployment (R&D)>Research, development and technology transfer	Multiple RE Sources

Example

Parsing HTML into data frame

- close the connection to Selenium server with `closeServer()`
- proceed with business as usual (`rvest` package)

R code

```
35 remDr$closeServer()
36 content <- read_html("iea-renewables.html", encoding = "utf8")
37 tabs <- html_table(content, fill = TRUE)
38 tab <- tabs[[1]]
39 "target")
40 tab[1,]

          title country year    status
                           type
1 Energy Concept Germany 2010 In Force Regulatory Instruments, Policy
Support>Strategic planning, Policy Support
          target
1 Multiple RE Sources
```

Summary

Summary

- Selenium is an excellent tool to scrape data from dynamic, JavaScript-enriched webpages where ordinary scraping methods fail
- once the setup is established (which can be troublesome, as many software components have to work together), its use is pretty simple
- I do not recommend it as a substitute for every scraping task because it is too slow and unreliable for that purpose



By W. Oelen (CC BY-SA 3.0),
<https://commons.wikimedia.org/w/index.php?curid=15369617>

Campus Luzern

A Primer to Web Scraping with R

Legal Issues

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

Is web scraping legal?

Is web scraping legal?

Disclaimer

Obviously, I am not a lawyer. Do not rely on any of my comments on this topic. If you are seriously worried about the legality of your scraping work, please consult a legal expert.



Is web scraping legal?

Scraping vs. crawling

- **Web scraping:** downloading data from a very specific page in a (semi-)automated manner
- **Web crawling:** automatically downloading webpage data, extracting hyperlinks, following links, downloading webpage data, ... (e.g.: Googlebot, BaiduSpider)

This course mostly is scraping, not crawling or web harvesting!

- web scraping per se is not illegal
- there is no unambiguous **yes** or **no** for specific applications in any country according to current jurisdiction
- so far, legal cases (especially in the US) often (but not always) dealt with commercial interest, crawling applications, and often (but not always) huge masses of data, e.g., *eBay vs. Bidder's Edge*, *AP vs. Meltwater*, *Facebook vs. Pete Warden*

Is web scraping legal?

Why web scraping can be problematic

- violation of copyrights, Terms of Service, consumption of bandwidth

Some counter-arguments

- data is publicly accessible
 - but the website as a "creative arrangement" might be copyrighted
- this is fair use → depends on your use
- it's the same what my browser does
 - not exactly, and many ToS prohibit automated uses of their data
- this is unfair—Google's business model is built on crawling the whole web
 - true, but you are not Google

Some interesting comments by Pablo Hoffman, co-founder of Scrapinghub

- As long as they don't crawl at a disruptive rate, scrapers do not breach any contract (in the form of terms of use) or commit a crime (as defined in the Computer Fraud and Abuse Act).
- Website's user agreement is not enforceable as a browse-wrap agreement because companies do not provide sufficient notice of the terms to site visitors.
- Scrapers accesses website data as a visitor, and by following paths similar to a search engine. This can be done without registering as a user (and explicitly accepting any terms).

(found on <https://goo.gl/jTt4ER>)

Things webpage administrators can do to prevent you from scraping massive amounts of data from their pages

- block your IP address
- identify your approximate geolocation from your IP address, then block
- move content exclusively to web services / APIs
- block bots with a particular user agent string (more on that later)
- challenge-response tests like **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part (CAPTCHA)
- obfuscation of data
- frequent changes in HTML/CSS

Summary

Summary

- there is no unconditional "legal" or "illegal" status of web scraping
- your use of the data can violate the data owner's rights
- targeted scraping efforts with limited traffic usually do not cause any problems



Campus Luzern

A Primer to Web Scraping with R

Good Practice

Simon Munzert

Hertie School of Governance, Berlin

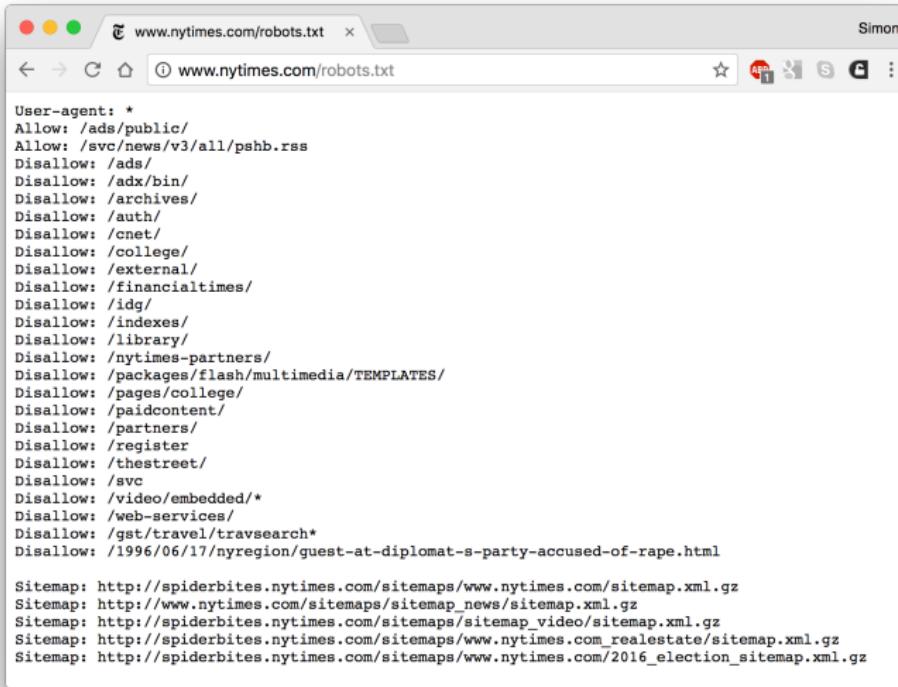
May 20-21, 2019

Understanding robots.txt

What's robots.txt?

- ‘Robots Exclusion Protocol’, informal protocol to prohibit web robots from crawling content
- located in the root directory of a website (e.g.,
<http://www.google.com/robots.txt>)
- documents which bot is allowed to crawl which resources (and which not)
- not a technical barrier, but a sign that asks for compliance

Example: NY Times's robots.txt



A screenshot of a web browser window titled "www.nytimes.com/robots.txt". The address bar also shows "www.nytimes.com/robots.txt". The page content is the NY Times's robots.txt file, which includes rules for various user-agents and specific URLs.

```
User-agent: *
Allow: /ads/public/
Allow: /svc/news/v3/all/pshb.rss
Disallow: /ads/
Disallow: /adx/bin/
Disallow: /archives/
Disallow: /auth/
Disallow: /cnet/
Disallow: /college/
Disallow: /external/
Disallow: /financialtimes/
Disallow: /idg/
Disallow: /indexes/
Disallow: /library/
Disallow: /nytimes-partners/
Disallow: /packages/flash/multimedia/TEMPLATES/
Disallow: /pages/college/
Disallow: /paidcontent/
Disallow: /partners/
Disallow: /register/
Disallow: /thestreet/
Disallow: /svc
Disallow: /video/embedded/*
Disallow: /web-services/
Disallow: /gst/travel/travsearch*
Disallow: /1996/06/17/nyregion/guest-at-diplomat-s-party-accused-of-rape.html

Sitemap: http://spiderbites.nytimes.com/sitemaps/www.nytimes.com/sitemap.xml.gz
Sitemap: http://www.nytimes.com/sitemaps/sitemap_news/sitemap.xml.gz
Sitemap: http://spiderbites.nytimes.com/sitemaps/sitemap_video/sitemap.xml.gz
Sitemap: http://spiderbites.nytimes.com/sitemaps/www.nytimes.com_realestate/sitemap.xml.gz
Sitemap: http://spiderbites.nytimes.com/sitemaps/www.nytimes.com/2016_election_sitemap.xml.gz
```

Syntax in robots.txt

Syntax

- not an official W3C standard, partly inconsistent syntax
- rules listed bot by bot
- general, bot-independent rules under '*' (most interesting entry for R-based crawlers)
- directories/folders listed separately

```
1 User-agent: Googlebot
2 Disallow: /images/
3 Disallow: /private/
```

```
1 User-agent: *
2 Disallow: /private/
```

Syntax in robots.txt

Universal ban

```
1 User-agent: *
2 Disallow: /
```

Separation of bots by empty line

```
1 User-agent: Googlebot
2 Disallow: /images/
4 User-agent: Slurp
5 Disallow: /images/
```

Allow declaration

```
1 User-agent: *
2 Disallow: /images/
3 Allow: /images/public/
```

Syntax in robots.txt

Crawl-delay (in seconds)

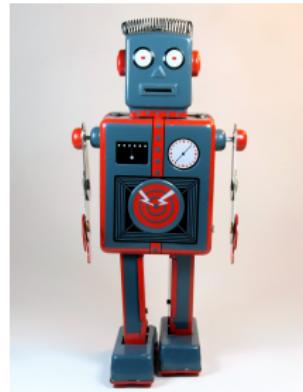
```
1 User-agent: *
2 Crawl-delay: 2
3 User-Agent: Googlebot
4 Disallow: /search/
```

Robots <meta> tag

```
1 <meta name="robots" content="noindex,nofollow" />
```

How to deal with robots.txt?

- not clear if robots.txt is legally binding or not, and if yes for which activities
- originally not thought of as protection against small-scale web scraping applications, but against large-scale indexing bots
- guide to a webmaster's preferences with regards to visibility of content
- my advice: take `robots.txt` into account! If the data you are interested in are excluded from crawling: contact webmaster
- there is even an R package, `robotstxt`, that helps you parse `robots.txt` documents and stick to the rules. It's available here:
<https://github.com/ropenscilabs/robotstxt>



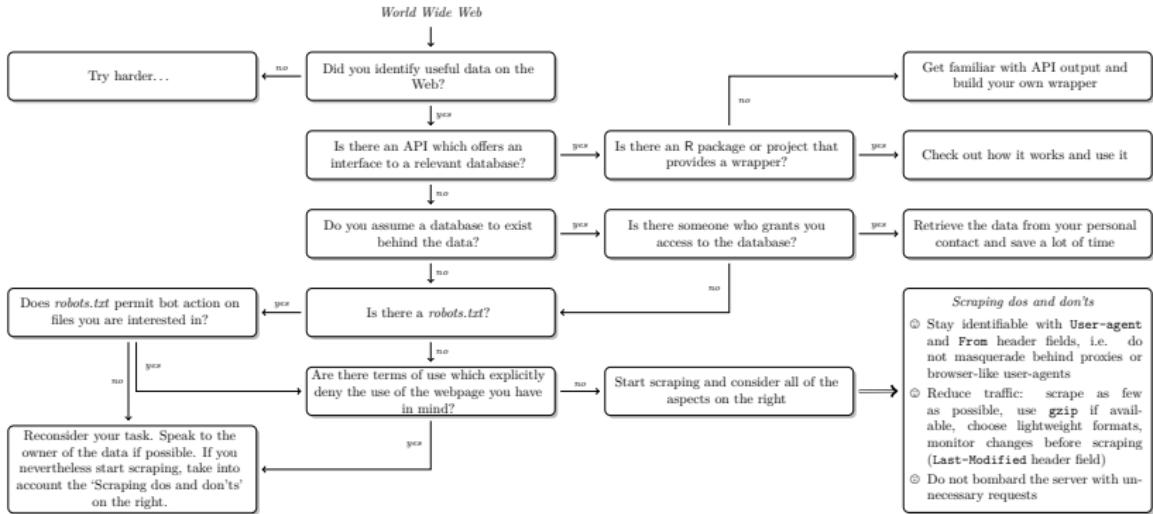
By D J Shin - My Toy Museum, [https://commons.wikimedia.org/wiki/File:QSH_Tin_Wind_Up_Mechanical_Robot_\(Giant_Easelback_Robot\)_Front.jpg](https://commons.wikimedia.org/wiki/File:QSH_Tin_Wind_Up_Mechanical_Robot_(Giant_Easelback_Robot)_Front.jpg)

Scraping etiquette

Some advice for your work

1. You take all the responsibility for your web scraping work.
2. Take all copyrights of a country's jurisdiction into account.
3. If you publish data, do not commit copyright fraud.
4. If possible, stay identifiable.
5. If in doubt, ask the author/creator/provider of data for permission—if your interest is entirely scientific, chances aren't bad that you get data.
6. Consult current jurisdiction, e.g. on
<http://blawgsearch.justia.com> or from a lawyer specialized on internet law.

Scraping etiquette



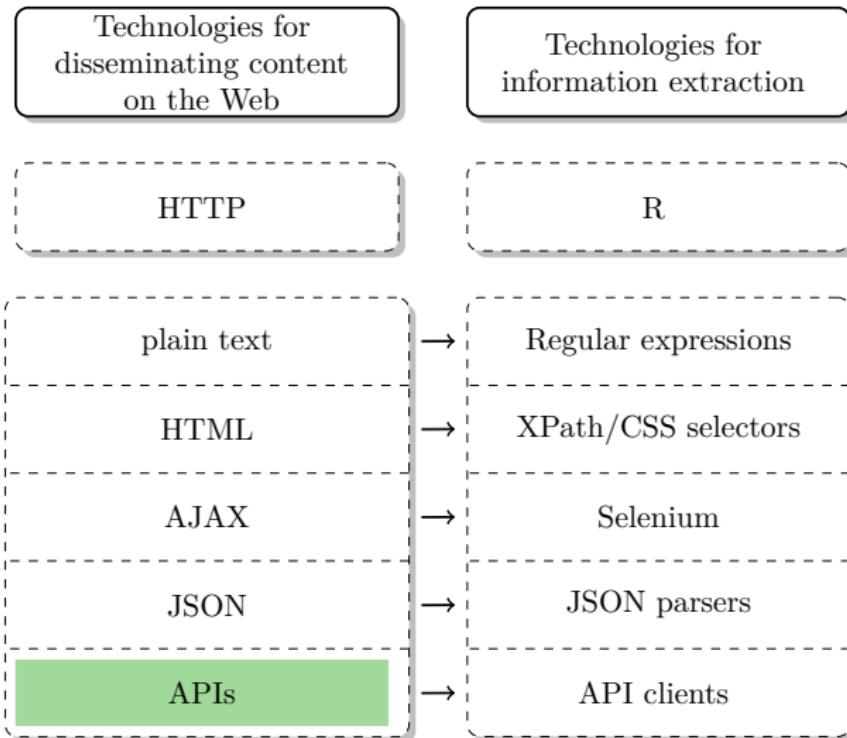
Campus Luzern

A Primer to Web Scraping with R

APIs

Simon Munzert
Hertie School of Governance, Berlin
May 20-21, 2019

Technologies of the World Wide Web



What are APIs?

What are APIs?

Definition

- Application Programming Interface
- "data search engine": you pose a request, the API answers with a bulk of data
- let you/your program query a provider for specific data
- common data formats: XML, JSON
- many popular web services provide APIs (Twitter, Google, Facebook, Wikipedia, ...)

Why we should care about APIs

- provide instant access to clean data
- free us from building manual scrapers
- API usage implies mutual data collection agreement

Example

Example

Google Maps API

- Google provides access to powerful location services
- free service (at least when used modestly)
- input/output: places, names, coordinates, maps, ...
- see also: <https://developers.google.com/maps/documentation/>



Google Maps

Potential use cases

- geocode observations based on address, city, post code, ...
- calculate distances between observations
- map observations

Example

Access the API with R

- the **ggmap** package provides high-level functions to access API
- in this case, all we have to do is to figure out how the R function works – the communication with the API is processed completely in the background

R code

```
1 library(ggmap)
2 geocode("Berlin, Germany")
```

```
Information from URL : http://maps.googleapis.com/maps/api/geocode/json?
address=Berlin,%20Germany&sensor=false
      lon      lat
1 13.40495 52.52001
```

end

Example

Excerpt from raw JSON behind the call

```
1  {
2      "results" : [
3          {
4              "address_components" : [
5                  {
6                      "long_name" : "Berlin",
7                      "short_name" : "Berlin",
8                      "types" : [ "locality", "political" ]
9                  },
10                 ],
11                 "formatted_address" : "Berlin, Germany",
12                 "location" : {
13                     "lat" : 52.52000659999999,
14                     "lng" : 13.404954
15                 }
16             },
17             "place_id" : "ChIJAVkDPzd0qEcRcDteWOYgIQQQ",
18             "types" : [ "locality", "political" ]
19         }
20     ]
21 }
```

Example

Map the location

R code

```
3 get_googlemap("Berlin, Germany",
  zoom = 12, maptype = "hybrid")
%>% ggmap()
  _____ end
```



Summary

Summary

Advantages of API use

- collecting data from the web using APIs provided by the data owner represents **the gold standard of web data retrieval**
- pure data collection without ‘layout waste’
- standardized data access
- de facto automatic agreement
- robustness of calls



[http://maxpixel.
freegreatpicture.com/
photo-1461569](http://maxpixel.freegreatpicture.com/photo-1461569)

Disadvantages of API use

- (sometimes) requires knowledge of API architecture
- dependent upon API suppliers
- use not always free

Campus Luzern

A Primer to Web Scraping with R

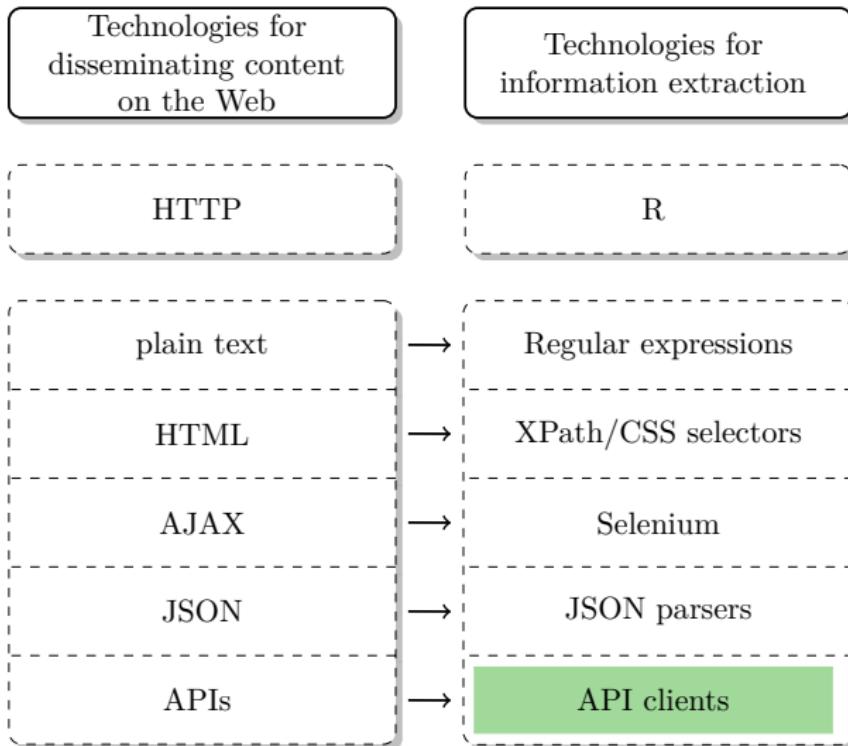
API Clients

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

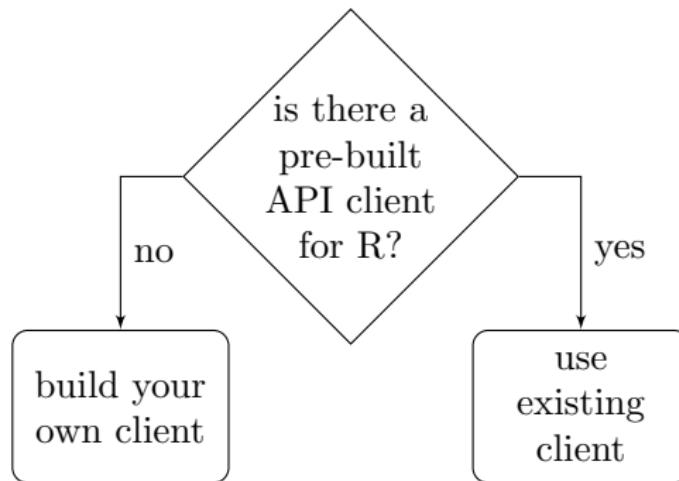
Technologies of the World Wide Web



Accessing APIs with R

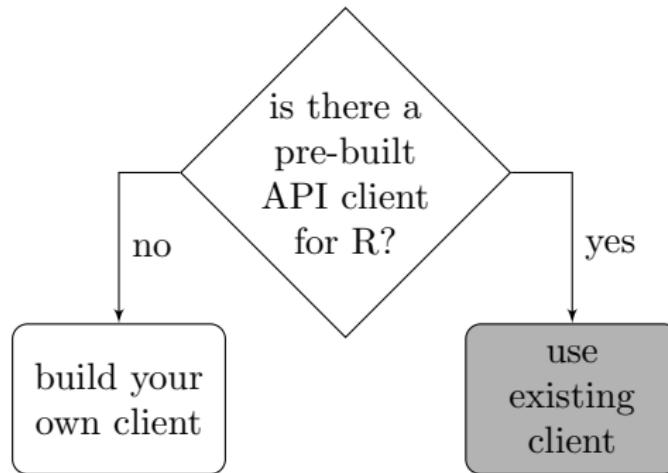
API access with R

There are basically **two scenarios:**



API access with R

There are basically **two scenarios:**



Here, we talk about the case where a client already exists.

API clients

- provide interface to APIs
- hide API back-end
- let you stay in your programming environment

Example

- the `rtweet` package provides an R client for Twitter
- lets you query data from the Twitter API with R commands
- you don't have to work with unfamiliar data formats (JSON) that is provided by the API—the client automatically transforms the incoming data into R objects

Finding API clients on the Web

General resources

List of APIs: <http://www.programmableweb.com/apis>

rOpenSci – collection of R API clients:

<https://github.com/ropensci/opendata>

CRAN Task View:

<http://cran.r-project.org/web/views/WebTechnologies.html>

How to find the API client you need

- google "R package + name of website"
- search on the website for a "Developer" or "API" section (only if you don't find an R package that works)

Popular R API clients

package name	access to	more info
<code>rtweet</code>	Twitter Stream and REST API	http://rtweet.info/
<code>Rfacebook</code>	Facebook API	https://github.com/pablobarbera/Rfacebook
<code>ipapi</code>	ip-api.com's API	https://github.com/hrbrmstr/ipapi
<code>ggmap</code>	Google Maps/OpenStreetMap APIs	https://github.com/dkahle/ggmap
<code>eurostat</code>	Eurostat database	https://github.com/ropengov/eurostat
<code>rftimes</code>	New York Times APIs	https://cran.rstudio.com/web/packages/rftimes/index.html

Summary

Summary

- if the website you want to use as a data basis for your research provides access via an API, you can consider yourself lucky
- if there is a working R-based API client available, you hit the jackpot



Final considerations

- please always cite package authors when you use their work. run `citation("<package name>")` to see how
- sometimes API clients are not up to date. Consider to notify the author or help update the package
- sometimes API clients provide functionality for only a fraction of the API's capability. It's always worth to check out the API documentation if you are looking for specific features

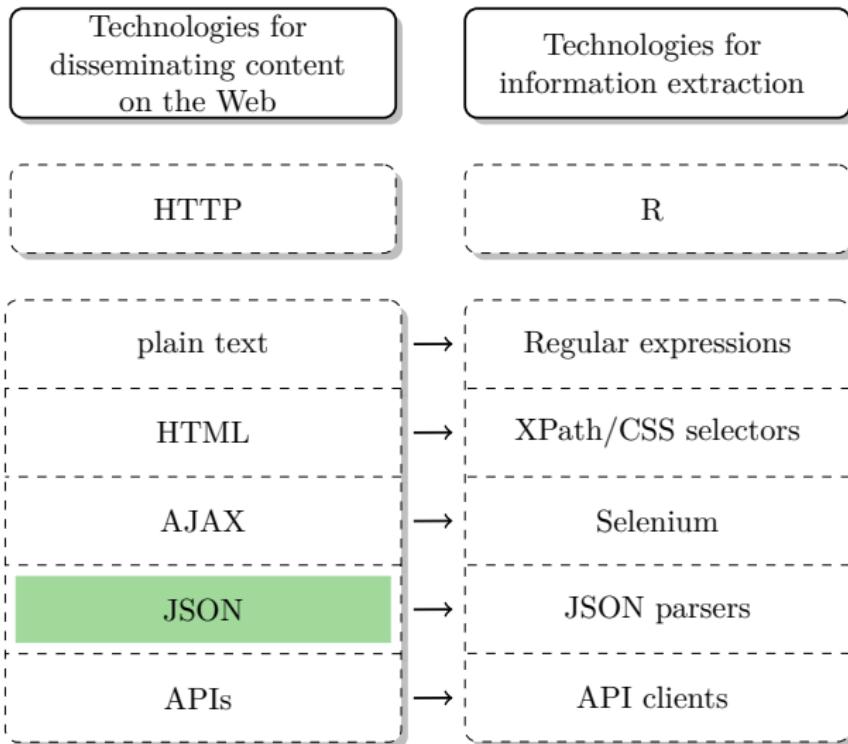
Campus Luzern

A Primer to Web Scraping with R

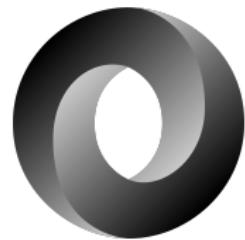
JSON

Simon Munzert
Hertie School of Governance, Berlin
May 20-21, 2019

Technologies of the World Wide Web



- **JavaScript Object Notation**
- popular data exchange format for web services / APIs
- ‘the fat-free alternative to XML’
- JSON ≠ Java, but a subset of JavaScript
- however, very flexible and not dependent upon any programming language
- import of JSON data into R is relatively straightforward with the **jsonlite** package



Example

```
1 {"indy movies": [
2     {"name": "Raiders of the Lost Ark",
3      "year": 1981,
4      "actors": {
5          "Indiana Jones": "Harrison Ford",
6          "Dr. Rene Belloq": "Paul Freeman"
7      },
8      "producers": ["Frank Marshall", "George Lucas", "Howard Kazanjian"],
9      "budget": 18000000,
10     "academy_award_ve": true},
11     {"name": "Indiana Jones and the Temple of Doom",
12      "year": 1984,
13      "actors": {
14          "Indiana Jones": "Harrison Ford",
15          "Mola Ram": "Amish Puri"
16      },
17      "producers": ["Robert Watts"],
18      "budget": 28170000,
19      "academy_award_ve": true}
20 ]
21 }
```

Types of brackets

1. curly brackets, ‘{’ and ‘}', embrace **objects**. Objects work similar to elements in XML/HTML and can contain other objects, key-value pairs or arrays
2. square brackets, ‘[’ and ‘]’, embrace **arrays**. An array is an ordered sequence of objects or values.

Key-value pairs

1. Keys are put in quotation marks; values only if they contain string data.

```
1 "name" : "Indiana Jones and the Temple of Doom"  
2 "year" : 1984
```

2. Keys and values are separated by a colon.

```
1 "year" : 1981
```

3. Key-value pairs are separated by commas.

```
1 {"Indiana Jones": "Harrison Ford",  
2 "Dr. Rene Belloq": "Paul Freeman"}
```

4. Values within arrays are separated by commas.

```
1 ["Frank Marshall", "George Lucas", "Howard Kazanjian"]
```

Data types

Summary

- JSON allows a basic set of data types
- often equivalent data types available in different programming languages
- compatibility with R partly given, but no isomorphic translation possible

data type	meaning
number	integer, real, or floating point (e.g., 1.3E10)
string	whitespace, zero or more Unicode characters (except " or \; \ introduces some escape sequences)
boolean	<code>true</code> or <code>false</code>
null	<code>null</code> , an unknown value
Object	content in curly brackets
Array	ordered content in square brackets

Parsing software

- different packages available for R: `rjson`, `RJSONIO`, `jsonlite`
- choose `jsonlite`: it's under active development and provides convincing mapping rules

JSON and R

Parsing JSON with `jsonlite`

R code

```
1 (indy <- fromJSON("../materials/indy.json"))
$`indy movies`  
name year actors.Indiana Jones  
1 Raiders of the Lost Ark 1981 Harrison Ford  
2 Indiana Jones and the Temple of Doom 1984 Harrison Ford  
3 Indiana Jones and the Last Crusade 1989 Harrison Ford  
actors.Dr. Rene Belloq actors.Mola Ram actors.Walter Donovan  
1 Paul Freeman <NA> <NA>  
2 <NA> Amish Puri <NA>  
3 <NA> <NA> Julian Glover  
producers budget academy_award_
ve  
1 Frank Marshall, George Lucas, Howard Kazanjian 18000000  
TRUE  
2 Robert Watts 28170000  
TRUE  
3 Robert Watts, George Lucas 48000000  
FALSE
```

Mapping rules of jsonlite

R code

```
2 library(jsonlite)
3 x <- '[1, 2, true, false]'
4 fromJSON(x)
[1] 1 2 1 0
5 x <- '["foo", true, false]'
6 fromJSON(x)
[1] "foo"    "TRUE"   "FALSE"
7 x <- '[1, "foo", null, false]'
8 fromJSON(x)
[1] "1"      "foo"    NA       "FALSE"
```

end

- there is no ultimate JSON-to-R converting function
- **jsonlite** simplifies matters a lot
- usually, JSON files returned by web services are not too complex



Campus Luzern

A Primer to Web Scraping with R

Accessing APIs from Scratch

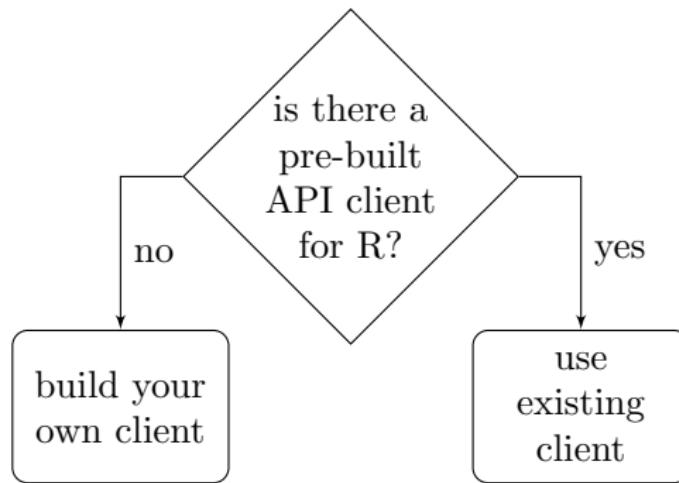
Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

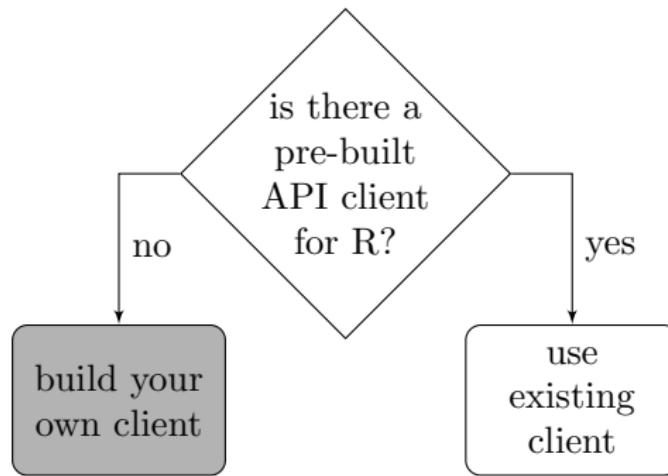
API access with R

There are basically **two scenarios**:



API access with R

There are basically **two scenarios:**



Here, we talk about the case where you have to build your own API binding.

Accessing APIs from Scratch

Steps to be taken

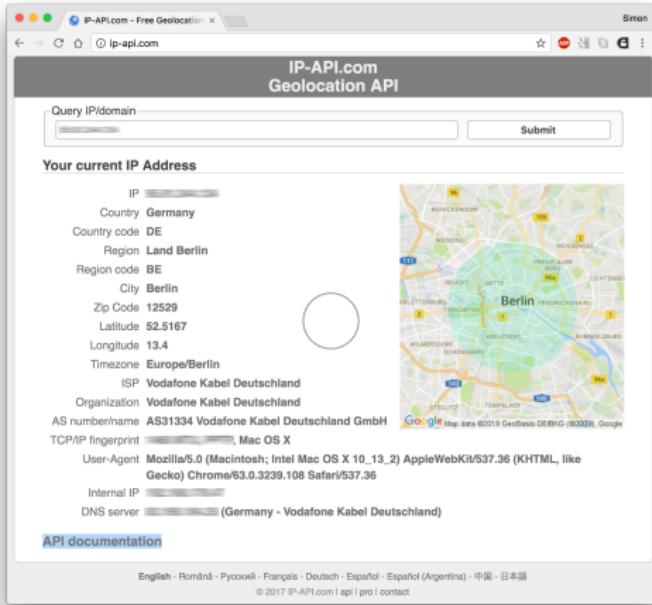
1. figure out how the API works: every API has a human-readable documentation for developers
2. build access to the API via R
3. build functionality that processes the data output (e.g., turns it into R objects)

Example

Example

The IP API

- available at
<http://ip-api.com/>
- its purpose: parses your IP (or a bunch of given IPs) and returns some values, including guessed location, service provider, and organization behind the IP



The screenshot shows the IP-API.com geolocation API interface. At the top, there's a search bar labeled "Query IP/domain" with the placeholder "125.29.52.167" and a "Submit" button. Below the search bar, it says "Your current IP Address". To the right is a map of Berlin, Germany, with various districts like Mitte, Kreuzberg, and Friedrichshain highlighted in green and yellow. A circular placeholder icon is positioned over the map. On the left side of the map, there's a list of geographical and technical details:

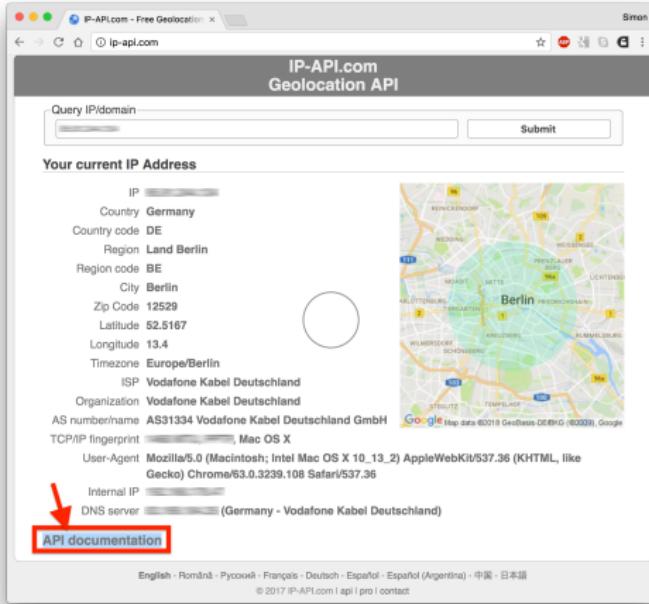
- IP 125.29.52.167
- Country Germany
- Country code DE
- Region Land Berlin
- Region code BE
- City Berlin
- Zip Code 12529
- Latitude 52.5167
- Longitude 13.4
- Timezone Europe/Berlin
- ISP Vodafone Kabel Deutschland
- Organization Vodafone Kabel Deutschland
- AS number/name AS31334 Vodafone Kabel Deutschland GmbH
- TCP/IP fingerprint [REDACTED] Mac OS X
- User-Agent Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.108 Safari/537.36
- Internal IP [REDACTED]
- DNS server [REDACTED] (Germany - Vodafone Kabel Deutschland)

At the bottom of the interface, there's a link to "API documentation" and a footer with language links: English · Română · Русский · Français · Deutsch · Español · Español (Argentina) · 中文 · 日本語. The footer also includes copyright information: © 2017 IP-API.com | api | pro | contact.

Example

The IP API

- available at
<http://ip-api.com/>
- its purpose: parses your IP (or a bunch of given IPs) and returns some values, including guessed location service provider, and organization behind the IP
- check out API documentation



The screenshot shows the IP-API.com geolocation API interface. A red arrow points to the "API documentation" link at the bottom of the page.

Query IP/domain: [REDACTED]

Your current IP Address

IP: [REDACTED]

Country: Germany

Country code: DE

Region: Land Berlin

Region code: BE

City: Berlin

Zip Code: 12529

Latitude: 52.5167

Longitude: 13.4

Timezone: Europe/Berlin

ISP: Vodafone Kabel Deutschland

Organization: Vodafone Kabel Deutschland

AS number/name: AS31334 Vodafone Kabel Deutschland GmbH

TCP/IP fingerprint: [REDACTED], Mac OS X

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.108 Safari/537.36

Internal IP: [REDACTED]

DNS server: [REDACTED] (Germany - Vodafone Kabel Deutschland)

[API documentation](#)

English · Română · Русский · Français · Deutsch · Español · Español (Argentina) · 中国 · 日本語
© 2017 IP-API.com | [api](#) | [pro](#) | [contact](#)

Example

The IP API

- access is free for non-commercial use (up to 150 requests per minute)
- various response formats available
- paid pro service available

The screenshot shows a web browser window displaying the documentation for the IP Geolocation API at ip-api.com/docs/. The page title is "IP Geolocation API". On the left, there's a sidebar with a "Table of Contents" section containing links to "IP Geolocation API", "About", "Response formats and examples", and "Usage limits". The main content area has several sections: "About" (describing free usage), "Response formats and examples" (listing CSV, JSON, Newline Separated, Serialized PHP, XML, Change Log, Statistics, DNS API, IP Geolocation API, and Urban IP), "Usage limits" (warning about a ban for over 150 requests/min), and a note for commercial users. At the bottom right is a "Back to top" button.

Example

The IP API

- to access the API, we have to send a **GET** request to <http://ip-api.com/json>
- we can provide any IP we want
- the response is raw JSON code

The screenshot shows a web browser displaying the IP Geolocation API documentation at <http://ip-api.com/docs/json>. The page has a sidebar with links to CSV, Error messages, JSON, Batch JSON, Nested, Separated, Return values, Serialized PHP, XML, Change Log, Statistics, DNS API, IP Geolocation API, and Urban IP. The main content area is titled "JSON" and contains sections for "Usage" and "Response". Under "Usage", it says "To receive the response in JSON format, send a GET request to" and provides a link to <http://ip-api.com/json>. It also notes that you can supply an IP address or domain to lookup, or none to use your current IP address. Under "Response", it says "A successful request will return, by default, the following:" and shows a JSON object structure:

```
{  
    "status": "success",  
    "country": "COUNTRY",  
    "countryCode": "COUNTRY CODE",  
    "region": "REGION CODE",  
    "regionName": "REGION NAME",  
    "city": "CITY",  
    "zip": "ZIP CODE",  
    "lat": LATITUDE,  
    "lon": LONGITUDE,  
    "timezone": "TIME ZONE",  
    "isp": "ISP NAME",  
    "org": "ORGANIZATION NAME",  
    "as": "AS NUMBER / NAME",  
    "query": "IP ADDRESS USED FOR QUERY"  
}
```

Below the JSON object, there is a section titled "List of returned values" and a note about failed requests.

Example

IP API access in R

A **GET** request essentially means that we assemble a URL using

- the API **endpoint** (a basic URL) and
- **parameters-value pairs** that are added to the URL
- here, we only add the literal IP address
- we use **fromJSON** to pose the request and directly parse the data from the API

R code

```
1 url <- "http://ip-api.com/json"
2 url_ip <- paste0(url, "/208.80.152.201")
3 ip_parsed <- jsonlite::fromJSON(url_ip)
```

end

Example

Investigating the output

R code

```
4 names(ip_parsed)
[1] "as"          "city"        "country"      "countryCode"  "isp"
[6] "lat"         "lon"         "org"         "query"       "region"
[11] "regionName" "status"      "timezone"    "zip"

5 ip_parsed
$as
[1] "AS14907 Wikimedia Foundation Inc."

$city
[1] "San Francisco"

$country
[1] "United States"

$countryCode
[1] "US"
```

Example

Bringing it into shape

- in our case, `fromJSON()` has created an R list object
- we turn it into a data frame

R code

```
6 ip_parsed %>% as.data.frame(stringsAsFactors = FALSE)
               as      city      country
1 AS14907 Wikimedia Foundation Inc. San Francisco United States
   countryCode          isp      lat      lon
1           US Wikimedia Foundation Inc. 37.7895 -122.403
               org      query region regionName  status
1 Wikimedia Foundation Inc 208.80.152.201       CA California success
   timezone     zip
1 America/Los_Angeles 94104
```

end

Summary

Summary

- accessing APIs from scratch can be almost as easy as using a readily available R client
- often however, APIs are more complex (provide much more parameters and variations in data output)
- ideally, you build functions around your API calls to make it even more accessible



Campus Luzern

A Primer to Web Scraping with R

API Authentication

Simon Munzert

Hertie School of Governance, Berlin

May 20-21, 2019

API access limits

Why API access can be restricted

- service provider wants to know who uses their API
- hosting APIs is costly—API usage limits can help control costs
- commercial interest of API hoster: you pay for access
(sometimes for advanced features or massive amounts of queries only)

Access tokens

- access tokens serve as the key to the API
- they usually come in form of a randomly generated string, such as `dk5nSj485jJZP3847kjU`
- obtaining a token requires registration (often email address is sufficient)
- sometimes you have to disclose your intentions
- once you have the token, you usually pass it along with your regular API query

Example

Example

The OpenWeatherMap API

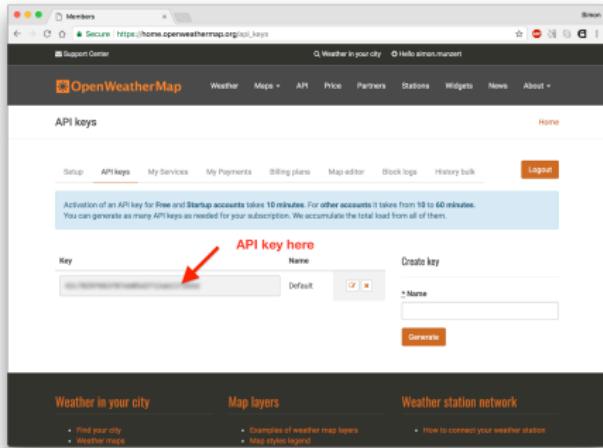
- available at <http://openweathermap.org>
- free access to basic weather data (paid plans for historical and more detailed data)
- sign-up necessary

The screenshot shows the OpenWeatherMap API documentation page. At the top, there's a navigation bar with links for Members, Support Center, Weather, Maps, API, Price, Partners, Stations, Widgets, News, and About. Below the navigation, the main content area has a heading "Weather API". A sub-section titled "Our weather API is simple, clear and free. We also offer higher levels of support, please see our [paid plan options](#). To access the API you need to sign up for an [API key](#) if you are on a free or paid plan." is displayed. The page is divided into several sections: "Current weather data", "5 day / 3 hour forecast", "16 day / daily forecast", "Historical data", "History Bulk", and "Weather map layers". Each section contains a brief description and a "View API" button.

Example

The OpenWeatherMap API

- available at <http://openweathermap.org>
- free access to basic weather data (paid plans for historical and more detailed data)
- sign-up necessary
- sign-up via browser—name and email address suffices



Example

Accessing the API from R

- copy the key to a string and store it in a local file
- import the key when you want to tap the API

R code

```
1 openweathermap <- "k4875jHkdf9kBbiuZ208d7s"  
2 save(openweathermap, file = "/Users/s.munzert/rkeys.RDa")
```

end

R code

```
3 load("/Users/s.munzert/rkeys.RDa")  
4 apikey <- paste0("&appid=", openweathermap)
```

end

Example

Accessing the API from R

- send along the API key when you make queries to the API

R code

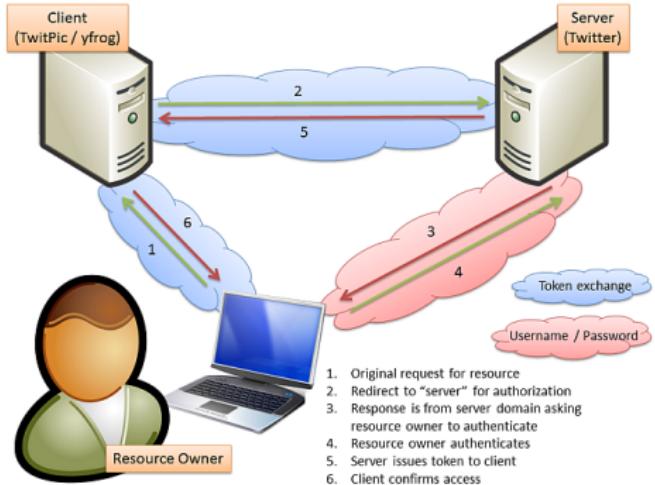
```
5 endpoint <- "http://api.openweathermap.org/data/2.5/find?"  
6 city <- "Berlin, Germany"  
7 metric <- "&units=metric"  
8 url <- paste0(endpoint, "q=", city, metric, apikey)  
9 jsonlite::fromJSON(url, flatten = TRUE)$list[1, ]  
  
          id      name           dt snow             weather coord.lat  
1 2950159 Berlin 1557240313    NA 800, Clear, Sky is Clear, 01d      52.517  
   coord.lon main.temp main.pressure main.humidity main.temp_min  
1     13.3889      13.43        1016            32        12.78  
   main.temp_max wind.speed wind.deg wind.gust sys.country rain.1h  
1            15        4.6        270            NA         DE       NA  
   clouds.all  
1            0  
  
end
```

OAuth authorization

Accessing APIs with OAuth authorization

What's OAuth?

- authorization standard
- used to provide client applications access to owner's resources—without giving them passwords
- frequently used by major companies (Twitter, Facebook, Amazon, Google, ...) to allow third party applications to interact with user's accounts



Source: <http://www.ubelly.com/wp-content/uploads/2010/02/OAuth1.png>

Accessing APIs with OAuth authorization

The OAuth workflow with `httr`

- `oauth_endpoint()`: define OAuth endpoints for the request and access token
- `oauth_app()`: bundle key and secret to request access credentials
- `oauth1.0_token()` and `oauth2.0_token()`: exchange consumer key and secret for access key and secret

... but often the R API client simplifies matters (e.g., the `rtweet` package)