getpatent: Scraping patent data into Stata

Demetris Christodoulou (Sydney)
Le Ma (UTS)
Hadi Mostafavi (Sydney)

Methodological and Empirical Advances in Financial Analysis (MEAFA)

September 27, 2016

Outline

Outline

Create database of patent attributes

- To enable research in innovation activity and the generation of intangible assets, we require detailed data on the outcome of the innovation process - the most observable and measurable being the number of patents and quality measures.

Create database of patent attributes

- To enable research in innovation activity and the generation of intangible assets, we require detailed data on the outcome of the innovation process - the most observable and measurable being the number of patents and quality measures.
- Although patent data is public and freely searchable, regional patent offices have restrictions on access and their data is limited to basic patent bibliographic information e.g. identifiers, date, title, classification, applicants and inventors. Their free data does not include information on patent citations, legal claims, legal status etc.

Create database of patent attributes

- To enable research in innovation activity and the generation of intangible assets, we require detailed data on the outcome of the innovation process - the most observable and measurable being the number of patents and quality measures.
- Although patent data is public and freely searchable, regional patent offices have restrictions on access and their data is limited to basic patent bibliographic information e.g. identifiers, date, title, classification, applicants and inventors. Their free data does not include information on patent citations, legal claims, legal status etc.

  - The EPO (Europe) provides free raw patent data in XML format.
  - The WIPO (World) allows downloads of up to $10,000$ records.
  - The SIPO (China) requires *domestic* account registration.
  - The exception is USPTO which provides all data in tab-delimited format.

Create database of patent attributes

- To enable research in innovation activity and the generation of intangible assets, we require detailed data on the outcome of the innovation process - the most observable and measurable being the number of patents and quality measures.
- Although patent data is public and freely searchable, regional patent offices have restrictions on access and their data is limited to basic patent bibliographic information e.g. identifiers, date, title, classification, applicants and inventors. Their free data does not include information on patent citations, legal claims, legal status etc.
    - The EPO (Europe) provides free raw patent data in XML format.
    - The WIPO (World) allows downloads of up to $10,000$ records.
    - The SIPO (China) requires *domestic* account registration.
    - The exception is USPTO which provides all data in tab-delimited format.
- There is also the issue of non-standardisation when working across multiple sources.

Google Patent Search

- Google Patent Search consolidates 87 million patent publications from 17 patent offices around the world including the US, Europe, Japan, China, South Korea, WIPO, Russia, Germany, The United Kingdom, Canada, France, Spain, Belgium, Denmark, Finland, Luxembourg, and the Netherlands.

Google Patent Search

- Google Patent Search consolidates 87 million patent publications from 17 patent offices around the world including the US, Europe, Japan, China, South Korea, WIPO, Russia, Germany, The United Kingdom, Canada, France, Spain, Belgium, Denmark, Finland, Luxembourg, and the Netherlands.

- This is free data and even though Google does not like mining its website, an efficient and careful code can scrape this information into a database.

Google Patent Search

- Google provides this data from several locations. The US servers are indexed in https://patents.google.com.

Google Patent Search

- Google provides this data from several locations. The US servers are indexed in https://patents.google.com.
- The US-based data is then mirrored onto local services, e.g. in Australia as https://www.google.com.au/patents, in Greece as https://www.google.gr/patents and so on.

Google Patent Search

- Google provides this data from several locations. The US servers are indexed in https://patents.google.com.
- The US-based data is then mirrored onto local services, e.g. in Australia as https://www.google.com.au/patents, in Greece as https://www.google.gr/patents and so on.
- There are two advantages in working with local servers: (1) they speak your language, (2) they give information for the 'cooperative' classification scheme.

Google Patent Search

- Google provides this data from several locations. The US servers are indexed in https://patents.google.com.
- The US-based data is then mirrored onto local services, e.g. in Australia as https://www.google.com.au/patents, in Greece as https://www.google.gr/patents and so on.
- There are two advantages in working with local servers: (1) they speak your language, (2) they give information for the 'cooperative' classification scheme.
- The US server contains the more widely recognised standard for international classification for patents, and importantly for us it applies a more consistent structure in its source code making it easier to scrape.

Outline

HTML source code

- HTML source code can be unpredictable and may follow any structure from page to page. Programmers do not need to follow any specific structural rules when writing code for webpages - they can write dirty and the browser will still interpret.

HTML source code

- HTML source code can be unpredictable and may follow any structure from page to page. Programmers do not need to follow any specific structural rules when writing code for webpages - they can write dirty and the browser will still interpret.
- We tried writing something with Stata that is more generalisable and could be interpreted in any HTML situation, but the task is beyond our capabilities and patience.

HTML source code

- HTML source code can be unpredictable and may follow any structure from page to page. Programmers do not need to follow any specific structural rules when writing code for webpages - they can write dirty and the browser will still interpret.
- We tried writing something with Stata that is more generalisable and could be interpreted in any HTML situation, but the task is beyond our capabilities and patience.
- The point being that scraping source code with Stata must be coded as a webpage-specific task. What works for Google Patent Search does not have to work with any other website.

Google Search Patent HTML source code

```
<html>
   <head>
      <meta> .... </meta>
      <script> .... </script>
      <style> .... </style>
   </head>
   <body>
      <h1 itemprop="title">Component name extraction system and method </h1>
      <h2>Info</h2>
         <dl>
            <dt>Publication number</dt>
              <dd itemprop="publicationNumber">CN102455997A</dd>
                ...
            <dt>Authority</dt>
              <dd itemprop="countryCode">CN</dd>
                ...
            <dt>Inventor</dt>
               <dd itemprop="inventor" repeat>Donald J. Leary</dd>
           ...
      <h2>Links</h2>
         ...
      <h2>Classifications</h2>
         ...
   </body>
</html>
```

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.
    2. Segment the remaining <body></body> by headings as sections.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.
    2. Segment the remaining <body></body> by headings as sections.
    3. There is only one <h1></h1> that holds the patent's title.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.
    2. Segment the remaining <body></body> by headings as sections.
    3. There is only one <h1></h1> that holds the patent's title.
    4. The remaining <body> is segmented by <h2></h2>.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.
    2. Segment the remaining <body></body> by headings as sections.
    3. There is only one <h1></h1> that holds the patent's title.
    4. The remaining <body> is segmented by <h2></h2>.
    5. Within a given <h2></h2> we search for the **itemprop=""** attribute, e.g. **itemprop="inventor"**. This is the item's property name that ends up as a variable name in the new dataset.

Segmenting the HTML code

- Think of the source code as a very long string, and strings are memory hungry.
    1. Purge <head></head> that contains mostly formatting code, that is taking up about half of the length of the string.
    2. Segment the remaining <body></body> by headings as sections.
    3. There is only one <h1></h1> that holds the patent's title.
    4. The remaining <body> is segmented by <h2></h2>.
    5. Within a given <h2></h2> we search for the **itemprop=""** attribute, e.g. **itemprop="inventor"**. This is the item's property name that ends up as a variable name in the new dataset.
    6. **itemprop=""** contains a *value* that ends up as the observation for that variable and that patent code, e.g. itemprop="inventor">Donald J. Leary<.

Outline

Read source code

- The source code is read as a single very long string, i.e. one source code is a single observation, as for example:

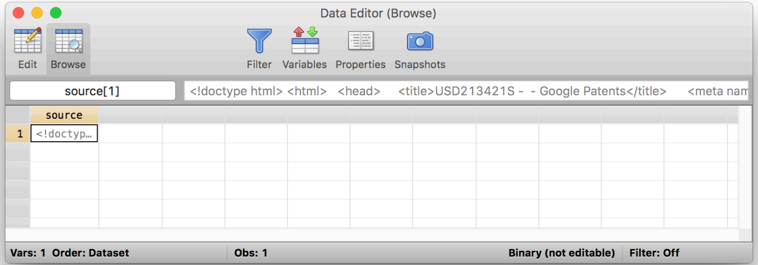
```
generate source = fileread("https://patents.google.com/patent/USD213421S")
```

Read source code

- The source code is read as a single very long string, i.e. one source code is a single observation, as for example:

```
generate source = fileread("https://patents.google.com/patent/USD213421S")
```

- **filereaderror()==0** checks that the URL exists. If not, then that observation is recorded as missing.



source[1]   <!doctype html> <html> <head> <title>USD213421S - - Google Patents</title> <meta nam

	source
1	<!doctyp…

Vars: 1 Order: Dataset   Obs: 1   Binary (not editable)   Filter: Off

Simplify source code

- We simplify the source code by removing all conflicting characters with Stata's syntax, including the tab, carriage return, double quotes, single quotes and the grave-accent. Using the ASCII characters:

```
foreach j in char(9) char(10) char(34) char(39) char(96)  {
   replace source = subinstr(source,`j',"",.)
}
```

Simplify source code

- We simplify the source code by removing all conflicting characters with Stata's syntax, including the tab, carriage return, double quotes, single quotes and the grave-accent. Using the ASCII characters:

```
foreach j in char(9) char(10) char(34) char(39) char(96)  {
   replace source = subinstr(source,`j',"",.)
}
```

- We trim all external and internal extra spaces:

```
replace source = strtrim(stritrim(source))
```

Simplify source code

- We simplify the source code by removing all conflicting characters with Stata's syntax, including the tab, carriage return, double quotes, single quotes and the grave-accent. Using the ASCII characters:

```
foreach j in char(9) char(10) char(34) char(39) char(96)  {
   replace source = subinstr(source,`j',"",.)
}
```

- We trim all external and internal extra spaces:

```
replace source = strtrim(stritrim(source))
```

- And make everything lowercase as it is easier to match string patterns and work with regular expressions:

```
 replace source = lower(source)
```

A crash course in regular expressions (ASCII capabilities)

Regular expressions: matching patterns in strings

Operator	Description	Example
Anchors to match the location of expression		
^	Match expression at beginning of the string	^sun matches "sunrise"
$	Match expression at end of the string	sun$ matches "Monsun"
Wildcards for counting matches		
?	Match preceding expression zero or one times	A? matches nothing or A
+	Match preceding expression one or more times	A+ matches A, AA, AAA
*	Match preceding expression zero or more times	A* matches nothing, A, AA, AAA
List operators		
.	Match any character except new lines	.* matches anything any times
-	Match range of alpha characters or integers	[0-1] matches numbers 0 or 1
[]	Match one character in brackets	[aeiou] matches a lowercase vowel
[^]	Match one character except those in brackets	[^0-9] matches non-numerical
()	Match sub-expression to be extracted as string	<(.*)> capture anything within <>
|	The OR operator	[A|B] matches A or B
Escape operator		
\	Match ^$.?*[]()|+ as string literals	\^ match ^ and \\ matches \

Purge <head> and any remaining <script>

- First, get rid of the <head></head>:

```
replace source = regexr(source,"<head>.*</head>","")
```

Purge <head> and any remaining <script>

- First, get rid of the <head></head>:

```
replace source = regexr(source,"<head>.*</head>","")
```

- Then purge any remaining formatting <script></script>:

```
local check = 1
while `check'==1 {
  qui replace `pos1'   = strpos(`source',"<script")
  qui replace `pos2'   = strpos(`source',"</script>")
  qui replace `subt'   = substr(`source',`pos1',`pos2'-`pos1'+9)
  qui replace `source' = subinstr(`source',`subt',"",1)
  qui sum `pos1' if `touse'
  if r(max)==0 local check = 0
}
```

Purge <head> and any remaining <script>

- First, get rid of the <head></head>:

```
replace source = regexr(source,"<head>.*</head>","")
```

- Then purge any remaining formatting <script></script>:

```
local check = 1
while `check'==1 {
  qui replace `pos1'   = strpos(`source',"<script")
  qui replace `pos2'   = strpos(`source',"</script>")
  qui replace `subt'   = substr(`source',`pos1',`pos2'-`pos1'+9)
  qui replace `source' = subinstr(`source',`subt',"",1)
  qui sum `pos1' if `touse'
  if r(max)==0 local check = 0
}
```

- We have since learned that there is a more elegant approach to this using **uregexr()**.

Scrape patent title from within <h1></h1>

- To scrape the patent title, first take an **extract** from the **source** that contains everything within <h1></h1> inclusive (extracting smaller strings increases computational efficiency). Then, locate **itemprop=title** and scrape the patent title:

```
generate extract = regexs(regexm(source,"(<h1.*</h1>)"))
generate title   = strtrim(regexs(regexm(extract,"itemprop=title>(.*)</h1>")))
replace  title   = regexr(title,"^([a-z])",regexs(regexm(title,"^([a-z])")))
```

Scrape rest of the data from <h2></h2>

- The remaining data is segmented in <h2></h2> sections.

Scrape rest of the data from <h2></h2>

- The remaining data is segmented in <h2></h2> sections.
- We repeat a similar process as in <h1></h1> for every <h2> section, each time accounting for the specific complexity that is pertinent to the data that is scraped.

Scrape rest of the data from <h2></h2>

- The remaining data is segmented in <h2></h2> sections.
- We repeat a similar process as in <h1></h1> for every <h2> section, each time accounting for the specific complexity that is pertinent to the data that is scraped.
- For example, from <h2>information</h2> we scrape the patent office authority, with **itemprop=countrycode**, using the following regular expression:

```
generate auth = regexs(regexm(`extract',"itemprop=countrycode>([^><][a-z \&\.\-]+)</"))
```

Scrape rest of the data from <h2></h2>

- The remaining data is segmented in <h2></h2> sections.
- We repeat a similar process as in <h1></h1> for every <h2> section, each time accounting for the specific complexity that is pertinent to the data that is scraped.
- For example, from <h2>information</h2> we scrape the patent office authority, with **itemprop=countrycode**, using the following regular expression:

```
generate auth = regexs(regexm(`extract',"itemprop=countrycode>([^><][a-z \&\.\-]+)</"))
```

- For **itemprop=inventor** there may be multiple inventors, so the process is recursive until these is none left to scrape. The regular expression for inventors is:

```
gen invent = regexs(regexm(`extract',"itemprop=inventor.+>([^><][a-z /:\.\-\(\)\\]+)</"))
```

Scrape rest of the data from <h2></h2>

- The remaining data is segmented in <h2></h2> sections.
- We repeat a similar process as in <h1></h1> for every <h2> section, each time accounting for the specific complexity that is pertinent to the data that is scraped.
- For example, from <h2>information</h2> we scrape the patent office authority, with **itemprop=countrycode**, using the following regular expression:

```
generate auth = regexs(regexm(`extract',"itemprop=countrycode>([^><][a-z \&\.\-]+)</"))
```

- For **itemprop=inventor** there may be multiple inventors, so the process is recursive until these is none left to scrape. The regular expression for inventors is:

```
gen invent = regexs(regexm(`extract',"itemprop=inventor.+>([^><][a-z /:\.\-\(\)\\]+)</"))
```

- The are other specific complexities, too many to list here.

getpatent.ado

gepatent requires access to a list of patent codes for reaching the dynamic URLs. If some codes are not valid then it returns missing values. There are two sets of options related to (1) which information should be scraped and (2) how quickly or carefully should this be done:

getpatent *codevar* [**if**] [**in**] , [*options*]

- There are actually too many *options* to list here related to (1) and they follow the HTML segmented structure.

getpatent.ado

- Specifying the option **all** scrapes every `itemprop=""` from the webpage which is fine for small datasets but would be problematic for large data because **all** would also scrape narrative text, such as **itemprop="abstract"** and **itemprop="description"**.

getpatent.ado

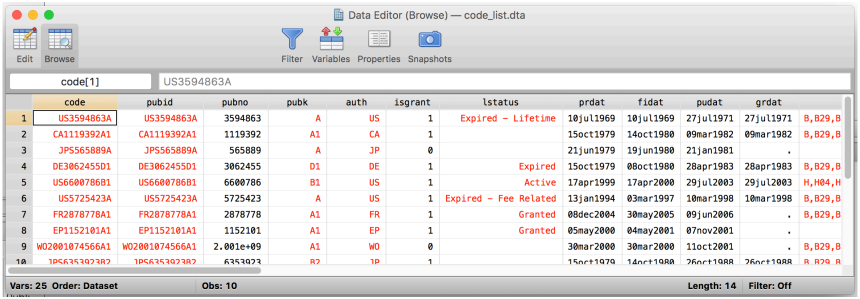
- Specifying the option **all** scrapes every `itemprop=""` from the webpage which is fine for small datasets but would be problematic for large data because **all** would also scrape narrative text, such as **itemprop="abstract"** and **itemprop="description"**.
- So, for large data be parsimonious. Specify only what you need. You should definitely specify **info** that gets all patent identifiers (e.g. **pubid**, **auth**, **invent**, **dates**) and then see what you need, e.g. **classifications**, **freferences**, **breferences**.

getpatent.ado

- Specifying the option **all** scrapes every `itemprop=""` from the webpage which is fine for small datasets but would be problematic for large data because **all** would also scrape narrative text, such as **itemprop="abstract"** and **itemprop="description"**.
- So, for large data be parsimonious. Specify only what you need. You should definitely specify **info** that gets all patent identifiers (e.g. **pubid**, **auth**, **invent**, **dates**) and then see what you need, e.g. **classifications**, **freferences**, **breferences**.
- There are also some utility options that specify how often should the program visit the Google website and how many calls it should make each time, as there is a risk of being uncovered as a robot and banned from visiting.

Example

```
. getpatent code, pubid pubno pubk auth isgrant lstatus dates class
```



	code	pubid	pubno	pubk	auth	isgrant	lstatus	prdat	fidat	pudat	grdat	
1	US3594863A	US3594863A	3594863	A	US	1	Expired - Lifetime	10jul1969	10jul1969	27jul1971	27jul1971	B,B29,B
2	CA1119392A1	CA1119392A1	1119392	A1	CA	1		15oct1979	14oct1980	09mar1982	09mar1982	B,B29,B
3	JPS565889A	JPS565889A	565889	A	JP	0		21jun1979	19jun1980	21jan1981	.	
4	DE3062455D1	DE3062455D1	3062455	D1	DE	1	Expired	15oct1979	08oct1980	28apr1983	28apr1983	B,B29,B
5	US6600786B1	US6600786B1	6600786	B1	US	1	Active	17apr1999	17apr2000	29jul2003	29jul2003	H,H04,H
6	US5725423A	US5725423A	5725423	A	US	1	Expired - Fee Related	13jan1994	03mar1997	10mar1998	10mar1998	B,B29,B
7	FR2878778A1	FR2878778A1	2878778	A1	FR	1	Granted	08dec2004	30may2005	09jun2006	.	B,B29,B
8	EP1152101A1	EP1152101A1	1152101	A1	EP	1	Granted	05may2000	04may2001	07nov2001	.	
9	WO2001074566A1	WO2001074566A1	2.001e+09	A1	WO	0		30mar2000	30mar2000	11oct2001	.	B,B29,B
10	JPS6353923B2	JPS6353923B2	6353923	B2	JP	1		15oct1979	14oct1980	26oct1988	26oct1988	B,B29,B

Vars: 25 Order: Dataset Obs: 10 Length: 14 Filter: Off

To do list

- ASCII regular expressions have limited capabilities in Stata by comparison to Perl and POSIX, plus they are not well documented - StataCorp people please note the small grumble.

To do list

- ASCII regular expressions have limited capabilities in Stata by comparison to Perl and POSIX, plus they are not well documented - StataCorp people please note the small grumble.
- We have recently discovered that Unicode regular expressions have slightly increased capability, e.g. they can do conditional lookahead assertions which is very useful for extracting repeated strings as in `itemprop=="inventor"` and `itemprop=="classification"`. Thus, migrating from ASCII to Unicode regular expressions would simplify our code considerably.

To do list

- ASCII regular expressions have limited capabilities in Stata by comparison to Perl and POSIX, plus they are not well documented - StataCorp people please note the small grumble.
- We have recently discovered that Unicode regular expressions have slightly increased capability, e.g. they can do conditional lookahead assertions which is very useful for extracting repeated strings as in `itemprop=="inventor"` and `itemprop=="classification"`. Thus, migrating from ASCII to Unicode regular expressions would simplify our code considerably.
- At this stage, **getpatent** requires access to a list of patent codes to get to the URLs. The ultimate aim is to design **getpatent** to require access to only 1 patent code and then build a database by expanding forwards and backwards to all patents that are cited *ad infinitum*, or at a cut-off point.