Week 7 Agenda

- Text data – Unicode/UTF-8 quick recap
- Special text formats: JSON, HTML, XML
- JSON examples: Twitter
- Reading files line by line
  - Basic file operations
- Retrieving data from the web
  - This will usually be HTML for pre-existing tables, JSON or XML for “dynamic” data
- Web scraping examples
  - Free-form vs. via REST API

NAVAL
POSTGRADUATE
SCHOOL

OS4118

Statistical and Machine Learning

Text Data

Prof. Sam Buttrey

Fall AY 2020

The Nation's Premiere Defense Research University

Monterey, California
WWW.NPS.EDU

Unicode

- **Unicode** is a scheme to represent characters from all the world’s languages
- Can display about 1M characters (“code points”), broken into 17 “planes”
- The characters in plane 0, the “basic multilingual plane,” have two-byte addresses 0x0000 to 0xFFFF
  - – Superset of ASCII, which is 0x00 to 0x7F

UTF-8

- The **UTF-8** encoding of Unicode is byte-ordered and is the standard for web pages, XML, and, really, everything else
  - Except that Microsoft’s “Save as Unicode” uses the different UTF-16
- ASCII require 1 byte (backwards compatibility); others require 2-6
- Example: Euro sign € is \U20ac
- Represented in UTF-8 as hex e2 a2 8c

The NUL Character

- NUL (0x00) requires special thought
- For years it served as a de facto “end of string” character because it serves this role in C programs (see ?readBin)
  - Issues: learning the string length requires traversing the whole string; programmers often forgot to allocate the extra character; strings and binary must be treated differently…

NUL Continued

- UTF-8 encodes NUL as 0x00, but…
- Lots of legacy code uses UTF-8 + NUL-terminated strings
  - “Modified UTF-8” treats NUL in a special way, encoding it as 0xC080, so that no single UTF-8 byte is ever 0x00
  - This is not permitted by the UTF-8 standard
  - NUL characters are generally not permitted in R, so if they’re in your file, watch out
  - Moral: Be aware of NUL issues

Text Formats

- **Delimited** (e.g. CSV, tab-separated)
  - – Flexible
  - – The separator itself takes up room
  - – The separator can be mistaken for legitimate contents – commas are common in free text
  - – Europeans more often use ;
- **Fixed-format**
  - – Sometimes easier to handle, smaller...
  - – Although format specs can be a pain to use

Delimited Text

```
Name,ID,State,Amount1,Date1
Buttrey,003682X,CA,102.45,11/09/2016
Jenkins,001926Z,WA,2130.40,11/30/2016
Lee,000411R,MI,-1.36,11/18/2016
Rimsky-Korsakov,6076T,QB,642.28,11/28/2016
```

- Generally carries column headers, but you’ll still need documentation
- Again, the comma is a common choice for delimiter but causes trouble with free-form text (even “comma-free” entries like city names!)

Fixed-Format Data

Data:

```
Buttrey.....003682XCA 0001024520161109
Jenkins.....001926ZWA 0021304020161130
Lee.........000411RMI-0000013620161118
Rimsky-Korsa006076TQB 0006422820161128
```

Format (shown here COBOL style):

```
Name        PIC X(12)
ID          PIC X(07)
Amt1        PIC S9(06)V99
Dt1         PIC 9(08)
```

Text Formats (cont'd)

- HyperText Markup Language (**HTML**) is a scheme for telling browsers how to display pages
- Text together with "markup" tags
- `<html><head>…</head>`
  `    <body>Text goes here</body></html>`
  - "Head" includes title, metadata like author
- Often tags come in pairs: <b>bold</b>, `<font color="red">red</font>`

HTML, Continued

- Markup gives display information to the browser, which is in charge of "rendering" (displaying) the page
- The set of tags is fixed by standard, although inevitably browers disagree
- HTML supports a list of special characters; usually our tools interpret these properly
- HTML supports UTF-8, which will be the default in the upcoming **HTML5**
- One important tag is `<table>`

HTML Tables

- Enclosed in `<table></table>`
- Rows in `<tr></tr>`, entries in `<td></td>`
  - `colspan, rowspan` to span multiple col/rows
  - HTML tags are not case-sensitive
  - Table rows can span many lines of text file
- Header in row 1 or inside `<th></th>`
- Tables often nested for formatting reasons

HTML Tables in R

- `readHTMLTable()` in R library XML or the newer library `htmltab` will try to extract table data…
  - …producing a list of tables found…
- …but HTML is often non-standard; you may have to acquire, decode it yourself
  - With, e.g., `GET {httr}`, `getURLContent {RCurl}`
- Examples: EIA tables

XML

- **XML** is a standard mechanism for storing and transporting (but not displaying) data
- It looks a little like HTML, but HTML's focus is on the display
- XML is very Unicode-friendly, usually using UTF-8
- XML is text-based, so it's not particularly well-suited for floating-point data

XML (cont’d)

- **XML**
  - – Very structured, easier to impose content restrictions at creation time (properly-formatted dates, factor levels…)
  - – Much bigger than just the data
    - Zillions of tags take up most of the room
  - – Not friendly to binary data
  - – Special editors helpful
  - – We will never write a parser for XML; there are libraries available for every serious language

HTML vs XML

- HTML comes with a fixed, defined set of tags (`<p>`, `<i>`, `<b>`, `<table>` etc.)
- In XML, we define our own (case-sensitive) tags for the purpose at hand
- So XML isn’t quite a language; it’s a set of tools for defining a new language
- E.g.: one such language, called **KML**, is what is used by Google Earth
  - Tags like `Placemark`, `Polygon`, `coordinates`…

XML Example

```
<customer><Name>Buttrey</Name>
<ID>003682X</ID><State>CA</State>
<Amount1>102.45</Amount1>
<Date1>20161109</Date1> ... </customer>

<customer><Name>Jenkins</Name>
<ID>001926Z</ID><State>WA</State>
<Amount1>2130.40</Amount1>
<Date1>20161130</Date1>...</customer>
```

XML Anecdote

- For NAVAIR Data Challenge 1, participants were given XML files with aircraft maintenance data
- Saved “as Unicode” in Windows, which means UTF-16, not UTF-8
  - Every ASCII character – so, every character in the file – took up two bytes instead of one
- No “new-line” characters were present
- It’s important that producers and consumers of data be on the same page!

Handling XML in R

- XML library at CRAN
  - Includes our `readHTMLTable()` function
- Functions to ingest, deparse, store, and write XML
- **XPath** and **XQuery** are query languages to select nodes from XML trees
  - Also can extract and compute on values
  - Xpath {XML}

Text Formats (cont’d)

- **JSON** (“JavaScript Object Notation”)
- Began as a mechanism for server-browser communication
  - Every browser understands JavaScript
- Now used as a storage format
  - E.g. Twitter data
- Braces surround name-value pairs, separated by commas

JSON

- `{"Name":"Buttrey", "ID":"003682X", "State":"CA", "Amount":102.45, "Date":"2016-11-09"}`
- Special values `true`, `false`, `null` and numbers don’t need quotes
- Can contain nested arrays (in square brackets) or objects (in braces)
- Always best handled with a library

Text Formats (cont’d)

- Free-form
  - Free-form text is never your friend
  - Inconsistencies in format, spelling and typography; end-of-line hyphens, all cause problems
  - Extracting meaning is hard

```
Buttrey, a Californian, sent his pay-
ment on November ninth of last year, in
the amount of $102.45...
```

Some Niceties of Text in R

- Individual string lengths given by `nchar()` (compare `length()`)
- How many chars in the Euro symbol?
  - A: One, but it takes three bytes
  - Compare nchar `("\t\t\n")` = 3
  - And `nchar("\Ud7a1", type="width")`
- We often see the empty string `""`, which has length 0 – but it's hard to distinguish that from strings like "   "

Niceties of R Text, cont'd

- Missing character values look like NA
  - This is not quite the same as numeric NA
  - Definitely different from characters `"NA"`
  - nchar (NA) = NA by default*
- Strings can be unreadably long
  - `strwrap()` formats paragraphs
- There are lots of tools for common tasks like tokenization, "lemmatization," removal of stopwords…

Niceties of R Text, cont'd

- Lots of R text is stored as **factors**, which are internally <integer + labels>
- Factor strengths: persistent levels – but this is also their weakness
- Factors are more suitable for R "long vectors" (length ≥ 2^{31}) than characters
  - Otherwise, I advise you to avoid factors
- "Never" use `as.numeric()` on a factor
  - Use `as.character(f)` or `levels(f)[f]`

Tools for Text in R

- `substring(vec, 4, 7)` produces a new vector, same length as `vec`, with characters 4-7 from each element of `vec`
  - Can also be used to assign (but not extend)
  - Start and stop arguments are vectorized; this allows one command to do a replacement for every element in a vector
  - Or, e.g. `substring (a, 1, nchar(a)-3)# truncate 3 chars from each`
  - See also `startsWith()`, `endsWith()`

Tools for Text in R

- `paste()` combines arguments in a vectorized way
- Within one element, separator is `sep`, defaults to space (`sep=""`, `paste0()`)
- Collapse into one string: `collapse=""`
  - I often use `cat (paste (vec, collapse ="\n"),"\n", file = outfile)`
  - Compare `write.table()`
  - `file="clipboard"/"clipboard-128"` on Windows or `pipe("pbcopy", "w")` and `pipe("pbpaste")` for Mac clipboard

Tools for Text in R (cont’d)

- `sprintf ()` formats data as text
  - – Including leading/trailing zeros, scientific notation, hex, strings
  - – Vectorized
- `format()` is handy for lining things up in reports and, particularly, for dates
- R has a big set of tools that use **regular expressions** to search, split, replace
  - – (The subject of a separate lecture)