

## Reading and Writing Data Files

## Unstructured vs Structured Plain-text Data

### State of the Union Speeches

\*\*\*  
State of the Union Address  
George Washington  
December 8, 1790

Fellow-Citizens of the Senate and House of Representatives:  
In meeting you again I feel much satisfaction in being able to repeat my congratulations on the favorable prospects which continue to distinguish our public affairs. The abundant fruits of another year have blessed our country with plenty and with the means of a flourishing commerce.

### Web Log Entries

169.237.46.168 -- [26/Jan/2004:10:47:58-0800]  
"GET /stat141/Winter04 HTTP/1.1" 301 328  
"http://anson.ucdavis.edu/courses/"  
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"

169.237.46.168 -- [26/Jan/2004:10:47:58-0800]  
"GET /stat141/Winter04/ HTTP/1.1" 200 2585  
"http://anson.ucdavis.edu/courses/"  
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR 1.1.4322)"

## Web Log Entries – **extract** & **omit**

```
169.237.46.168 -- [26/Jan/2004:10:47:58-0800]
"GET /stat141/Winter04 HTTP/1.1" 301 328
"http://anson.ucdavis.edu/courses/"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
1.1.4322) "
```

```
169.237.46.168 -- [26/Jan/2004:10:47:58-0800]
"GET /stat141/Winter04/ HTTP/1.1" 200 2585
"http://anson.ucdavis.edu/courses/"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0; .NET CLR
1.1.4322)"
```

```
wlist = strsplit(wl, " \" | -- \\[|\\\" ")  SPLIT on either
                                           blank" or
wlist[[1]]                                blank--blank[ or
[1] "169.237.46.168"                      "blank"
[2] "26/Jan/2004:10:47:58 -0800]"
[3] "GET /stat141/Winter04 HTTP/1.1"
[4] "301 328"
[5] "http://anson.ucdavis.edu/courses/"
[6] "\"Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.0; .NET CLR 1.1.4322)\""
```

## **readLines()**

- The **readLines()** function reads each line of text in a file and creates a character vector with one element per line
- We can then use regular expressions to extract the data we want.

```
> wlist[[1]][3]
[1] "GET /stat141/Winter04 HTTP/1.1"
```

Eliminate the unwanted characters in the third element

```
> sapply(wlist,
         function(x) gsub(" .*$", "", x[3]))
[1] "GET" "GET"
```

## read.fwf()

### Fixed-width formats

- The `read.fwf()` function is handy if the pieces of information are always the same width
- “fwf” stands for fixed-width-format
- The web log data is close to a fwf

```
123456789012345678901234567890123456789...
```

```
ip is 1-14 | skip| |
```

```
169.237.46.168 -- [26/Jan/2004:10:47:58...
```

```
read.fwf(fileLoc,  
         widths = c(14,5,2,1,3,1,4,18,3))
```

	V1	V2	V3	V4	V5	V6	V7	...
1	169.237.46.168	--	[	26	/	Jan	/	2004 ...
2	169.237.46.168	--	[	26	/	Jan	/	2004 ...

### Delimited data

# Reading data into R

- Many data sets are stored in text files.
- The easiest way to read these into R is using either the `read.table` or `read.csv` function, both of which return a data frame.
- Consider the data at the following site

```
fileLoc = "http://www-958.ibm.com/software/data/cognos/manyeyes/datasets/olympic2012withgdp/versions/1.txt"
```

## `read.table()` or `read.csv()`

- These functions are useful for reading delimited plain text files
- They have quite a few options. Some of the important ones are:
  - file - name or URL
  - header - are column names at the top of the file?
  - sep - what divides elements of the table
  - na.strings - symbol for missing values, like 9999
  - skip - number of lines at the top of the file to ignore

# The data

ISO	Gold/medals					Silver/medals					Bronze/medals					...
ABW	0	0	0	0	0	0	2,456,000,000.00	108,000...								
AFG	0	0	1	1	1	1	20,343,461,030.00	34,385,000								...
AGO	0	0	0	0	0	0	100,990,000,000.00									...
ALB	0	0	0	0	0	0	12,959,563,902.00	3,205,000								...

These data are tab delimited

The variable names have slashes in them

The numbers have commas in them

```
> ctry = read.csv(
  fileLoc, skip = 1, sep = "\t", header = FALSE,
  colClasses = c("character", rep("numeric", 5),
    rep("character", 3)))
```

```
> head(ctry)
  V1 V2 V3 V4 V5 V6          V7          V8          V9
1 ABW  0  0  0  0  0    2,456,000,000.00    108,000 22740.7407
2 AFG  0  0  1  1  1   20,343,461,030.00   34,385,000   591.6377
3 AGO  0  0  0  0  0   100,990,000,000.00   19,082,000   5292.4222
4 ALB  0  0  0  0  0    12,959,563,902.00    3,205,000   4043.5457
5 AND  0  0  0  0  0     3,491,000,000.00     84,864  41136.4065
6 ARE  0  0  0  0  0   360,245,000,000.00    7,512,000  47955.9372
```

Note we skipped the first row because the names would be problematic

Next we need to:

Clean up the GDP and POP by removing ,s and converting character strings to numeric

# Data Available on the Web

```
> head(data$V7)
[1] "2,456,000,000.00"  "20,343,461,030.00"  "100,990,000,000.00"
[4] "12,959,563,902.00"  "3,491,000,000.00"   "360,245,000,000.00"

> fix7 = as.numeric(
      gsub(",", "", data$V7))

> head(fix7)
[1] 2456000000 20343461030 100990000000
[4] 12959563902 3491000000 360245000000
```

## HTML

- HTML
  - Table (e.g., your simulation results)
  - plain text format (e.g., the ManyEyes data)
- Other Format:
  - JSON
  - XML

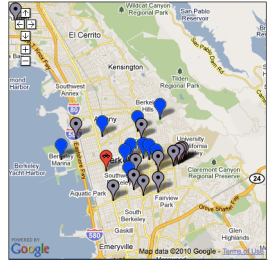
## Scraping data from a Web page

- Means to write code to automatically extract data from one or more web pages.
- HTML is like XML – We can use parsing capabilities in the XML package.  
`htmlParse()` can create a tree structure from ill-formed HTML.
- The information is all in text and we may need to use regular expressions to extract the relevant pieces

- Much of the data available on the web is not provided as a separate downloadable file; it's embedded in the website itself.

#### All incidents within 7 days

Displaying 1 - 25 of 28 incidents. < Previous | Next >



Top categories	
Theft	10 36%
Disturbances	5 18%
VANDALISM	5 18%
Aggravated Assault	3 11%
Burglary	3 11%
Stolen Auto	1 4%
Robbery	1 4%

Case num	Date	Category	Offense	Location
10055504	2010-11-01 03:40PM	Disturbances	Disturbance	1600 Block Milvia St
10055476	2010-11-01 11:45AM	Burglary	BURGLARY RESIDENTIAL	2400 Block Warring St
10055502	2010-11-01 10:30AM	Aggravated Assault	ASSAULT/BATTERY FELONY	600 Block Gilman St
10055442	2010-11-01 09:59AM	Theft	THEFT FELONY	2100 Block McGee Av
10055453	2010-11-01 09:35AM	Theft	THEFT FROM AUTO	1600 Block Carleton
10055431	2010-11-01 07:55AM	Disturbances	DOMESTIC VIOLENCE	3000 Block Martin Ln
10055419	2010-11-01 03:05AM	Robbery	Robbery	2700 Block Haste St

#### Popular Baby Names

Popular Names by Birth Year  
November 7, 2010

#### Popularity in 2009

Rank	Male name	Female name
1	Jacob	Isabella
2	Ethan	Emma
3	Michael	Olivia
4	Alexander	Sophia
5	William	Ava
6	Joshua	Emily
7	Daniel	Madison
8	Jayden	Abigail
9	Noah	Chloe
10	Anthony	Mia
11	Christopher	Elizabeth
12	Aiden	Addison
13	Matthew	Alexis
14	David	Ella
15	Andrew	Samantha
16	Joseph	Natalie
17	Logan	Grace
18	James	Lily
19	Ryan	Alyssa
20	Benjamin	Ashley
21	Elijah	Sarah
22	Gabriel	Taylor

- Web pages are created when your browser software represents or “renders” a specially formatted (HTML) text file. Most browsers allow you to see this file under something like View > Page Source.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<title>IMDb Charts</title>
<link rel="canonical" href="http://www.imdb.com/chart/" />
<meta name="title" content="IMDb Charts">
<meta name="description" content="IMDb: The biggest, best, most award-winning movie site on the planet.">
<meta name="keywords" content="movies, films, movie database, actors, actresses, directors, hollywood, stars, quotes">
<link rel="stylesheet" type="text/css" media="screen" href="http://i.media-imdb.com/images/SF05d2bd4730e135f1c4bceb">
<script type="text/javascript" src="http://i.media-imdb.com/images/SF3ee6861263732f8e66aacfd1850b466/a/js/ads.js">
</script>
<script type="text/javascript">
generic.monitoring.set_twilight_info("chart", "US", "83e50b7a3b50b8f7118fc7864f59906f20c5aeb4", "2009-10-07T18:00:00Z");
</script>
<script type="text/javascript">
generic.monitoring.start_timing("page_load");
</script>
<script type="text/javascript">
var aan = {
url:"http://aan.amazon.com/2009-05-01/imdb/default?slot=sitewide-iframe&ord=[CLIENT_SIDE_ORD]",
oncall:custom.amazon.aan.iframe_oncall
};
</script>
<iframe src="/images/SFed0def01846066a8fbf875202fe91fcd/a/js/scriptloader.html#aan" style="width:0px;height:0px;dis">
</iframe>
<link rel="icon" href="http://i.imdb.com/favicon.ico" />
<link rel="apple-touch-icon" href="http://i.media-imdb.com/apple-touch-icon.png" />
```

IMDb Top 250

IMDb Charts

Main Index

IMDb Top 250

IMDb Bottom 100

US Box Office

USA Top 10

USA Archive

UK Box Office

UK Top 10

UK Archive

All-Time Box Office

USA

Non-USA

World-wide

DVD Rentals

USA Weekly Top 20

Top 250 movies as voted by our users

For this top 250, only votes from regular voters are considered.

Track which films you've seen from the Top 250 [right here!](#)

Rank	Rating	Title	Votes
1	9.2	<a href="#">The Shawshank Redemption</a> (1994)	730,730
2	9.2	<a href="#">The Godfather</a> (1972)	549,136
3	9.0	<a href="#">The Godfather: Part II</a> (1974)	346,262
4	8.9	<a href="#">Pulp Fiction</a> (1994)	577,751
5	8.9	<a href="#">The Good, the Bad and the Ugly</a> (1966)	229,684
6	8.9	<a href="#">12 Angry Men</a> (1957)	180,224
7	8.9	<a href="#">Schindler's List</a> (1993)	386,132
8	8.8	<a href="#">The Dark Knight</a> (2008)	669,650
9	8.8	<a href="#">The Lord of the Rings: The Return of the King</a> (2003)	516,592
10	8.8	<a href="#">One Flew Over the Cuckoo's Nest</a> (1975)	309,947
11	8.8	<a href="#">Star Wars: Episode V - The Empire Strikes Back</a> (1980)	374,476
12	8.8	<a href="#">Fight Club</a> (1999)	556,117
13	8.8	<a href="#">Inception</a> (2010)	524,009

Our goal: Extract this information and put it in a dataframe

JSON

JavaScript Object Notation

# JSON Structure

- Text format
- Lightweight data-interchange
- Easy for humans to read and write.
- Easy for machines to parse and generate

- JSON is built on two structures:
- An unordered collection of comma-separated name:value pairs

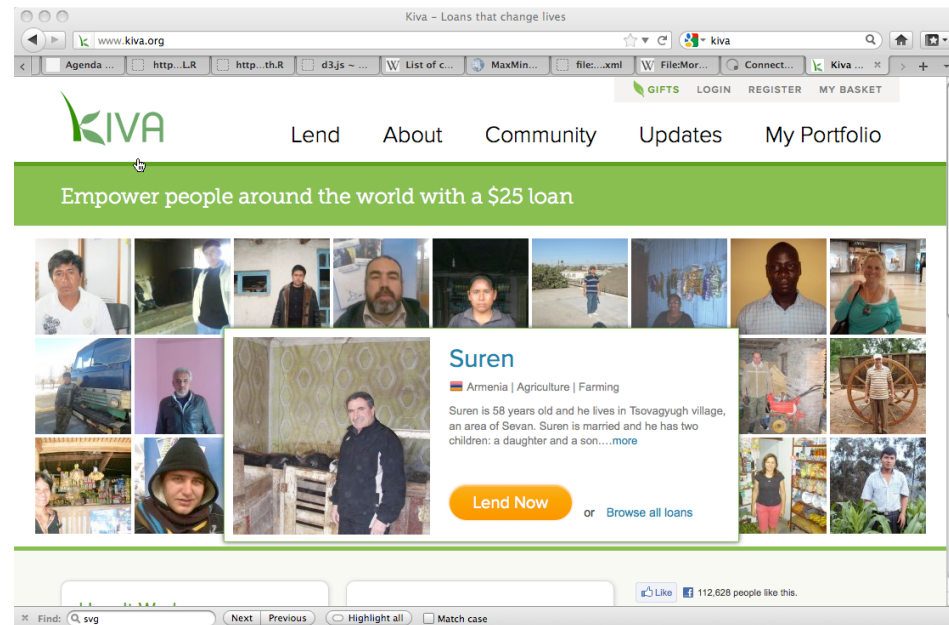
`{"lender_id":"matt", "loan_count":23}`

- An ordered array of values

`[1, [true, false, true], [1, 2, 10, 20], {"lender_id":"skylar", "loan_count":1}]`

## Comparison to XML

- JSON is simpler
- Not as rich – no attributes, unordered, no schema for describing acceptable format
- Compressed JSON and XML not much different in size



```
{
  "header": {
    "total": "576803",
    "page": "1",
    "date": "2010-01-29T20:00:23Z",
    "page_size": "1000"
  },
  "lenders": [
    {
      "lender_id": "matt",
      "name": "Matt",
      "image": {
        "id": "12829",
        "template_id": "1",
        "whereabouts": "San Francisco CA",
        "country_code": "US",
        "uid": "matt",
        "member_since": "2006-01-01T09:01:01Z",
        "personal_url": "www.socialedge.org/blogs/kiva-chronicles",
        "occupation": "Entrepreneur",
        "loan_because": "I love the stories. ",
        "occupational_info": "I co-founded a startup nonprofit (this one!) and I work with an amazing group of people dreaming up ways to alleviate poverty through personal lending. ",
        "loan_count": "89",
        "invitee_count": "23"
      },
      "lender_id": "jessica",
      "name": "Jessica",
      "image": {
        "id": "197292",
        "template_id": "1",
        ...
      }
    }
  ]
}
```

## XML

eXtensible Markup Language

Most of the data sets we have seen have been in the form of ASCII tables.

Date	Time	Lat	Lon	Depth	Mag
1968/01/12	22:19:10.35	36.6453	-121.2497	6.84	3.00
1968/02/09	13:42:37.05	37.1527	-121.5448	8.49	3.00
1968/02/21	14:39:48.10	37.1783	-121.5780	6.95	3.80
1968/03/02	04:25:53.94	36.8343	-121.5447	5.35	3.00
1968/03/17	15:07:02.12	37.3088	-121.6615	4.39	3.00
1968/03/21	21:54:59.94	37.0378	-121.7407	11.86	4.30

- **Advantages:**
  - easy to read, write, and process
  - in standard cases, don't need a lot of extra information
- But these advantages can quickly disappear....

XML is a standard for *semantic, hierarchical* representation of data

```
<state>
  <gml:name abbreviation="AL"> ALABAMA </gml:name>
  <county>
    <gml:name> Autauga County </gml:name>
    <gml:location>
      <gml:coord>
        <gml:X> -86641472</gml:X>
        <gml:Y> 32542207</gml:Y>
      </gml:coord>
    </gml:location>
  </county>
```

Relationships between pieces of data reflect relationships in the real world.



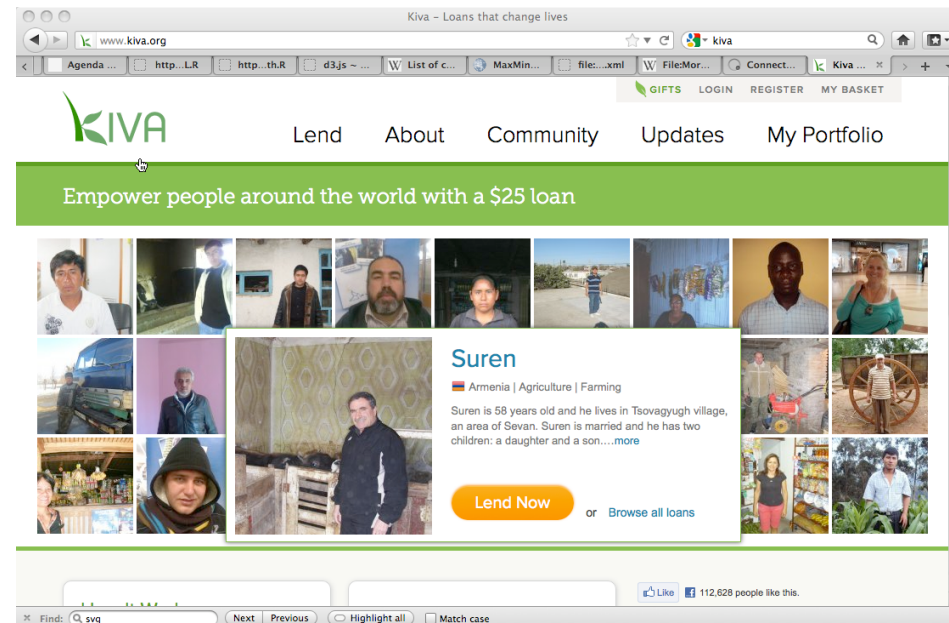
## Pros

- data is self-describing
- format separates content from structure
- data can be easily merged and exchanged
- file is human-readable
- file is also easily machine-generated
- standards are widely adopted

## Cons

- XML documents can be very verbose and hard to read
- It's so general that it's hard to develop tools for all cases
- Files can be quite large due to high amount of redundancy

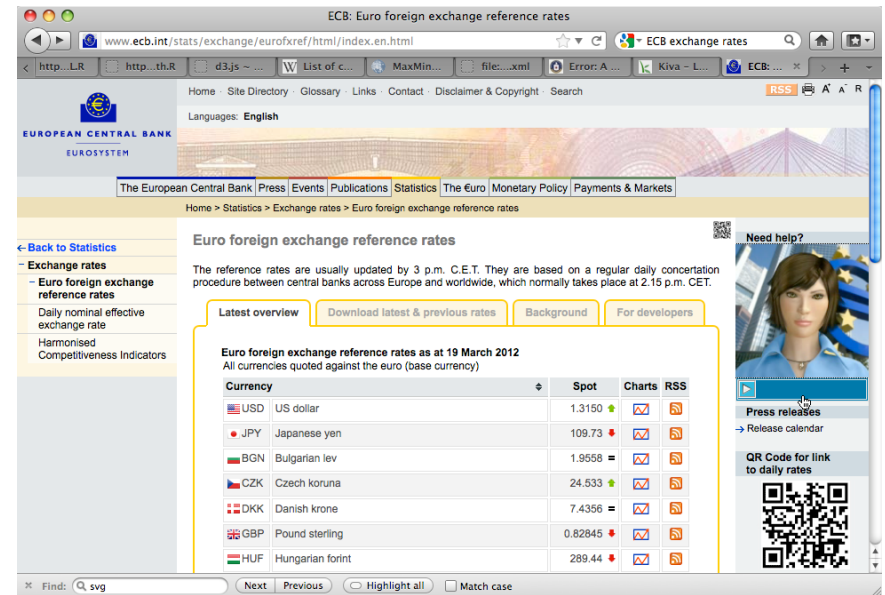
- XML is has become quite popular in many scientific fields, and it is standard in many web applications for the exchange and visualization of data.
- Well learn how to
  - create it and
  - read/process it.
- Well do both of these things from within R, but first let's start with an overview of what XML documents look like.



<lender>

```
<lender_id>matt</lender_id>
<name>Matt</name>
<image>
  <id>12829</id>
  <template_id>1</template_id>
</image>
<whereabouts>San Francisco CA</whereabouts>
<country_code>US</country_code>
<uid>matt</uid>
<member_since>2006-01-01T09:01:01Z</member_since>
<personal_url>www.socialedge.org/blogs/kiva-chronicles
</personal_url>
<occupation>Entrepreneur</occupation>
<loan_because>I love the stories. </loan_because>
<occupational_info>I co-founded a startup nonprofit (this one!)
  and I work with an amazing group of people dreaming up ways to
  alleviate poverty through personal lending.
</occupational_info>
<loan_count>89</loan_count>
<invitee_count>23</invitee_count>
</lender>
```

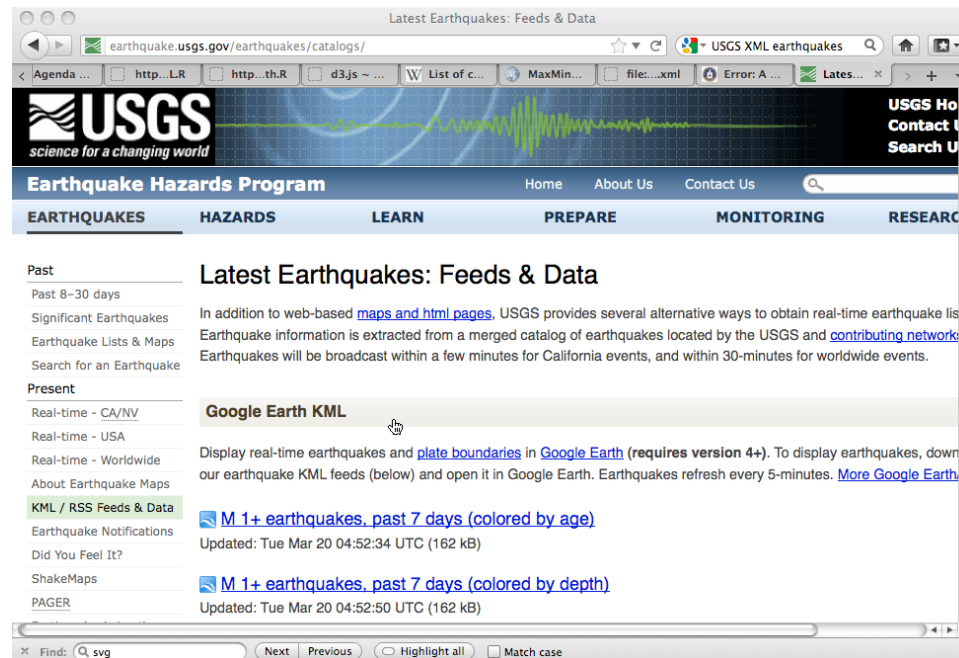
## Snippet of Kiva Data for one lender



## Snippet of exchange data

<Cube>

```
<Cube time="2008-04-21">
  <Cube currency="USD" rate="1.5898"/>
  <Cube currency="JPY" rate="164.43"/>
  <Cube currency="BGN" rate="1.9558"/>
  <Cube currency="CZK" rate="25.091"/>
</Cube>
<Cube time="2008-04-17">
  <Cube currency="USD" rate="1.5872"/>
  <Cube currency="JPY" rate="162.74"/>
  <Cube currency="BGN" rate="1.9558"/>
  <Cube currency="CZK" rate="24.975"/>
</Cube>
</Cube>
```



```
<event id="00068404" network-code="ak"
  time-stamp="2008/09/16_22:17:31" version="2">
  <param name="year" value="2008"/>
  <param name="month" value="09"/>
  <param name="day" value="14"/>
  <param name="hour" value="00"/>
  <param name="minute" value="59"/>
  <param name="second" value="04.0"/>
  <param name="latitude" value="51.8106"/>
  <param name="longitude" value="-175.9250"/>
  <param name="depth" value="146.0"/>
  <param name="magnitude" value="3.8"/>
  <param name="num-stations" value="10"/>
  <param name="num-phases" value="15"/>
  <param name="dist-first-station" value="126.1"/>
  <param name="azimuthal-gap" value="53"/>
  <param name="magnitude-type" value="L"/>
  <param name="magnitude-type-ext"
    value="Ml = local magnitude (synthetic Wood-Anderson)"/>
  <param name="location-method" value="a"/>
  <param name="location-method-ext"
    value="Auryn (Confirmed by human review)"/>
</event>
<event>
```

## Snippet of USGS earthquake catalog data

```
<actions>
<action datetime="2009-01-26">
  <text>Referred to the Committee on Appropriations, and in addition
    to the Committee on the Budget, for a period to be
    subsequently determined by the Speaker, in each case for
    consideration of such provisions as fall within the
    jurisdiction of the committee concerned.
  </text>
</action>
<action datetime="2009-01-26">
  <text>Referred to House Appropriations</text>
</action>
</actions>
..
<relatedbills>
  <bill relation="rule" session="111" type="hr" number="88" />
</relatedbills>
```

## Snippet of US Congress data

```

<Placemark id="217">
  <name>8.2</name>
  <description>
    Date: 2008-9-15
    Magnitude: 1.5
    Depth: 8.2 km
  </description>
  <styleUrl>#ball1-2</styleUrl>
  <Point>
    <coordinates>-147.426, 60.929, 0</coordinates>
  </Point>
</Placemark>

```

Snippet of  
KML for one  
earthquake

## XML Syntax

### Syntax

The basic unit of XML code is called an “element” or “node.” It is made up of both *markup* and content. Markup consists of *tags*, *attributes*, and *comments*.

```

<CYL> 6 </CYL> <!-- elem with content 6 -->

```

Start tag      Content      End tag      Comment - can go anywhere

```

<CYL> </CYL>
<CYL type="numeric" />
<CYL size="2"> 6 </CYL>

```

Elements with no content

An attribute

### Well-formed

- Tag names are case-sensitive; start and end tags must match exactly.
- No spaces are allowed between the `<` and the tag name.
- Tag names must begin with a letter and contain only alphanumeric characters.
- An element must have both an open and closing tag unless it is empty.
- An empty element that does not have a closing tag must be of the form `<tagname/>`.
- Tags must nest properly. (Inner tags must close before outer ones.)

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Edited with XML Spy v2006 (http://www.altova.com) -->
<CATALOG>
  <PLANT>
    <COMMON>Bloodroot</COMMON>
    <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
    <ZONE>4</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$2.44</PRICE>
    <AVAILABILITY>031599</AVAILABILITY>
  </PLANT>
  <PLANT>
    <COMMON>Columbine</COMMON>
    <BOTANICAL>Aquilegia canadensis</BOTANICAL>
    <ZONE>3</ZONE>
    <LIGHT>Mostly Shady</LIGHT>
    <PRICE>$9.37</PRICE>
    <AVAILABILITY>030699</AVAILABILITY>
  </PLANT>
</CATALOG>

```

XML declaration  
and processing  
instructions

Note how indentation  
makes it easier to  
check that the tags  
are correctly nested.

## Well formed XML ctd.:

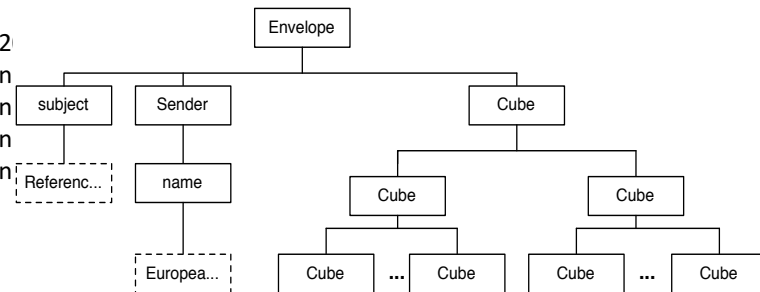
- All attributes must appear in quotes in the format:  
**name = "value"**
- Isolated markup characters must be specified via entity references. < is specified by &lt;; and > is specified by &gt;.
- All XML documents must contain a *root node* containing all the other nodes.

## Tree Representation

```

<Envelope>
  <subject>Reference rates</subject>
  <Sender>
    <name>European Central Bank</name>
  </Sender>
  <Cube>
    <Cube time="2008-04-21">
      <Cube currency="USD" rate="1.5898"/>
      <Cube currency="JPY" rate="164.43"/>
      <Cube currency="BGN" rate="1.9558"/>
      <Cube currency="CZK" rate="25.091"/>
    </Cube>
  </Cube>
</Envelope>

```



# Tree terminology

- There is only one *root or document node* in the tree, and all the other nodes are contained within it.
- We think of these other nodes as being *descendants* of the root node.
- We use the language of a family tree to refer to relationships between nodes. *Parents, children, siblings, ancestors, descendants*
- The *terminal nodes* in a tree are also known as *leaf nodes*. Content always falls in a leaf node.

## Note:

- We'll learn to *create* and *process* XML documents from within R, but always keep in mind that R and XML are two separate things.
- In particular, it will be helpful to have in your mind the structure of the XML document *before* you do anything in R, especially when you're creating a new XML document.