Topics

- Data
  - JSON
  - XML, XPath
  - SQL and SELECT statement
- Analysis
  - Naïve Bayes
  - Bootstrap and Bagging
  - Cross-validation

Data Analysis Issues

- Model selection: How to choose the k in K-NN, the complexity parameter in recursive partitioning, the threshold in Naïve Bayes?
- Variability: How accurate is our predictor?
- Generalizability: What can our data and findings say about other situations?

Structured Plain-text Data

Recall

- Delimited data, e.g.,
  - comma separated values
  - Tab-delimited
- Fixed-width format data
- Key:value pairs

Data Available on the Web

- HTML
  - HTML Table
  - plain text formats (e.g., delimited)
- Other Formats:
  - JSON
  - XML

JSON

JavaScript Object Notation

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

JSON Primitive Data Types

- Number
  - like numeric in R
  - no NaN, NA, Inf
- String (in double quotes)
- Boolean (true or false)
- These are scalars
- null – empty object (like NULL in R)

JSON Structures: Array

- Unnamed
- Ordered
- Comma-separated
- Square brackets
- Possibly heterogeneous

`[value, value, value, …]`

JSON Structures: Object

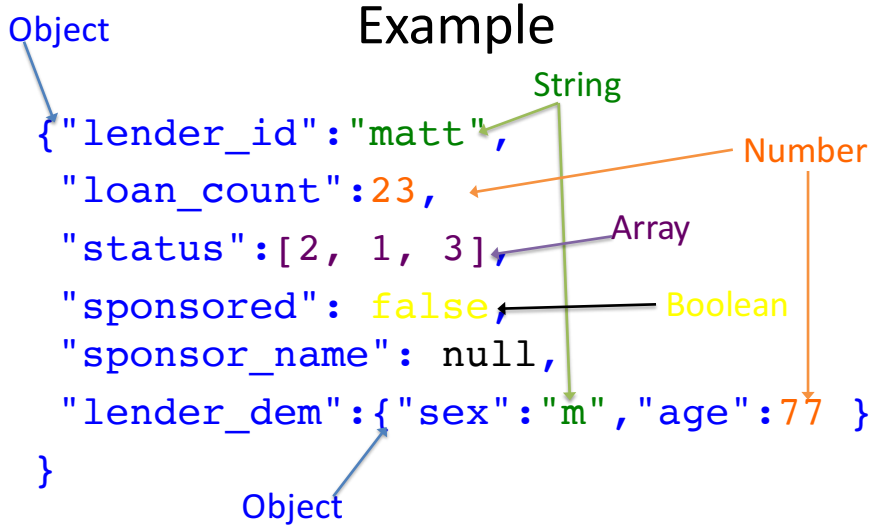
- Named
- Unordered
- Comma-separated
- Curly brackets
- Possibly heterogeneous
- AKA: Associative array

{"key": value, "key": value, "key": value, …}

Example

```
{"lender_id":"matt",
 "loan_count":23,
 "status":[2, 1, 3],
 "sponsored": false,
 "sponsor_name": null,
 "lender_dem":{"sex":"m","age":77 }
}
```

Example



Twitter

One Abbreviated Tweet

[
{
"favorite_count": 11495,
"favorited": false,
"user": {"verified": true, "profile_sidebar_fill_color": "C5CEC0", … }
"id_str": "786204978629185536",
"text": "Crooked Hillary Clinton likes to talk about the things she will do but she has been there for 30 years - why didn't she do them?",
"id": 7.862e+17,
"source": "Twitter for iPhone",
"lang": "en",
"geo": null,
"created_at": "Wed Oct 12 14:00:48 +0000 2016",
"place":{"full_name": "Florida, USA", … }, …]

Translate into a Structure in R

```
> length(tweetList)
[1] 3243

> class(tweetList[[1]])
[1] "list"

> names(tweetList[[1]])
 [1] "favorite_count"  "favorited"       "user"
 [4] "id_str"          "text"            "id"
 [7] "source"          "lang"            "geo"
[10] "created_at"      "place"
```

Translate into a Structure in R

```
> length(tweetList[[1]]$user)
[1] 41

> tweetList[[1]]$text
[1] "PAY TO PLAY POLITICS.
\n#CrookedHillary
https://t.co/wjsl8ITVvk"

> tweetList[[1]]$created_at
[1] "Wed Oct 12 14:00:48 +0000 2016"
```

Translate into a Structure in R

- Unnamed list of 3243 tweets
- Each tweet is a named list of 11 elements (this is an abbreviated file)
- Most of these elements are vectors of length 1 or NULL vectors
- Two are named lists with 41 (user) and 10 (place) elements. These elements are a mix of vectors and lists

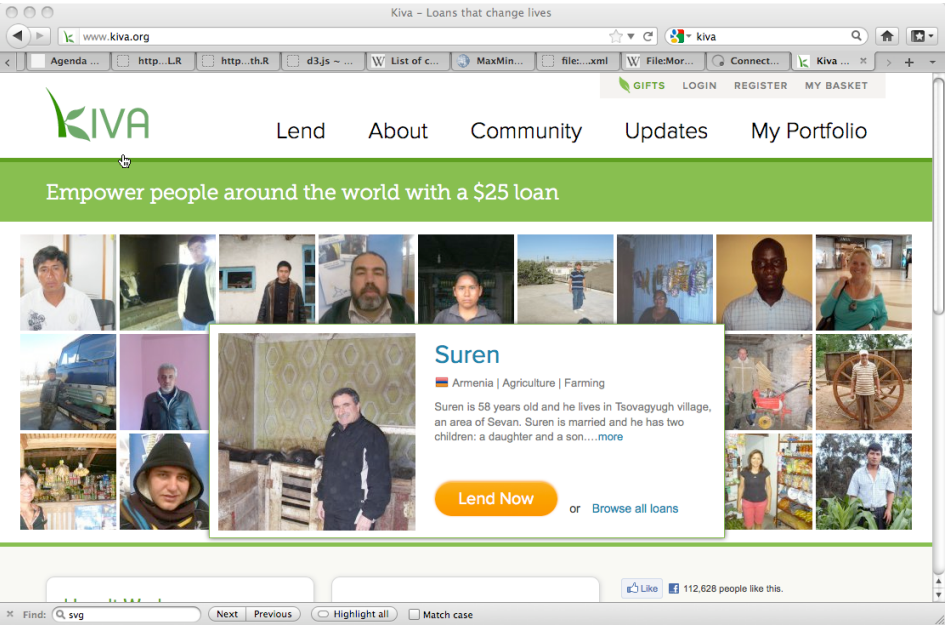
XML

eXtensible Markup Language

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

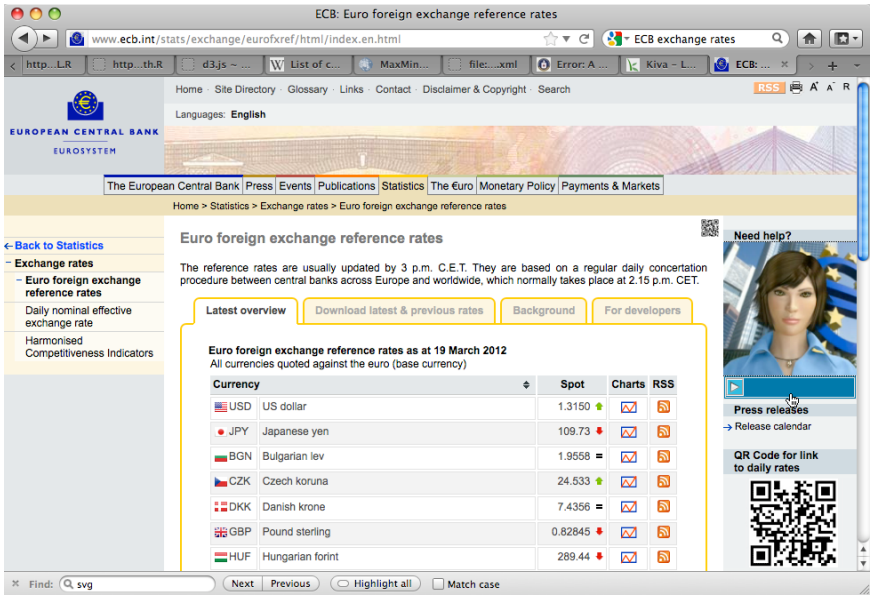
```
<catalog>
  <plant>
    <common>Bloodroot</common>
    <botanical>Sanguinaria canadensis</botanical>
    <zone>4</zone>
    <light>Mostly Shady</light>
    <price>2.44</price>
    <availability>03/15/2006</availability>
    <indoor>true</indoor>
  </plant>
…
</catalog>
```

Examples of XML




```
<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

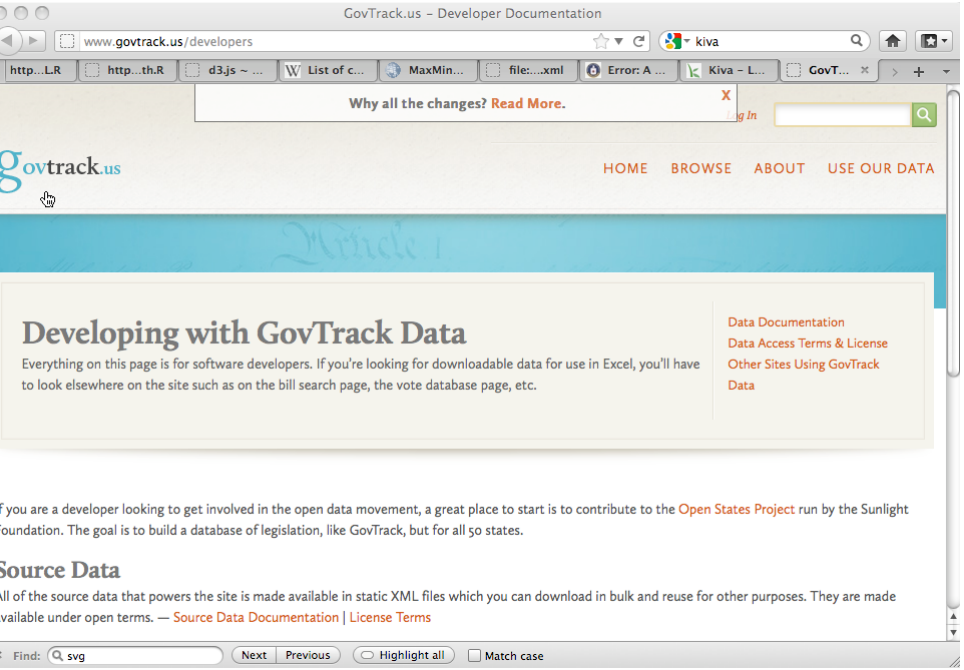


Snippet of exchange data

```
<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>
```

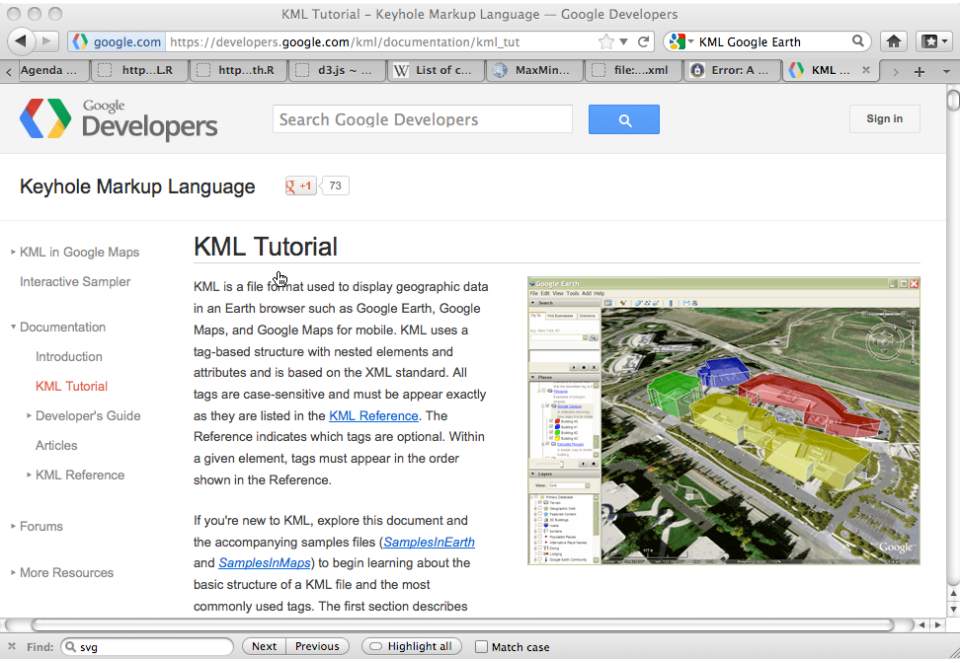

```
<event id="00068404" network-code="ak"
    time-stamp="2008/09/16_22:17:31 " version="2">
    <param name="year" value="2008"/>
    <param name="month" value="09"/>
    <param name="day" value="14"/>
    <param name="hour" value="00"/>
    <param name="minute" value="59"/>
    <param name="second" 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 (quakeml)




```
<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>
```

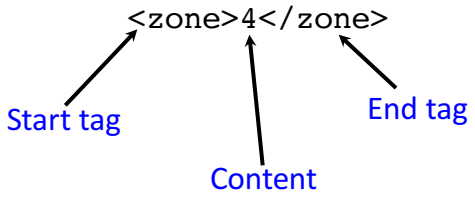
Snippet of KML for one earthquake placemark

XML Syntax

Syntax

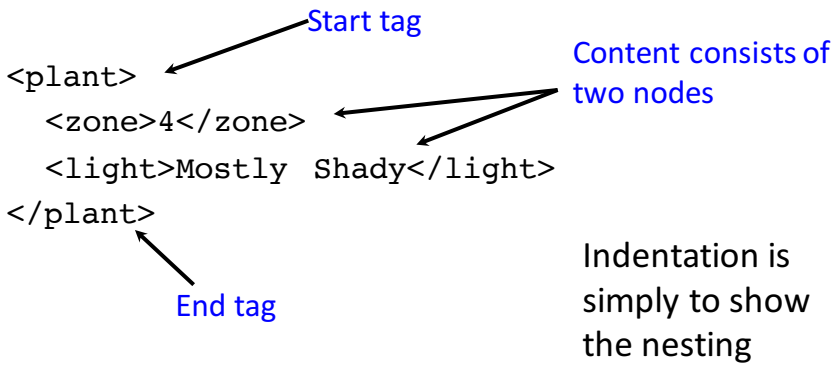
The basic unit of XML code is called an "element" or "node"

Each Node has a start tag and end tag



Syntax

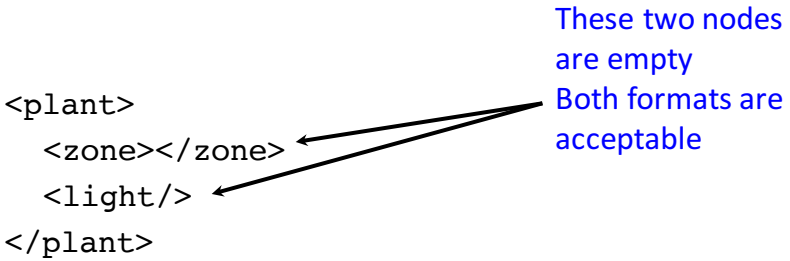
A node may have other other nodes (children) in it in addition to plain text content.



Indentation is simply to show the nesting

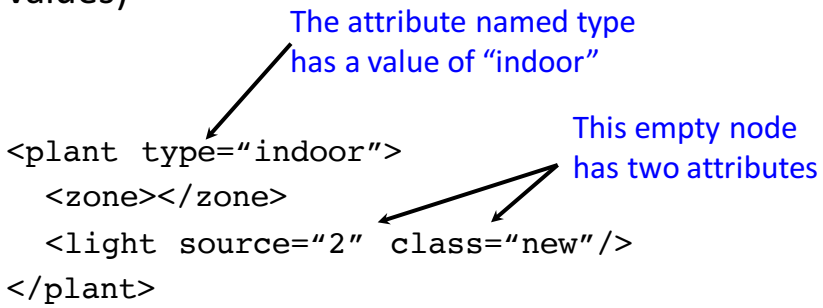
Syntax

Nodes may be empty



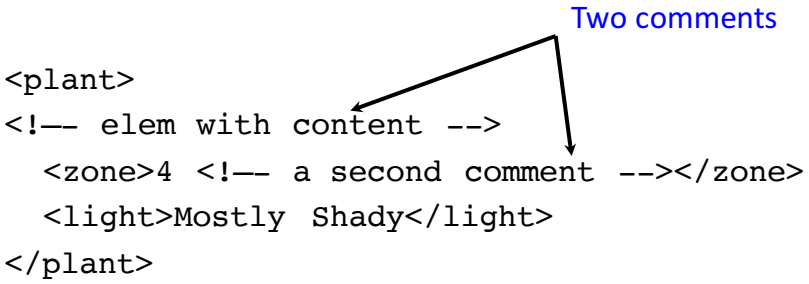
Syntax

Nodes may have attributes (and attribute values)



Syntax

Comments can appear anywhere



Well-formed XML

- An element must have both an open and closing tag unless it is empty. If empty, it can be of the form `<tagname/>`.
- Tags must nest properly. (Inner tags must close before outer ones.)
- Tag names are case-sensitive; start and end tags must match exactly.
- No spaces are allowed between `<` and tag name.
- Tag names must begin with a letter and contain only alphanumeric characters.

```
<?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.:

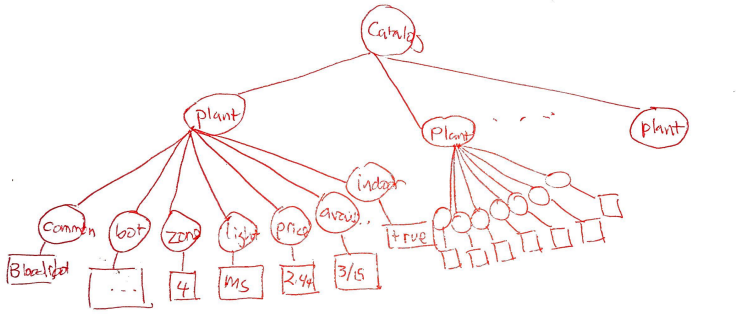
- All attributes must appear in quotes in the format:

  `name = "value"`

- Isolated markup characters must be specified via entity references. `<` is specified by `&lt;` and `>` is specified by `&gt;`.
- All XML documents must have *one root node* that contains all the other nodes.

Tree Representation

```
<catalog>
 <plant>
   <common>Bloodroot</common>
   <botanical>Sanguinaria canadensis</botanical>
   <zone>4</zone>
   <light>Mostly Shady</light>
   <price>$2.44</price>
   <availability>031599</availability>
 </plant>
   …
</catalog>
```



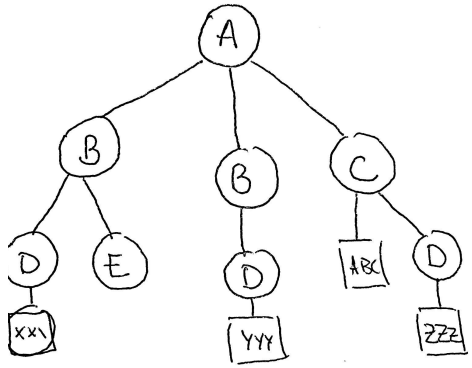
Tree terminology

- 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.

```
<A>
 <B>
   <D>xxx</D>
   <E/>
 </B>
 <B>
   <D> yyy </D>
 </B>
 <C>
   ABC
   <D> ZZZZ </D>
 </C>
</A>
```

Which statement is TRUE?

A. All B nodes are siblings
B. E's parent is a B node
C. All D nodes are siblings
D. A and B
E. A, B, and C




```
<A>
 <B>
   <D>xxx</D>
   <E/>
 </B>
 <B>
   <D> yyy </D>
 </B>
 <C>
   ABC
   <D> ZZZZ </D>
 </C>
</A>
```

Which statement is TRUE?

A. All B nodes are siblings
B. E's parent is a B node
C. All D nodes are siblings
D. A and B
E. A, B, and C

Working with XML in R

XML package

- Handy functions for parsing XML
  - **xmlParse**: read an XML file into R
  - **xmlValue**: retrieve text content of a node (including content of all child nodes)
  - **xmlSize**: return the number of child nodes
  - **xmlName**: return the tag name of a node
  - **xmlGetAttr**: return the attribute value of the specified attribute

To read an XML file into R, use `xmlParse`

```
doc = xmlParse("plant.xml")
```

and extract the root node using `xmlRoot`.

```
catalog = xmlRoot(doc)
class(catalog)
[1] "XMLNode"
```

To illustrate how we manipulate an XML object in R, we take this data and reformat it into a data frame with one row for each plant.

The XML document is a special object in R.

It is similar to a list, but we need special functions to extract content

`xmlParse` implements what is called the DOM (Document Object Model) parser.

We don't have time to cover it, but you should be aware of another parsing model called SAX (Simple API for XML). It reads the document incrementally and is more memory efficient, but it is trickier to use.

The tree structure is represented in R as a list of lists.

We can access an element within a node (i.e., a child), using the usual `[[ ]]` indexing for lists.

- XML versions of `lapply` and `sapply`, named `xmlApply` `xmlSApply`. Each takes an XMLNode object as its primary argument. They iterate over the node's children nodes, invoking the given function.

- Like `lapply`, `xmlApply` returns a list. Like `sapply`, `xmlSApply` returns a simpler data structure if possible.

Look at the first plant node

```
> oneplant = catalog[[1]]
> class(oneplant)
[1] "XMLNode"
> oneplant
<plant>
 <common>Bloodroot</common>
 <botanical>Sanguinaria canadensis</botanical>
 <zone>4</zone>
 <light>Mostly Shady</light>
 <price>$2.44</price>
 <availability>031599</availability>
</plant>
```

Here is a leaf node
It contains the content
"Bloodroot"

We can drill down further into the list:

```
> oneplant[['common']]
<common>Bloodroot</common>
```

Note that this doesn't remove the markup.

To do this, use the function `xmlValue`

```
> xmlValue(oneplant[['common']])
[1] "Bloodroot"
> xmlValue(oneplant[['botanical']])
[1] "Sanguinaria canadensis"
```

TASK: Create a vector of common names:

```
common = xmlSApply(catalog, function(el){
                   xmlValue(el[['common']])})
head(common)
          plant                     plant                     plant
"Bloodroot"                "Columbine"    "Marsh Marigold"
          plant                     plant                     plant
  "Cowslip"    "Dutchman's-Breeches"      "Ginger, Wild"
```

The elements of the root node are all plant nodes, like `oneplant`.

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
- It's so general that it can be difficult to develop tools for all cases
- Files can be quite large due to high amount of redundancy

Comparison JSON & XML

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