

UseR!

Deborah Nolan
Duncan Temple Lang

XML and Web Technologies for Data Sciences with R

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 $y \sim a + b \mid \text{time}$. 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.xml
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

Preface	vii
Part I Data Formats: XML and JSON	1
1 Getting Started with XML and JSON	5
1.1 Introduction	5
1.2 Reading Data from <i>HTML</i> Tables	5
1.3 Reading Data from <i>XML</i> -formatted Documents	8
1.3.1 Extracting Data from <i>XML</i> Attributes	13
1.4 Reading Data from <i>JSON</i> -formatted Documents	14
1.5 Summary of Functions to Read <i>HTML</i> , <i>XML</i> , and <i>JSON</i> into <i>R</i> Data Frames and Lists	17
1.6 Further Reading	17
References	18
2 An Introduction to XML	19
2.1 Overview	19
2.2 Essentials of <i>XML</i>	23
2.2.1 Syntax Checkers	28
2.3 Examples of <i>XML</i> Grammars	29
2.3.1 A Discussion of <i>XML</i> Features	35
2.4 Hierarchical Structure	36
2.5 Additional <i>XML</i> Elements	39
2.6 <i>XML</i> Namespaces	42
2.7 Describing the Structure of Classes of <i>XML</i> Documents: Schema and <i>DTDs</i>	45
2.7.1 The <i>DTD</i>	45
2.7.2 Schema	46
2.8 History of <i>XML</i>	50
2.9 Further Reading	50
References	50
3 Parsing XML Content	53
3.1 Introduction to Reading <i>XML</i> in <i>R</i>	53
3.2 The Document Object Model (<i>DOM</i>)	54
3.3 Accessing Nodes in the <i>DOM</i>	56
3.4 Parsing Other <i>XML</i> Element Types	63
3.5 Parsing <i>HTML</i> Documents	66
3.6 Reading <i>XML</i> from Different Input Sources	67

3.7	Character Encoding	68
3.8	Greater Control over the Parser	69
3.9	Three Representations of the <i>DOM</i> Tree in <i>R</i>	71
3.10	Summary of Functions for Parsing and Operating on the <i>XML</i> Hierarchy	73
3.11	Further Reading	74
	References	74
4	<i>XPath</i>, <i>XPointer</i>, and <i>XInclude</i>	75
4.1	Getting Started with <i>XPath</i>	75
4.2	<i>XPath</i> and the <i>XML</i> Tree	79
4.3	<i>XPath</i> Syntax	83
4.3.1	The Axis	84
4.3.2	The Node Test	86
4.3.3	The Predicate	87
4.4	<i>XPath</i> Functions and Logical Operators	89
4.5	Multiple Predicates in a Node Test	92
4.6	Combining <i>XPath</i> Location Paths in a Single Query	94
4.6.1	Programmatically Generating <i>XPath</i> Queries in <i>R</i>	94
4.7	Examples of Accessing Data with <i>XPath</i>	97
4.8	Namespaces and <i>XPath</i> Queries	104
4.9	<i>XInclude</i> and <i>XPointer</i>	107
4.10	Summary of Functions for Applying <i>XPath</i> Expressions to <i>XML</i> Documents	111
4.11	Further Reading	112
	References	112
5	Strategies for Extracting Data from <i>HTML</i> and <i>XML</i> Content	115
5.1	Introduction	115
5.2	Using High-level Functions to Read <i>XML</i> Content	116
5.2.1	Simple <i>HTML</i> Access	116
5.2.2	Extracting Data from <i>HTML</i> Tables	119
5.2.2.1	Extracting Other Information from <i>HTML</i> Table Cells	120
5.2.3	<i>XML</i> Property List Documents	121
5.2.4	Helper Functions for Converting Nodes	124
5.3	Examples of Scraping Content from <i>HTML</i> Pages	127
5.4	Extracting Multiple Variables From <i>XML</i> Content	146
5.4.1	Extracting an Entire Observation: A Different Approach	150
5.4.2	Modifying the Tree Before Extracting Variables: A Final Approach	150
5.5	Integrating Parts of Documents with <i>XInclude</i>	151
5.6	Reading <i>XML</i> Data into <i>R</i> Using Schema	152
5.7	Element Handler Functions	152
5.8	SAX: Simple API for <i>XML</i>	158
5.9	Managing State Across Handler Functions	164
5.9.1	Using State Objects	165
5.10	Higher-level SAX: Branches	166
5.10.1	Nested Nodes and Branches	169
5.10.2	Deferred Node Creation	169
5.11	Accessing the Parser Context	171
5.12	Parsing <i>XML</i> Content from <i>R</i> Connections	172

5.13	Comparing <i>XML</i> Parsing Techniques in <i>R</i>	172
5.13.1	The Standard <i>DOM</i> Approach	174
5.13.2	The <i>DOM</i> Approach with Handler Functions	175
5.13.3	<i>SAX</i>	176
5.13.4	Timings	178
5.13.5	<i>SAX</i> Branches	179
5.14	Summary of Functions for Parsing <i>XML</i>	180
5.15	Further Reading	182
	References	182
6	Generating <i>XML</i>	183
6.1	Introduction: A Few Ideas on Building <i>XML</i> Documents	183
6.2	A Simple Top-down Approach to Generating <i>XML</i>	184
6.3	Overview of Essential Functions for Constructing and Modifying <i>XML</i>	189
6.3.1	Changing a Node	193
6.3.2	Removing Nodes and Attributes	194
6.3.3	Generating Text Nodes	195
6.3.4	Creating Other Kinds of <i>XML</i> Nodes	196
6.3.5	Copying Nodes	196
6.3.6	Creating an <i>XML</i> Document	197
6.4	Combining Nodes to Construct an <i>XML</i> Document	198
6.5	Vectorized Generation of <i>XML</i> Using Text Manipulation	206
6.6	<i>XML</i> Namespaces	210
6.6.1	Adding Namespaces to Child Nodes	215
6.6.2	Namespaces on Attributes	218
6.6.3	Using Namespace Reference Objects	219
6.7	Working with Alternative Tree Representations to Generate <i>XML</i>	220
6.7.1	Building an <i>XML</i> Tree Entirely with Regular <i>R</i> Objects	220
6.8	Summary of Functions to Create and Modify <i>XML</i>	223
6.9	Further Reading	224
	References	225
7	<i>JavaScript</i> Object Notation	227
7.1	Introduction: Sample <i>JSON</i> Data	227
7.2	The <i>JSON</i> Format	229
7.2.1	Converting from <i>JSON</i> to <i>R</i>	231
7.2.2	Creating <i>JSON</i> from <i>R</i>	236
7.3	Validating <i>JSON</i>	238
7.4	Examples	239
7.4.1	Reading <i>JSON</i> Content from Kiva Files	239
7.4.2	Putting Data into <i>JavaScript</i> Documents	241
7.4.3	Searching Text Documents with <i>ElasticSearch</i> and <i>JSON</i>	243
7.5	Comparing <i>XML</i> and <i>JSON</i>	248
7.6	Related Work	250
7.7	Possible Enhancements and Extensions	250
7.8	Summary of Functions to Read and Write <i>JSON</i> in <i>R</i>	251
7.9	Further Reading	252
	References	252

Part II Web Technologies – Getting Data from the Web	255
8 HTTP Requests	259
8.1 Introduction	259
8.2 Overview of <i>HTTP</i>	261
8.2.1 The Simple GET Method	261
8.2.1.1 Adding Fields to the <i>HTTP</i> Header	262
8.2.1.2 Understanding the Server’s Response	263
8.2.1.2.1 Processing the Body in <i>R</i>	265
8.2.1.2.2 Manipulating the Header in <i>R</i>	267
8.2.2 GET Requests with Input Parameters	267
8.2.3 POST ’ing a Form	269
8.2.3.1 Two POST Formats	271
8.2.3.2 Uploading the Contents of Files	272
8.2.4 Specifying Request Options in <code>getForm()</code> and <code>postForm()</code>	272
8.2.5 The General POST Method for Data in the <i>HTTP</i> Request Body	273
8.2.6 <i>HTTP</i> ’s PUT Method	275
8.2.7 <i>HTTP</i> ’s HEAD Method	276
8.2.8 <i>HTTP</i> ’s DELETE Method	276
8.2.9 <code>customrequest</code> and Extended Methods	277
8.3 Character Encoding	277
8.4 Using a Connection Across Requests	278
8.4.1 Setting Options in a <code>curl</code> Handle	281
8.5 Multiple Requests and Handles	283
8.5.1 The <code>Multihandle</code> Interface in <i>R</i>	284
8.6 Overview of <code>libcurl</code> Options	286
8.6.1 Asynchronous Callback Function Options	287
8.6.1.1 Customizing the <code>writelfunction</code> and <code>headerfunction</code> Options	288
8.6.1.2 The <code>readfunction</code> and <code>readdata</code> Options	291
8.6.1.3 The <code>progressfunction</code> Option	292
8.6.1.4 Using <i>C</i> Routines as Callbacks	293
8.6.2 Passwords for Web Pages	294
8.6.3 Cookies	296
8.6.4 Working with <i>SSL</i> and Certificates	299
8.6.5 Using a Proxy Server	300
8.7 Getting Information About a Request	301
8.8 Getting Information About <code>libcurl</code> and Its Capabilities	302
8.9 Other Protocols	303
8.9.1 Secure Copy (<code>scp</code>)	303
8.10 <i>HTTP</i> Errors and <i>R</i> Classes	304
8.11 Debugging Web Requests	306
8.12 <code>Curl</code> Command Line Arguments and <code>RCurl</code>	309
8.13 Summary of <code>RCurl</code> Functions	311
8.14 Further Reading	312
References	312
9 Scraping Data from <i>HTML</i> Forms	315
9.1 Introduction	315

9.1.1	GET and POST Methods of Form Submission	318
9.2	Generating Customized Functions to Handle Form Submission	321
9.2.1	Adding a Function to Convert the Result	324
9.3	Supplying the curl Handle and Modifying the Form	325
9.3.1	Saving State Across Submission of Different Forms	325
9.3.2	Changing the Form Description	330
9.4	Forms and Elements that Use <i>JavaScript</i>	333
9.5	Further Reading	338
	References	338
10	REST-based Web Services	339
10.1	Introduction	339
10.1.1	Key Concepts	340
10.1.2	A Brief Contrast of <i>REST</i> and <i>SOAP</i>	342
10.2	Simple <i>REST</i>	343
10.2.1	Accessing the <i>NoSQL</i> Database CouchDB via <i>REST</i>	349
10.3	Simple Authentication	351
10.4	Changing State with <i>REST</i>	357
10.4.1	Establishing a Connection with Google Docs from <i>R</i>	359
10.4.2	Managing Documents in Google Docs	361
10.4.3	Using an Access Token to Digitally Sign Requests	366
10.5	Web Application Description Language: <i>WADL</i>	369
10.5.1	Reflection Methods for <i>REST</i> Methods and Services	369
10.5.2	Working with <i>WADL</i> Documents	370
10.6	Possible Enhancements and Extensions	377
10.7	Summary of Functions for <i>REST</i> in <i>R</i>	377
10.8	Further Reading	378
	References	378
11	Simple Web Services and Remote Method Calls with XML-RPC	381
11.1	Using <i>XML</i> for Remote Procedure Calls: XML-RPC	381
11.2	Classes for Representing the XML-RPC Server	384
11.3	Writing <i>R</i> Functions to Use XML-RPC	385
11.3.1	Programmatically Accessing a Blog	385
11.3.2	Interactive and Dynamic Network Graphs with Ubigraph	388
11.4	Handling Errors in XML-RPC	393
11.5	Under the Hood of <code>xml.rpc()</code>	395
11.5.1	The <i>HTTP</i> Request	399
11.6	Possible Enhancements and Extensions	399
11.7	Summary of Functions to use XML-RPC from <i>R</i>	400
11.8	Further Reading	400
	References	400
12	Accessing SOAP Web Services	403
12.1	Introduction: What Is <i>SOAP</i> ?	403
12.2	The Basic Workflow: Working with <i>SOAP</i> in <i>R</i>	404
12.2.1	Accessing the KEGG Web Service	405
12.2.2	Accessing Chemical Data via the ChemSpider <i>SOAP</i> API	407

12.2.3	Other Useful Features of <code>genSOAPClientInterface()</code>	409
12.3	Understanding the Generated Wrapper Functions	411
12.4	The Basics of <i>SOAP</i>	413
12.5	The <code>.SOAP()</code> Function	416
12.5.1	The <i>server</i> Parameter	417
12.5.2	The <i>method</i> Parameter	417
12.5.3	Arguments for the <i>SOAP</i> Method: <code>...</code> and <code>.soapArgs</code> Parameters	418
12.5.4	The <i>action</i> Parameter	419
12.5.5	Passing Curl Options via the <code>.opts</code> Parameter	420
12.5.6	The <code>.convert</code> Parameter	420
12.5.7	Additional Arguments	424
12.6	Handling Errors in <i>SOAP</i> Calls	424
12.7	Using the <code><Header></code> Element in a <i>SOAP</i> Request for Authentication and Security	425
12.8	Customizing the Code Generation	428
12.8.1	Specifying the Port and Bindings	428
12.8.2	Processing Only Relevant Functions	429
12.8.3	Changing and Adding Formal Parameters	430
12.8.3.1	Changing the Default Server	430
12.8.3.2	Changing the Default Value of Service-level Parameters in All Functions	431
12.8.3.3	Adding a Parameter to a Function	432
12.8.3.4	Changing How the Functions Are Generated	434
12.9	Serializing <i>R</i> Values to <i>XML</i> for <i>SOAP</i>	435
12.10	Possible Enhancements and Extensions	437
12.11	Summary of Functions for Working with <i>SOAP</i> in <i>R</i>	437
12.12	Further Reading	438
	References	438
13	Authentication for Web Services via <i>OAuth</i>	441
13.1	Introduction: Securely Accessing Private Data with <i>OAuth</i>	441
13.1.1	The <i>OAuth</i> Model and <i>R</i>	442
13.1.2	Creating/Registering an Application with the Provider	444
13.2	The <code>ROAuth</code> Package	444
13.2.1	The Basic Workflow in <i>R</i> for <i>OAuth</i> 1.0	444
13.2.2	Using an Access Token Across <i>R</i> Sessions	449
13.2.3	Keeping the Consumer Key and Secret Private	449
13.2.4	Extending the <code>OAuthCredentials</code> Class	449
13.2.5	An Alternative Syntax for Invoking <i>OAuth</i> Requests	450
13.2.6	Low-level Details of <i>OAuth</i> 1.0: The Handshake	451
13.2.7	Low-level Details of <i>OAuth</i> 1.0: The Digital Signature	452
13.3	<i>OAuth</i> 2.0 and Google Storage	453
13.3.1	Getting the User's Permission and the Authorization Token	454
13.3.2	Exchanging the Authorization Token for an Access Token	456
13.3.3	Using the Access Token in an API Request	457
13.3.4	Refreshing an <i>OAuth2</i> Access Token	459
13.4	Summary of Functions for Using <i>OAuth</i> in <i>R</i>	460
13.5	Further Reading	460
	References	461

Part III General XML Application Areas 463

14 Meta-Programming with XML Schema 467

- 14.1 Introduction: Using Information from XML Schema 467
- 14.2 Reading XML Schema and Generating Code and Classes 471
 - 14.2.1 Writing the Generated Code to a File 473
 - 14.2.2 Customizing the Code Generation 474
- 14.3 Reading XML Schema in R 475
- 14.4 R Classes for Describing XML Schema Types 480
- 14.5 Mapping Schema Type Descriptions to R Classes and Converter Methods 484
 - 14.5.1 Mapping Simple Elements to R Types 484
 - 14.5.2 Class Inheritance in R for Schema Derived Types 487
 - 14.5.3 Collections, Lists, and Recurring Elements 491
 - 14.5.3.1 Collections of Simple Types 494
- 14.6 Working with Included and Imported Schema 496
 - 14.6.1 Processing Sub-schema 496
 - 14.6.2 Local Schema Files and XML Catalogs 496
 - 14.6.3 Computations on a Schema Hierarchy 497
- 14.7 Possible Enhancements and Extensions 498
- 14.8 Summary of Functions to Work with XML Schema 499
- 14.9 Further Reading 499
- References 499

15 Spreadsheets 501

- 15.1 Introduction: A Background in Spreadsheets 501
- 15.2 Simple Spreadsheets 503
 - 15.2.1 Extracting a Spreadsheet into a Data Frame 504
 - 15.2.2 Extracting Multiple Sheets from a Workbook 504
- 15.3 Office Open XML 508
 - 15.3.1 The *xlsx* Archive 508
 - 15.3.2 The Workbook 510
 - 15.3.3 Cells and Worksheets 511
- 15.4 Intermediate-Level Functions for Extracting Subsets of a Worksheet 512
 - 15.4.1 The Excel Archive in R 513
 - 15.4.2 The Excel Workbook in R 514
 - 15.4.3 The Excel Worksheet in R 514
- 15.5 Accessing Highly Formatted Spreadsheets 516
- 15.6 Creating and Updating Spreadsheets 520
 - 15.6.1 Cloning the Excel Document and Entering Cell Values and Formulae 521
 - 15.6.2 Working with Styles 523
 - 15.6.3 Inserting Other Content into the Archive 524
- 15.7 Using Relationship and Association Information in the Archive 525
- 15.8 Google Docs and Open Office Spreadsheets 531
- 15.9 Possible Enhancements and Extensions 532
- 15.10 Summary of Functions in RExcelXML 533
- 15.11 Further Reading 534
- References 534

16 Scalable Vector Graphics	537
16.1 Introduction: What Is SVG?	537
16.1.1 A Model for Adding Interactivity to SVG Plots	538
16.1.2 Other Approaches to Making Interactive SVG Plots in R	540
16.2 Simple Forms of Interactivity	542
16.3 The Essentials of SVG	545
16.4 General Interactivity on SVG Elements via JavaScript	548
16.4.1 Adding JavaScript Event Handlers to SVG Elements	549
16.4.2 Using JavaScript to Create Graphical Elements at Run-time	552
16.4.3 Interaction with HTML User Interface Elements	556
16.4.4 Adding Event Handlers to SVG Elements via JavaScript Code in HTML	559
16.4.5 Embedding GUI Controls Within an SVG Plot	561
16.5 Animation	562
16.5.1 Declarative Animation with SVG	563
16.5.2 Programming Animation with JavaScript	566
16.6 Understanding Low-level SVG Content	568
16.6.1 The SVG Display for an R Plot	569
16.6.2 Text in the SVG Display	571
16.6.3 Styles in SVG	572
16.6.4 SVG Animation Elements	573
16.7 Possible Enhancements and Extensions	575
16.8 Summary of Functions in SVGAnnotation	576
16.9 Further Reading	578
References	578
17 Keyhole Markup Language	581
17.1 Introduction: Google Earth as a Graphics Device	581
17.1.1 The Google Earth and Google Maps Interfaces	583
17.2 Simple Displays of Spatial Data	586
17.2.1 Adding Points to the Google Earth and Google Maps Canvas	586
17.2.2 Associating Time with Points	587
17.2.3 Using Styles to Customize Graphical Elements	589
17.2.3.1 Styles for Placemarks and Lines	590
17.2.3.2 Creating Icons in R and Using HTML in Pop-up Windows	592
17.3 Zipped KML Documents	595
17.4 A Formula Language for Making KML Plots	596
17.4.1 Including Time in the Formula for Geospatial–Temporal Plots	597
17.4.2 Grouping Placemarks into Folders on Google Earth	597
17.5 The KML Grammar	599
17.5.1 A Sample KML Document	599
17.5.2 Strategies for Working with and Debugging KML Documents	602
17.6 Working More Directly with KML to Create Custom Displays	603
17.6.1 Overlaying Images Made in R on Google Earth	603
17.6.2 KML-Formatted Plots on Google Earth	607
17.7 Embedding Google Earth in a Web Page	609
17.7.1 Using the Google Earth Plug-in	610
17.7.2 Linking the Plug-in to Other Elements in a Web Page	613
17.8 Possible Enhancements and Extensions	616

17.9 Summary of Functions in RKML 616

17.10 Further Reading 617

References 617

18 New Ways to Think about Documents 619

18.1 The Process of Authoring and Creating Documents 619

18.2 Validating a Document 620

18.3 Treating a Document as *R* Code 625

 18.3.1 Accessing Code Chunks via Variables 626

18.4 Reusing Content in Different Documents 627

18.5 Capturing the Process and Paths of the Workflow 628

18.6 Using *XSL* to Transforming *XML* Documents 629

 18.6.1 *XSL* in *R* 632

18.7 Further Reading 633

References 634

Bibliography 635

General Index 647

R Function and Parameter Index 653

R Package Index 659

R Class Index 661

Colophon 663

List of Examples

1-1	Extracting Country Populations from a Wikipedia <i>HTML</i> Table	6
1-2	Converting <i>XML</i> -formatted Kiva Data to an <i>R</i> List or Data Frame	11
1-3	Retrieving Attribute Values from <i>XML</i> -formatted Bills in the US Congress	13
1-4	Converting <i>JSON</i> -formatted Kiva Data into an <i>R</i> List	15
2-1	A <i>DocBook</i> Document	30
2-2	A Climate Science Modelling Language (CSML) Document	32
2-3	A Statistical Data and Metadata Exchange (SDMX) Exchange Rate Document	33
2-4	A <i>DTD</i> for <i>XHTML</i>	45
2-5	Examining Schema for the Predictive Model Markup Language	47
3-1	Subsetting a Kiva Document to Extract Lender's Occupation	54
3-2	Retrieving Content from Subnodes in a Kiva Document	58
3-3	Retrieving Attribute Values from a USGS Earthquake Document	60
4-1	Efficient Extractions from a Michigan Molecular Interactions (MiMI) Document	76
4-2	Simplifying <i>XPath</i> Axes to Locate SDMX Nodes	84
4-3	Using Parent and Attribute Axes to Locate Dates in an SDMX Document	85
4-4	Creating Multiple <i>XPath</i> Queries for Exchange Rates	95
4-5	Using <i>XPath</i> Functions to Retrieve Loan Information for a Large Number of Kiva Loans	95
4-6	Retrieving Attribute Values with <i>XPath</i> for Bills in the US Congress	97
4-7	Retrieving Magnitude and Time with <i>XPath</i> for Earthquakes in a USGS Catalog	97
4-8	Extracting Text Content from a Kiva Document	98
4-9	Locating Content in Metadata Object Description Schema (MODS) Entries	98
4-10	Building a Data Set from Fragments with <i>XInclude</i>	108
4-11	Using <i>XInclude</i> to Create <i>R</i> Data Structures with Shared Sub-components	110
5-1	Extracting and Formatting Information from a Wikipedia Table on US Public Debt	119
5-2	Extracting Hyperlinks to KMZ Files from Attributes in <i>HTML</i> Table Cells	120
5-3	Reading Baseball Player Information from Attributes into an <i>R</i> Data Frame	124
5-4	Converting Player Information into <i>S4</i> Objects	126
5-5	Extracting Headlines from The <i>New York Times</i> ' Web Pages	127
5-6	Scraping Job Postings from Kaggle Web Pages	133
5-7	Getting the Content of Kaggle Job Posts Across Web Pages	136
5-8	Extracting Data About Pitches in a Baseball Game	141
5-9	Extracting Loan Counts from Kiva Using Unique Identifiers on Nodes	148
5-10	Reading Earthquake Data with Handler Functions	154
5-11	Extracting Earthquake Information Using Handler Functions with Closures	157
5-12	Extracting Exchange Rates via <i>SAX</i> Parsing	159
5-13	Creating a Table of Counts of Nodes with a <i>SAX</i> Parser Using Reference Classes	164
5-14	Creating a Table of Counts of Nodes with a <i>SAX</i> Parser Using a State Object	165
5-15	Extracting Revision History from Wikipedia Pages Using <i>SAX</i> Branches	166

5-16	Extracting a Random Sample of Revisions to Wikipedia Pages	170
6-1	Generating an <i>HTML</i> Table from a Data Frame	185
6-2	Creating a Great Circle in the Keyhole Markup Language (<i>KML</i>)	198
6-3	Modifying an Existing <i>HTML</i> Table	202
6-4	Creating <i>KML</i> Using Text Manipulation	207
6-5	Generating <i>SDMX</i> Exchange Rates with Namespaces	215
6-6	Generating an <i>XHTML</i> Table from an <i>R</i> Structure	221
7-1	Creating <i>HTML</i> Tables Using Election Results Stored in <i>JSON</i>	242
7-2	Using <i>ElasticSearch</i> to Search Google News	244
7-3	Inserting Email Messages into <i>ElasticSearch</i>	246
8-1	Retrieving a Gzipped Mail Archive from a Secure Site	265
8-2	Retrieving a CSV File from National Stock Exchange (NSE) India	265
8-3	Requesting Stock Prices via a Form on Yahoo	268
8-4	Posting a Form to Obtain Historical Consumer Price Index (CPI) Data	270
8-5	Using PUT to Rename a Google Docs Document	275
8-6	Specifying the Character Encoding for a US Census Bureau CSV File	278
8-7	Using the Same Connection to Retrieve Multiple Files in a Mail Archive	279
8-8	Making Multiple Web Requests to NSE India	283
8-9	Using Cookies to Access the Caltrans Performance Measurement System (PeMS) Site	296
8-10	Making a Web Request Through a Proxy Server	300
8-11	Catching Errors in a Request to Open Street Map	305
8-12	Comparing Command Line and RCurl Requests for GlobalGiving	309
8-13	Posting a Tweet with a <code>curl</code> Command versus <code>httpPOST()</code>	310
9-1	The Google Search Form	316
9-2	Accessing Historical Consumer Price Index Data with a Form	321
10-1	Accessing the European Bioinformatics Institute Protein Databases via <i>REST</i>	343
10-2	Parameterizing <i>REST</i> Requests for Climate Data from the World Bank	346
10-3	Authenticating a Request to Zillow for Housing Prices	351
10-4	Using an Access Token to Obtain Historical Weather Data from NOAA	354
10-5	The Google Docs API	358
10-6	Digitally Signing <i>REST</i> Requests to Amazon S3	366
10-7	The EuPathDB Gene Search Web Service	372
10-8	A <i>WADL</i> Interface to the NOAA Web Service	374
11-1	Creating an Ubigraph Network Display	392
11-2	Understanding an Input Type Error in XML-RPC	393
11-3	An <i>HTTP</i> Error in an XML-RPC Request	394
11-4	Serializing the <code>timeDate</code> Class to XML-RPC	398
11-5	Serializing an <i>S3</i> <code>lm</code> Object to XML-RPC	398
11-6	Serializing Specific Classes in XML-RPC	398
14-1	Generating <i>S4</i> Classes Programmatically for <i>PMML</i>	470
14-2	Reading <i>KML</i> Schema	472
14-3	Exploring the <i>KML</i> Schema via the <code>SchemaCollection</code> Class	477
14-4	A Description of <i>PMML</i> Schema Elements in <i>R</i>	480
15-1	Extracting Data from a World Bank Spreadsheet	504
15-2	Extracting Federal Exchange Commission (FEC) Data from Multiple Worksheets	505
15-3	Extracting a Rectangular Region from an FEC Worksheet	515
15-4	Working with Detailed Titles and Footnotes in a US Census Spreadsheet	516
15-5	Generating an Excel Report from a Template	522

15-6	Adding Styles to Cells for an Excel Report	524
15-7	Adding an rda File to an Excel Archive	524
15-8	Adding a Worksheet to a Workbook	526
15-9	Adding an Image to a Worksheet	530
15-10	Inserting a Worksheet into a Google Docs Spreadsheet	531
16-1	Adding Tool Tips and Hyperlinks to an <i>SVG</i> Plot of Earthquakes	542
16-2	Pointwise Linking Across <i>SVG</i> Scatter Plots	551
16-3	Drawing Nearest Neighbors on an <i>SVG</i> Scatter Plot with <i>JavaScript</i>	552
16-4	Using Forms to Highlight Points in a Lattice Plot	556
16-5	Adding Event Handlers to <i>SVG</i> Maps When the Map Is Loaded into an <i>HTML</i> Page . .	559
16-6	Animating Scatter Plots Through Snapshots in Time	564
16-7	Animating a Map with <i>JavaScript</i> and <i>SVG</i>	566
17-1	Plotting Earthquake Locations as Paddles on Google Maps	586
17-2	Plotting Elephant Seal Locations on Google Earth	587
17-3	Customizing a <i>KML</i> Display of an Elephant Seal's Movements	590
17-4	Annotating Earthquake Locations in <i>KML</i> with Depth and Magnitude	592
17-5	Plotting Earthquake Locations in <i>KML</i> via the Formula Language	597
17-6	<i>R</i> Plots of Average Daily Temperature as Ground Overlays on Google Earth	605
17-7	Creating Temperature Boxplots with a <i>KML</i> Graphics Device	608
17-8	Displaying San Francisco Housing Market Data in a Google Earth Plug-in	610
17-9	Linking <i>SVG</i> Scatter Plots and a Google Earth Plug-in	614

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 *DTDs* (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

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
```