Web Technologies
Accessing Data

Topics
• HTML pages
• XPath
• HTML forms
• REST
• SOAP
• XML-RPC
• (You don’t have to teach them all, but there are interesting aspects to all.)

Consumer Price Index
• Suppose we have a financial time series and need to adjust for inflation. We need the CPI values for the relevant period.
• We can look this up on the Web, e.g. http://www.rateinflation.com/consumer-price-index/usa-historical-cpi.php
The data for the most recent 5 years is in the main table.
There is also an HTML form that allows the reader to specify the interval of interest. We'll return to this.

How to read the data for the 5 years for each month?
Simple answer: readHTMLTable() in the XML package.

```r
tbls = readHTMLTable("http://www.rateinflation.com/consumer-price-index/usa-historical-cpi.php")
length(tbls)
sapply(tbls, nrow)
```
We want the last one – 6 rows, including the header.

```r
cpi = readHTMLTable("http://www.rateinflation.com/consumer-price-index/usa-historical-cpi.php", which = 11, header = TRUE)
```
Fix up the types of each column, converting from a factor to a number.

```r
cpi = as.data.frame(lapply(cpi, function(x) as.numeric(as.character(x)))))
```

Details

Interesting answer is how that function is implemented
Examine the HTML
- find all <table> elements
- process each of these to convert to a data frame
  - find <tr> elements for each row
  - recognize <th> elements or <thead> for header
  - <td> for data value
  - Unravel into data frame
Details in the XML package and readHTMLTable()
But general concepts in Xpath and finding <table> nodes.
XPath

- XPath is yet another DSL – domain specific language
- XML documents are trees and XPath is a mechanism for finding nodes anywhere within the tree based on a “pattern”
- Pattern is a path that identifies sequence of nodes by
  - direction or “axis” (parent, child, ancestor, descendant, sideways (&lt;‐&gt;))
  - node test – i.e. the name (e.g. table, head, tr, td)
  - predicate test (has an attribute href, has an attribute href = "foo")

Parse the XML/HTML document

Find the <table> elements
- tbs = getNodeSet(doc, "//table")

getNodeSet() takes a document or a node and searches through the sub-tree using a language for describing how to find the nodes of interest.

- // is srt-hand for "/descendant::table",
  / is the top-level/root node
descendant is an “axis”
table is the node-test
- If the <table> of interest had an id attribute, we could add a predicate, e.g.
  - getNodeSet(doc, "//table[@id='cpi']")

getNodeSet() returns a list of matching nodes.
- We can then recursively extract the nodes of interest, e.g. the <tr> and the <td> elements
  - can walk the tree ourselves if shallow
  - or use getNodeSet() to query the subtree easily
- Convert the values in these sub-nodes to R values and combine into data structure.
Walking the tree

• A node has a name
  – xmlName(node)
• Attributes
  – xmlAttrs(node),
  xmlGetAttr(node, “attrName”)
• Children
  – xmlChildren(node) – list of child nodes
• Parent node
  – xmlParent(node)

• rows = getNodeSet(tbl, “//tr”)
  do.call(“rbind”, lapply(rows, getRowValues))
• getRowValues gets all the <td> within a <tr>
  xpathSApply(row, “//td”, xmlValue)

Back to the HTML form

• Xpath is similar to regular expressions
  – It is a way of expressing complex patterns very tersely and having the Xpath engine implement the search.
• Works for any XML document, so very general.
• Can build up very precise or general queries
  – contextual knowledge important to catch all the nodes we want, but no more.
• We use Xpath for processing XML from many different sources.

• What if we want more or different years?
  – Use the HTML form?
• But how can we mimic selecting the Start and End years from within R, i.e. programmatically?
• An HTML form is like an R function
  – takes inputs, returns an result – an HTML document
• Need to mimic a Web browser to pass arguments to Web server.
RCurl

• The RCurl package provides an R interface to a very general and powerful library that can perform Web queries programmatically and that are very customizable.
• 3 main functions:
  – getURLContent()
  – getForm()
  – postForm()

• Similar functionality to download.url(), but much more customizable and general
• Can handle
  – Secure HTTP – https
  – cookies, passwords
  – many additional important options
  – maintain state across requests
  – multiple concurrent requests

• Examine HTML document and look for the <form>.
  Find the parameter names and use these as named parameters in getForm()
• `x = postForm("http://www.example.com/formname/form.html",` form = "usacpi",
  fromYear = "1945",
  toYear = "1965",
  `_submit_check` = "1"
)
• Then pass this to readHTMLTable(), which =

REST

• Representational State Transfer
• URL represents a state which can be queried or even updated via remote calls/queries.
• Send parameterized Web query via getForm()
  – specify URL
  – name value pairs for parameters
• Get back a "document"
  – may be
    • raw text
    • XML
    • JSON
    • binary data
Process result

- Raw text – use text manipulation, regular expressions, connections to read into R object
- JSON – JavaScript Object Notation
  – use RJSONIO or rjson
- XML – parseXML() and Xpath (getNodeSet())
- Binary data – treat as is, or if compressed, uncompress in-memory via Rcompression

Zillow

- Zillow provides information and price estimates of homes
- REST API info at http://www.zillow.com/howto/api/APIOverview.htm
- Register to get a Zillow Web Service ID (ZWSID) that you pass in each call to a Zillow API method

Getting the Result Info

- Call GetZEstimate for a property giving street address
  - getForm("http://www.zillow.com/webservice/
    GetSearchResults.htm",
    'zws-id' = ZWSID,
    address = "1292 Monterey Ave",
    citystatezip = "Berkeley, CA")
  Result is a text string which contains an XML document

- XML contains <request>, <message>, <response>
- Extract property id, price estimate, lat./long., comparables link, etc.
- Use Xpath and xmlValue().
- doc = xmlParse(txt, asText = TRUE)
- est = doc["//result/zestimate"]
- as.numeric(xmlValue(est["amount"]))
R package Zillow provides functions for several of the API methods and hides all the details.

Yahoo Search

- Yahoo Web Search Service
- out = getForm("http://search.yahooapis.com/WebSearchService/V1/webSearch",
  appid = yahooAppIdString,
  query = "REST XML Yahoo",
  results = 100,
  output = "json")

```r
library(RJSONIO)
ans = fromJSON(out)
```

ans is a list with 1 element named ResultSet
- length(ans$ResultSet) # 6
- names(ans$ResultSet)
  - [1] "type" "totalResultsAvailable"
  - [3] "totalResultsReturned" "firstResultPosition"
  - [5] "moreSearch" "Result"

Individual Search Result Item
- names(ans$ResultSet$Result[[1]])
  - [1] "Title" "Summary" "Url"
  - [4] "ClickUrl" "DisplayUrl" "ModificationDate"
  - [7] "MimeType" "Cache"
REST

- **Pros:**
  - simple and easy to get started
  - natural exploitation of URLs as resources
- **Cons:**
  - cannot send or retrieved complex/hierarchical data structures
  - have to process result manually
  - have to find methods and inputs manually by reading documentation.
    - Do this once and build R functions to hide the details.

SOAP

- **Simple Object Access Protocol**
- **Richer and more complex than REST**
  - can send highly structured data via XML
  - Send request in an Envelope containing a request to invoke a method in the server’s object
    - Send arguments as self-describing objects
- **SOAP allows us to define new data types and structures**
  - application specific data types

SOAP

- **Would have to construct the SOAP request**
  - the envelop and the message
  - Too many details to do manually.
- **Instead, SOAP service publishes a description of its methods and data types**
  - WSDL document – Web Service Description Language
- **Code reads this and generates R functions to invoke each of the methods, coercing the R arguments to their XML representation and converting the XML result to an R object.**
- **Transparent to user**
KEGG (Kyoto Encyclopedia of Genes and Genomes) provides a SOAP Web Service (among other services) to access its system functionality (API).

From R

- library(SSOAP)
- u = "http://soap.genome.jp/KEGG.wsdl"
- kegg.wsdl = processWSDL(u)
- kegg.iface = genSOAPClientInterface(kegg.wsdl)
- Now we have an S4 object containing class definitions and a list of functions
- names(kegg.iface@functions)

• Invoke the list_databases method
  - kegg.iface@functions$list_databases()
  - returns a list of S4 Definition objects
  - e.g. An object of class "Definition"
    - Slot "entry_id":
      - [1] "nt"
    - Slot "definition":
      - [1] "Non-redundant nucleic acid sequence database"

• Get enzymes for a specific gene id
  - iface@functions$get_enzymes_by_gene('eco:b0002')
    - [1] "ec:1.1.1.3" "ec:2.7.2.4"