The GovData Project
MIT-Harvard Winter Course 2011

Module 2:
Data Wrangling Techniques
Data Wrangling Outline

Three easy pieces:

1. Scraping & Parsing Tools & Techniques
2. Data “Cleaning” Tools & Techniques
3. A bit of database technology (MongoDB primer)
Data Wrangling Outline

Motivations:

1. Scraping & Parsing Tools & Techniques
   
   because the web, especially complex data portals, contains lots of data

2. Data “Cleaning” Tools & Techniques
   
   because the data, even coming from DB-backed sites, is often “dirty”

3. A bit of database technology (MongoDB primer)
   
   because you want to be able to serve up the data too
Data Wrangling Outline

Three easy pieces:

\textit{prelude: How the web works: request / response}

1. Scraping

\textit{interlude: How the web works: HTML}

... and then Parsing Tools & Techniques

2. Data “Cleaning” Tools & Techniques

3. A bit of database technology (MongoDB primer)
How the Web Works (sort of)

SERVER

A (powerful) computer that hosts a webpage

CLIENT

Your computer, where you view the webpage
How the Web Works (sort of)

(the web)
How the Web Works (sort of)

(the web)
How the Web Works (sort of)

How the Web Works (sort of)

How the Web Works (sort of)
How the Web Works (sort of)

(the web)
How the Web Works (sort of)
How the Web Works (sort of)

http://thissite.com/thispage
How the Web Works (sort of)

[Diagram of client-server network with various nodes labeled as 'CLIENT' and 'SERVER', and arrows indicating data flow. The diagram illustrates the complex interactions between clients and servers, highlighting the dynamic nature of web traffic.]
How the Web Works (sort of)

Key fact: all the servers know all the other servers’ address, and know how to forward on the message properly
How the Web Works (sort of)

http://thissite.com/thispage

GET me: thissite.com/thispage
How the Web Works (sort of)
How the Web Works (sort of)
Server computes the response.

```
thissite.com/thispage.html
```

Simple web page = static = little computation
Complex page = dynamic (e.g. DB-backed) = more computation
How the Web Works (sort of)

A typical request/response pair looks like:

Request URL: https://mail.google.com/mail/?shva=1#inbox
Request Method: GET
Status Code: 200 OK

Request Headers
- Accept: application/xml, application/xhtml+xml, text/html; q=0.9, text/plain; q=0.8,
- Accept-Charset: ISO-8859-1, utf-8; q=0.7, *; q=0.3
- Accept-Encoding: gzip, deflate, sdch
- Accept-Language: en-US, en;q=0.8
- Cookie: Tz=380; GMsf_1AT=A66bu0OGIP2Td2d8vNqJ3v6Y6RBx7Zrxw; gmac=5x6qhh68b8c-f85drl_aZPcRreuDVfUCVcUQ8rFxbmUHVAMW0tHx6ahQCM_78RLhQ0YmhHKvrm0LajqEzXab3jaZFr-Y; PREF-Id=9f85d408194c6fab:u-a9958d6b946b47ba:FF=4; LD=en:NR=10:TM=129421912.1294281932.1294281932.1;
- User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_5; en-US) AppleWebKit/533.18.5 (KHTML, like Gecko) Version/5.1.7 Safari/533.18.5

Response Headers
- cache-control: no-cache, no-store, max-age=0, must-revalidate
- content-encoding: gzip
- content-type: text/html; charset=UTF-8
- date: Sat, 08 Jan 2011 16:39:33 GMT
- expires: Fri, 01 Jan 1990 00:00:00 GMT
- pragma: no-cache
- server: GSE
- status: 200 OK
- version: HTTP/1.1
- x-content-type-options: nosniff
- x-dns-prefetch-control: off
- x-frame-options: SAMEORIGIN
- x-xss-protection: 1; mode=block
How the Web Works (sort of)

The real contents of the response.

It’s HTML.
Your web browser renders HTML into something meaningful.
Data Wrangling Outline Revisited

Three easy pieces:

1. Scraping & Parsing Tools & Techniques
   - issue the right requests / transform resulting HTML into a data structure more suited to analytical manipulation

2. Data “Cleaning” Tools & Techniques
   - correct and enrich the data structure

3. A bit of database technology (MongoDB primer)
   - repackage the data structure
   - make it available to others just it was made available to you (but better)
Scraping

The Idea Of Scraping:
Issue a GET request not through the browser, but instead through some other route, so as to be able to direct the response to your analyze its contents & extract its structured information (as opposed to having it rendered in the browser window).

Sub-issues of Scraping:
- How to issue the request
- How to figure out which requests to issue in the first place
- How to extract (that is, parse) data from the response into a useable data structure.
Scraping

How to issue the request:

Command Line Tools

- wget
- curl
- python urllib, urllib2
- mechanize
- selenium

GUI Web scrapers

More like programming

More like browsing.
You can integrate it into your python scripts trivially:

```python
In [1]: import os

In [2]: os.system("wget http://www.nytimes.com")
```

```bash
Resolving www.nytimes.com... 199.239.136.200
Connecting to www.nytimes.com|199.239.136.200|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: `index.html'

[ <=> ] 125,392 125,392 --.-K/s in 0.1s

2011-01-08 13:08:34 (1.05 MB/s) - `index.html' saved [125392]
```
Scraping: wget / curl
Scraping: wget / curl
Scraping: `wget` / `curl`

`wget` has all kinds of options, for recursively getting many subpages of a page, following links in different ways, using passwords with secure pages, controlling how input is named, configuring response headers, &c

`curl` is basically the same
Scraping: `wget / curl`

NYTimes was a pretty simple example. Some are harder.
Scraping: `wget / curl`

NYTimes was a pretty simple example. Some are harder.

The resulting page just doesn’t have the stuff in it.

But it had to have gotten to your computer *somehow*.

Time for Firebug.
Scraping: `wget` / `curl`

NYTimes was a pretty simple example. Some are harder.
Scraping: \texttt{wget} / \texttt{curl}

NYTimes was a pretty simple example. Some are harder.
Scraping: `wget / curl`

NYTimes was a pretty simple example. Some are harder.
Scraping: `wget / curl`

NYTimes was a pretty simple example. Some are harder.

```bash
wget
--post-data="{"searchQueryString":"p+1-n+12-c+287458-s+5-r+101323338-t+ri++ni+1-x+"},"isSearchMode":false"
--no-cache
--max-redirect=0
--header="Content-Type: application/json"
http://www.tiffany.com/Shopping/CategoryBrowse.aspx/GetCategoriesXmlBySearchQS
-O "test.xml"
```

http://www.tiffany.com/shared/media/products/26598044_l_over_M_3.jpg
## Scrapping: `wget / curl`

<table>
<thead>
<tr>
<th>Categories</th>
<th>Level1</th>
<th>Level2</th>
<th>Image</th>
<th>Item</th>
<th>Price</th>
<th>sku</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Metro five-row ring of diamonds in 18k white gold.</td>
<td>$3,750</td>
<td>GRP03613</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Locks heart pendant in sterling silver, mini.</td>
<td>$100</td>
<td>26914973</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Return to Tiffany™ mini heart tag in sterling silver on a bead bracelet.</td>
<td>$125</td>
<td>GRP03577</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Return to Tiffany™ double heart tag pendant in silver with Tiffany Blue®.</td>
<td>$100</td>
<td>27125107</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Locks vintage charm in sterling silver and 18k rose gold on a chain.</td>
<td>$225</td>
<td>GRP03791</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Locks heart pendant in 18k rose gold, mini.</td>
<td>$300</td>
<td>26914663</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Petals cuff in sterling silver and 18k rose gold with diamonds, medium.</td>
<td>$1,700</td>
<td>GRP03892</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Paloma Picasso® Love &amp; Kisses ring with diamonds in 18k white gold, narrow.</td>
<td>$1,250</td>
<td>GRP03859</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Keys twist heart key pendant in 18k gold on a chain.</td>
<td>$750</td>
<td>GRP03737</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Keys heart key charm with enamel finish in sterling silver on a chain.</td>
<td>$175</td>
<td>GRP03674</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Return to Tiffany™ heart lock, peace tag and Tiffany 1837™ lock bead bracelets.</td>
<td>$375</td>
<td>GRP03766</td>
</tr>
<tr>
<td>Jewelry</td>
<td>Jewelry</td>
<td>New Jewelry</td>
<td>1</td>
<td>Tiffany Locks vintage charm in sterling silver on a chain.</td>
<td>$130</td>
<td>GRP03790</td>
</tr>
</tbody>
</table>

*Note: Images are not included in the text representation.*
<table>
<thead>
<tr>
<th>Category</th>
<th>Price Range</th>
<th>Quantity</th>
<th>Description</th>
<th>Price</th>
<th>Item Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jewelry</td>
<td>$500 &amp; Under</td>
<td>1</td>
<td>Paloma Picasso® Love pendant in 18k rose gold, mini.</td>
<td>$500</td>
<td>26230597</td>
</tr>
<tr>
<td>Jewelry</td>
<td>$500 &amp; Under</td>
<td>1</td>
<td>Tiffany Sparklers peridot earrings in sterling silver.</td>
<td>$500</td>
<td>23259478</td>
</tr>
</tbody>
</table>
Scraping: `wget` / `curl`

Python’s `urllib` and `urllib2` are pure-python libraries for doing similar things.

```python
In [1]: import urllib

In [2]: fh=urllib.urlopen("http://www.nytimes.com")

In [3]: html = fh.read()

In [4]: html[:100]
Out[4]: '

In [5]:
```
So let’s look e.g. at BEA. Navigate our way thru to:

What happens when we click on “Download All Years”. Something downloads. From where?
Scraping: `wget / curl`

Let's look at the source:
Scraping: `wget / curl`

All right, let’s try it:

```
[1] 17918
[2] 17919
[3] 17920

Resolving www.bea.gov... 198.76.170.20
Connecting to www.bea.gov|198.76.170.20|:80... connected.
HTTP request sent, awaiting response... 302 Object moved
Location: SvrIsBusy.asp [following]

Reusing existing connection to www.bea.gov:80.
HTTP request sent, awaiting response... -bash: -Do: command not found
[2]-  Done  FirstYear=1900
[3]+  Done  LastYear=2011
```

Saving to: `SvrIsBusy.asp.1`

```
100%[--------------------------] 8898 (8.7K) [text/html]
```

Stuck on this ... Go to Firebug and look for something fancy? NO!
We forgot you use ""s.
### Scraping: `wget / curl`

OK!

<table>
<thead>
<tr>
<th>Line</th>
<th>Table 1.1.1. Percent Change From Preceding Period in Real Gross Domestic Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Seasonally adjusted at annual rates</td>
</tr>
<tr>
<td>2</td>
<td>Bureau of Economic Analysis</td>
</tr>
<tr>
<td>3</td>
<td>Downloaded on 1/8/2011 At 2:27:14 PM Last Revised December 22, 2010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>-0.6</td>
<td>-0.3</td>
<td>6.2</td>
<td>6.5</td>
<td>7.6</td>
<td>2.2</td>
<td>0.6</td>
<td>0.6</td>
<td>-5.4</td>
<td>-1.4</td>
<td>4.8</td>
<td>-3.7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>6.8</td>
<td>1.3</td>
<td>0.1</td>
<td>2.0</td>
<td>4.7</td>
<td>0.5</td>
<td>3.2</td>
<td>0.6</td>
<td>6.3</td>
<td>0.9</td>
<td>6.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7.4</td>
<td>2.7</td>
<td>1.5</td>
<td>0.6</td>
<td>3.7</td>
<td>-0.4</td>
<td>2.7</td>
<td>-0.6</td>
<td>9.1</td>
<td>2.0</td>
<td>9.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>8.6</td>
<td>5.7</td>
<td>28.7</td>
<td>-2.1</td>
<td>2.8</td>
<td>6.9</td>
<td>-5.5</td>
<td>-7.1</td>
<td>39.5</td>
<td>23.5</td>
<td>16.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>7.0</td>
<td>1.9</td>
<td>-5.1</td>
<td>1.3</td>
<td>4.0</td>
<td>-2.4</td>
<td>5.3</td>
<td>1.5</td>
<td>1.1</td>
<td>-3.5</td>
<td>6.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5.9</td>
<td>-0.9</td>
<td>-2.0</td>
<td>4.3</td>
<td>6.4</td>
<td>2.2</td>
<td>4.1</td>
<td>2.6</td>
<td>2.1</td>
<td>-3.8</td>
<td>1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>-28.5</td>
<td>-12.2</td>
<td>112.3</td>
<td>50.9</td>
<td>29.1</td>
<td>4.4</td>
<td>-18.5</td>
<td>-45.1</td>
<td>-45.3</td>
<td>39.9</td>
<td>-20.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>-10.2</td>
<td>14.3</td>
<td>40.1</td>
<td>14.2</td>
<td>-0.4</td>
<td>-4.2</td>
<td>-4.6</td>
<td>-12.0</td>
<td>-12.1</td>
<td>-1.0</td>
<td>15.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-6.2</td>
<td>-7.7</td>
<td>13.5</td>
<td>27.2</td>
<td>-13.2</td>
<td>1.0</td>
<td>10.7</td>
<td>-17.6</td>
<td>-14.9</td>
<td>-13.4</td>
<td>-6.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-2.1</td>
<td>2.7</td>
<td>-5.6</td>
<td>8.6</td>
<td>11.6</td>
<td>12.5</td>
<td>5.9</td>
<td>-7.3</td>
<td>-9.0</td>
<td>-15.6</td>
<td>-10.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>-8.2</td>
<td>-12.8</td>
<td>25.4</td>
<td>38.2</td>
<td>-24.0</td>
<td>-5.0</td>
<td>13.5</td>
<td>-22.9</td>
<td>-18.1</td>
<td>-19.4</td>
<td>-4.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>-18.5</td>
<td>76.9</td>
<td>102.7</td>
<td>-4.7</td>
<td>25.5</td>
<td>-12.2</td>
<td>-26.5</td>
<td>-26.8</td>
<td>-6.8</td>
<td>38.9</td>
<td>59.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>-4.9</td>
<td>-18.8</td>
<td>-38.2</td>
<td>-17.0</td>
<td>-32.5</td>
<td>9.3</td>
<td>-13.4</td>
<td>57.2</td>
<td>-5.4</td>
<td>-23.9</td>
<td>-43.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>-20.3</td>
<td>-41.0</td>
<td>-16.6</td>
<td>-34.4</td>
<td>10.7</td>
<td>-15.0</td>
<td>64.0</td>
<td>-6.6</td>
<td>-29.3</td>
<td>-44.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>38.4</td>
<td>-8.5</td>
<td>-17.1</td>
<td>-25.5</td>
<td>-18.7</td>
<td>4.0</td>
<td>-3.5</td>
<td>23.4</td>
<td>10.2</td>
<td>-35.3</td>
<td>-40.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>8.0</td>
<td>-38.2</td>
<td>23.5</td>
<td>61.7</td>
<td>16.8</td>
<td>22.6</td>
<td>-6.3</td>
<td>-11.4</td>
<td>-5.7</td>
<td>-15.4</td>
<td>6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>9.5</td>
<td>-43.9</td>
<td>5.3</td>
<td>88.4</td>
<td>16.7</td>
<td>8.8</td>
<td>-1.3</td>
<td>-10.3</td>
<td>-12.3</td>
<td>-13.6</td>
<td>13.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>3.4</td>
<td>-14.8</td>
<td>97.8</td>
<td>5.7</td>
<td>17.4</td>
<td>73.9</td>
<td>-19.5</td>
<td>-15.1</td>
<td>15.7</td>
<td>-25.7</td>
<td>-13.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>2.6</td>
<td>3.7</td>
<td>-5.8</td>
<td>8.2</td>
<td>20.7</td>
<td>8.0</td>
<td>17.9</td>
<td>11.9</td>
<td>19.4</td>
<td>4.1</td>
<td>-4.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0.8</td>
<td>0.2</td>
<td>-13.2</td>
<td>14.3</td>
<td>28.7</td>
<td>8.4</td>
<td>23.5</td>
<td>6.3</td>
<td>15.9</td>
<td>-4.3</td>
<td>-18.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>-23.7</td>
<td>22.3</td>
<td>2.8</td>
<td>-1.1</td>
<td>-3.3</td>
<td>-5.9</td>
<td>21.5</td>
<td>11.0</td>
<td>7.5</td>
<td>-1.1</td>
<td>-22.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>231.8</td>
<td>-60.0</td>
<td>-667.1</td>
<td>106.4</td>
<td>222.7</td>
<td>62.9</td>
<td>29.1</td>
<td>1.6</td>
<td>40.9</td>
<td>-11.7</td>
<td>-5.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>5.5</td>
<td>9.7</td>
<td>6.6</td>
<td>-0.6</td>
<td>12.0</td>
<td>7.3</td>
<td>10.0</td>
<td>17.6</td>
<td>24.8</td>
<td>17.7</td>
<td>7.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>5.6</td>
<td>7.0</td>
<td>17.1</td>
<td>9.7</td>
<td>11.1</td>
<td>10.1</td>
<td>1.9</td>
<td>-7.3</td>
<td>-5.4</td>
<td>2.2</td>
<td>-3.8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GovData MongoDB Metadata**

Scraping: `wget / curl`
Scraping: `wget` / `curl`

How to figure out which requests to get in the first place?

`<-- Scrape this page first.`
How to figure out which requests to get in the first place?

<-- Scrape this page first.

The key point is that there’s a simple discoverable correspondence between the data on this page, and the urls of the actual data files.
How to figure out which requests to get in the first place?

<-- Scrape this page first.

In fact, there’s more to be gleaned as well, like metadata, stuff that may not be obvious in the actual data file.
Scraping: wget / curl

So your parser program goes something like this:

1) First wget and parse the “top page” (I’ll get to how you parse it later.)

2) Use the results of 1) to wget a whole bunch more pages and then clean those.

We’ll get to you how you deal with the parsing step in a bit ...
Scraping: mechanize

First, let’s try another one!

**U.S. direct investment abroad**
- Balance of payments and Direct Investment position data (Selected Tables, Interactive Tables)
- Financial and operating data (Selected Tables, Interactive Tables)
- Product Guide for U.S. Direct Investment Abroad

**Foreign direct investment in the U.S.**
- Balance of payments and Direct Investment position data (Selected Tables, Interactive Tables)
- Financial and operating data (Selected Tables, Interactive Tables)
- Product Guide for Foreign Direct Investment in the U.S.

**Articles and presentations**
- U.S. Direct Investment Abroad
- Foreign Direct Investment in the U.S.
- Globalization

From:

http://bea.gov/international/index.htm

E.g. the same portal, different data set.
Scraping: mechanize

Here, the first screen you encounter is this:

And you haven’t got a “list.” but notice the URL:

http://bea.gov/international/ii_web/timeseries2.cfm?econtypeid=1&dirlevel1id=1&Entitytypeid=1&stepnum=1
Scraping: mechanize

Our choices are being recorded by the data set, along the way:
Scraping: mechanize

Eventually we get to:

Looking at the source and watching firebug, we see the url of interest is

http://bea.gov/international/ii_web/timeseries7-2.cfm
Scraping: **mechanize**

And sure enough, if we wget this url and send the right data along with this URL, we get the final page.

But there’s a problem.  **How to figure out the right values of these things to send, to get a comprehensive dataset?**

Well, one could try parsing down the hierarchy.  But this is quite annoying, because the hierarchy has different numbers of levels in different branches.

Enter **mechanize**.
Scraping: mechanize

Mechanize is a Python library that allows you to *(sort of)* load a page as if you were browsing it, find and “click” on buttons in forms, etc... *(It’s like a browser w/o JS interpreter.)*

So we can “browse” this page programmatically.
Scraping: mechanize

Load up a browser:

```
In [1]: from mechanize import Browser

In [2]: br = Browser()

In [3]: br.open("http://bea.gov/international/indid=1&dirlevelid=1&Entitytypeid=1&stepnum=1")
Out[3]: <response_seek_wrapper at 0x10263a368 whose response at 0x102639dd0 whose fp = <socket._fileobject object at 0x10263a9c0>>
```
Scraping: mechanize

See what forms it finds:

```
In [8]: print(list(br.forms()))
[<mechanize._form.HTMLForm instance at 0x10263ed40>, <mechanize._form.HTMLForm instance at 0x10263ee18>]
```

Select that form; see it’s the right one:

```
In [10]: br.select_form(nr=1)
In [11]: print(br)
<Browser visiting http://bea.gov/international/ii_web/timeseries2.cfm?ecolid=1&dirlevellid=1&Entitytypeid=1&stepnum=1
   selected form:
   <step3 POST http://bea.gov/international/ii_web/timeseries3.cfm multipart-data
   <RadioControl(seriesid=[30, 31, 32, 25, 33, 27, 34, 39, 35, 40])>
   <HiddenControl(ecotypeid=1) (readonly)>
   <HiddenControl(dirlevellid=1) (readonly)>
   <HiddenControl(entitytypeid=1) (readonly)>
   <HiddenControl(stepnum=2) (readonly)>>
```
Scraping: mechanize

Select the control for the form:

```python
In [12]: control = br.find_control(predicate=lambda x: x.name == 'seriesid' and not x.readonly)
In [13]: control
Out[13]: <mechanize._form.RadioControl instance at 0x10263ee60>
In [14]: print(control)
<RadioControl(seriesid=[30, 31, 32, 25, 33, 27, 34, 39, 35, 40])>
```
Scraping: mechanize

Here’s where all the control data is buried:

```python
In [15]: control.items
Out[15]:
[<Item name='30' id='seriesid30' value='30' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid30'>,
 <Item name='31' id='seriesid31' value='31' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid31'>,
 <Item name='32' id='seriesid32' value='32' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid32'>,
 <Item name='25' id='seriesid25' value='25' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid25'>,
 <Item name='33' id='seriesid33' value='33' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid33'>,
 <Item name='27' id='seriesid27' value='27' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid27'>,
 <Item name='34' id='seriesid34' value='34' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid34'>,
 <Item name='39' id='seriesid39' value='39' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid39'>,
 <Item name='35' id='seriesid35' value='35' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid35'>,
 <Item name='40' id='seriesid40' value='40' type='radio' name='seriesid' check='document.step3.submit()' id='seriesid40'>]
```
Scraping: mechanize

Select the value:

```
In [16]: br['seriesid'] = [control.items[0].attrs['value']]
```

```
In [17]: br['seriesid']
Out[17]: ['30']
```

“Click” the button:

```
In [18]: br.submit()
Out[18]: <response_seek_wrapper at 0x10263a320 whose wrapped object = <closeable
le_response at 0x1026c9128 whose fp = <socket._fileobject object at 0x1026b630>>
```

Capture the result:

```
In [36]: resp = br.response().read()
```

```
In [37]: open
KeyboardInterrupt
```

```
In [37]: F = open('test.html','w')
F.
In [38]: F.write(resp)
```

```
In [39]: F.close()
```
Scraping: mechanize

OK!

Balance of Payments and Direct Investment Position Data

U.S. Direct Investment Abroad, U.S. Direct Investment Position Abroad on a Historical

Step 2. Choose Classification:
- Aggregate Totals Only
- By Industry of Affiliate Only (All Industries)
- By Country Only (Major Countries)
- By Country Only (All Countries)
- By Country and Industry

Continue

Back

Repeat.
Scraping: selenium

Sometimes you have to execute the site’s Javascript code to figure out what all the URLs you need are.

Get started with Selenium!

0. Watch: see the magic.

1. Begin: write and run tests in Firefox.
   Selenium IDE is a Firefox add-on that records clicks, typing, and other actions to make a test, which you can play back in the browser.
   Download Selenium IDE  Learn more  Screenshots

2. Customize: your language, your browser.
   Selenium Remote Control (RC) runs your tests in multiple browsers and platforms. Tweak your tests in your preferred language.
   Download Selenium RC  Learn more

3. Deploy: scale out, speed up.
   Selenium Grid extends Selenium RC to distribute your tests across multiple servers, saving you time by running tests in parallel.
Scraping

Sub-issues of Scraping:

- How to issue the request
  -- watch in the browser debug mode
  -- then wget (curl /urllib)

- How to figure out which requests to issue in the first place
  -- parse the tree page hierarchy to get metadata and links
  then wget in a loop

  -- or, write a mechanize spider

- Now: How to extract (that is, parse) data from the response into a useable data structure.
Scrapping: Parsing

How to extract (that is, parse) data from the response into a useable data structure.

It depends on the kind of response. If it’s HTML:

BeautifulSoup
lxml
python’s native xml lib
pyquery

If it’s excel:  xlrdf

If it’s csv: well, then you’re sort of done with the “parsing” phase and read for cleaning.
<html>
  <head>
  
  </head>
  <body>
    
    <div class="box" id="main">
      <p>Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    
    <div class="box" id="footer">
      <p>More text</p>
    </div>
    
  </body>
</html>
tags : basic part-types of a page  ex: <a>, <div>, <p> ;

How the Web Works: HTML

```html
<html>
  <head>
  </head>
  <body>
    <div class="box" id="main">
      <p>Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    <div class="box" id="footer">
      <p>More text</p>
    </div>
  </body>
</html>
```
attributes: information about part  ex: class, id
How the Web Works: HTML

Basic syntax of HTML:

```html
<tag attr1="val1" attr2="val2">
  <tag2 attr="val3">
    Contents
  </tag2>
</tag>
```

- nested structure; one tag object contains many others

- `<tag>` always (should) “close” with `<tag>`

- attributes are put in at the beginning of the tag using name = “value” syntax
<html>
  <head>
  
  </head>
  <body>
    <div class="box" id="main">
      <p>Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    
    <div class="box" id="footer">
      <p>More text</p>
    </div>
  </body>
</html>
How the Web Works: HTML

Some basic tag types:

- `<html>`: begin / end the whole page
- `<head>`: the page header
- `<body>`: the page body
- `<div>`, `<span>`: basic “container” types
- `<a>`: a hyper link
- `<p>`: a paragraph, for text
- `<ul>`, `<li>`: a bulleted list / element of a list
- `<img>`: an image
- `<script>`: JS code
<html>
  <head>
  </head>
  <body>
    <div class="box" id="main">
      <p>Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    <div class="box" id="footer">
      <p>More text</p>
    </div>
  </body>
</html>
How the Web Works: HTML

Basic attributes:

class: a user-defined type, many instances per page

id: a user-defined name, only one per page

src: for images

href: for hyper links

style: for CSS style
How the Web Works: HTML

```html
<html>
  <head>
  </head>
  <body>
    <div class="box" id="main">
      <p style="color:red;font-size:40px">Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    <div class="box" id="footer">
      <p>More text</p>
      <img src="http://www.harvard.edu/images/harvard.gif" />
    </div>
  </body>
</html>
```
<html>
  <head>
    <link type="text/css" href="style.css" rel="Stylesheet" />
  </head>
  <body>
    <div class="box" id="main">
      <p>Some text</p>
      <a href="http://www.nytimes.com">A link to the NYTimes.</a>
    </div>
    <div class="box" id="footer">
      <p>More text</p>
      <img src="http://www.harvard.edu/images/harvard.gif" />
    </div>
  </body>
</html>

```css
#main p{
  color : red;
  font-size : 40px;
}
```
Scraping: BeautifulSoup

http://www.crummy.com/software/BeautifulSoup/

You didn't write that awful page. You're just trying to get some data out of it. Right now, you don't really care what HTML is supposed to look like.

Neither does this parser.

Beautiful Soup

"A tremendous boon." -- Python411 Podcast

[ Download | Documentation | Source | What's New | Contributors | Discussion group ]

If Beautiful Soup has saved you a lot of time and money, please share the wealth.

Donate
Scraping: BeautifulSoup

```
In [1]: import BeautifulSoup
```

```
In [2]: S = BeautifulSoup(BeautifulSoup(open('index.html')))
```

Basic operation: selecting objects

```
Soup.findAll("tagname", attr1=val1, attr2=val2)
```

```
Soup.findAll("tagname")   -- get ‘em all
```

```
Soup.findAll("tagname",classname)   -- for class spec
```

(use “/home/nytimes.html”)
Scraping: BeautifulSoup

Basic operation: selecting objects

```
Soup.findAll("tagname",classname)     --for class spec
```

```
In [6]: stories = S.findAll('div','story')

In [7]: len(stories)
```

57 minutes ago
Scraping: BeautifulSoup

```python
In [1]: import BeautifulSoup
   
In [2]: S = BeautifulSoup.BeautifulSoup(open('index.html'))
   
Out[2]:
```

```python
In [6]: stories = S.findAll('div','story')

Out[7]: 40
```

```html
<span class="timestamp">57 minutes ago</span>

In [6]: stories = S.findAll('div','story')

at conflated the religious and political, the populist Iraqi
candidate has not had a chance to state that he had once derided as a
`Out[7]: 40
```

```python
In [14]: ByLines = [story.findAll(True,'byline') for story in stories]
   
In [15]: ByLines[0]
   
Out[15]:
``` `By J. DAVID GOODMAN
```
```html
</h6>
```
Scraping: BeautifulSoup

In [12]: Summaries = [story.find_all('p','summary') for story in stories]

In [13]: Summaries[0]
Out[13]:
<p class="summary">
The Democratic congresswoman from Arizona was shot in the head at a public event at a grocery store, according to her spokesman and news reports. Several other people were injured. Their conditions remain unclear.  
</p>

from starflow import utils

In [10]: Contents = [utils.Contents(s[0]).strip() for s in Summaries if s]

In [11]: Contents[0]
Out[11]: 'The Democratic congresswoman from Arizona was shot in the head at a public event at a grocery store, according to her spokesman and news reports. Several other people were injured. Their conditions remain unclear.'
Scraping: lxml

http://codespeak.net/lxml/

» lxml takes all the pain out of XML

Stephan Richter

lxml is the most feature-rich and easy-to-use library for working with XML and HTML in the Python language.

» Introduction

lxml is a Pythonic binding for the libxml2 and libxslt libraries. It is unique in that it combines the speed and feature completeness of these libraries with the simplicity of a native Python API, mostly compatible but superior to the well-known ElementTree API. See the introduction for more information about background and goals. Some common questions are answered in the FAQ.

For commercial consulting and customisations, please contact Stefan Behnel.

This page describes the current stable version of lxml. See also the web page of the development version.

» Documentation

The complete lxml documentation is available for download as PDF documentation. The HTML documentation from this web site is part of the normal source download.

- ElementTree:
  - ElementTree API
  - compatibility and differences of lxml.etree
  - benchmark results
- lxml.etree:
  - the lxml.etree Tutorial
Scraping: etree

http://docs.python.org/library/xml.etree.elementtree.html

19.13. xml.etree.ElementTree — The ElementTree
XML API

New in version 2.5.

The Element type is a flexible container object, designed to store hierarchical data structures in memory. The type can be described as a cross between a list and a dictionary.

Each element has a number of properties associated with it:

- a tag which is a string identifying what kind of data this element represents (the element type, in other words).
- a number of attributes, stored in a Python dictionary.
- a text string.
- an optional tail string.
- a number of child elements, stored in a Python sequence

BeautifulSoup does XML really well, too.
(BeautifulSoup.BeautifulStoneSoup)
Scraping: pyquery

http://pypi.python.org/pypi/pyquery

You can use the PyQuery class to load an XML document from a string, an XML document, from a file or from an URL:

```python
>>> from pyquery import PyQuery as pq
>>> from lxml import etree
>>> import urllib
>>> d = pq("<html></html>"
>>> d = pq(etree.fromstring("<html></html>"))
>>> d = pq(url='http://google.com/')
>>> d = pq(url='http://google.com/', opener=lambda url: urllib.urlopen(url).read())
>>> d = pq(filename='path_to_html_file')
```

Now `d` is like the `$` in jQuery:

```python
>>> d("#hello")
[<p id="hello">Hello world !</p>]
>>> p = d("#hello")
>>> p.html()
'Hello world !'
>>> p.html("you know <a href="http://python.org/">Python</a> rocks")
[<p id="hello">you know Python rocks</p>]
>>> p.html()
u'you know <a href="http://python.org/">Python</a> rocks'
>>> p.text()
'you know Python rocks'
```
Scraping: excel with xlrd

http://pypi.python.org/pypi/xlrd

xlrd 0.7.1

Library for developers to extract data from Microsoft Excel (tm) spreadsheet files

Extract data from new and old Excel spreadsheets on any platform. Pure Python (2.1 to 2.6). Strong support for Excel dates. Unicode-aware.

Forum for comments, questions, and bug reports: http://groups.google.com/group/python-excel
Welcome to ScraperWiki

Scraper: a computer program that copies structured information from webpages into a database

ScraperWiki: a website where people can write and repair public web scrapers and invent uses for the data
Scraping: ScraperWiki

Basically organizes the various underlying tools:

**Scraping**

How to retrieve HTML pages from websites.

```python
scraperwiki.scrape(url, [params=], )
```

Downloads a web page, and returns you the HTML. The page will appear in the "sources" pane in the scraper code editor, so you can easily view what was downloaded.

- `url` The address of the web page, e.g. `http://www.parliament.uk/mpsloordsoffices/government_and_opposition/hmg.cfm`
- `params` (optional)
  If present, makes the HTTP request a POST request, such as if a form had been submitted. `params` is a dictionary, whose keys are the names of the fields being posted, and values are their values.

**Standard libraries**

You can download pages using all the standard Python functions. The page will not appear in the "sources" pane.

- `urllib2`, `urllib.parse` Standard python libraries for opening urls. [docs](http://docs.python.org/library/urllib2.html)
- `BeautifulSoup` For parsing html. [docs](http://www.crummy.com/software/BeautifulSoup/)
- `mechanize` For navigating form fillings. [docs](http://mechanize.readthedocs.org/)
- `xlrd` For reading Excel files. [docs](http://pythonhosted.org/xlrd/
- `Python Google Chart` For generating charts using the Google Chart API. [docs](http://code.google.com/apis/chart/)
Scraping: Our htools

```python
import utils.htools as htools

def TiffanyCatalogInstantiator(creates = protocolroot + 'Tiffany.py'):
    L = [ParseTiffany1, {"Parser":ParseTiffany2, "Getter":JSONWgetter},ParseTiffanyFinal]

def ParseTiffany1(page, x):
    Soup = BeautifulSoup(open(page), convertEntities=BeautifulStoneSoup.HTML_ENTITIES)
    G = [a for a in Soup.findAll('a', 'globalnav') if a.findNext().name == 'ul']
    Recs = []
    for g in G:
        lev1 = Contents(g)
        LI = g.findNext().findAll('li')
        newrecs = [(lev1, Contents(li), str(dict(li.findAll('a')[0].attrs)['href'])) for li in LI if Contents(li).strip('\x2\xa0')]
        for i in range(len(newrecs)):
            l = newrecs[i]
            if not l[2].startswith('http://h');
                newrecs[i] = (l[0], l[1], 'http://www.tiffany.com' + l[2])
            l = newrecs[i]
            newrecs[i] = (l[0], l[1], l[2].replace('Category.aspx', 'CategoryBrowse.aspx'))

    Recs += newrecs
    return tb.tabarray(records = Recs, names = ['Level1', 'Level2', 'URL'], coloring=["Categories"[:"Level1", "Level2"]])
```
Data Wrangling Outline

Three easy pieces:

1. Scraping & Parsing Tools & Techniques
2. Data “Cleaning” Tools & Techniques
3. A bit of database technology (MongoDB primer)
Data Wrangling Outline

Even if you have this in CSV format:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.6</td>
<td>-0.3</td>
<td>6.2</td>
<td>6.5</td>
<td>7.6</td>
<td>2.2</td>
<td>0.6</td>
<td>-5.4</td>
<td>-1.4</td>
<td>4.8</td>
<td>-3.7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6.8</td>
<td>1.3</td>
<td>0.1</td>
<td>2.0</td>
<td>4.7</td>
<td>0.6</td>
<td>3.2</td>
<td>0.6</td>
<td>6.3</td>
<td>0.9</td>
<td>6.0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7.4</td>
<td>2.7</td>
<td>1.5</td>
<td>0.6</td>
<td>3.7</td>
<td>-0.4</td>
<td>2.7</td>
<td>-0.6</td>
<td>9.1</td>
<td>2.0</td>
<td>9.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-5.1</td>
<td>1.3</td>
<td>4.0</td>
<td>-2.4</td>
<td>5.3</td>
<td>1.5</td>
<td>-3.5</td>
<td>1.2</td>
<td>6.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7.0</td>
<td>-0.9</td>
<td>-2.0</td>
<td>4.3</td>
<td>6.4</td>
<td>2.2</td>
<td>4.1</td>
<td>2.6</td>
<td>-0.8</td>
<td>1.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-12.2</td>
<td>0.8</td>
<td>4.3</td>
<td>6.4</td>
<td>2.2</td>
<td>4.1</td>
<td>2.6</td>
<td>-0.8</td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>50.9</td>
<td>29.1</td>
<td>15.1</td>
<td>10.7</td>
<td>10.7</td>
<td>10.7</td>
<td>10.7</td>
<td>15.1</td>
<td>15.1</td>
<td>15.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>14.3</td>
<td>14.2</td>
<td>-0.4</td>
<td>-4.2</td>
<td>-4.6</td>
<td>-21.0</td>
<td>-12.1</td>
<td>-1.0</td>
<td>15.1</td>
<td>15.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-6.2</td>
<td>-7.7</td>
<td>13.5</td>
<td>27.2</td>
<td>13.2</td>
<td>1.0</td>
<td>10.7</td>
<td>10.7</td>
<td>10.7</td>
<td>10.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-2.1</td>
<td>2.7</td>
<td>-5.6</td>
<td>8.6</td>
<td>11.6</td>
<td>12.5</td>
<td>5.9</td>
<td>-7.3</td>
<td>-15.6</td>
<td>-15.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>-0.2</td>
<td>-12.8</td>
<td>25.4</td>
<td>39.2</td>
<td>-24.0</td>
<td>-5.0</td>
<td>13.5</td>
<td>22.9</td>
<td>-18.1</td>
<td>-18.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>-18.5</td>
<td>76.9</td>
<td>102.7</td>
<td>-4.7</td>
<td>25.5</td>
<td>-12.2</td>
<td>26.5</td>
<td>-6.8</td>
<td>39.9</td>
<td>59.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Change in private inventories</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Net exports of goods and services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>-4.9</td>
<td>-18.8</td>
<td>-38.2</td>
<td>-17.0</td>
<td>-32.5</td>
<td>9.3</td>
<td>-13.4</td>
<td>57.2</td>
<td>-3.4</td>
<td>-29.4</td>
<td>-43.9</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-10.7</td>
<td>-20.9</td>
<td>-41.0</td>
<td>-16.6</td>
<td>-34.6</td>
<td>10.7</td>
<td>-16.0</td>
<td>64.0</td>
<td>-6.6</td>
<td>-29.3</td>
<td>-44.6</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>38.4</td>
<td>-8.5</td>
<td>-17.1</td>
<td>-25.5</td>
<td>-18.7</td>
<td>4.0</td>
<td>-3.5</td>
<td>23.4</td>
<td>10.2</td>
<td>-35.3</td>
<td>-40.4</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>8.0</td>
<td>-38.2</td>
<td>23.5</td>
<td>61.7</td>
<td>16.6</td>
<td>22.6</td>
<td>-6.3</td>
<td>-11.4</td>
<td>-5.7</td>
<td>-15.4</td>
<td>6.1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>9.5</td>
<td>-43.9</td>
<td>5.5</td>
<td>86.4</td>
<td>16.7</td>
<td>8.8</td>
<td>-1.3</td>
<td>-10.3</td>
<td>-12.3</td>
<td>-13.6</td>
<td>13.8</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>3.4</td>
<td>-14.8</td>
<td>97.8</td>
<td>5.7</td>
<td>17.4</td>
<td>73.9</td>
<td>-19.5</td>
<td>-15.1</td>
<td>15.7</td>
<td>-25.7</td>
<td>-13.4</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>2.6</td>
<td>3.7</td>
<td>-5.8</td>
<td>8.2</td>
<td>20.7</td>
<td>8.0</td>
<td>17.9</td>
<td>11.9</td>
<td>19.4</td>
<td>4.1</td>
<td>-8.2</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0.8</td>
<td>0.2</td>
<td>-13.2</td>
<td>14.3</td>
<td>28.7</td>
<td>8.4</td>
<td>23.5</td>
<td>8.3</td>
<td>15.9</td>
<td>-4.3</td>
<td>-18.4</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>-23.7</td>
<td>22.3</td>
<td>2.8</td>
<td>2.8</td>
<td>-3.3</td>
<td>-5.9</td>
<td>21.5</td>
<td>11.0</td>
<td>7.5</td>
<td>-1.1</td>
<td>-22.9</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>231.8</td>
<td>-50.0</td>
<td>-56.7</td>
<td>106.4</td>
<td>222.7</td>
<td>62.9</td>
<td>29.1</td>
<td>1.6</td>
<td>40.9</td>
<td>-11.7</td>
<td>-5.7</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5.5</td>
<td>9.7</td>
<td>6.6</td>
<td>-0.6</td>
<td>12.0</td>
<td>7.3</td>
<td>10.0</td>
<td>17.6</td>
<td>24.8</td>
<td>17.7</td>
<td>7.9</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Gross domestic product</td>
<td>5.6</td>
<td>7.0</td>
<td>17.1</td>
<td>9.7</td>
<td>11.1</td>
<td>10.1</td>
<td>1.9</td>
<td>-7.3</td>
<td>-5.4</td>
<td>2.2</td>
<td>-3.8</td>
</tr>
</tbody>
</table>
Data Wrangling Outline

Even if you have this in CSV format, supposed clean:

It still needs cleaning. Let me count the ways.
1. Header needs to be removed. (and its data used)

2. Empty column header needs filling in. It’s the “category” column.

3. Column headers refer to dates in a weird way.

4. Line 14 is empty

5. The “category” column is actually several columns collapsed via formatting.
You have several options for something like this job

Python itself (esp csv module)

Numpy/Tabular

Google Refine
Data Wrangling: Cleaning

Python itself

Use “/home/NIPA_Table_1.1.1.csv”
(make sure you open it ‘rU’)

```python
In [20]: X = open('/Users/dyamins/Desktop/test.csv','rU').read()
In [21]: Lines = X.split('
')
In [22]: Lines[0]
Out[22]: '"Table 1.1. Percent Change From Preceding Period in Real Gross Domestic Product"
```

```python
In [23]: Lines[5]
```
Data Wrangling: Cleaning

Python itself

```python
In [24]: ColumnHeaders = Lines[5].split(', '

In [25]: ColumnHeaders[:10]
Out[25]:
```

Notice the error
Data Wrangling: Cleaning

The native CSV module can often help:

```python
In [31]: ColumnHeaders[0]
Out[31]: 'Line',''

In [25]: ColumnHeaders[0]
Out[25]: ['Line','']

In [32]: import csv

In [33]: H = csv.reader([Lines[5]])
   23.7, 22.195
   -50.0
   196
   current

In [34]: ColumnHeaders = H.next()

In [35]: ColumnHeaders[:10]
```
Data Wrangling: Cleaning

http://entabular.com/tabular/

Tabular

Tabular data can be easily represented in Python using the language’s native objects – e.g. by lists of tuples to create, these kind of representations typically do not enable important tabular data manipulations, like spreadsheet-style operations.

Tabular is a package of Python modules for working with tabular data. Its main object is the tabarray class. Data. By putting data into a tabarray object, you’ll get a representation of the data that is more flexible and specifically, tabarray provides:

- ultra-fast filtering, selection, and numerical analysis methods, using convenient Matlab-style matrix operations
- spreadsheet-style operations, including row & column operations, ‘sort’, ‘replace’, ‘aggregate’, ‘pivot’
- flexible load and save methods for a variety of file formats, including delimited text (CSV), binary, and
- sophisticated inference algorithms for determining formatting parameters and data types of input files
- support for hierarchical groupings of columns, both as data structures and file formats
Data Wrangling: Cleaning

import tabular as tb

use “/home/TiffanyCatalog.csv”

In [33]: X = tb.tabarray(SVfile = '/Users/dyamins/gotdata/Data/catalogs/Tiffany/catalog.tsv')
Setting metadata attribute from dialect delimiter to equal specified value: '\t'
Setting dialect attribute delimiter to equal specified value: '\t'

In [34]: X.dtype
Out[34]: dtype([('Level1', 'S23'), ('Level2', 'S28'),
('Page', 'S2'), ('Image', 'S75'), ('Item', 'S83'),
('Price', 'S10'), ('sku', 'S8'), ('Image2', 'S80'),
('Image3', 'S81'), ('Image4', 'S75'))]

Y = X[['Level1', 'Level2', 'Price']]
Data Wrangling: Cleaning

```python
from starflow import utils

In [4]: utils.uniqify(X[['Level1']])
Out[4]:
['Jewelry',
 'Diamonds',
 'Watches',
 'Men's',
 'Designers & Collections',
 'Accessories',
 'Gifts',
 'Table']

In [6]: utils.uniqify(X[['Level1','Level2']].tolist())
Out[6]:
['Jewelry', 'New Jewelry'),
 ('Jewelry', '$250 & Under'),
 ('Jewelry', '$500 & Under'),
 ('Jewelry', '$1,000 & Under'),
 ('Jewelry', 'Bracelets'),
 ('Jewelry', 'Brooches'),
 ('Jewelry', 'Tiffany Charms'),
 ('Jewelry', 'Earrings'),
 ('Jewelry', 'Necklaces & Pendants'),
 ('Jewelry', 'Rings'),
 ('Jewelry', 'Tiffany Celebration\xo2\xae Rings'),
 ('Jewelry', 'Tiffany Wedding Bands'),
 ('Jewelry', 'Tiffany Keys'),
 ('Jewelry', 'Tiffany Silver Jewelry'),
 ('Jewelry', 'Statement Jewelry'),
]```
Data Wrangling: Cleaning

In [16]: Prices = [int(x.strip('$')).replace(',', '').replace(',', '') for x in X['Price']]

In [17]: Y1 = Y.addcols(Prices, names=["Price"])
('Replacing columns', ['Price'])

In [18]: Y1
Out[18]:

darray([("Jewelry", "New Jewelry", 3750), ("Jewelry", "New Jewelry", 100
   ("Jewelry", "New Jewelry", 125), ...
   ("Table", "Wedding Gifts", 155), ("Table", "Wedding Gifts", 115),
   ("Table", "Wedding Gifts", 330)
   dtype=[("Level", '<i8'), ("Level2", '<i8'), ("Price", '<i8')]]
Data Wrangling: Cleaning

```python
import numpy as np

In [19]: AggFuncDict = {}

In [20]: AggFuncDict['Price'] = lambda x: np.array(x).mean()

In [21]: AggFuncDict['Level2'] = lambda x: 'Average'

In [22]: Z = Yl.aggregate(On=['Level1'],AggFuncDict = AggFuncDict)

In [26]: Z
Out[26]:
tabarray([(Accessories', 496.4193548387097, 'Average'),
         ('Designers & Collections', 3726.367137355584, 'Average'),
         ('Diamonds', 3554.987037037037, 'Average'),
         ('Gifts', 1452.8847406664995, 'Average'),
         ('Jewelry', 7793.111458333334, 'Average'),
         ('Men's', 1573.1723484848485, 'Average'),
         ('Table', 541.8221343873518, 'Average'),
         ('Watches', 11600.310559006211, 'Average')],
        dtype=[('Level1', 'S23'), ('Price', '<f8'), ('Level2', 'S7')])

Z.sort(order=['Price'])
Data Wrangling: Cleaning

http://code.google.com/p/google-refine/
Data Wrangling Outline

Three easy pieces:

1. Scraping & Parsing Tools & Techniques

2. Data “Cleaning” Tools & Techniques

   A simple example to pull things together ...

3. A bit of database technology (MongoDB primer)
Data Wrangling Outline

An example to pull things together:

cp /home/parse_FEC.py .
cp /home/runFEC.sh .
emacs parse_FEC.py
execute and understand all the lines with your partner

uncomment “ALL DATA” line and comment “SOME DATA” line, using ‘#’ char to start comment

qsub runFEC.sh
Data Wrangling Outline

Three easy pieces:

1. Scraping & Parsing Tools & Techniques
2. Data “Cleaning” Tools & Techniques
3. A bit of database technology (MongoDB primer)
Data Wrangling: Databases

A Database is a program that allows data to be retrieved by topically-meaningful query (as opposed to loading from a file).

The basic “process” you do is:

1. Start your DB program.

2. Insert data (like putting files into a directory, but the file system structure is hidden)

3. Then you get data by:
   
   **INPUT:** a query

   **OUTPUT:** one or more records
Data Wrangling: Databases

One of the nice things about modern databases is that they allow many users to insert and retrieve data simultaneously.

Query In, Records Out
MongoDB is a good modern database program.

MongoDB has three levels in its “schema”:

1) The “database” level; you’re allowed to have multiple databases on your machine.

2) The “collection” level; each collection is a little like a tabular data file. One database can contain many collections.

3) The “document” level; each document is like a record in a CSV. Each collection is composed of many documents.
Data Wrangling: Databases

1) The “database” level; you’re allowed to have multiple databases on your machine.
2) The “collection” level; each collection is a little like a tabular data file. One database can contain many collections.
Data Wrangling: Databases

3) The “document” level; each document is like a record in a CSV. Each collection is composed of many documents.
Data Wrangling: Databases

Let’s do an example.

```python
In [1]: import pymongo
In [2]: connection = pymongo.Connection()
In [3]: connection
Out[3]: Connection('localhost', 27017)
In [4]: database = connection['testdb']
In [5]: collection = database['testcollection']
In [6]: collection
Out[6]: Collection(Database(Connection('localhost', 27017), u'testdb'), u'testcollection')
```

How many records are in our collection?

```python
In [7]: collection.count()
Out[7]: 0
```

0, since we just started it.
**Data Wrangling: Databases**

In [8]: collection.insert({'State': 'Alabama', 'Employment': 22})
Out[8]: ObjectId('4d28edfab90022493b000000')

In [9]: collection.insert({'State': 'Alaska', 'Employment': 3})
Out[9]: ObjectId('4d28ee05b90022493b00001')

In [10]: collection.insert({'State': 'Massachusetts', 'Employment': 54})
Out[10]: ObjectId('4d28ee13b90022493b00002')

In [11]: collection.count()
Out[11]: 3

In [32]: collection.find().distinct('State')
Out[32]: [u'Alabama', u'Alaska', u'Massachusetts']

In [12]: collection.find_one()
Out[12]:
{u'Employment': 22, 
  u'State': u'Alabama', 
  u'_id': ObjectId('4d28edfab90022493b000000')}
Data Wrangling: Databases

In [15]: collection.insert({'State': 'Massachusetts', 'Employment': 4, 'Year': 2001})
Out[15]: ObjectId('4d28ee96b90022493b000003')

In [16]: collection.insert({'State': 'Massachusetts', 'Employment': 5, 'Year': 2002})
Out[16]: ObjectId('4d28ee9ab90022493b000004')

In [17]: collection.insert({'State': 'Massachusetts', 'Employment': 10, 'Year': 2003})
Out[17]: ObjectId('4d28ee9fb90022493b000005')

In [18]: collection.insert({'State': 'Massachusetts', 'Employment': 3, 'Year': 2004})
Out[18]: ObjectId('4d28eea4b90022493b000006')

In [19]: collection.insert({'State': 'Massachusetts', 'Employment': 6, 'Year': 2005})
Out[19]: ObjectId('4d28eeab90022493b000007')

In [20]: collection.insert({'State': 'Massachusetts', 'Employment': 4, 'Year': 2006})
Out[20]: ObjectId('4d28eeb3b90022493b000008')

In [80]: collection.find({'State': 'Massachusetts'}).count()
Out[80]: 7

In [81]: collection.find({'State': 'Massachusetts'}).distinct('Year')
Data Wrangling: Databases

In [21]: collection.find_one({'State': 'Massachusetts'})
Out[21]:
{u'Employment': 54,
 u'State': u'Massachusetts',
 u'_id': ObjectId('4d28ee13b90022493b000002')}

In [22]: collection.find_one({'State': 'Massachusetts', 'Year': 2005})
Out[22]:
{u'Employment': 6,
 u'State': u'Massachusetts',
 u'Year': 2005,
 u'_id': ObjectId('4d28eeae90022493b000007')}

In [26]: collection.find_one({'State': 'Massachusetts', 'Employment': {'$lt': 5}})
Out[26]:
{u'Employment': 4,
 u'State': u'Massachusetts',
 u'Year': 2001,
 u'_id': ObjectId('4d28ee96b90022493b00003')}
Data Wrangling: Databases

In [27]: data = collection.find({"State": 'Massachusetts'})

Out[27]:

In [28]: data.next()

Out[28]:

{u'Employment': 54,
 u'State': u'Massachusetts',
 u'_id': ObjectId('4d28ee13b90022493b000002')}

In [29]: data.next()

Out[29]:

{u'Employment': 4,
 u'State': u'Massachusetts',
 u'Year': 2001,
 u'_id': ObjectId('4d28ee96b90022493b000003')}

In [30]: data.next()

Out[30]:

{u'Employment': 5,
 u'State': u'Massachusetts',
 u'Year': 2002,
 u'_id': ObjectId('4d28ee9ab90022493b000004')}
Data Wrangling: Databases

In [33]: collection.insert({'Location': {'State': 'Massachusetts', 'City': 'Cambridge'}, 'Employment': 20, 'Year': 2010})
Out[33]: ObjectId('4d28f3b3b90022493b000009')

In [34]: collection.insert({'Location': {'State': 'New York', 'City': 'Albany'}, 'Employment': 4, 'Year': 2010})
Out[34]: ObjectId('4d28f3fbb90022493b00000a')

In [35]: collection.find_one({'Location.State': 'Massachusetts'})
Out[35]:
{u'Employment': 20,
 u'Location': {u'City': u'Cambridge', u'State': u'Massachusetts'},
 u'Year': 2010,
 u'_id': ObjectId('4d28f3b3b90022493b000009')}
In [36]: collection.find_one({‘Location.City’: ‘Albany’, ‘Employment’: 22})

In [37]: collection.find_one({‘Location.City’: ‘Albany’, ‘Employment’: 4})
Out[37]:
{u’Employment’: 4,
 u’Location’: {u’City’: u’Albany’, u’State’: u’New York’},
 u’Year’: 2010,
 u’_id’: ObjectId(‘4d28f3fbb90022493b00000a’)}

In [56]: data = collection.find({‘Employment’: {$in: [3, 4, 6]}})

In [57]: data.next()
Out[57]:
{u’Employment’: 3,
 u’State’: u’Alaska’,
 u’_id’: ObjectId(‘4d28ee05b90022493b000001’)}

In [58]: data.next()
Out[58]:
{u’Employment’: 4,
 u’State’: u’Massachusetts’,
 u’Year’: 2001,
 u’_id’: ObjectId(‘4d28ee96b90022493b000003’)}
In [40]: data = collection.find({'$or':[{'Location.City': 'Albany'},{'Employment': 4}]})

In [41]: data.next()
Out[41]:
{u'Employment': 4,
 'State': u'Massachusetts',
 'Year': 2001,
 '_id': ObjectId('4d28ee96b90022493b000003')}

In [42]: data.next()
Out[42]:
{u'Employment': 4,
 'State': u'Massachusetts',
 'Year': 2006,
 '_id': ObjectId('4d28eeb3b90022493b000008')}

In [43]: data.next()
Out[43]:
{u'Employment': 4,
 'Location': {u'City': u'Albany', u'State': u'New York'},
 'Year': 2010,
 '_id': ObjectId('4d28f3fbb90022493b00000a')}
Data Wrangling: Databases

In [52]: collection.find_one({'Location.City':{'$exists':True}})
Out[52]:
{u'Employment': 20,
 u'Location': {u'City': u'Cambridge', u'State': u'Massachusetts'},
 u'Year': 2010,
 u'_id': ObjectId('4d28f3b3b90022493b000009')}

In [53]: collection.find_one({'Year':{'$exists':False}})
Out[53]:
{u'Employment': 22,
 u'State': u'Alabama',
 u'_id': ObjectId('4d28edf90022493b000000')}
Data Wrangling: Databases

```python
In [61]: data = collection.find().sort('Employment')
    ...:
    ...: [u'Employment': 20,
    ...:  u'Location': {u'City': u'Camb'}
    ...:  u'Year': 2010,
    ...:  u'_id': ObjectId('4d28f3b3b90000001')}

In [62]: data.next()
Out[62]:
    ...: {u'Employment': 3,
    ...:  u'State': u'Alaska',
    ...:  u'_id': ObjectId('4d28ee05b9002493b000001')}

In [63]: data.next()
Out[63]:
    ...: {u'Employment': 3,
    ...:  u'State': u'Massachusetts',
    ...:  u'Year': 2004,
    ...:  u'_id': ObjectId('4d28ee4b9002493b00006')}

In [64]:

In [65]: data.next()
Out[65]:
    ...: {u'Employment': 4,
    ...:  u'State': u'Massachusetts',
    ...:  u'Year': 2001,
    ...:  u'_id': ObjectId('4d28ee96b9002493b00003')}
```
In [68]: data = collection.find().sort('Employment', direction=-1)

In [69]: data.next()
Out[69]:
{u'Employment': 54,
 'State': u'Massachusetts',
 '_id': ObjectId('4d28ee13b90022493b000002')}

In [70]: data.next()
Out[70]:
{u'Employment': 22,
 'State': u'Alabama',
 '_id': ObjectId('4d28edfab90022493b000000')}

In [77]: data = collection.find().sort('Employment').skip(5)

In [78]: data.next()
Out[78]:
{u'Employment': 5,
 'State': u'Massachusetts',
 'Year': 2002,
 '_id': ObjectId('4d28ee9ab90022493b000004')}

In [65]: data.next()
Out[65]:
{u'Employment': 4,
 'State': u'Massachusetts',
 'Year': 2002,
 '_id': ObjectId('4d28ee96b90022493b000000')}

In [64]: data.next()
Out[64]:
{u'Employment': 2,
 'State': u'Alabama',
 'Year': 2002,
 '_id': ObjectId('4d28ee96b90022493b000001')}

In [63]: data.next()
Out[63]:
{u'Employment': 1,
 'State': u'Alabama',
 'Year': 2002,
 '_id': ObjectId('4d28ee96b90022493b000002')}
Data Wrangling: Databases

http://www.mongodb.org/

Agile and Scalable

MongoDB (from "hmongous") is a scalable, high-performance, open source, document-oriented database. Written in C++, MongoDB features:

- **Document-oriented storage**
  JSON-style documents with dynamic schemas offer simplicity and power.

- **Full Index Support**
  Index on any attribute, just like you’re used to.

- **Replication & High Availability**
  Mirror across LANs and WANs for scale and peace of mind.

- **Auto-Sharding**
  Scale horizontally without compromising functionality.

- **Querying**

- **Fast In-Place Updates**
  Atomic modifiers for contention-free performance.

- **Map/Reduce**
  Flexible aggregation and data processing.

- **GridFS**
  Store files of any size without complicating your stack.

Advanced Queries

- Introduction
- Retrieving a Subset of Fields
- Conditional Operators
  - `<`, `<=`, `>`, `>=`
  - `$all`
  - `$exists`
  - `$mod`
  - `$ne`
  - `$in`
  - `$nin`
  - `$nor`
  - `$or`
  - `$size`
  - `$type`
- Regular Expressions
- Value in an Array
  - `SelemMatch`
- Value in an Embedded Object
- Meta operator: `$not`
- Javascript Expressions and `$where`
- Cursor Methods
  - `count()`
  - `limit()`
  - `skip()`
  - `snapshot()`
  - `sort()`
- Special operators
  - `group()`
- See Also