Plan today

- The nature of data
- Units of data
- Data types
- More about git
- Representing text data: Unicode
- Representing dates
- Representing sparse matrix formats
- Data and datasets
  - "tidy" data
  - reshaping data
  - normalization forms
- Lab preview
Data is Fundamental

"You can have data without information, but you cannot have information without data." – Daniel Keys Moran, an American computer programmer and science fiction writer.

"In God we trust. All others must bring data." – W. Edwards Deming, statistician, professor, author, lecturer, and consultant.
Structured data: Index cards

- Origins in the 19th century, with botanist Carl Linnaeus (https://en.wikipedia.org/wiki/Carl_Linnaeus), who needed to record species that he was studying

- This was a form of *database*
  - each piece of information about a species formed a *field*
  - each species' entry in the system formed a *record*
  - the records were *indexed* using some reference system
Heyday: Use in libraries to catalog books
a record looked like this
Dewey decimal system

- a proprietary library classification system first published in the United States by Melvil Dewey in 1876
- scheme is made up of ten classes, each divided into ten divisions, each having ten sections
- the system's notation uses Arabic numbers, with three whole numbers making up the main classes and sub-classes and decimals creating further divisions

Example:

500 Natural sciences and mathematics
  510 Mathematics
    516 Geometry
      516.3 Analytic geometries
        516.37 Metric differential geometries
          516.375 Finsler Geometry
How to index?

- Problem: Could only sort the cards one way
- Re-referencing was literally a manual operation
• Contrast with the idea of electronic indexes, where assets are stored once, and many indexing and reference systems can be applied.

• In most photographic software, edits are "non-destructive", and stored separately from the original images.
Modern database manager

- "Normalizes" data into relational tables, linked by "keys" (more on this in Week 3 (https://lse-st445.github.io/#week-3-creating-and-managing-databases))
Punch cards (and legacy systems)

- How we used to enter programs into the computer

- Pre-computer origins: 18-th century use in textile looms
- Responsible for the 80-character legacy
Who knows what this is?
The point: Information is more than data

Data
- raw, unorganized facts that need to be processed
- unusable until it is organized

Information
- created when data is processed, organized, structured
- needs to be situated in an appropriate context in order to become useful

There are important differences in how humans and computers treat data as information
Changes in the world of data

- volume of data in the modern world: 90% of the world's data generated in the last two years (https://www.sciencedaily.com/releases/2013/05/130522085217.htm)

- and that was in 2013

- SKA: Square Kilometer Array
  - a southern hemisphere radio telescope with a total of 1 km$^2$ of data sensors
  - will generate 1 exabyte daily $= 1 \times 10^{18}$ bytes
  - $= 1,000,000,000,000,000,000,000$ bytes
• compare this with the Apollo Guidance Computer (1966), which guided the first humans to the moon
  ▪ 16-bit wordlength, 2048 words RAM (magnetic core memory) = 4KB
  ▪ 36,864 words ROM (core rope memory) = 73KB
Basic units of data

- Bits
  - smallest unit of storage, a 0 or 1
  - anything that can store two states - now "transistors", used to be vacuum tubes
  - with $n$ bits, can store $2^n$ patterns - so one byte can store 256 patterns
- **Bytes**
  - eight bits = one byte
  - "eight bit encoding" - represented characters, such as A represented as 65

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Char</th>
<th>Dec</th>
<th>Hx</th>
<th>Oct</th>
<th>Html</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>NUL (null)</td>
<td>32</td>
<td>20</td>
<td>040</td>
<td>Space</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>SOH (start of heading)</td>
<td>33</td>
<td>21</td>
<td>041</td>
<td>!</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>02</td>
<td>STX (start of text)</td>
<td>34</td>
<td>22</td>
<td>042</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>03</td>
<td>ETX (end of text)</td>
<td>35</td>
<td>23</td>
<td>043</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>04</td>
<td>EOT (end of transmission)</td>
<td>36</td>
<td>24</td>
<td>044</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>05</td>
<td>ENQ (enquiry)</td>
<td>37</td>
<td>25</td>
<td>045</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>06</td>
<td>ACK (acknowledge)</td>
<td>38</td>
<td>26</td>
<td>046</td>
<td>&amp;</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>07</td>
<td>BEL (bell)</td>
<td>39</td>
<td>27</td>
<td>047</td>
<td>'</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>08</td>
<td>BS (backspace)</td>
<td>40</td>
<td>28</td>
<td>050</td>
<td>(</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>09</td>
<td>HT (horizontal tab)</td>
<td>41</td>
<td>29</td>
<td>051</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>010</td>
<td>LF (NL line feed, new line)</td>
<td>42</td>
<td>2A</td>
<td>052</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>013</td>
<td>VT (vertical tab)</td>
<td>43</td>
<td>2B</td>
<td>053</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>014</td>
<td>FF (form feed, new page)</td>
<td>44</td>
<td>2C</td>
<td>054</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>015</td>
<td>CR (carriage return)</td>
<td>45</td>
<td>2D</td>
<td>055</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>016</td>
<td>SO (shift out)</td>
<td>46</td>
<td>2E</td>
<td>056</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>017</td>
<td>SI (shift in)</td>
<td>47</td>
<td>2F</td>
<td>057</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>020</td>
<td>DLE (data link escape)</td>
<td>48</td>
<td>30</td>
<td>060</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>021</td>
<td>DC1 (device control 1)</td>
<td>49</td>
<td>31</td>
<td>061</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>022</td>
<td>DC2 (device control 2)</td>
<td>50</td>
<td>32</td>
<td>062</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>023</td>
<td>DC3 (device control 3)</td>
<td>51</td>
<td>33</td>
<td>063</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>024</td>
<td>DC4 (device control 4)</td>
<td>52</td>
<td>34</td>
<td>064</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>025</td>
<td>NAK (negative acknowledge)</td>
<td>53</td>
<td>35</td>
<td>065</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>026</td>
<td>SYN (synchronous idle)</td>
<td>54</td>
<td>36</td>
<td>066</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>027</td>
<td>ETB (end of trans. block)</td>
<td>55</td>
<td>37</td>
<td>067</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>030</td>
<td>CAN (cancel)</td>
<td>56</td>
<td>38</td>
<td>070</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>031</td>
<td>EM (end of medium)</td>
<td>57</td>
<td>39</td>
<td>071</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>20</td>
<td>032</td>
<td>SUB (substitute)</td>
<td>58</td>
<td>3A</td>
<td>072</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>21</td>
<td>033</td>
<td>ESC (escape)</td>
<td>59</td>
<td>3B</td>
<td>073</td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>22</td>
<td>034</td>
<td>FS (file separator)</td>
<td>60</td>
<td>3C</td>
<td>074</td>
<td>e</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>23</td>
<td>035</td>
<td>GS (group separator)</td>
<td>61</td>
<td>3D</td>
<td>075</td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>24</td>
<td>036</td>
<td>RS (record separator)</td>
<td>62</td>
<td>3E</td>
<td>076</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>25</td>
<td>037</td>
<td>US (unit separator)</td>
<td>63</td>
<td>3F</td>
<td>077</td>
<td>h</td>
<td></td>
</tr>
</tbody>
</table>

Source: [www.LookupTables.com](http://www.LookupTables.com)
### multi-byte units

<table>
<thead>
<tr>
<th>unit</th>
<th>abbreviation</th>
<th>total bytes</th>
<th>nearest decimal equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>kilobyte</td>
<td>KB</td>
<td>1,024^1</td>
<td>1000^1</td>
</tr>
<tr>
<td>megabyte</td>
<td>MB</td>
<td>1,024^2</td>
<td>1000^2</td>
</tr>
<tr>
<td>gigabyte</td>
<td>GB</td>
<td>1,024^3</td>
<td>1000^3</td>
</tr>
<tr>
<td>terabyte</td>
<td>TB</td>
<td>1,024^4</td>
<td>1000^4</td>
</tr>
<tr>
<td>petabyte</td>
<td>PB</td>
<td>1,024^5</td>
<td>1000^5</td>
</tr>
<tr>
<td>exabyte</td>
<td>EB</td>
<td>1,024^6</td>
<td>1000^6</td>
</tr>
<tr>
<td>zettabyte</td>
<td>ZB</td>
<td>1,024^7</td>
<td>1000^7</td>
</tr>
<tr>
<td>yottabyte</td>
<td>YB</td>
<td>1,024^8</td>
<td>1000^8</td>
</tr>
</tbody>
</table>

- this is why 1GB is greater than 1 billion bytes
Comparison of decimal and binary units

Metric storage capacity (log scale)

Percent less than equivalent in binary units

- Kilobyte: 2.34%
- Megabyte: 4.63%
- Gigabyte: 6.87%
- Terabyte: 9.05%
- Petabyte: 11.18%
- Exabyte: 13.26%
Data types: Generically

- objects are *bound* to an identifier, e.g.

```
In [2]:
    temperature = 98.6
    print(temperature)
```

98.6

- here, `temperature` is a variable name assigned to the literal floating-point object with the value of 98.6
- in Python, this is an instance of the `float` class
- identifiers in R and Python are *case-sensitive*
- some identifiers are typically reserved, e.g.
  ```
  False, True, None, or, and    # Python
  FALSE, TRUE, NA              # R
  ```

- All programming languages use comments, for humans to read
  - this is anything that follows the `#` character in both Python and R
"Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do." -- Donald Knuth, Literate Programming (1984)
Instantiation

- objects have classes, meaning they represent a "type" of object
- instantiation means creating a new instance of that class
- "immutable" objects cannot be subsequently changed

<table>
<thead>
<tr>
<th>Python class</th>
<th>Immutable</th>
<th>Description</th>
<th>R class</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>Yes</td>
<td>Boolean value</td>
<td>logical</td>
</tr>
<tr>
<td>int</td>
<td>Yes</td>
<td>integer number</td>
<td>integer</td>
</tr>
<tr>
<td>float</td>
<td>Yes</td>
<td>floating-point number</td>
<td>numeric</td>
</tr>
<tr>
<td>list</td>
<td>No</td>
<td>mutable sequence of objects</td>
<td>list</td>
</tr>
<tr>
<td>tuple</td>
<td>Yes</td>
<td>immutable sequence of objects</td>
<td>-</td>
</tr>
<tr>
<td>str</td>
<td>Yes</td>
<td>character string</td>
<td>character</td>
</tr>
<tr>
<td>set</td>
<td>No</td>
<td>unordered set of distinct objects</td>
<td>-</td>
</tr>
<tr>
<td>frozenset</td>
<td>Yes</td>
<td>immutable form of set class</td>
<td>-</td>
</tr>
<tr>
<td>dict</td>
<td>No</td>
<td>dictionary</td>
<td>(named) list</td>
</tr>
</tbody>
</table>
How to index data

- Endian debate
  - Big Endian Byte Order: The most significant byte (the "big end") of the data is placed at the byte with the lowest address
    - Little Endian Byte Order: The least significant byte (the "little end") of the data is placed at the byte with the lowest address
    - Comes from *Gulliver's Travels*
(indexing data cont.)

- index from 0 or from 1?
  - where an index begins counting, when addressing elements of a data object
  - most languages index from 0 (https://en.wikipedia.org/wiki/Comparison_of_programming_languages_%28array_reference_list)
  - human ages - do they index from 0?

In [6]:
string_example = "Hello World"
string_example[0:5]

Out[6]: 'Hello'

- Python indexes from 0, R from 1. See R example here (R_indexing.ipynb)
git

- git: a revision control system
- Allows for complete history of changes, branching, staging areas, and flexible and distributed workflows

- simplified workflow (from Anita Cheng's excellent blog post [http://anitacheng.com/git-for-non-developers])
• Works through the command line, or through GUI clients, or through most IDEs and (good) editors
• or through editors (here, the excellent Atom (https://atom.io) editor)
GitHub

- a website and hosting platform for git repositories
- and so much more...
  - "continuous integration" hooks: https://github.com/kbenoit/spacyr (https://github.com/kbenoit/spacyr), (for instance - see the badges)
- GitHub classroom (https://classroom.github.com).
More great resources for using git/GitHub

- git - the simple guide (http://rogerdudler.github.io/git-guide/), by Roger Dudler

Some people have entire, open-source, user-commented books online, such as:

- R for Data Science (http://r4ds.had.co.nz)
  - with source code here (https://github.com/hadley/r4ds)
  - with GitHub issues here (https://github.com/hadley/r4ds/issues)
  - and pull requests - examples here (https://github.com/hadley/r4ds/pulls)
Markdown (and other markup languages)

- Idea of a "markup" language: HTML, XML, LaTeX
- "Markdown"
  - Created by John Gruber as a simple way for non-programming types to write in an easy-to-read format that could be converted directly into HTML
  - No opening or closing tags
  - Plain text, and can be read when not rendered
- Markdown has many "flavours" (https://github.com/commonmark/CommonMark/wiki/Markdown-Flavors)
# Markdown example

This is a markdown example.

- bullet list 1
- bullet list 2

"I love deadlines. I like the whooshing sound they make as they fly by. (https://www.brainyquote.com/quotes/quotes/d/douglasada134151.html?src=t_funny)"
-- Douglas Adams

# Markdown example

This is a markdown example
* bullet list 1
* bullet list 2

> "[I love deadlines. I like the whooshing sound they make as they fly by.](https://www.brainyquote.com/quotes/quotes/d/douglasada134151.html?src=t_funny)"
-- _Douglas Adams_
How to represent text data: encoding

- A “character set” is a list of character with associated numerical representations.

- ASCII: the original character set, uses just 7 bits ($2^7$) – see [http://ergoemacs.org/emacs/unicode_basics.html](http://ergoemacs.org/emacs/unicode_basics.html).


Solution: Unicode
But: Unicode must still be encoded

- and displayed... Different devices will render displayed emoji differently (https://unicode.org/emoji/charts/full-emoji-list.html), for instance

- another problem: there are more far code points than fit into 8-bit encodings. Hence there are multiple ways to encode the Unicode code points

- variable-byte encodings use multiple bytes as needed. Advantage is efficiency, since most ASCII and simple extended character sets can use just one byte, and these were set in the Unicode standard to their ASCII and ISO-8859 equivalents

- two most common are UTF-8 and UTF-16, using 8 and 16 bits respectively
Doh! Byte Order Marks!

- Back to the big-endian v. little-endian issue: Where do you store the start of the data representation?

Example: UCS-2/UTF-16 (little-endian UCS-2 is the native format on Windows)

```
FF FE  4800  6500  6C00  6C00  6F00
header H e l l o
```

But in "big-endian":

```
FE FF  0048  0065  006C  006C  006F
header H e l l o
```
Doh! Byte Order Marks! (cont’d)

- Solution: A "byte-order mark" (U+FEFF) at the header of each file, indicating the byte order
  - FFFE for little-endian, FEFF for big-endian
- But this has a problem: The BOM is actually a valid Unicode character (http://www.fileformat.info/info/unicode/char/FEFF/index.htm)!
  - What if someone sent a file without a header, and that character was actually part of the file?
Solution: UTF-8

• One problem with fixed-length schemes (such as UTF-16) is that they are wasteful: they use two bytes even when this is unnecessary

• UTF-8 is designed to be more efficient: only as many bytes as you need

  EF BB BF 48 65 6C 6C 6F
  header   H e l l o

  • Code points 0 – 007F are stored as regular, single-byte ASCII
  • Code points 0080 and above are converted to binary and stored (encoded) in a series of bytes
Solution: UTF-8 (cont’d)

- UTF-8 is a variable-length encoding
  - The first “count” byte indicates the number of bytes for the codepoint, including the count byte. These bytes start with 11..0:
    - 110xxxxx (The leading “11” is indicates 2 bytes in sequence, including the “count” byte)
    - 1110xxxx (1110 -> 3 bytes in sequence)
    - 11110xxx (11110 -> 4 bytes in sequence)
  - Bytes starting with 10… are “data” bytes and contain information for the codepoint. A 2-byte example looks like this: 110xxxxx 10xxxxxx
    - This means there are 2 bytes in the sequence. The X’s represent the binary value of the codepoint, which needs to squeeze in the remaining bits

- Recommendation: Use UTF-8 (http://utf8everywhere.org)
Text encoding: Caution

- Input texts can be very different

- Many text production software (e.g. MS Office-based products) still tend to use proprietary formats, such as Windows-1252

- Windows tends to use UTF-16, while Mac and other Unix-based platforms use UTF-8

- Your eyes can be deceiving: a client may display gibberish but the encoding might still be as intended

- No easy method of detecting encodings (except in HTML meta-data)
A note on "meta-data"

- Data that provides information about other (primary) data
- Usually not meant to be analyzed as data itself

- Example: HTML

```html
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Metadata - Wikipedia</title>
</head>
</html>
```

- Example of a standard attempting to address this need: Dublin Core Metadata Initiative (http://dublincore.org/documents/dc-text/)
Representing dates: Different formats?

<table>
<thead>
<tr>
<th>Color</th>
<th>Order styles</th>
<th>Main regions and countries (approximate population of each region in millions)</th>
<th>Approximate population in millions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyan</td>
<td>DMY</td>
<td>Asia (Central, SE, West), Australia (25), New Zealand (5), parts of Europe (c. 640), Latin America (625), North Africa (195), India (1315), Indonesia (265), Bangladesh (165), Russia (145)</td>
<td>3565</td>
</tr>
<tr>
<td>Yellow</td>
<td>YMD</td>
<td>Bhutan, Canada (35), China (1385), Korea (75), Taiwan (24), Hungary (10), Iran (80), Japan (125), Lithuania (5), Mongolia (5). Known in other countries due to ISO 8601.</td>
<td>1745</td>
</tr>
<tr>
<td>Magenta</td>
<td>MDY</td>
<td>United States (325), Federated States of Micronesia, Marshall Islands</td>
<td>325</td>
</tr>
<tr>
<td>Red</td>
<td>DMY, MDY</td>
<td>Malaysia (35), Nigeria (190), Philippines (105), Saudi Arabia (35), Somalia (10)</td>
<td>380</td>
</tr>
<tr>
<td>Green</td>
<td>DMY, YMD</td>
<td>Afghanistan (28), Albania (3), Austria (9), Czech Republic (11), Kenya (49), Macau (1), Maldives, Montenegro, Namibia (2), Nepal (29), Singapore (6), South Africa (56), Sri Lanka (21), Sweden (10)</td>
<td>225</td>
</tr>
</tbody>
</table>
Representing dates: Different formats?
## Representing dates

<table>
<thead>
<tr>
<th>Description</th>
<th>Format</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>American month and day</td>
<td>mm/dd</td>
<td>&quot;5/12&quot;, &quot;10/27&quot;</td>
</tr>
<tr>
<td>American month, day and year</td>
<td>mm/dd/y</td>
<td>&quot;1/17/2006&quot;</td>
</tr>
<tr>
<td>Four digit year, month and day with slashes</td>
<td>YY/mm/dd</td>
<td>&quot;2008/6/30&quot;, &quot;1978/12/22&quot;</td>
</tr>
<tr>
<td>Four digit year and month (GNU)</td>
<td>YY-mm</td>
<td>&quot;2008-6&quot;, &quot;2008-06&quot;, &quot;1978-12&quot;</td>
</tr>
<tr>
<td>Year, month and day with dashes</td>
<td>y-mm-dd</td>
<td>&quot;2008-6-30&quot;, &quot;78-12-22&quot;, &quot;8-6-21&quot;</td>
</tr>
<tr>
<td>Day, month and four digit year</td>
<td>dd-mm-YY</td>
<td>&quot;30-6-2008&quot;</td>
</tr>
<tr>
<td>Day, month and two digit year</td>
<td>dd.mm.yy</td>
<td>&quot;30.6.08&quot;</td>
</tr>
<tr>
<td>Day, textual month and year</td>
<td>dd-m y</td>
<td>&quot;30-June 2008&quot;</td>
</tr>
<tr>
<td>Textual month and four digit year</td>
<td>m YY</td>
<td>&quot;June 2008&quot;, &quot;March 1879&quot;</td>
</tr>
<tr>
<td>Four digit year and textual month</td>
<td>YY m</td>
<td>&quot;2008 June&quot;</td>
</tr>
<tr>
<td>Textual month, day and year</td>
<td>m dd, y</td>
<td>&quot;April 17, 1790&quot;</td>
</tr>
<tr>
<td>Day and textual month</td>
<td>d m</td>
<td>&quot;1 July&quot;</td>
</tr>
<tr>
<td>Month abbreviation, day and year</td>
<td>M-DD-m</td>
<td>&quot;May-09-78&quot;, &quot;Apr-17-1790&quot;</td>
</tr>
<tr>
<td>Description</td>
<td>Format</td>
<td>Examples</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>Year, month abbreviation and day</td>
<td>y-M-DD</td>
<td>&quot;78-Dec-22&quot;, &quot;1814-MAY-17&quot;</td>
</tr>
<tr>
<td>Year (and just the year)</td>
<td>YY</td>
<td>&quot;1978&quot;, &quot;2008&quot;</td>
</tr>
<tr>
<td>Textual month (and just the month)</td>
<td>m</td>
<td>&quot;March&quot;, &quot;jun&quot;, &quot;DEC&quot;</td>
</tr>
</tbody>
</table>
ISO 8601: Imposing common standards

- Purpose: to provide unambiguous and well-defined method of representing dates and times
- Goal: to avoid misinterpretation of numeric representations of dates and times, particularly when data are transferred between countries with different conventions for writing numeric dates and time
- First published in 1988
- Introduces a common notation, and a common order (most-to-least-significant order [YYYY]-[MM]-[DD])
  - matches lexicographical order with chronological order
- Uses codes for date and time elements, to represent dates (and times) in either a basic format (no separators) or in an extended format with added separators (to enhance human readability)
### ISO8601 formatting components

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY</td>
<td>4-digit year</td>
<td>2017</td>
<td>Avoids the &quot;Y2K&quot; problem</td>
</tr>
<tr>
<td>MM</td>
<td>2-digit day of the month</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>DD</td>
<td>2-digit day of the month</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td>Www</td>
<td>Week number</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>Weekday number</td>
<td>2</td>
<td>Starts on Monday!</td>
</tr>
<tr>
<td>hh</td>
<td>hour (0-24)</td>
<td>10</td>
<td>24 is only used to denote midnight at the end of a calendar day</td>
</tr>
<tr>
<td>mm</td>
<td>minute (0-59)</td>
<td>05</td>
<td></td>
</tr>
<tr>
<td>ss</td>
<td>second (0-60)</td>
<td>20</td>
<td>60 is only used to denote an added leap second</td>
</tr>
</tbody>
</table>
Coordinated Universal Time (UTC)

- World standard for time
- Does not include Daylight Savings Time
- Interchangeable with Greenwich Mean Time (GMT), but GMT is no longer precisely defined by the scientific community
- Time zones around the world (https://en.wikipedia.org/wiki/List_of.UTC_time_offsets) are expressed using positive or negative offsets from UTC

- French v. English
  - English speakers originally proposed **CUT** (for "coordinated universal time")
  - French speakers proposed **TUC** (for "temps universel coordonné")
  - Compromise: **UTC**
POSIX Time

- a system for describing a point in time, defined as the number of seconds that have elapsed since 00:00:00 UTC, Thursday, 1 January 1970
- Also known as "Unix time", or "[epoch time](https://en.wikipedia.org/wiki/Epoch_(reference_date)#Computing)", because it represents elapsed time from a defined "epoch"
- Problem: How much elapsed time can you store?

- The Year 2038 Problem
  - Many Unix-like operating systems which keep time as seconds elapsed from the epoch date of January 1, 1970
  - For signed 32-bit integers, this means that cannot encoding times after 03:14:07 UTC on 19 January 2038
  - Times beyond that will wrap around and be stored internally as a negative number, which these systems will interpret as having occurred on 13 December 1901
  - A solution: 64-bit signed integers allow a new wraparound date that is 20x greater than the estimated age of the universe: approximately 290 billion years in the future
Compression

- Seeks to economize on space by representing recurring items using patterns that represent the uncompressed data
- Common in formats for graphics and video encoding
- "Lossless" formats compress data without reducing information - examples are .zip and .gz compression
- This (and avoiding errors) is also a principle in normalized relational data forms
- Also very important for sparse matrix representations, where many of the cells are zero, but it would be very wasteful to record a double precision numeric zero for each of these non-informative cells
Compression: Sparse matrix formats

- used because many forms of matrix are very sparse - for example, document-term matrixes, which are commonly 80-90% sparse

In [1]:

```r
suppressPackageStartupMessages(library("quanteda"))
mydfm <- dfm(data_corpus_inaugural)
mydfm
```

Document-feature matrix of: 58 documents, 9,357 features (91.8% sparse).

In [22]:

```r
head(mydfm, nfeature = 8)
```

Document-feature matrix of: 6 documents, 8 features (37.5% sparse).
6 x 8 sparse Matrix of class "dfmSparse"
features

docs      fellow-citizens of the senate and house representatives :
1789-Washington 1  71 116 1  48 2
1793-Washington 0  11  13 0  2 0
1797-Adams     3 140 163 1 130 0
1801-Jefferson 2 104 130 0  81 0
1805-Jefferson 0 101 143 0  93 0
1809-Madison   1  69 104 0  43 0
```
"simple triplet" format

- "simple triplet" format
  - $i$: indexes row
  - $j$: indexes column
  - $x$: indicates value

(indexes will be from zero)
"simple triplet" example

This matrix:

\[
\begin{array}{cccc}
[1,] & 1 & 99 & 3 & 12 \\
[2,] & 0 & 0 & 10 & 1 \\
[3,] & 2 & 0 & 0 & 0 \\
\end{array}
\]

Would be represented by:

- \(i: 0 2 0 0 1 0 1\)
- \(j: 0 0 1 2 2 3 3\)
- \(x: 1 2 9 9 3 1 0 1 2 1\)
In [13]:
# in R

    m <- matrix(c(1, 0, 2, 99, 0, 0, 3, 10, 0, 12, 1, 0), nrow = 3)
    print(m)

[1,]   1   99    3  12
[2,]   0   0  10   1
[3,]

In [14]:
library("Matrix")
    m_st <- as(m, "dgTMatrix")
    print(m_st@i)
    print(m_st@j)
    print(m_st@x)

    [1] 0 2 0 0 1 0 1
[1] 0 0 1 2 2 3 3
[1] 1 2 99 3 10 12 1
"compressed sparse column" format

- More efficient than the STF
  - \( i \): indexes row
  - \( p \): indexes the first nonzero element in each column of the matrix
  - \( x \): indicates value

- "compressed sparse row" format is also possible
**“compressed sparse column” example**

This matrix:

\[
\begin{bmatrix}
[1,] & 1 & 99 & 3 & 12 \\
[2,] & 0 & 0 & 10 & 1 \\
[3,] & 2 & 0 & 0 & 0
\end{bmatrix}
\]

Would be represented by:

- \(i: 0200101\)
- \(p: 02357\)
- \(x: 1299310121\)
In [15]: # in R

    m_sc <- as(m, "dgCMatrix")
    print(m_sc$i)
    print(m_sc$p)
    print(m_sc$x)

    [1] 0 2 0 0 1 0 1
    [1] 0 2 3 5 7
    [1]  1  2 99  3 10 12  1
**Dataset manipulation**

**What is a "Dataset"?**

- A dataset is a “rectangular” formatted table of data in which all the values of the same variable must be in a single column.

- A dataset is not:
  - the results of tabulating a dataset
  - any set of summary statistics on a dataset
  - a series of relational tables

- Many of the datasets we use have been artificially reshaped in order to fulfill this criterion of rectangularity.
  - This means "non-normalized" data
  - Often confounds variables with their values
The difference between a dataset and table

- This is a *table*:

<table>
<thead>
<tr>
<th>Lost</th>
<th>Won</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenger</td>
<td>266 60</td>
</tr>
<tr>
<td>Incumbent</td>
<td>32 106</td>
</tr>
</tbody>
</table>

- This is a (partial) dataset:

<table>
<thead>
<tr>
<th>district</th>
<th>incumbf</th>
<th>wonseatf</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Carlow Kilkenny Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>2</td>
<td>Carlow Kilkenny Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>5</td>
<td>Carlow Kilkenny Incumbent</td>
<td>Won</td>
</tr>
<tr>
<td>100</td>
<td>Donegal South West Challenger</td>
<td>Lost</td>
</tr>
<tr>
<td>459</td>
<td>Wicklow Incumbent</td>
<td>Won</td>
</tr>
<tr>
<td>464</td>
<td>Wicklow Challenger</td>
<td>Lost</td>
</tr>
</tbody>
</table>
Hadley Wickham’s three rules for "tidy" datasets

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.
Example from *R for Data Science*

- Example of non-tidy data

```r
# In [4]:
suppressPackageStartupMessages(library("tidyverse"))
print(table4a)

# A tibble: 3 x 3
  country `1999` `2000`
  *   <chr>  <int>  <int>
1  Afghanistan    745   2666
2      Brazil  37737  80488
3       China 212258 213766
```
In [5]: print(gather(table4a, `1999`, `2000`, key = "year", value = "cases"))

# A tibble: 6 x 3
   country    year cases
   <chr>      <chr> <int>
1 Afghanistan 1999    745
2      Brazil 1999  37737
3       China 1999 212258
4 Afghanistan 2000   2666
5      Brazil 2000  80488
6       China 2000 213766

<table>
<thead>
<tr>
<th>country</th>
<th>year</th>
<th>cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Afghanistan</td>
<td>1999</td>
<td>745</td>
</tr>
<tr>
<td>Afghanistan</td>
<td>2000</td>
<td>2666</td>
</tr>
<tr>
<td>Brazil</td>
<td>1999</td>
<td>37737</td>
</tr>
<tr>
<td>Brazil</td>
<td>2000</td>
<td>80488</td>
</tr>
<tr>
<td>China</td>
<td>1999</td>
<td>212258</td>
</tr>
<tr>
<td>China</td>
<td>2000</td>
<td>213766</td>
</tr>
</tbody>
</table>
"Spreading"

In [6]: print(table2)

```r
# A tibble: 12 x 4
  country       year  type      count
     <chr> <int> <chr>    <int>
 1 Afghanistan  1999 cases      745
 2 Afghanistan  1999 population 19987071
 3 Afghanistan  2000 cases      2666
 4 Afghanistan  2000 population 20595360
 5      Brazil  1999 cases   37737
 6      Brazil  1999 population 172006362
 7      Brazil  2000 cases   80488
 8      Brazil  2000 population 174504898
 9       China  1999 cases  212258
10       China  1999 population 1272915272
11       China  2000 cases  213766
12       China  2000 population 1280428583
```
In [7]: spread(table2, key = type, value = count) %>% print()

# A tibble: 6 x 4
  country year cases population
  <chr> <int> <int>      <int>
1 Afghanistan 1999    745   19987071
2 Afghanistan 2000   2666   20595360
3      Brazil 1999  37737  172006362
4        Brazil 2000 80488  174504898
5       China 1999 212258 1272915272
6       China 2000 213766 1280428583
Long v. wide formats (R)

- More fine-grained control with functions and packages for this (alternative to \texttt{dplyr} functions of \texttt{spread} \texttt{gather}, etc.)

- \texttt{base::reshape()}  
  - the “old” R way to do this, using ‘base::reshape()‘  
  - problem: confusing and difficult to use

- \texttt{reshape2} package  
  - Hadley Wickham’s \texttt{reshape2} package  
  - data is first ‘melt’ed into long format  
  - then ‘cast’ into desired format
Example with Manifesto Data (https://manifesto-project.wzb.eu).
In [38]:

cmpdataLong <- melt(cmpdata, 
    id.vars = c("countryname", "party", "date"), 
    measure.vars = names(cmpdata)[21:76], 
    variable.name = "category", 
    value.name = "catcount")

head(cmpdataLong, 30)

countryname  party  date    category  catcount
Austria     42420  199010  per101      0
Austria     42110  199010  per101      0
Austria     42320  199010  per101      0
Austria     42520  199010  per101      5
Austria     42420  199410  per101      0
Austria     42421  199410  per101      0
Austria     42110  199410  per101      0
Austria     42320  199410  per101      0
Austria     42520  199410  per101      0
Austria     42110  199512  per101      0
Austria     42320  199512  per101      0
Austria     42520  199512  per101      0
Austria     42110  199512  per101      0
Austria     42320  199512  per101      0
Austria     42520  199512  per101      0
Austria     42110  199512  per101      0
Austria     42320  199512  per101      0
Austria     42420  199512  per101      0
Austria     42421  199512  per101      0
<table>
<thead>
<tr>
<th>countryname</th>
<th>party</th>
<th>date</th>
<th>category</th>
<th>catcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Austria</td>
<td>42420</td>
<td>199910</td>
<td>per101</td>
<td>1</td>
</tr>
<tr>
<td>Austria</td>
<td>42110</td>
<td>199910</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42320</td>
<td>199910</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42520</td>
<td>199910</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42520</td>
<td>200211</td>
<td>per101</td>
<td>7</td>
</tr>
<tr>
<td>Austria</td>
<td>42320</td>
<td>200211</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42110</td>
<td>200211</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42220</td>
<td>200211</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42420</td>
<td>200211</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42110</td>
<td>200610</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42710</td>
<td>200610</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42420</td>
<td>200610</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42520</td>
<td>200610</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42320</td>
<td>200610</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42320</td>
<td>200809</td>
<td>per101</td>
<td>0</td>
</tr>
<tr>
<td>Austria</td>
<td>42110</td>
<td>200809</td>
<td>per101</td>
<td>0</td>
</tr>
</tbody>
</table>
In [29]: tail(cmpdataLong, 30)

<table>
<thead>
<tr>
<th>countryname</th>
<th>party</th>
<th>date</th>
<th>category</th>
<th>catcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>39675</td>
<td>Switzerland</td>
<td>43530</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39676</td>
<td>Switzerland</td>
<td>43420</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39677</td>
<td>Switzerland</td>
<td>43810</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39678</td>
<td>Switzerland</td>
<td>43110</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39679</td>
<td>Switzerland</td>
<td>43951</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39680</td>
<td>Switzerland</td>
<td>43320</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39681</td>
<td>Switzerland</td>
<td>43710</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39682</td>
<td>Switzerland</td>
<td>43520</td>
<td>199110</td>
<td>per706</td>
</tr>
<tr>
<td>39683</td>
<td>Switzerland</td>
<td>43321</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39684</td>
<td>Switzerland</td>
<td>43110</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39685</td>
<td>Switzerland</td>
<td>43810</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39686</td>
<td>Switzerland</td>
<td>43320</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39687</td>
<td>Switzerland</td>
<td>43520</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39688</td>
<td>Switzerland</td>
<td>43530</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39689</td>
<td>Switzerland</td>
<td>43420</td>
<td>199510</td>
<td>per706</td>
</tr>
<tr>
<td>39690</td>
<td>Switzerland</td>
<td>43110</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>39691</td>
<td>Switzerland</td>
<td>43520</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>countryname</td>
<td>party</td>
<td>date</td>
<td>category</td>
<td>catcount</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>39692</td>
<td>Switzerland</td>
<td>43320</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>39693</td>
<td>Switzerland</td>
<td>43810</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>39694</td>
<td>Switzerland</td>
<td>43420</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>39695</td>
<td>Switzerland</td>
<td>43530</td>
<td>199910</td>
<td>per706</td>
</tr>
<tr>
<td>39696</td>
<td>Switzerland</td>
<td>43320</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39697</td>
<td>Switzerland</td>
<td>43530</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39698</td>
<td>Switzerland</td>
<td>43110</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39699</td>
<td>Switzerland</td>
<td>43520</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39700</td>
<td>Switzerland</td>
<td>43711</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39701</td>
<td>Switzerland</td>
<td>43810</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39702</td>
<td>Switzerland</td>
<td>43420</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39703</td>
<td>Switzerland</td>
<td>43531</td>
<td>200310</td>
<td>per706</td>
</tr>
<tr>
<td>39704</td>
<td>Switzerland</td>
<td>43220</td>
<td>200310</td>
<td>per706</td>
</tr>
</tbody>
</table>
```r
In [34]:
cmpdataLong <- mutate(cmpdataLong, year = floor(date / 100))
cmpdataLong %>%
  group_by(category, year) %>%
  summarize(total = sum(catcount)) %>%
  filter(category %in% c("per108", "per110")) %>%
  ggplot(aes(x = year, y = total, colour = category, group = category)) + geom_line()
```