Journal of the Midwest Association for Information Systems

Volume 2021 | Issue 1 Article 5

Date: 01-31-2021

Web Scraping in the R Language: A Tutorial

Vlad Krotov
Murray State University, vkrotov@murraystate.edu

Matthew F. Tennyson
Murray State University, mtennyson@murraystate.edu

Abstract

Information Systems researchers can now more easily access vast amounts of data on the World Wide Web to answer both familiar and new questions with more rigor, precision, and timeliness. The main goal of this tutorial is to explain how Information Systems researchers can automatically "scrape" data from the web using the R programming language. This article provides a conceptual overview of the Web Scraping process. The tutorial discussion is about two R packages useful for Web Scraping: "rvest" and "xml2". Simple examples of web scraping involving these two packages are provided. This tutorial concludes with an example of a complex web scraping task involving retrieving data from Bayt.com - a leading employment website in the Middle East.

Keywords: Web Scraping, R, RStudio, HTML, CSS, XML, rvest, xml2

DOI: 10.17705/3jmwa.000066

1. Introduction

Data available on the World Wide Web is measured in zettabytes (Cisco Systems, 2017). This vast volume of data presents researchers and practitioners with a wealth of opportunities for gaining additional insights about individuals, organizations, or even macro-level socio-technical phenomena in real time. Not surprisingly, Information Systems researchers are increasingly turning to the web for data that can be used to address their research questions (e.g. see Geva et al., 2017; Gunarathne et al., 2018; Triche and Walden 2018; Vaast et al., 2017)

Harnessing the vast data from the web often requires a programmatic approach and a good foundation in various web technologies. Besides vast volume, there are three other common issues associated with accessing and parsing the data available on the web: variety, velocity, and veracity (Goes, 2014; Krotov & Silva, 2018). First, web data comes in a variety of formats that rely on different technological and regulatory standards (Basoglu & White, 2015). Second, this data is characterized by extreme velocity. The data on the web is in a constant state of flux, i.e., it is generated in real time and is continuously updated and modified. Another characteristic of web data is veracity (Goes, 2014). Due to the voluntary and often anonymous nature of the interactions on the web, quality and availability of web data are surrounded with uncertainty. A researcher can never be completely sure if the data he or she needs will be available on the web and whether the data is reliable enough to be used in research (Krotov & Silva, 2018).

Given these issues associated with "big web data", harnessing this data requires a highly customizable, programmatic approach. One functional and easily-customizable platform for retrieving and analyzing web data is R - one of the most widely-used programming languages in Data Science (Lander, 2014). R can be used not only for automating web data collection, but also for analyzing this data using multiple techniques. Currently, there are more than 16,000 R packages for various data analysis techniques - from basic statistics to advanced machine learning (CRAN, 2020). Some packages are useful for web crawling and scraping, pre-processing, and organizing data stored on the web in various formats.

The following sections introduce the reader to the web scraping infrastructure in R. First, a general overview of the web scraping process is provided. This overview provides a high-level understanding of the steps and technologies involved in automatically sourcing data from the web. Second, the tutorial provides an overview of the "rvest" and "xml2" R packages. These packages are useful for writing web crawling applications and retrieving data from the web in various formats. Third, the article provides simple examples of web scraping code written in R together with a longer and more complex example of automating a web scraping task. The final section discusses implications for researchers and practitioners and the usefulness of the web scraping approach.

2. Web Scraping in R: An Overview

In this tutorial, web scraping is broadly defined as using technology tools for automatic retrieval and organization of data from the web for the purpose of further analysis of this data (Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020). Web scraping consists of the following main, intertwined phases: website analysis, website crawling, and data organization (see Figure 1) (Krotov & Silva, 2018). Although listed in order, these phases are often intertwined. A researcher has to go back and forth between those phases until a clean, tidy dataset suitable for further analysis is obtained.

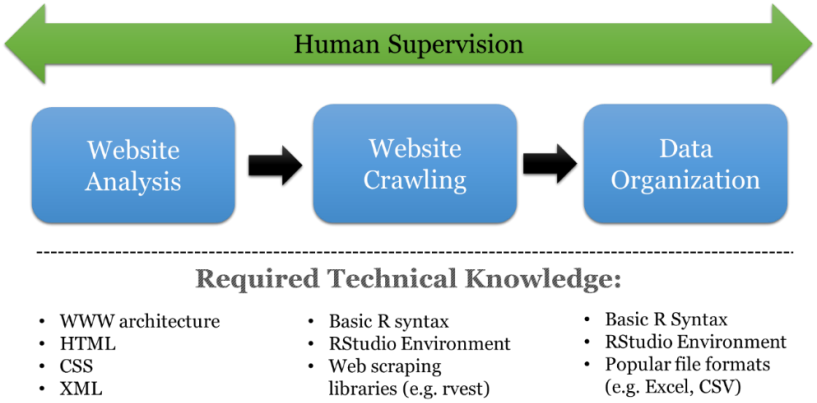


Figure 1. Web Scraping (Adapted from Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020)

The "Website Analysis" phase involves examining the underlying structure of a website or a web repository in order to understand how the needed data is stored at the technical level (Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020). This is often done one web page at a time. This analysis requires a basic understanding of the World Wide Web architecture and some of the most commonly used Web technologies used for storing and transmitting data on the web: HTML, CSS, and XML.

The "Web Crawling" phase involves developing and running a script that automatically browses (or "crawls") the web and retrieves the data needed for a research project (Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020). These crawling applications (or scripts) are often developed using programming languages such as R or Python. We argue that R is especially suitable for this purpose. This has to do with the overall popularity of R in the Data Science community and availability of various packages for automatic crawling (e.g. the "rvest" package in R) and parsing (e.g. the "xml2" package in R) of web data. Furthermore, once data is retrieved using R, it can be subjected to various forms of analysis available in the form of R packages. Thus, R can be used to automate the entire research process – from the time data is acquired to the time visualizations and written reports are produced for a research paper or presentation (the latter can be accomplished with the help of the package called "knitr").

The "Data Organization" phase involves pre-processing and organizing data in a way that enables further analysis (Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020). In order to make further analysis of this data easy, the data needs be to be clean and tidy. Data is in a tidy format when each variable comprises a column, each observation of that variable comprises a row, and the table is supplied with intuitive linguistic labels and the necessary metadata (Wickham, 2014b). A dataset is "clean" when each observation is free from redundancies or impurities (e.g. extra white spaces or mark-up tags) that can potentially stand in the way of analyzing the data and arriving to valid conclusions based on this data. This often requires the knowledge of various popular file formats, such as Excel or CSV. Again, R contains various packages and built-in functions for working with a variety of formats. This is another set of features that make R especially suitable for web scraping.

Most of the time, at least some of the processes within these three phases cannot be fully automated and require at least some degree of human involvement or supervision (Krotov & Tennyson, 2018; Krotov & Silva, 2018; Krotov et al., 2020). For example, "ready-made" web scraping tools often select wrong data elements from a web page. This often has to do with poor instructions supplied by the user of such tools or an ambiguous mark-up used to format data. These tools also often fail to save the necessary data elements in the "tidy data" format (Wickham, 2014b), as data cleaning often requires human interpretation of what this data represents. Moreover, numerous networking errors are possible during web crawling (e.g. an unresponsive web server) that require troubleshooting by a human. Finally, things change so fast on the web! A ready-made tool that is working now may not work in the future due to changes made to a website. This is due to the fact that a ready-made tool may not be fully customizable or modifiable (at least, not from the perspective of a regular user). Thus, changes in the underlying technology behind a website may not be accommodated using the existing features of the tool. All these problems become more acute for large, complicated web scraping tasks, making these "ready-made" tools practically useless. Thus, at least with the current state of technology, web scraping often cannot be fully automated and requires a "human touch" together with a highly customizable approach.

3. R Packages Needed for Web Scraping

Developing custom tools for web scraping requires a general understanding of the web architecture; a good foundation in R programming and RStudio Environment, and at least basic knowledge of some of the most commonly used mark-up languages on the web, such as HTML, CSS, and XML. This tutorial assumed that the readers have the necessary foundation in the aforementioned tools and technologies. If not, then the readers can use the resources listed in Appendix A to get the necessary background knowledge.

Skipping all these foundational knowledge areas, this part of the tutorial focuses on the functionality of two R packages available from the Comprehensive R Archive Network (CRAN): "rvest" and "xml2". Both packages can be downloaded and installed for free from CRAN (see https://cran.r-project.org/). The primary use of the "rvest" package is simulating browser sessions necessary for "crawling" a website. Although the "rvest" package also has some tools for data parsing (e.g. saving HTML code as text), this functionality is often derived from the "xml2" package, which contains a wide variety of tools necessary for parsing data. The functionalities of both packages are explained in detail in the sections below. Short web scraping example that rely on these packages are provided at the end of this section.

3.1 Rvest Package

This section contains an overview of the R package called "rvest". Some tables, examples, and related explanations in this section come from Krotov & Tennyson (2018) and Wickham (2016). There are many features that make "rvest" useful for accessing and parsing data from the web. First, "rvest" contains many functions that can be used for simulating sessions of a web browser. These features come in handy when one needs to browse through many webpages to download

the needed data (this is referred to as the "web crawling" process). Second, "rvest" contains numerous functions for accessing and parsing data from web documents in HTML, XML, CSS, and JSON formats.

Some of the most essential functions of the rvest package are listed in Table 1. Many other functions are available as a part of "rvest" package (Wickham, 2016). One should refer to the official "rvest" documentation to learn more about the package and its usage (see Wickham, 2016).

Table 1. Some Functions of the rvest Package (Reprinted from Wickham, 2016; Krotov & Tennyson, 2018)

Function Usage and Purpose	**Arguments**
Usage: read_html(x, ..., encoding = "") Purpose: This function reads HTML code of a web page from which data is to be retrieved. Can be used for reading XML as well.	• x: A url, a local path, or a string containing HTML code that needs to be read • …: Additional arguments can be passed to a URL using the GET() metod • encoding: specify encoding of the web page being read
Usage: html_nodes(x, css, xpath) Purpose: This function is used to select specific elements of a web document. To select these specific elements one can use CSS elements which contain the needed data or use XPath language to specify the "address" of an element of a web page	• x: A document, a node, or a set of nodes from which data is selected • css, xpath: a name of a CSS element or an XPath 1.0 link can be used to select a node
Usage: html_session(url, ...) Purpose: This function allows to start a web browsing session to browse HTML pages for the purpose of collecting data from them.	• url: address of a web page where browsing starts • …: Any additional httr config commands to use throughout session
Usage: html_table(x, header = NA, trim = TRUE, fill = FALSE, dec = ".") Purpose: This function can be used to read HTML tables into data frames (a commonly used data structure in R). Can be especially useful for reading HTML tables containing financial data.	• x: A node, node set or document • header: if NA, then the first row contains data and not column labels • trim: if TRUE, the function will remove leading and trailing whitespace within each cell • fill: If TRUE, automatically fill rows with fewer than the maximum number of columns with NAs • dec: The character used as decimal mark for numbers

3.2 Xml2 Package

In the past, the rvest package was also used to with XML documents using such functions as xml_node(), xml_attr(), xml_attrs(), xml_text() and xml_tag(). Eventually, these XML functions branched out into "xml2" package designed specifically to work with XML files (including with XML files retrieved from the web). The package has numerous functions for working with XML data. Some of the most commonly used functions of this package together with their commonly used arguments are listed in Table 2 below. Additional details about the functions listed in Table 2 (together with many other functions omitted here) are provided in Wickham et al. 2018.

Table 2. Some Functions of the xml2 Package (Reprinted from Wickham et al., 2018)

Function Usage and Purpose	**Arguments**
Usage: read_xml(x, encoding = "", ..., as_html = FALSE, options = "NOBLANKS") Purpose: This function reads XML and HTML files.	• x: a string, a connection, or a raw vector • encoding: specify a default encoding for the document • ...: additional arguments passed on to method • as_html: optionally parse an XML file as if it's HTML • options: set parsing options for the libxml2 parser
Usage: xml_children(x); xml_contents(x); xml_parents(x); xml_siblings(x) Purpose: These functions are used for navigating through an XML document structure. xml_children returns only elements, xml_contents returns all nodes, xml_parents returns all parents up to the root, xml_siblings returns all nodes at the same level,	• x: a document, node, or node set.
Usage: xml_find_all(x, xpath) Purpose: Finds all XML nodes that match a particular XPath expression	• x: a document, node, or node set. • xpath: a string containing an xpath (1.0) expression
Usage: xml_text(x, trim = FALSE); xml_set_text(x, value) Purpose: Used to extract or replace text in an XML document, node, or node set.	• x: a document, node, or node set • trim: If TRUE will trim leading and trailing spaces • value: character vector with replacement text

4. Simple Web Scraping Examples

The four examples provide below show how to use the "xml2" and "rvest" packages for accessing data in XML, HTML, and CSS format.

The first example comes from Krotov and Tennyson (2018). The example illustrates how "xml2" package is used to extract financial data from an online XML document. The document can be found here:

https://www.sec.gov/Archives/edgar/data/1067983/000119312518238892/brka-20180630.xml

The XML document is a 10-Q statement for Berkshire Hathaway, Inc. posted on EDGAR (an open web database of financial statements of publicly listed companies). Technically, the document is an XBRL (eXtensible Business Reporting Language) instance file. XBRL is an extension of XML language used specifically for interexchange of financial reporting data over the web. Since XBRL is based on XML, one can use the common functions of "xml2" package to work with this data. More specifically, the example below extracts the Net Income data reported by Berkshire Hathaway and saves this data into a data frame named Net_Income_Data.

```
#This example requires xml2 R package
require(xml2)

#Parse XML from an online document found at the URL specified below
URL <- "https://www.sec.gov/Archives/edgar/data/1067983/000119312518238892/brka-20180630.xml"
XML_data <- read_xml(URL)

#Save all nodes related to Net Income or Loss as defined by US GAAP
Nodes <- xml_find_all(XML_data, ".//us-gaap:NetIncomeLoss")

#Convert the node structure into a character vector
Nodes_Vector <- as.character(Nodes)

#Retreive values for all Net Income or Loss elements and save them into a vetcor
#These are dollar values for each Net Income or Loss item reported in the 10Q document
Nodes_Values <- xml_text(xml_find_all(XML_data, ".//us-gaap:NetIncomeLoss"))

#Bind the vectors together as columns and convert the structure into a data frame
#The data frame contains two columns and four rows
Net_Income_Data <- data.frame(cbind(Nodes_Vector,Nodes_Values))
```

The resulting data frame (Net_Income_Data) contains various XBRL attributes of each node containing Net Income or Loss data together with the dollar value of Net Income or Loss (in dollars) reported for each of these items. The definitions of each of the four Net Income or Loss items can be explored further by analyzing the values of attributes for each of the nodes saved in the data frame.

The next example also comes from Krotov and Tennyson (2018). This example illustrates how the "rvest" package can be used for accessing data from an HTML page available on the web:

https://www.sec.gov/Archives/edgar/data/1067983/000119312516760194/d268144d10q.htm

This HTML page also contains a 10-Q statement posted by Berkshire Hathway on EDGAR, albeit from a different period. This time, the goal is to retrieve Balance Sheet data from the statement. This data is available in the form of an HTML table. The HTML table is a part of the HTML web page containing the entire 10-Q statement by Berkshire Hathaway. The table data is retrieved and saved into a data frame called "balance_sheet_data" using the R code below.

```
#This script requires the rvest package to be installed and activated
require(rvest)

#The variable url contains a URL to the 10-Q document published via EDGAR
url <- "https://www.sec.gov/Archives/edgar/data/1067983/000119312516760194/d268144d10q.htm"

#read_html() function from rvest package used to read HTML code from page
tenqreport_html <- read_html(url)

#xpath used to retreive HTML code specifically for balance sheet table
balance_sheet_html <- html_nodes(tenqreport_html,
xpath='/html/body/document/type/sequence/filename/description/text/table[7]')

#HTML code from the balance sheet table is obtained
balance_sheet_list <- html_table(balance_sheet_html)

#Balance sheet data is saved from a list into a data frame
balance_sheet_table <- balance_sheet_list[[1]]
```

Once balance sheet data is saved into a data frame (named "balance_sheet_table" in this example), individual values from the balance sheet can be accessed and used in calculations. For example, certain accounting ratios can be calculated. Alternatively, one can match the balance sheet with other data related to the company and available on the web. The "rvest" package can be used in a similar fashion to obtain quantitative and qualitative data from the same HTML document or other sources on the web.

The code below illustrates how to access top reviews for the iPadPro product listed on Amazon.com. This time, a CSS selector is used to retrieve top reviews for this particular product listed on Amazon. The SelectorGadget tool was used to find an XPath for the CSS element containing the top reviews. The resulting variable review_text is a character vector containing 6 reviews. Note that only a portion of one review was displayed to save space.

```
#Require the rvest package
require(rvest)

#Read HTML code for Apple iPad Pro
ipad_page <-
read_html("https://www.amazon.com/gp/product/B01CGXU0GM/ref=s9_dcacsd_dcoop_bw_c_x_17_w")

#Access top reviews for the product and save them in review_text
review <- html_nodes(ipad_page, xpath='//*[contains(concat( " ", @class, " " ), concat( " ", "a-expander-
partial-collapse-content", " " ))]')
review_text <- html_text(review)

#Display top customer reviews
review_text

## [6] "To fully understand my review, must explain a few things. I come to owning the iPad Pro 9.7\" via
a long lineage of Macs, iPods, iPhones and iPads.
```

The final example is supplied with the "rvest" package (Wickham, 2016). It involves retrieving rating and cast data for "The Lego Movie" found on IMDb website from corresponding HTML or CSS elements. Again, the SelectorGadget can be used to find XPath link associated with each of these elements – this should save time and eliminate the need to be knowledgeable in the particularities of XPath syntax. The "%>%" string in the example below is the so called "pipe" operator used to pass results from one function to another function. Thus, the operator simplified the code by creating a "pipeline" that performs work on data using various functions. The contents of the "rating" and "cast" variables are displayed.

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

rating <- lego_movie %>%
  html_nodes("strong span") %>%
  html_text() %>%
  as.numeric()
rating
#> [1] 7.8

cast <- lego_movie %>%
  html_nodes("#titleCast .itemprop span") %>%
  html_text()
cast
#>  [1] "Will Arnett"     "Elizabeth Banks" "Craig Berry"
#>  [4] "Alison Brie"     "David Burrows"   "Anthony Daniels"
#>  [7] "Charlie Day"     "Amanda Farinos"  "Keith Ferguson"
#> [10] "Will Ferrell"    "Will Forte"      "Dave Franco"
#> [13] "Morgan Freeman"  "Todd Hansen"     "Jonah Hill"
```

As one can see from the examples provided this section, "rvest" is a robust package that can be fine-tuned to automatically scrape data for virtually any Information Systems research project requiring web data. The next section contains a more elaborate example of a web scraping project relying on the "rvest" package.

5. A Complex Web Scraping Example

This section contains an example of a web scraping project involving Bayt.com, a leading employment website in the Middle East. At any point in time, the website contains thousands of employment ads from various industries and for

a diverse set of roles. The data collected from the website can be used to answer a number of interesting questions about IT competencies together with competencies in other industries and roles. The example is structured in accordance with the three phases of web scraping discussed previously: Website Analysis, Website Crawling, and Data Organization.

Apart from being larger and more complex, there are three features of this more elaborate example that make it different from the examples provided earlier. First, the dataset retrieved is fairly large. Second, retrieving the dataset involves "crawling" the website one page at a time. This is in contrast to the previous examples where data is accessed from a single web page. Third, the data is saved into a file on the computer after the web scraping task is completed.

5.1 Website Analysis

The web scraping project aimed at retrieving data from Bayt.com starts with the examination of the underlying structure of the website. The structure of the Bayt.com site is fairly typical for a job posting site. One way to browse job postings is by sector (http://www.bayt.com/en/international/jobs/sectors/). When viewing the "Jobs by Sector" page, a list of sectors (such as "Technology and Telecom") is displayed, each as a hyperlink (see Figure 2).

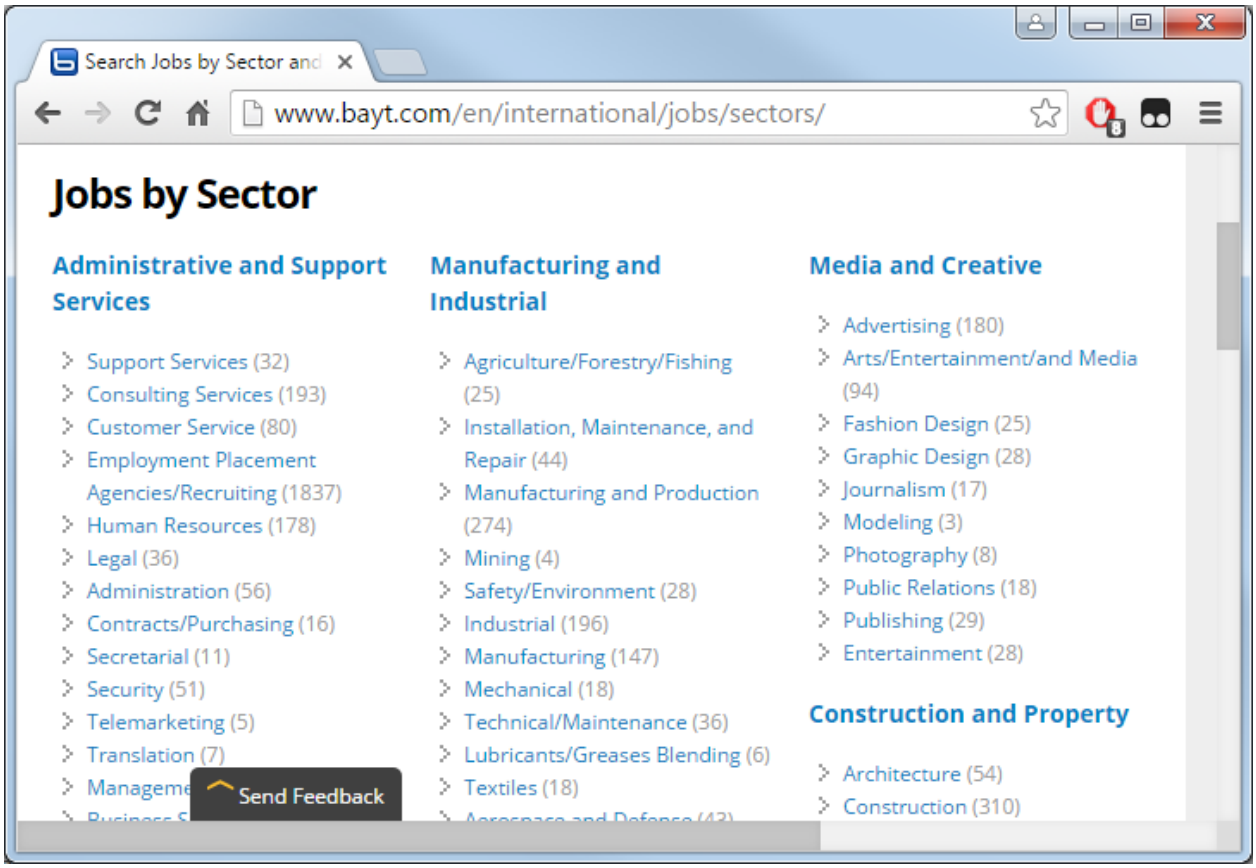


Figure 2. The Bayt.com website when viewing "Jobs by Sector"

Each job sector, therefore, can be accessed using a unique URL. The "Technology and Telecom" and "Banking and Finance" sectors, for example, can be accessed using the following URLs:

https://www.bayt.com/en/international/jobs/sectors/technology-telecom/

https://www.bayt.com/en/international/jobs/sectors/banking-finance/

Note that each URL shares the same base URL, and each is distinguished only by its respective subfolder. This observation will be useful when scraping the website sector by sector later during the Web Crawling phase.

When a particular sector's webpage is viewed, links to the individual job postings are listed, but only in "pages" of twenty at a time. Each page of jobs can be accessed through its own URL, as follows:

https://www.bayt.com/en/international/jobs/sectors/banking-finance/?page=1

https://www.bayt.com/en/international/jobs/sectors/banking-finance/?page=2

https://www.bayt.com/en/international/jobs/sectors/banking-finance/?page=3

On each of these pages, its twenty unique job postings are represented in HTML as a series of "div" elements. Each "div" element contains a hyperlink with the following structure:

```
<a data-js-aid="jobID" href="...">Job Title</a>
```

So, for each job posting, a hyperlink exists that contains all of the information we are scraping. Notice that the element is identified with a distinctive attribute (data-js-aid) that has "jobID" as the value. Therefore, we will need to collect all of the <a> elements that have that distinctive attribute/value pair. Then we will extract the content of the <a> element to get the title, and we will extract the value of the "href" attribute to get the URL.

Also, on each of these pages, there is a <link> element containing the URL to the next page of jobs. The <link> element has the following structure:

```
<link rel="next" href="..." />
```

This will be the element used to determine the URL of the next page of job listings. We will look for a <link> element with a "rel" attribute whose value is "next". Once that element is found, we will use the value of the "href" attribute to determine the URL of the next page of jobs. We will know that we've reached the end of the job listings for the current sector if such an element doesn't exist, at which point we will move on to the next sector of jobs and continue from there.

In this section, we have evaluated the structure of the website, identified how the website can be methodically and programmatically accessed, identified which HTML elements contain our desired information, and inspected the HTML to determine how those elements can be accessed. Therefore, the "Website Analysis" phase is complete.

5.2 Website Crawling

Once the elements that contain the target data have been identified and the respective HTML has been inspected (during the "Website Analysis" phase as discussed in the previous subsection), the data must actually be extracted. In this subsection, we will walk through the script shown in Appendix B step-by-step to demonstrate how the R language can be used to scrape data from the Bayt.com website.

In overview, the script will operate as follows:

1. The necessary library packages will be imported.
2. The working directory will be set.
3. Important input and output variables will be initialized
4. A loop will be set up to iterate through each job sector. At each iteration, one sector will be processed by visiting that sector's webpage.
5. Another loop will be set up to iterate through each page of jobs for that sector. At each iteration, one page of jobs will be processed. A list of the jobs accessible via that page will be collected.
6. Yet another loop will be set up to iterate through that list of jobs. At each iteration, one job posting will be processed. Each job's title and URL title will be extracted. The information will be saved.
7. Finally, once all of the jobs, pages, and sectors have been processed, a file containing all of the collected data will be written.

The remainder of this section will examine and discuss the corresponding lines of the script in detail.

Lines 1-4:

```
#Required packages
require(tm)
require(rvest)
require(XML)
```

These lines are used to load packages into the current session, so that their functions can be called. As it was explained earlier, the purpose of the rvest package is "to make it easy to download, then manipulate, both html and xml" documents (Wickham, 2016). The XML package contains functions for parsing XML documents, and is required by the rvest package. Various functions that are contained in these packages are used throughout the scripts, and will be discussed within the context of those portions of the script in which they are used.

Lines 6-19:

```
#Set working directory
setwd("C:/Users/.../Research/WebScraping")

#Identify job sectors from which to scrape data
sectors <- c("banking-finance",
             "technology-telecom",
             "media-creative")

#Create frame in which data results will be stored
job_data <- data.frame(Sector=character(0),
                       Title=character(0),
                       URL=character(0))

colnames(job_data) <- c("Sector", "Title", "URL")
```

These lines are used to set up the working environment for the script. The working directory specifies the local directory from which any input files will be read and to which any output files will be written. The directory should be set as desired. The "sectors" vector identifies which job sectors will be queried for job postings. Recall from the "Website Analysis" section that each sector has its own URL such that each URL shares the same base address but is distinguished only by its respective subfolder. The values in this vector represent those subfolders, and so they must match the URL subfolder exactly. The "job_data" frame will be used to store all of the extracted information. Each row of the frame will represent exactly one job posting, while the columns of the frame correspond to the bits of information being collected (i.e., "Sector, "Title", and "URL").

Lines 21-25:

```
#For each job sector (each job sector will be processed one at a time)
for (sector in 1:length(sectors))
{
  base_url <- "https://www.bayt.com/en/uae/jobs/sectors/"
  page_url <- paste(base_url, sectors[sector], sep="")
```

These lines are used to loop through each job sector, one at a time. The first line sets up the loop to iterate exactly as many times as there are sectors. The "base_url" variable represents the base URL for all of the job sector websites. That base URL is then concatenated with the name of the current job sector to create the "page_url" variable.

Lines 27-32:

```
  #For each page of jobs within the sector
  repeat
  {
    #Read website for the current sector page
    page_html <- read_html(html_session(page_url))
    cat(paste(page_url, "\n")) #display script progress
```

These lines are used to loop through each page of jobs within the current sector. The "html_session" function (from the rvest package) is used to send an HTTP request for the specified webpage and simulate the functionality of a browser. The "read_html" function is used to get the raw HTML code for the webpage associated with that session. The "cat" function is used to simply display output to the console as the script is executing to indicate to the user which page is currently being processed.

Lines 34-36:

```
    #Retreive the list of job links
    a_elements <- html_nodes(page_html, "a[data-js-aid=\"jobID\"]")
    cat(paste(length(a_elements), " jobs found\n", sep="")) #display progress
```

These lines are used to collect all of the <a> elements that contain the desired information (job title and URL). Recall

from the "Website Analysis" section that each page contains a list of job postings. For each job posting in the list, an <a> element exists, which contains a hyperlink to the respective job posting. Each of these <a> elements is identified with a distinctive "data-js-aid" attribute having "jobID" as the value. The "html_nodes" function is used to retrieve a list of these relevant <a> elements.

Lines 38-44:

```
#For each job within the current page (process one job at a time)
for (i in 1:length(a_elements))
{
  job_href <- xml_attr(a_elements[i], "href")
  job_url <- paste("https://www.bayt.com", job_href, sep="")
  job_title <- html_text(a_elements[i], trim=TRUE)
  cat(paste(" ", i, ". ", job_title, "\n", sep="")) #display progress
```

For each of the <a> elements that were collected in the previous step, we must extract the job title and the URL for the respective job posting. The value of the "href" attribute tells us the URL of the job posting, so we use the "xml_attr" function to extract it. The "href" attribute contains only a relative path, so the Bayt.com domain is prepended to create a complete URL and stored in the "job_url" variable. The content of the <a> element contains the job title, so the "html_text" function is used to extract it, which is stored inside the "job_title" variable. The "cat" function is used to display the progress of the script as it executes by printing the title of the job that was just extracted.

Lines 46-53:

```
  #Consolidate all the information about this job into a single dataframe
  job_info <- data.frame("Sector"=sectors[sector],
                         "Title"=job_title,
                         "URL"=job_url)

  #Write the current job info to the frame where all data is stored
  job_data <- rbind(job_data, job_info)
}
```

At this point in the script, all of the data for the current job posting will have been retrieved. These lines from the script are used to simply write that data to a single-row data frame, and then append that row to the "job_data" frame, where the aggregation of all the data from all the job postings will be stored. The closing brace "}" is used to end the loop that is being used to iterate through each job.

Lines 55-59:

```
#Get the URL for the next page
next_html <- html_node(page_html, "link[rel=\"next\"]")

if(length(next_html)==0)
  break
```

Recall from the "Website Analysis" section that there is a <link> element that contains the URL for the page of jobs. It is distinguished by having a "rel" attribute with the value "next", so the "html_node" function is used to extract that element from the HTML, which is stored in the "next_html" variable. If the element is not found, then the variable will have a length of zero, in which case we will break out of the loop. This will end the execution of the loop that is iterating through each page within the sector, so we will then move on to the next sector.

Lines 61-62:

```
next_xml <- xmlParse(next_html, asText=TRUE)
page_url <- xmlAttrs(xmlRoot(next_xml))["href"]
```

These lines of code will only be reached if we did not break out of the loop during the execution of the previous lines of code, which means that there are still more pages of jobs to process. So, we use the "xmlParse" function to read the HTML as XML, and use the "xmlAttrs" function to extract the value of the "href" attribute, which will contain the URL of the next page of job listings. We update the "page_url" variable with that new URL, and we are then ready for the next iteration of the loop, so that we can process the next page of job listings.

5.3 Data Organization

In the "Data Organization" phase of the web scraping process, the data retrieved during the previous phase is organized into a useful format, such as a spreadsheet. In this case, our data is conveniently collected into a data frame (called "job_data" in the script) that has the exact same structure as a columnar table. That data is written to a comma-separated CSV file in the very last line of the script, which is shown below:

```
#Finally, write the collected data to an output file
write.table(job_data, file="output_data.csv", sep=",", col.names=TRUE, row.names=FALSE)
```

The CSV file can be opened in a spreadsheet program such as Excel. In this example, no further manipulation of the data is necessary. Our data is clean and tidy. However, in general, if additional manipulation of the data is necessary, it would be performed at this stage. Conditional formatting could be added to the spreadsheet to highlight minimum or maximum values; pivot tables could be added to see different views of the data; and so on.

6. Implications for Researchers and Practitioners

We believe that web scraping using R offers a number of advantages to researchers and practitioners. First, when dealing with Big Data, even simple manipulations with quantitative data or text can be quite tedious and prone to errors if done manually. The R environment can be used to automate a number of simple and also complex data collection and transformation processes and techniques based on complex data transformation heuristics (Krotov & Tennyson, 2018). Second, using R for web scraping and subsequent analysis ensures reproducibility of research (Peng, 2011) – something that is central to the scientific method. Any manual manipulation of data may involve subjective choices and interpretations in relation to what data is retrieved and how it is formatted, pre-processed, and saved. Oftentimes, even simple research involving basic data analysis cannot be replicated by other researchers due to the errors made at various stages of data gathering and analysis (e.g. see The Economist, 2016). With R, all aspects of data retrieval and manipulation can be unambiguously described and then reproduced by other researchers by running the script used for data collection. Finally, once web data is retrieved using R, it can be subjected to virtually all known forms of analysis implemented via thousands user-generated R packages available from CRAN. The data can also be used for creating various data products (e.g. via the Shiny tool available with RStudio) – something that can be of relevance for industry researchers.

7. Conclusion

The World Wide Web is a vast repository of data. Many research questions can be addressed by retrieving and analyzing web data. Unfortunately, web data is often unstructured or semi-structured, based on somewhat loose standards, and generated or updated in real time. Retrieving such data for further analysis requires a programmatic approach. Developing web scraping scripts that automate data collection from the web, regardless of which programming language is used for that, requires a good understanding of web architecture and some of the key web technologies, such as HTML, CSS, and XML. As demonstrated in this tutorial, the R environment is an effective platform for creating and modifying automated tools for retrieving a wide variety of data from the web. We believe that the approach to web scraping outlined in this tutorial is general enough to serve as a good starting point for any academic or industry-based research project involving web data. Also, given the brief yet detailed coverage of all fundamental technologies underpinning web scraping in R, this tutorial can also be used by instructors in various Information Systems and Analytics courses to introduce students to web scraping in R.

8. References

Basoglu, K. A., & White, Jr., C. E. (2015). Inline XBRL versus XBRL for SEC reporting. *Journal of Emerging Technologies in Accounting*, 12(1), 189-199.

Cisco Systems. (2017). Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. Retrieved from: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

Comprehensive R Archive Network (CRAN). (2020). Retrieved from: https://cran.r-project.org/

Dynamic Web Solutions. (2017). A Conceptual Explanation of the World Wide Web. Retrieved from: http://www.dynamicwebs.com.au/tutorials/explain-web.htm

Geva, T., Oestreicher-Singer, G., Efron, N., & Shimshoni, Y. (2017). Using Forum and Search Data for Sales Prediction of High-Involvement Products. *MIS Quarterly,*41(3), 65-82.

Goes, P. (2014). Editor's comments: Big data and IS research, *MIS Quarterly,* 38(3), 3-8.

Gunarathne, P., Rui, H., & Seidmann, A. (2018). When social media delivers customer service: Differential customer treatment in the airline industry. *MIS Quarterly*, 42(2), 489-520.

Krotov, V., and Tennyson, M. F. (2018). Scraping financial data from the web using R language. *Journal of Emerging Technologies in Accounting*, 15(1), 169-181.

Krotov, V. and Silva, L. (2018). Legality and ethics of web scraping. The Twenty Fourth Americas Conference on Information Systems.

Krotov, V. , Johnson, L. and Silva, L. (2020). Legality and ethics of web scraping. *Communications of the AIS*, 47(1), 22.

Lander, J. P. (2014). R for Everyone: Advanced Analytics and Graphics. Boston, MA: Addison-Wesley.

Peng, R. D. (2011). Reproducible research in computational science. *Science*, 334(6060), 1226-1227.

The Economist (2016). "Excel errors and science papers". Retrieved from https://www.economist.com/graphic-detail/2016/09/07/excel-errors-and-science-papers

Triche, J., & Walden, E. (2018). The Use of Impression Management Strategies to Manage Stock Market Reactions to IT Failures. *Journal of the Association for Information Systems*, 19(4), 333-357.

Vaast, E., Safadi, H., Lapointe, L., & Negoita, B. (2017). Social Media Affordance for Connective Action: An Examination of the Microblogging Use During the Gulf of Mexico Oil Spill. *MIS Quarterly*, 41(4), 1179-1206.

Wickham, H. (2014a). Advanced R. Boca Raton, FL: CRC Press.

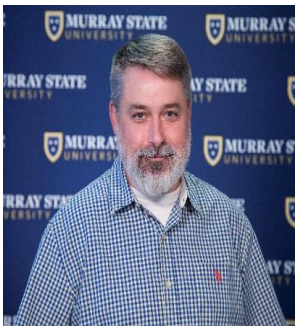
Wickham, H. (2014b). Tidy data. *Journal of Statistical Software*, 59(10), 1-23.

Wickham, H. (2016). Package 'rvest'. Retrieved from: https://cran.r-project.org/web/packages/rvest/rvest.pdf

Wickham, H., Hester, J. and Ooms, J. (2018). Package 'xml2'. Retrieved from https://cran.r-project.org/web/packages/xml2/xml2.pdf

Author Biographies

Dr. Vlad Krotov is an Associate Professor of Management Information Systems at the Department of Computer Science and Information Systems, Arthur J. Bauernfeind College of Business, Murray State University. Dr. Vlad Krotov received his PhD in Management Information Systems from the Department of Decision and Information Sciences, University of Houston (USA). His teaching, research and consulting work is devoted to helping managers and organizations to use Information and Communication Technologies for analyzing organizational data in a way that enhances organizational performance. His quantitative and qualitative research has appeared in a number of academic and practitioner-oriented journals and conferences, such as: CIO Magazine, Journal of Theoretical and Applied E-Commerce, Communications of the Association of Information Systems, Business Horizons, Blackwell Encyclopedia of Management, America's Conference on Information Systems (AMCIS), Hawaii International Conference on System Sciences (HICSS), International Conference on Mobile Business (ICMB). His research was recognized by the 2016 research was recognized by the 2016 Outstanding Researcher award and 2017 Emerging Scholar award at Murray State University.



Dr. Matthew Tennyson is an Associate Professor of Computer Science at the Department of Computer Science and Information Systems, Arthur J. Bauernfeind College of Business, Murray State University. Matthew earned his B.S. in Computer Engineering from Rose-Hulman in 1999. After graduating, he worked at Caterpillar, developing embedded systems for various types of earth-moving machinery. In 2004 he earned his M.S. in Computer Science from Bradley University. A few years later, he started pursuing a Ph.D. at Nova Southeastern University, graduating in 2013. Matthew's teaching and research interests include software engineering, programming practice and theory, and computer science education.

Appendix A: Additional Resources

Web Architecture

The following online tutorials are recommended for those who want to learn more about web architecture:

- Dynamic Web Solutions. 2017. A Conceptual Explanation of the World Wide Web. Available at: http://www.dynamicwebs.com.au/tutorials/explain-web.htm
- Tutorials Point. 2017. HTTP Tutorial. Available at: http://www.tutorialspoint.com/http/

The World Wide Web Consortium (W3C) is the ultimate source on the technologies and specifications related to the web:

- World Wide Web Consortium (W3C). 2017. Available at: https://www.w3.org/

HTML

A free online tutorial from w3schools.com on HTML:

- http://www.w3schools.com/html/default.asp

CSS

A free online tutorial from w3schools.com on CSS:

- http://www.w3schools.com/css/default.asp

XML

A free online tutorial from w3schools.com on XML and a number of related technologies:

- http://www.w3schools.com/xml/xml_exam.asp

In addition to providing a rather thorough treatment of XML, the tutorial also has sections devoted to related technologies, such as XML Namespaces, XML Schema, XPath and XLink

R and RStudio

We recommend the following book to people with basic understanding of computer programming but no previous knowledge of R:

- Lander, J. P. 2014. R for Everyone: Advanced Analytics and Graphics. Boston, MA: Addison-Wesley.

The book contains an excellent introduction into various aspects of R language and contains a manual on installing and using RStudio. Much of the examples found in this note are based on this book.

For those already familiar with R, the following advanced text on R can be recommended:

- Wickham, H. 2014. Advanced R. Boca Raton, FL: CRC Press.

Alternatively, one can access various articles and tutorials on R online:

- https://www.r-bloggers.com/how-to-learn-r-2/

For those wishing to get a practical introduction to data science in R and learn various related technologies and statistical techniques, the following online specialization from John Hopkins University is available in Coursera:

- https://www.coursera.org/specializations/jhu-data-science

Appendix B: Example Web Scraping Script

```
#Required packages
require(tm)
require(rvest)
require(XML)

#Set working directory
setwd("C:/Users/mtennyson/Documents/Research/WebScraping")

#Identify job sectors from which to scrape data
sectors <- c("banking-finance",
             "technology-telecom",
             "media-creative")

#Create frame in which data results will be stored
job_data <- data.frame(Sector=character(0),
                       Title=character(0),
                       URL=character(0))

colnames(job_data) <- c("Sector", "Title", "URL")

#For each job sector (each job sector will be processed one at a time)
for (sector in 1:length(sectors))
{
  base_url <- "https://www.bayt.com/en/uae/jobs/sectors/"
  page_url <- paste(base_url, sectors[sector], sep="")

  #For each page of jobs within the sector
  repeat
  {
    #Read website for the current sector page
    page_html <- read_html(html_session(page_url))
    cat(paste(page_url, "\n")) #display script progress

    #Retreive the list of job links
    a_elements <- html_nodes(page_html, "a[data-js-aid=\"jobID\"]")
    cat(paste(length(a_elements), " jobs found\n", sep="")) #display progress

    #For each job within the current page (process one job at a time)
    for (i in 1:length(a_elements))
    {
      job_href <- xml_attr(a_elements[i], "href")
      job_url <- paste("https://www.bayt.com", job_href, sep="")
      job_title <- html_text(a_elements[i], trim=TRUE)
      cat(paste(" ", i, ". ", job_title, "\n", sep="")) #display progress

      #Consolidate all the information about this job into single frame
      job_info <- data.frame("Sector"=sectors[sector],
                             "Title"=job_title,
                             "URL"=job_url)

      #Write the current job info to the frame where all data is stored
      job_data <- rbind(job_data, job_info)
    }
```

```
    #Get the URL for the next page
    next_html <- html_node(page_html, "link[rel=\"next\"]")

    if(length(next_html)==0)
      break

    next_xml <- xmlParse(next_html, asText=TRUE)
    page_url <- xmlAttrs(xmlRoot(next_xml))["href"]
  }
}

#Finally, write the collected data to an output file
write.table(job_data, file="output_data.csv", sep=",", col.names=TRUE, row.names=FALSE)
```