Use R!

Deborah Nolan
Duncan Temple Lang

XML and Web Technologies for Data Sciences with R

Springer

Use R!

Series Editors:
Robert Gentleman   Kurt Hornik   Giovanni Parmigiani

For further volumes:
http://www.springer.com/series/6991

Deborah Nolan • Duncan Temple Lang

XML and Web Technologies for Data Sciences with R

Springer

Deborah Nolan
Department of Statistics
University of California
Berkeley, CA, USA

Duncan Temple Lang
Department of Statistics
University of California
Davis, CA, USA

ISSN 2197-5736 ISSN 2197-5744 (electronic)
ISBN 978-1-4614-7899-7 ISBN 978-1-4614-7900-0 (eBook)
DOI 10.1007/978-1-4614-7900-0
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2013954669

© Springer Science+Business Media New York 2014
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Doris and Harriet,
and my teacher Winifred Asprey.

— Deborah

To Zoë and Suzana,
and my family farther away.

— Duncan

Preface

There has been a major change over the last decade in many aspects related to working with data and doing scientific work. As the bloggers at simplystatistics.org put it, this is a "new era where data are abundant and statisticians are scientists." The growth of the Web and the numerous data technologies that it has fostered have changed what we can do, and how we do it. It has helped to broaden the focus of statistics from mostly the modeling stage to all stages of data science: finding relevant data, accessing data, reading and transforming data, visualizing the data in rich ways, modeling, and presenting the results and conclusions with compelling, interactive displays. In this book, we describe many technologies and approaches related to all of these additional stages in the data scientist's work flow. We focus on important and fundamental technologies that are likely to stand the test of time, explain their roles, and compare them with other technologies and approaches. Importantly, we also illustrate how to work with them within the *R* programming environment through numerous comprehensive examples and case studies.

Our focus is on technologies related to the different stages of the data analysis work flow. We can now access so much data from many different sources, in different formats and via many different techniques. We can use formal Web services and application programming interfaces (APIs) or simply scrape data from human-readable Web pages. The data may come in some dialect of *XML*, *HTML* or as a *JSON* document or some other self-describing format. We will explore how we both make Web requests—both simple and sophisticated—and transform the content into data in *R*.

While we can use *R*'s rich graphical functionality, we can also visualize data in new ways with a collection of interactive plots and text in a Web browser, or use applications such as Google Earth to display spatio-temporal data and models. For these, we can use *R* to create these "external" displays as *JavaScript*, *SVG*, or *KML* documents. We can export data from *R* as *JSON* for use in *JavaScript* code to connect the data analysis with the visualization of the results.

Technologies such as Web services and requests, *XML*, and *JSON* are widely used in contexts other than the Web. All of the desktop office suites that provide spreadsheets, word processors, and presentation applications use *XML* to represent the contents of those documents. We also interact with online office suites such as GoogleDocs via authenticated Web requests using *OAuth* and then digest the *XML* content, and we also communicate with newer *NoSQL* databases and other applications using Web service technologies locally. Therefore, the technologies we discuss in this book are, in many ways, fundamental infrastructure for modern computing with data.

One of the important concepts motivating this book is that data analysts and modern statisticians increasingly are working on multi-disciplinary projects or posing their own questions. They need to access data and find auxiliary data, and wrangle them into a usable form. They expect to create rich and informative graphical displays in Web browsers, dashboards, or dynamic reports, and should be familiar with how to do this. They should also be able to understand the core technologies and know when and how to leverage them. We like to think that we foresaw this evolution when we first started developing the *R* interfaces for these technologies, back in December of 1999—a different century! We are very glad to see the rise of data science as the broadening of statistics and also see the technologies we cover in this book growing in importance in the practice of data analysis. This book is aimed at this new breed of scientist who thinks of statistics as spanning all the stages of the data work flow, and who wants to be involved in all of them.

The book describes a mixture of general infrastructure tools for *R* programmers, e.g., parse an *XML* document or make a Web request, and end-user tools *R* users can employ directly to perform a high-level task, e.g., read an *HTML* table as an *R* data frame or access the contents of a document from Google Docs. The aim in each chapter is to introduce to the reader an important technology and illustrate how it is relevant for modern statisticians. We introduce each of these technologies, provide a succinct overview of the essential concepts and elements, and describe *R*-based tools for working with the particular technology, including numerous packages developed as part of the Omegahat project (www.omegahat.org). We illustrate each technology with high-level, reasonably simple examples. We also provide more in-depth examples that put the different pieces in context and compare possible approaches. Combining these technologies into a book allows us to incrementally build on the descriptions of the fundamentals.

We have organized this book into three parts. The first part introduces the reader to *XML* and *JSON*. It assumes no prior knowledge of these data formats. It includes discussions of the tools in *R* for reading *XML* and *JSON* files and extracting data from them. It describes various different approaches and computational models for reading *XML* and explains why they are all useful in different circumstances, dealing with documents ranging from small to enormous in size. While we may spend most of our time processing content that was generated by other researchers and Web sites, etc., we often want to create our own *XML* and *JSON* documents, e.g., for displaying plots in Web browsers or Google Earth or upload to Google Docs. We also address this topic in the first part of the book.

Our focus in the second part is on how to obtain data over the Web from remote sites, services, and APIs. These methods include accessing data directly from *HTML* on static Web pages and also from dynamic *HTML* documents via *HTML* forms. We also explore using more structured access via Web services such as *SOAP* (Simple Object Access Protocol), *REST* (Representational State Transfer), and XML-RPC (*XML* Remote Procedure Call). Additionally, we may need to obtain data locally from an application running as a server, such as a *NoSQL* database. We will introduce and discuss the common technologies used for these methods. We will also explore the common approaches to authorization and authentication when accessing data from a remote site. Some of these are simple, e.g., passwords sent in a request, and others are more industrial-strength methods using technologies such as *OAuth*.

Accessing data from a remote host involves the client (*R*) communicating with that host. *R* already has several functions to do this. These are sufficient for many common tasks, but as we try to use *R* for more sophisticated network communications and interactions, we need richer tools and a richer, more flexible interface to use those tools. We will discuss the RCurl package that gives us these more general facilities. This infrastructure allows us now, and in the future, to harness new technologies built on *HTTP* and other protocols.

The final part of this book presents four in-depth examples that cover: 1) *XML* Schema, a grammar for describing the rules of an *XML* grammar; 2) *SpreadsheetML*, an Office Open *XML* vocabulary for spreadsheets and report writing; 3) Scalable Vector Graphics (*SVG*) for creating interactive graphical displays; and 4) Keyhole Markup Language (*KML*) for displaying geographic data on Google Earth and Maps. In each chapter, we describe the underlying computational model we have developed to work with these *XML* formats to, e.g., programmatically define *R* data structures, create reports, and design new graphics formats—several important aspects of data science. We hope that these examples serve as case studies for how someone might create an *R* interface to other *XML* vocabularies.

Throughout the book, we present several dozen *R* packages for data scientists to work with Web-related data technologies. Some of these packages are essentially complete and can be useful immediately to *R* users. Others are more infrastructural and/or speculative to illustrate what might be useful and how it might be done. We hope these packages will excite readers to think about how these technologies might be used to do new things and that some readers will want to extend the software to realize new possibilities. For those readers, we have provided ideas for enhancements at the end of many of the chapters and we welcome your contributions and suggestions. If you are interested in extending our work and developing new applications, we encourage you to read about writing *R* packages [145] and useful, slightly advanced aspects of the *R* language in texts such as [30, 56, 108].

We make no claim that *R* is the best tool for working with *XML*, *JSON*, and other Web technologies, or that people should use it in preference to other languages such as *Python*, *Java*, *PERL*, and *JavaScript*. However, we are suggesting that statisticians already using *R* should be able to work with these new technologies without an entire shift in their programming language and environment.

Code from the examples in this book and additional materials are available via the Web page for the book – `http://rxmlwebtech.org`.

Typographic Conventions

In this book, we discuss numerous different programming languages, and also how we use the languages in combination with one another. We illustrate code in these different languages both inline in the text and as separate code blocks. While the context should clearly indicate the language used, we use color to distinguish the languages and different fonts to distinguish the elements of a language, e.g., to differentiate a function name from a package name in *R*.

The following is a collection of example formats for these languages.

R

A block of *R* code appears as

```
doc = as(filename, "XMLInternalDocument")
xpathSApply(doc, "//section/title/", xmlValue)
```

Output from *R* commands appears as

```
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
```

And, *R* expressions appear inline as `getNodeSet(x, "//js:*", "js")`.

References to *R* function names appear in the form xmlParse(), and names of function parameters look like *filename*. Names of *R* variables appear as `variable`, names of *S4* classes in *R* appear as `XSLStyleSheet`, and *S3* class names as `XMLInternalDocument`. Named elements in an *R* list appear as `name` and slots in an *S4* object as *slot*. Options that we can query and set in *R* via the options() function appear as *warning.length*. Also, special values in *R* appear as `TRUE`, `FALSE`, `NULL`, `NA`, `NaN`, etc. Formulas are shown as `y ~ a + b | time`. Keywords in the language appear as `while`. We display regular *R* package names as `lattice`, Omegahat packages (i.e., packages distributed from `http://www.omegahat.org`) as RCurl, and **Bioconductor** packages as, e.g., graph.

File Names and Directories

We render file names as filename, file extensions as **xlsx**, and directories with a trailing / to differentiate them from file names, e.g., /home/frank/.

XML

The content for any of the various *XML* vocabularies (e.g., *XHTML*, *KML*, *SVG*, *SpreadsheetML*) is displayed as

```
<?xml version="1.0" encoding="UTF-8"?>
<snapshot>
 <header>
  <total>576803</total>
  <page>1</page>
  <date>2010-01-29T20:00:23Z</date>
  <page_size>1000</page_size>
 </header>
 ...
</snapshot>
```

Inline *XML* content is displayed as <xs:element name="ARIMA" />. Element names are shown as <*r:plot*>, while namespace prefixes are rendered as bioc. Attributes for an *XML* node are displayed as *href*. *XML* entities appear as *<* or *&*.

XPath

The code blocks for *XPath* or non-inlined expressions are displayed as

```
/Envelope/Cube/Cube
```

We see an expression inline as //a[@href] and names of *XPath* functions appear as starts-with(). An *XPath* axis is rendered as **ancestor**, and we show node test expressions as **comment**. We display the literal logical values in *XPath* as true.

JSON

Generic *JSON* content appears as

```
{
 "values": [ 1, 2, 3, 4, 5 ],
 "names": [ "ABC", "DEF", "GHI", "JKL", "MNO" ]
}
```

Inline *JSON* content is shown as arrays with *[1, 2, null, 4]*. Fields in *JSON* content appear as copyright. The *JSON* literal logical values are displayed as *true* and *false*. Similarly, the null value appears as *null*. *JSON* number and string values are shown as *3.1415* and *string*.

JavaScript

a *JavaScript* code block appears as

```
function highlightEdges(evt, row, color)
{
  var labels = edgeTable[row];
  var reset = false;

  if(typeof color == 'undefined') {
    color = "black";
```

```
    reset = true;
  }
}
```

References to *JavaScript* function names appear in the form *getElementById()*, and names of variables appear as *neighbors*. Fields of an object are shown as `myVar` while methods of an object appear as *getValue*. We see inline *JavaScript* expressions as `x = slider.getValue()`.

RCurl and `libcurl` Options
The names of curl options appear as, for example, *verbose*.

HTTP
We refer to different aspects of *HTTP* in various chapters. We represent *HTTP* operations, e.g., **GET** and **DELETE**. We also refer to elements/fields of the *HTTP* header with *ContentType*. We show some or all of the header information from a request or response with

```
GET /folder/docName HTTP/1.1
```

Shell
A block of shell (sh, bash, csh, tcsh, etc.) code appears as

```
xmllint myDoc.xml
```

Output from shell commands is displayed as

```
149 book.xml
426 springerLatex.xsl
575 total
```

When we refer to a shell command/executable, we see it as `xmllint`. Shell variables are displayed as *XML_CATALOG_FILES*. Options or flags for shell commands are rendered as *–noout*.

Acknowledgements

Many have contributed to this project by reading chapters, suggesting interesting ideas to pursue, and expressing support for our work. We thank them all, including those who participated in our NSF-sponsored Computing in Statistics Workshops. (The material in this book is based upon work supported by the National Science Foundation under Grant No. 0618865.) We especially mention: Gabe Becker, Carl Boettiger, Vince Carey, John Chambers, Robert Gentleman, Jeff Gentry, Tammy Greasby, Kurt Hornik, Ross Ihaka, Cari Kaufman, Bitao Liu, Paul Murrel, Balasubramanian Narasimhan, Karthik Ram, Steve Sein, and Phil Spector. We are also grateful to Daniel Veillard for his *XML* parser and *XSL* toolkit, `libxml2` and `liblibxslt`, and Jonathan Wallace for his *JSON* parser.

We wish to acknowledge the users of our software who have provided useful examples, asked questions that turned into examples, and submitted bug reports. Their contributions have served as inspiration for many of the examples in this book. Open source software improves because of the community of contributors. In that vein, we express our appreciation to the *R* Core for maintaining a valuable infrastructure in which we can explore these technologies.

We thank the students who have taken our computing classes over the past eight years (STAT 133 at Berkeley and STAT 141 and 242 at Davis). They have given us valuable feedback as they explored ideas and used our software when working on their projects and assignments. We also thank the computing support staff in our departments: Larry Tai at Davis; and Rick Kawin, Ryan Lovett, and the rest of the Statistical Computing Facility staff at Berkeley.

We are grateful to our original editor John Kimmel for his continual encouragement and the new folks at Springer—Jon Gurstelle, Marc Strauss and Hannah Bracken—who supported this project to completion.

Finally, we give special thanks to our families for their patience and encouragement throughout this project.

Contents

List of Examples

Part I
Data Formats: *XML* and *JSON*

Overview

The initial topic in this book is a brief introduction to both *XML* and *JSON*. We start with a practical hands-on approach by introducing some of the very high-level functions that we commonly use to read data from *HTML*, *XML*, and *JSON* documents. If you have tasks of this nature, you can hopefully read the first chapter and solve that problem. The remainder of this part of the book explains the details of both *XML* and *JSON* and how to work with these formats in *R*.

While the first chapter is a very high-level, detail-free introduction to *R* functionality, we follow it with a comprehensive introduction to *XML*. For readers who are not familiar with *XML*, this explains all of the concepts and elements of *XML*. For readers who already know the structure of *XML*, the chapter also explains some of the less common aspects such as namespaces, schema, and *DTD*s (Document Type Definitions). We also explain some of the potential and motivation for using *XML* and illustrate these with some examples of *XML* in action.

The next three chapters deal with how we extract data from *XML* documents in *R*. In Chapter 3 we start by introducing the core *R* functionality for parsing documents and working with *XML* trees and nodes. While we can process any *XML* document with these alone, the *XPath* language is a very powerful mechanism for locating particular nodes within a tree and so simplifies extracting data. We discuss *XPath* in Chapter 4. In Chapter 5, we go beyond the details of different functions and the "how-to's" of using them and discuss different strategies and approaches for extracting data from *XML* documents. In practice, we combine *XPath* and the functions for working with nodes when parsing a document. However, there are different techniques even within this hybrid approach. In this chapter, we also introduce the *SAX* approach for dealing with very large or streaming *XML* documents.

Having discussed how to read *XML* content in *R*, we turn to creating *XML* content in *R* so that we can create and use the documents in other applications such as Google Earth, Web service requests, spreadsheet and word processing software. Chapter 6 introduces the approaches and functions for creating *XML* from data in *R*.

The final chapter in this part of the book introduces the *JSON* format and the functions for both reading and writing *JSON* within *R*. *JSON* is much simpler than *XML* and we discuss the relative advantages and disadvantages of the two formats. *JSON* is not extensible in the same way that *XML* is, and it also has fewer concepts. As a result, we can cover all aspects of reading and creating *JSON* content in *R* in a single chapter, along with several examples.

Chapter 1
Getting Started with *XML* and *JSON*

Abstract The goal of this chapter is to provide a rapid introduction to a few high-level functions available to *R* users for parsing *XML* and *JSON* content. In many cases, these functions (readHTMLTable(), xmlToList(), xmlToDataFrame(), and fromJSON()) are all that you will need to read *XML*- or *JSON*- formatted data directly into an *R* *list* or `dataframe`. One of the purposes of this chapter is to introduce many of the functions you need for common applications for scraping data from Web pages, reading data from files, and working with *XML* and *JSON* data from Web services. We also want to give you a sense of the possibilities and entice you to learn more about these data formats.

1.1 Introduction

The eXtensible Markup Language (*XML*) [10] and *JavaScript* Object Notation (*JSON*) [3] are widely used on the Web to create Web pages and interactive graphics, display geographical data on, e.g., Google Earth, and transfer data between applications in an application-independent format. Being able to work with these data formats allows us to quickly and easily access and gather data from many different sources and present them in extraordinary new ways. It is exciting and relatively easy to get started gathering data in *R* [5] from Web pages and Web services and local *JSON* or *XML* files. Rather than begin by discussing details of the *XML* and *JSON* formats, we delegate this to the next chapters, and instead, jump in and learn about a few high-level functions that are available in the XML package [8] to work with *XML* content and the single function needed to import *JSON* found in the RJSONIO package [7]. These are often all we need, especially for working with *JSON* content. We hope to introduce readers to the tools needed to get them started on common tasks.

1.2 Reading Data from *HTML* Tables

We start with *HTML*, an *XML*-like vocabulary. It is quite common to find data that are available on a Web page, typically displayed in a table or a list. For example, Wikipedia [9] has a page giving the population counts for each country in the world available at `http://en.wikipedia.org/wiki/Country_population`. A screenshot of the table is shown in Figure 1.1. Tables such as this one are typically rendered to make it easy for humans to view. However, it is not necessarily easy to read such a table into a data analysis environment such as *R*. We can sometimes cut-and-paste the

D. Nolan and D. Temple Lang, *XML and Web Technologies for Data Sciences with R*, Use R!,
DOI 10.1007/978-1-4614-7900-0_1, © Springer Science+Business Media New York 2014

data into a spreadsheet and then export it from there. This approach is limited, awkward, and neither reproducible nor verifiable. Instead, we want to be able to read the data directly into the *R* environment in the same way we use the functions read.csv() and read.table(). At times we are lucky and the page is suitably formatted so we can use the readHTMLTable() function in the XML package [8] to do exactly this. The next example demonstrates this approach.

Rank	Country / Territory	Population	Date of estimate	% of World population	Source
-	World	6,965,300,000	September 30, 2011	100%	US Census Bureau's World Population Clock
1	China, People's Republic of[n2]	1,339,724,852	November 1, 2010	19.23%	2010 China Census
2	India	1,210,193,422	March 1, 2011	17.37%	Provisional 2011 Indian Census result
3	United States	312,325,000	September 30, 2011	4.48%	Official United States Population Clock
4	Indonesia	237,556,363	May 2010	3.41%	2010 Indonesian Census
5	Brazil	190,732,694	August 1, 2010	2.74%	2010 Official Brazilian Census results
6	Pakistan	177,376,000	September 30, 2011	2.55%	Official Pakistani Population clock

Figure 1.1: Wikipedia Table of Country Populations. This *HTML* table of country populations is one of five tables embedded in the Web page. With readHTMLTable(), the country populations can be extracted easily from the table of interest. This screenshot of the Wikipedia Web page `http://en.wikipedia.org/wiki/Country_population` was captured in September, 2011.

Example 1-1 Extracting Country Populations from a Wikipedia HTML Table

At its simplest, we pass readHTMLTable() either the *URL* or name of a local *HTML* file and it reads the data in the table within the document into a data frame. For the Wikipedia page, we use

```
u = "http://en.wikipedia.org/wiki/Country_population"
tbls = readHTMLTable(u)
```

This *HTML* document, like many documents on the Web, contains several tables that are used to format ads and other parts of the Web page. This can make it difficult to find the data we want. Fortunately, readHTMLTable(), by default, returns all the tables in the document so we can examine them to identify the one we want. For example, we can look at the number of rows in each table with

```
sapply(tbls, nrow)
```

```
 toc NULL NULL NULL NULL
   1  226    1    1   12
```