Lecture 2: Web Mining and Corpora

Aurelia Power

Hons. Degree in Computing

H4016 Text Analysis - 2024

Mining Data from the Web

The topic of web mining includes a number of data sources:

1. **Web Content Mining:** Text content on websites
   - Standard sites
   - Blogs
   - News groups and discussions
   - Software
   - Multimedia content on web sites
2. **Web Usage Mining**
   - Web logs generated from web activity
   - Associated databases recording additional information about that activity.
3. **Web Structure Mining / Web Link Mining**
   - Analysis of links between web pages

Web Mining Application Areas

- Product Mining: automated scanning of a range of sites for product information and price comparisons.
- Blog & News mining: analyse blog content to find out what topics are important, and how people feel about them.
- Sentiment analysis: analyse the content of reviews to discover what people are saying about a product or service.
- Website optimisation: analyse usage patterns on your website ➔ Design pages and links to optimise the user experience, and optimise sales.

Web Content Mining

- Today we will cover:
  - **Web crawlers and extracting text data from web pages**
  - **Creating a labelled text-based corpus**

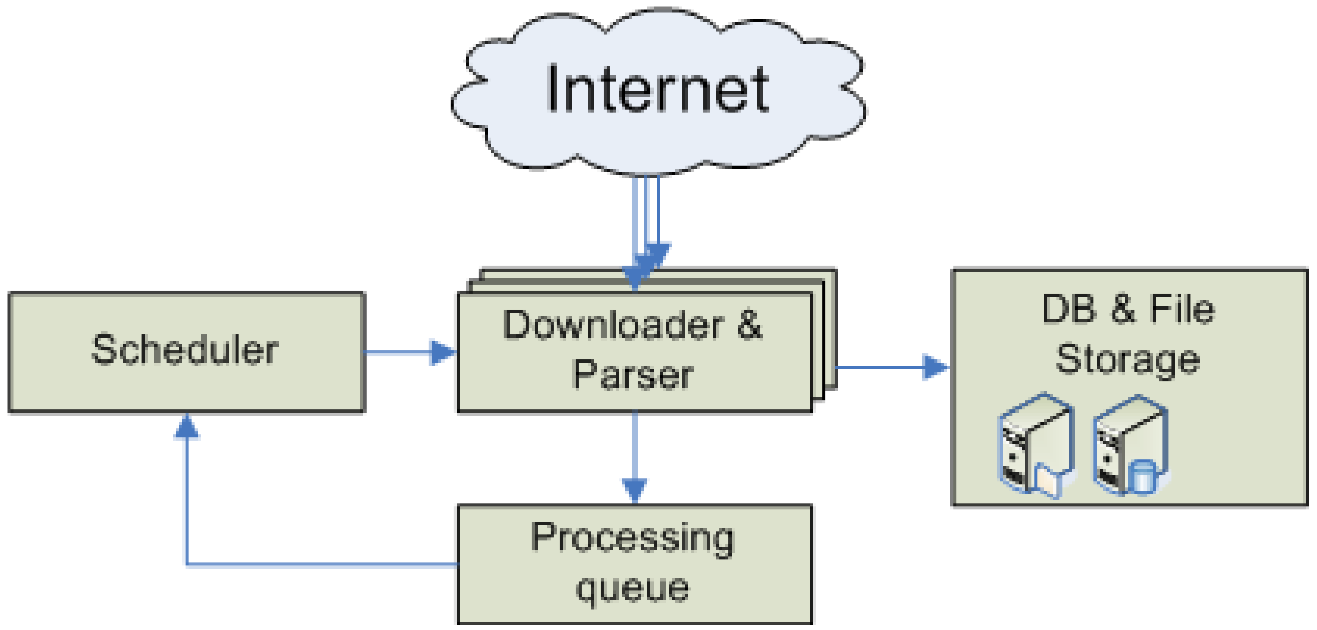
- The aim of ***Web Content Mining*** is to extract knowledge from data contained within the content of web pages

- **There are several stages/steps of web mining:**
  1. Crawl the web to download data (web pages)
  2. Extract relevant information from the web pages
  3. Pre-process the data for data mining
  4. Mine the data

Step 1: Web Crawling

- Web crawling is used to access the content on the web.
- This is done using web crawlers: programs that automatically download web pages;
  - Also known as spiders, robots, User Agent, wanderer or bots
- A web crawler can visit many different sites to collect information.
- This is often a never-ending process due to the rapid change and growth of the world wide web.
- Well known crawlers used by the search engine include googlebot, bingbot and yahoobot.

High Level Workflow of Web Crawlers



1. Get next URL to visit from the processing queue
2. Download the web document
3. Parse document's content to extract the set of URL links to other resources and update processing queue
4. Store web document for further processing

Web Crawling Policies

Web crawlers generally implement several **basic policies**:

1. **Selection policy** - strategy for URL selection from processing queue.

- This determines the order the order in which URLs are processed and crawled based on factors such as relevance, importance, or recency.
- It also determines the depth and breadth of the crawling process, employing algorithms based on simple breadth-first approach, or extended to more sophisticated evaluation of URLs, such as back-link count or page rank. :
  - Depth refers to the number of levels of links to follow from the seed URLs
  - Breadth refers to the number of links to explore at each level.

Web Crawling Policies

2. Revisit policy - to determine how often a URL should be revisited for updates.

There are several approaches for revisiting a URL, including:

- Time-Based: Set time intervals, after which previously crawled pages should be revisited, regardless of whether they have changed or not. This approach ensures a regular update of the crawled data but may result in unnecessary revisits if the content hasn't been modified.
- Delta-Based: uses delta-based mechanisms to track changes in web pages. It can compare the current version of a page with the previously crawled version and determine if there are significant differences or updates. If the changes exceed a predefined threshold, the page is considered for recrawling.
- Priority-Based: assigns priority levels to web pages based on their importance or significance. Higher-priority pages are revisited more frequently than lower-priority pages. The priority can be determined by factors such as the popularity of the page, user demand, or the importance of the content.
- Dynamic Discovery -Based: uses mechanisms to dynamically discover changes on web pages during the crawling process. This can involve techniques such as comparing checksums, tracking version numbers, or monitoring RSS feeds or APIs for updates. Dynamic discovery helps identify pages that require recrawling.

Web Crawling Policies

3. **Politeness policy** - to limit number of requests to the same website in a given period to prevent bottleneck effect on crawling domain.

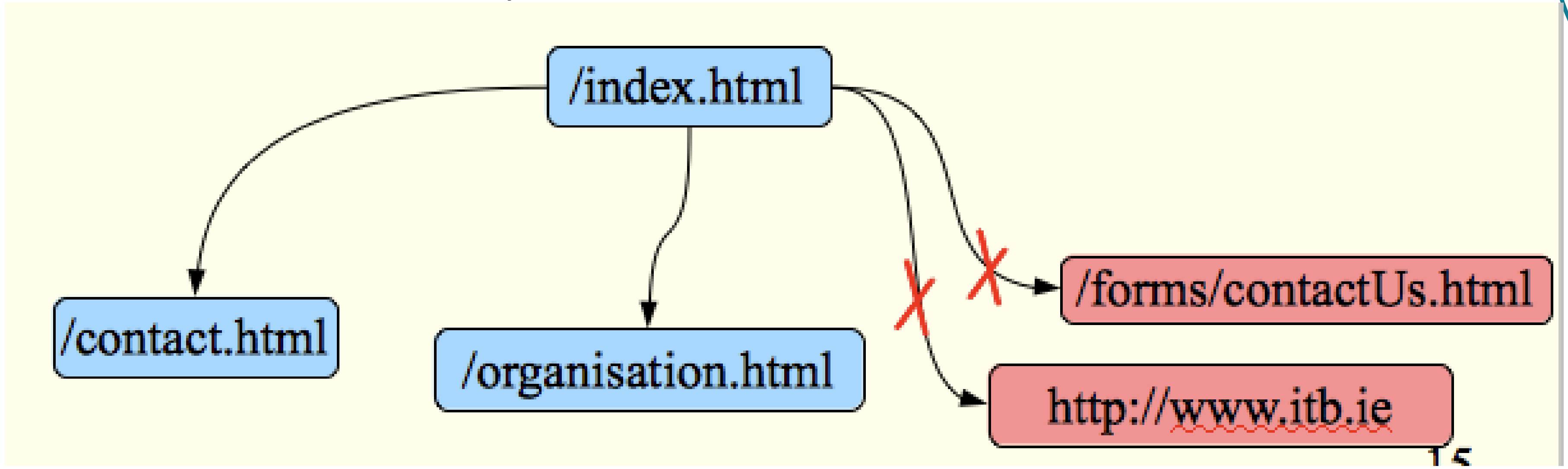
- There are several aspects to consider which are typically embedded in a Robots.txt; these include:
- Crawl delay – the time interval between consecutive requests made by the crawler to the same website is set so that it prevents overloading.
- Throttling - limiting the rate at which requests are sent to a specific website
- Bandwidth - avoiding excessive data transfer. The crawler should not download unnecessarily large files or make requests for resource-intensive content unless it is relevant to the crawling objectives.
- User-Agent Identification- use a descriptive and meaningful User-Agent header when making requests to the target website.

4. **Parallelization policy** – to coordinate the work of distributed web crawlers and to provide maximal download rate of crawling method. Aspects considered include:

- Task partitioning - dividing the crawling task into smaller units, such as URLs or batches of URLs, that can be processed independently
- Concurrent Requests - multiple requests to be made concurrently to different websites or server, so that the crawler can retrieve content from multiple sources in parallel, reducing the overall crawling time.
- Thread Synchronization – to ensure access to shared data or resources maintains data integrity.
- Resource Allocation –allocating resources, such as CPU, memory, and network bandwidth, among the concurrent tasks or threads so that each task or thread receives an appropriate share of resources to maintain optimal performance and avoid resource contention.

Web Crawling for Text Mining

- Information is generally scattered over many pages.
- A web crawler traverses the graph of linked pages to access the content.
- For a text analytics project, it is not feasible to gather all content on the web; the web crawling needs to be focused on specific pages which are likely to contain useful information. This can be done by:
  - Limiting the crawler to specific **domains**
  - Limiting the crawler to specific **depths** (the number of slashes in a URL from its site root)



Crawler Ethics and Rules

- Crawlers can upset web servers and administrators:
  - E.g., sending too many requests in quick succession can be seen as a Denial-of-Service attack.
  - This can lead to blacklisting of the crawlers IP.

✓ You need to identify yourself.

✓ Alternate requests between different hosts.

✓ Look out for re-direction loops.

✓ Look out for spider traps (also known as crawler traps), which are similar to an infinite loop, e.g., a calendar - dynamically creates links to the next day/month/year.

✓ Honour the robot.txt file which contains information as to whom can access which files/folders:

A recent study based on a large-scale analysis of robots.txt files showed that certain web crawlers were preferred over others, with Googlebot being the most preferred web crawler.

Web Crawlers: **the Robots.txt File**

```
....
User-agent: Googlebot
User-agent: Mediapartners-Google
User-agent: Adsbot-Google
User-agent: slurp
 . . . .
User-agent: FeedFetcher

Crawl-delay: 2
Disallow: /covers/
. . .
Allow: /product-search?cps=*&facet=type__book$
Allow: /product-search?cps=*&facet=type__journal$
 . . .
# extra lines for bing bot only
User-agent: bingbot
Crawl-delay:0
  . .
# all others
User-agent: *
Disallow: /
```

The crawlers that can access the content…

Time delay between requests.

Can not enter this directory

Directories that can be searched

Rules that apply to certain crawler(s).

All other agents are not allowed to crawl the site

You are not legally bound to robot.txt files, but you may be blocked if you ignore them.

check out http://www.springer.com/robots.txt

Type of Crawlers

1. Universal Crawler

- They are wanderers designed to gather all pages irrespective of their content.
- NOTE: achieving true universality is extremely difficult.
- Large Scale.
- Huge Cost in terms of network bandwidth.
- Incremental updates to existing data repositories.
- Heavily indexed.

- **2 important issues related to universal crawlers:**
  - Performance – need to crawl billions of objects.
  - Policy – what to prioritise in terms of coverage, freshness and bias towards 'important' pages (importance is subjective).
  - We will not use a universal crawler…

Type of Crawlers

2. Preferential Crawlers

- These crawlers are more targeted and only download pages of a certain topic or type.
- The crawler works with a priority queue rather than the FIFO (first in first out) concept.

❑ There are 2 main types of preferential crawlers:

1. **Topical crawlers:** Preferential crawlers that gather pages as they are needed (real time) based on set of seed pages or URLs.

2. **Focused crawlers**: Preferential crawlers that gather pages in advance that are focused on particular topics of interest… this is the one that we will be using.

Step 2: Extract relevant information from webpages

- Pages downloaded by the web crawler can be mined as a text file, as it is done with all unstructured documents.
- We can extract specific information that we are interested in from those pages:
  - e.g., product price, company name, etc.
- We can use XLML or regex, or both:
  - XLML – which identifies text based on its position within XML or HTML tags, so we can extract sections from a web page that we are interested in, such as paragraphs (p).
  - Regular expression – which identifies text based on the surrounding text or providing a matching template for the text itself.
- We can also use Xpath and other libraries such as selenium (not covered here).

Step 1 and 2 in Python…

- We will use the **requests** library (https://requests.readthedocs.io/en/master/) which is a simple HTTP library for python
- We will also use **Beautiful Soup** which is a library that makes it easy to scrape information from web pages.
  - It sits on top of an HTML or XML parser, providing Pythonic idioms for iterating, searching, and modifying the parse tree (https://pypi.org/project/beautifulsoup4/ or https://www.crummy.com/software/BeautifulSoup/bs4/doc)

```
## once imported, we retrieve the text of the page using the specified url and use it to construct a beautiful soup object;

url= "https://en.wikipedia.org/wiki/Global_warming_controversy"

page = requests.get(url)

all_data = page.text;

 soup = bs4.BeautifulSoup(all_data, 'html.parser')
```

Step 1 and 2 in Python…

- Other options for the second argument of the BeautifulSoup constructor: html5lib, lxml, etc.
- The soup object allows us to extract individual html elements using its named attributes

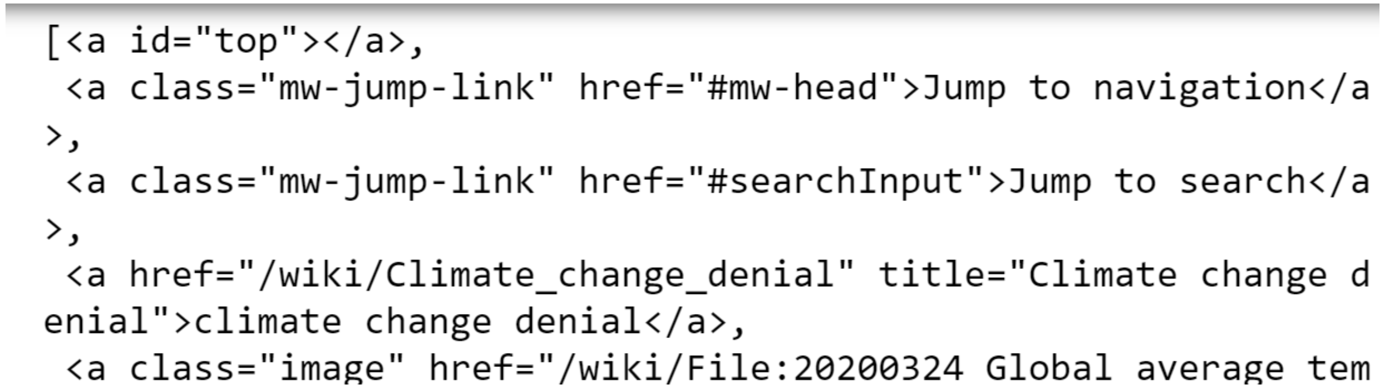
soup.**title** ## retrieves the element title
<title>Global warming controversy - Wikipedia</title>

soup.**title.get_text()** ## retrieves the text of the element title
‘Global warming controversy – Wikipedia’

soup.**a** ## retrieves the first anchor element: <a id=“top”></a>

- The soup object allows us also to extract groups of same html elements using the method find_all which returns a list of those elements

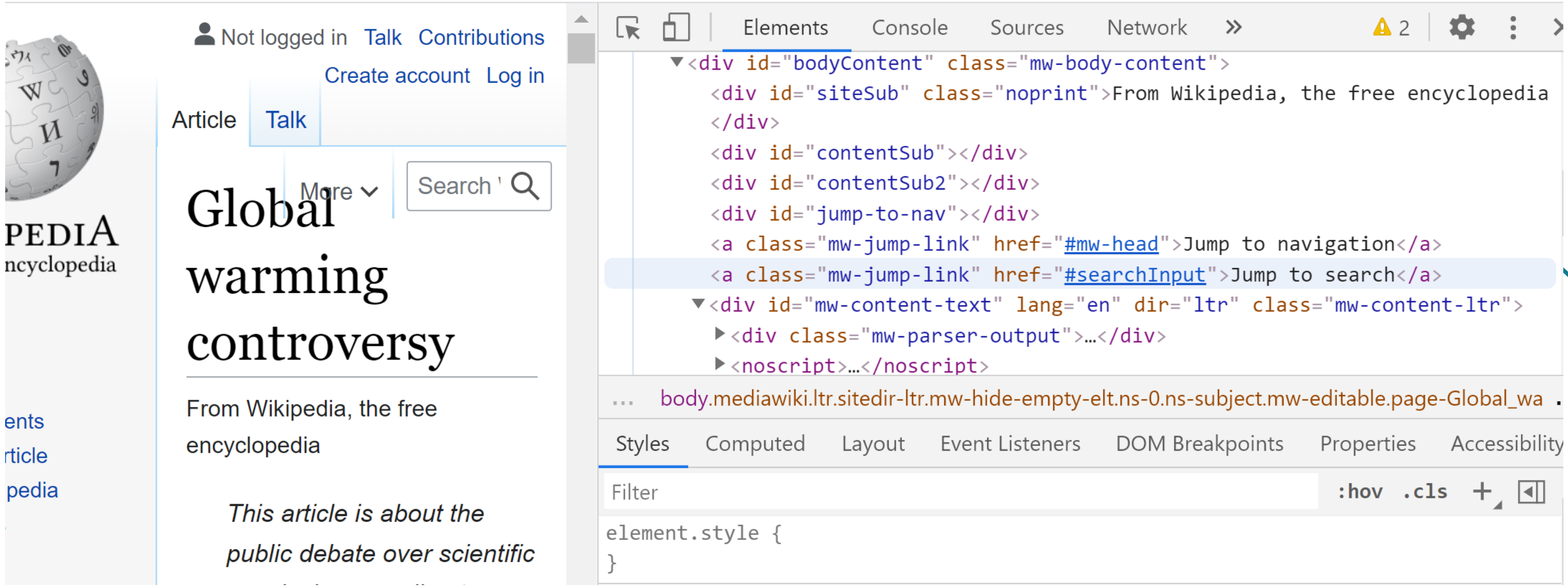
soup.find_all(‘a')## retrieves all anchors and puts them into a list



```
[<a id="top"></a>,
 <a class="mw-jump-link" href="#mw-head">Jump to navigation</a
>,
 <a class="mw-jump-link" href="#searchInput">Jump to search</a
>,
 <a href="/wiki/Climate_change_denial" title="Climate change d
enial">climate change denial</a>,
 <a class="image" href="/wiki/File:20200324 Global average tem
```

Step 1 and 2 in Python…

- Note: Retrieving and processing relevant elements from a page is highly dependent on the html code for that page.
- It is therefore paramount that we use the developer tools to understand the structure of a website page and identify elements that are relevant:



Not logged in Talk Contributions
Create account Log in
Article Talk
More
Search
Global warming controversy
From Wikipedia, the free encyclopedia
This article is about the public debate over scientific
Elements Console Sources Network
Styles Computed Layout Event Listeners DOM Breakpoints Properties Accessibility
Filter :hov .cls

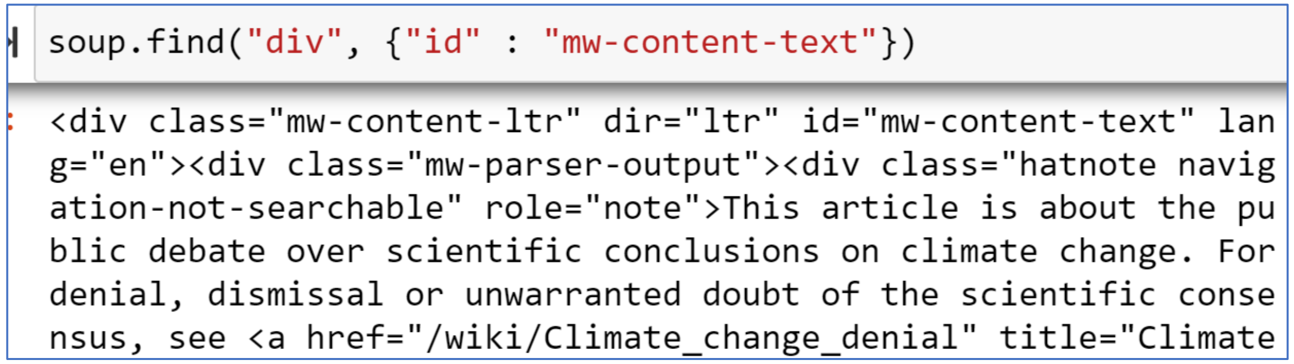
```
▼<div id="bodyContent" class="mw-body-content">
   <div id="siteSub" class="noprint">From Wikipedia, the free encyclopedia
   </div>
   <div id="contentSub"></div>
   <div id="contentSub2"></div>
   <div id="jump-to-nav"></div>
   <a class="mw-jump-link" href="#mw-head">Jump to navigation</a>
   <a class="mw-jump-link" href="#searchInput">Jump to search</a>
  ▼<div id="mw-content-text" lang="en" dir="ltr" class="mw-content-ltr">
    ▶<div class="mw-parser-output">…</div>
    ▶<noscript>…</noscript>
```

```
body.mediawiki.ltr.sitedir-ltr.mw-hide-empty-elt.ns-0.ns-subject.mw-editable.page-Global_wa
```

```
element.style {
}
```

Step 1 and 2 in Python…

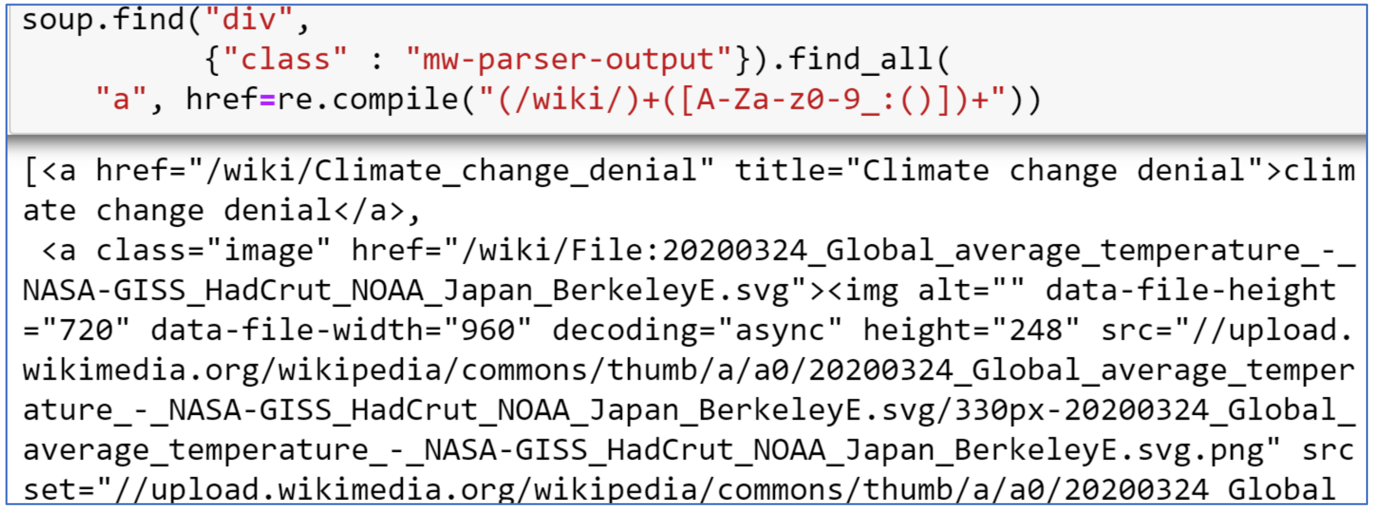
- We also have a **find** function that will allow us to search for a specific element by id or class:



```
soup.find("div", {"id" : "mw-content-text"})
```

```
<div class="mw-content-ltr" dir="ltr" id="mw-content-text" lan
g="en"><div class="mw-parser-output"><div class="hatnote navig
ation-not-searchable" role="note">This article is about the pu
blic debate over scientific conclusions on climate change. For
denial, dismissal or unwarranted doubt of the scientific conse
nsus, see <a href="/wiki/Climate_change_denial" title="Climate
```

- We can combine **find and find_all** to retrieve all containing elements in the given div(s) or section(s): for instance, we want all the local anchors from the wiki page in the divs that belong to the class "mw-parser-output"



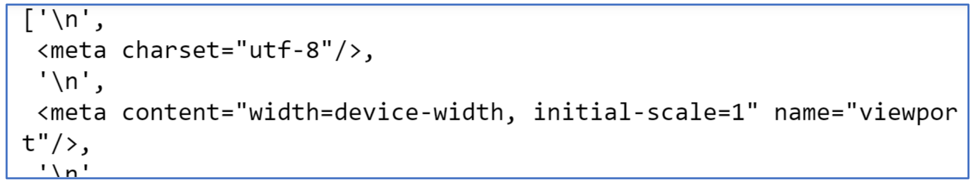
```
soup.find("div",
          {"class" : "mw-parser-output"}).find_all(
    "a", href=re.compile("(/wiki/)+([A-Za-z0-9_:()])+"))
```

```
[<a href="/wiki/Climate_change_denial" title="Climate change denial">clim
ate change denial</a>,
 <a class="image" href="/wiki/File:20200324_Global_average_temperature_-_
NASA-GISS_HadCrut_NOAA_Japan_BerkeleyE.svg"><img alt="" data-file-height
="720" data-file-width="960" decoding="async" height="248" src="//upload.
wikimedia.org/wikipedia/commons/thumb/a/a0/20200324_Global_average_temper
ature_-_NASA-GISS_HadCrut_NOAA_Japan_BerkeleyE.svg/330px-20200324_Global_
average_temperature_-_NASA-GISS_HadCrut_NOAA_Japan_BerkeleyE.svg.png" src
set="//upload.wikimedia.org/wikipedia/commons/thumb/a/a0/20200324_Global
```

Step 1 and 2 in Python…

- We can also access the contents of an element using the **contents** attribute:

soup.head.contents



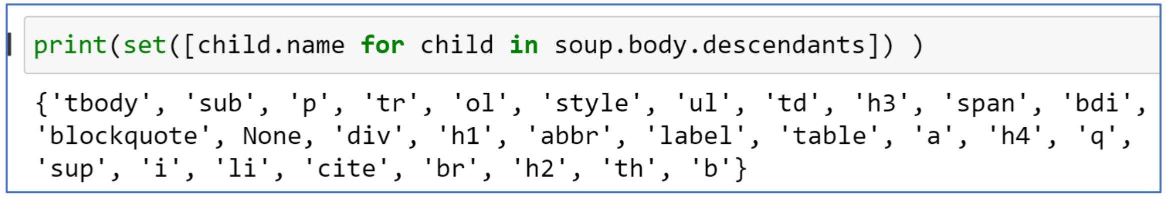
```
['\n',
 <meta charset="utf-8"/>,
 '\n',
 <meta content="width=device-width, initial-scale=1" name="viewpor
t"/>,
 '\n'
```

- We can go up and down the html tree using **parent** and **children** attributes, respectively:

soup.body.parent.name ## the name of the parent element of the body is html

set([child.name for child in soup.body.children]) ## {None, 'div', ‘footer', 'script'}

- Note that children attribute only provides the direct descendant elements; to get all descending elements, we can use the attribute **descendants**:



```
print(set([child.name for child in soup.body.descendants]) )

{'tbody', 'sub', 'p', 'tr', 'ol', 'style', 'ul', 'td', 'h3', 'span', 'bdi',
'blockquote', None, 'div', 'h1', 'abbr', 'label', 'table', 'a', 'h4', 'q',
'sup', 'i', 'li', 'cite', 'br', 'h2', 'th', 'b'}
```

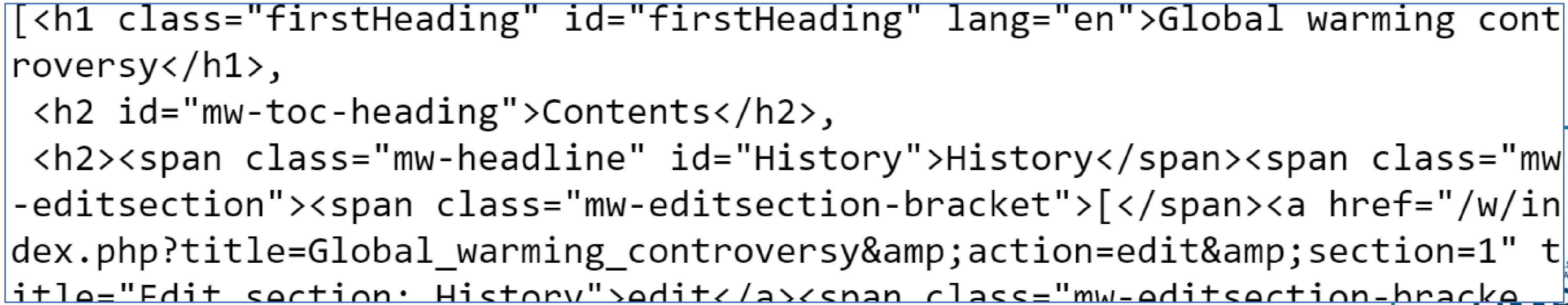
- We can also go sideways using **find_next_siblings(), find_next_sibling(), find_previous_siblings(),** and **find_previous_sibling()** -> check out the documentation
- **NOTE:** each of these will returns different results, depending on the website you are working with.

[

Step 1 and 2 in Python…

## We can retrieve elements using specific tags, for instance all levels of h

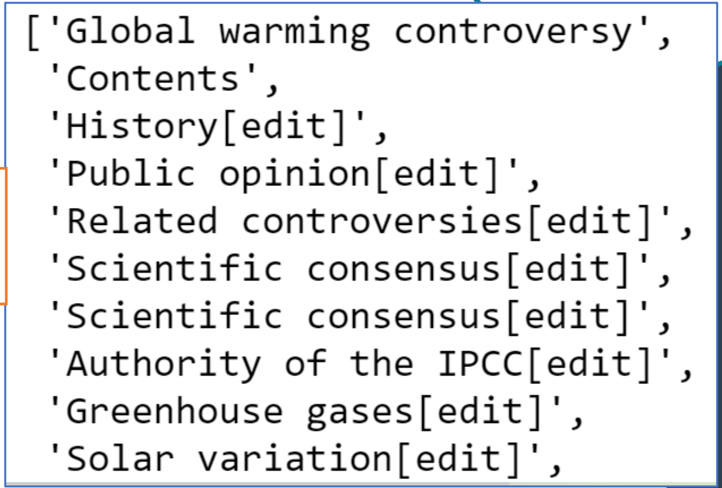
soup.find_all(re.compile('^h[1-6]’))



```
[<h1 class="firstHeading" id="firstHeading" lang="en">Global warming cont
roversy</h1>,
 <h2 id="mw-toc-heading">Contents</h2>,
 <h2><span class="mw-headline" id="History">History</span><span class="mw
-editsection"><span class="mw-editsection-bracket">[</span><a href="/w/in
dex.php?title=Global_warming_controversy&amp;action=edit&amp;section=1" t
itle="Edit section: History">edit</a><span class="mw-editsection-bracke
```

## We can then convert them to strings and remove the tags to keep only the text

```
hs = [str(h) for h in soup.find_all(re.compile('^h[1-6]'))]
hs = [re.sub(r'<.+?>', '', h) for h in hs]
```



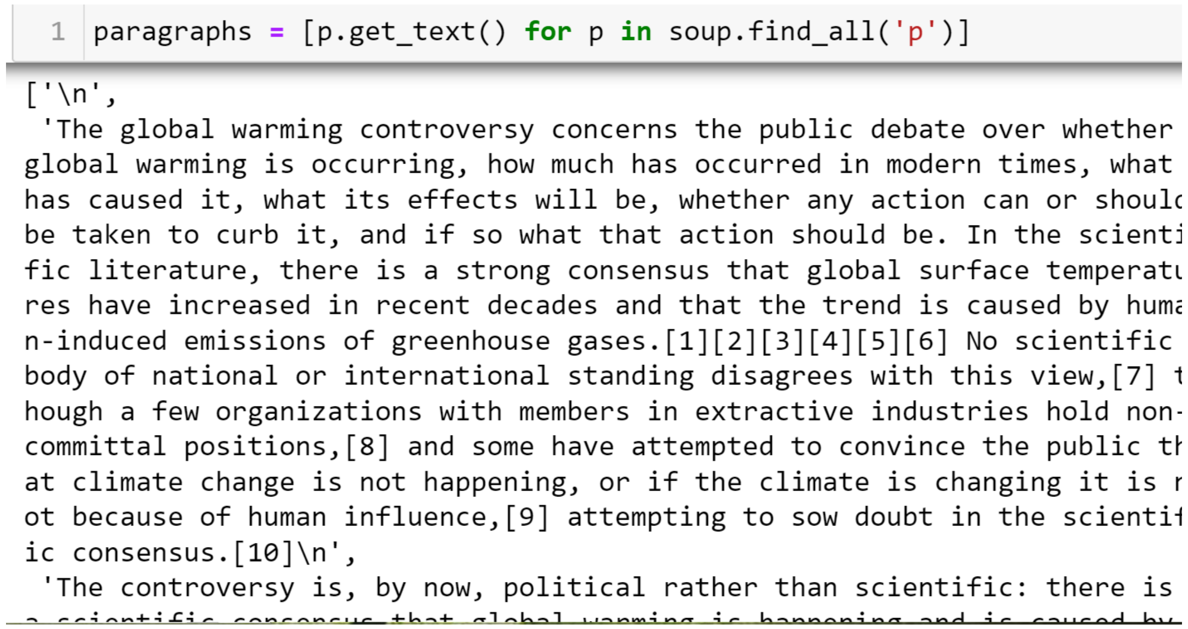
```
['Global warming controversy',
 'Contents',
 'History[edit]',
 'Public opinion[edit]',
 'Related controversies[edit]',
 'Scientific consensus[edit]',
 'Scientific consensus[edit]',
 'Authority of the IPCC[edit]',
 'Greenhouse gases[edit]',
 'Solar variation[edit]',
```

Why did I use the reluctant/non-greedy quantifier **‘?’**

<u>Exercise</u>: we can use the function **get_text().** Check out its documentation and write code to obtain the same output as on the right.

Step 1 and 2 in Python…

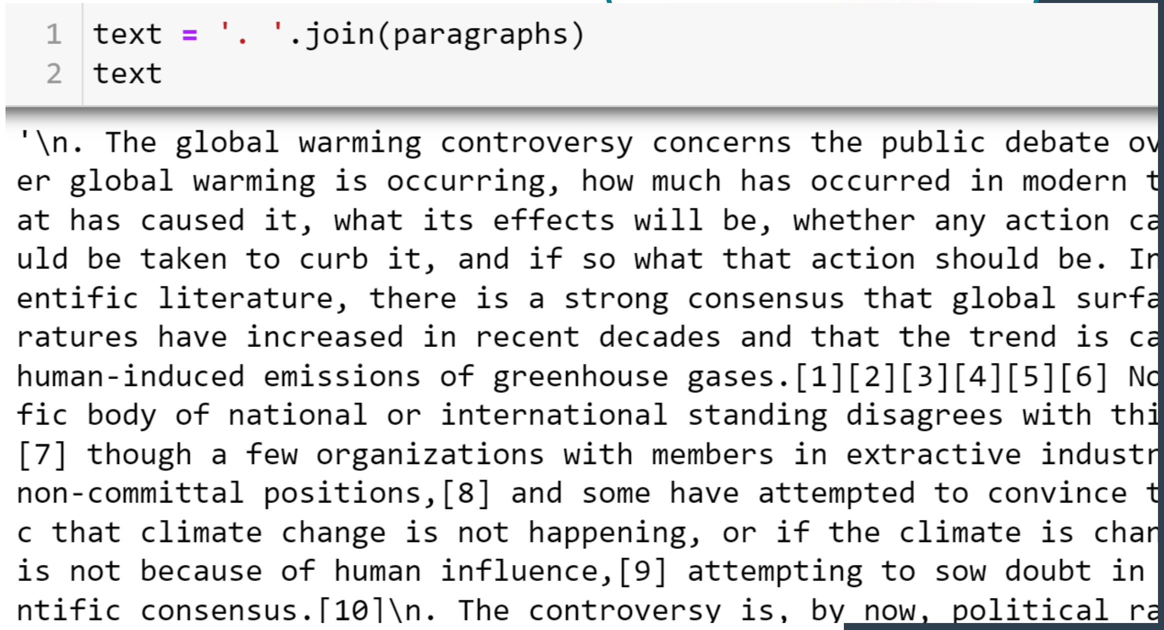
- The most important elements for an article are the p elements as they contain the text of an article.
- We should isolate them and get their text:



```
paragraphs = [p.get_text() for p in soup.find_all('p')]
```

```
['\n',
 'The global warming controversy concerns the public debate over whether
global warming is occurring, how much has occurred in modern times, what
has caused it, what its effects will be, whether any action can or should
be taken to curb it, and if so what that action should be. In the scienti
fic literature, there is a strong consensus that global surface temperatu
res have increased in recent decades and that the trend is caused by huma
n-induced emissions of greenhouse gases.[1][2][3][4][5][6] No scientific
body of national or international standing disagrees with this view,[7] t
hough a few organizations with members in extractive industries hold non-
committal positions,[8] and some have attempted to convince the public th
at climate change is not happening, or if the climate is changing it is n
ot because of human influence,[9] attempting to sow doubt in the scientif
ic consensus.[10]\n',
 'The controversy is, by now, political rather than scientific: there is
a scientific consensus that global warming is happening and is caused by
```

- Then join all elements of the list into a single text; optionally, we can use '. ' to maintain sentence boundaries or simply ' ' ➔ we now have our article/document



```
text = '. '.join(paragraphs)
text
```

```
'\n. The global warming controversy concerns the public debate ov
er global warming is occurring, how much has occurred in modern t
at has caused it, what its effects will be, whether any action ca
uld be taken to curb it, and if so what that action should be. In
entific literature, there is a strong consensus that global surfa
ratures have increased in recent decades and that the trend is ca
human-induced emissions of greenhouse gases.[1][2][3][4][5][6] No
fic body of national or international standing disagrees with thi
[7] though a few organizations with members in extractive industr
non-committal positions,[8] and some have attempted to convince t
c that climate change is not happening, or if the climate is chan
is not because of human influence,[9] attempting to sow doubt in
ntific consensus.[10]\n. The controversy is, by now, political ra
```

Step 1 and 2 in Python…

- Another approach to keep only relevant elements is to use the **extract()** method.

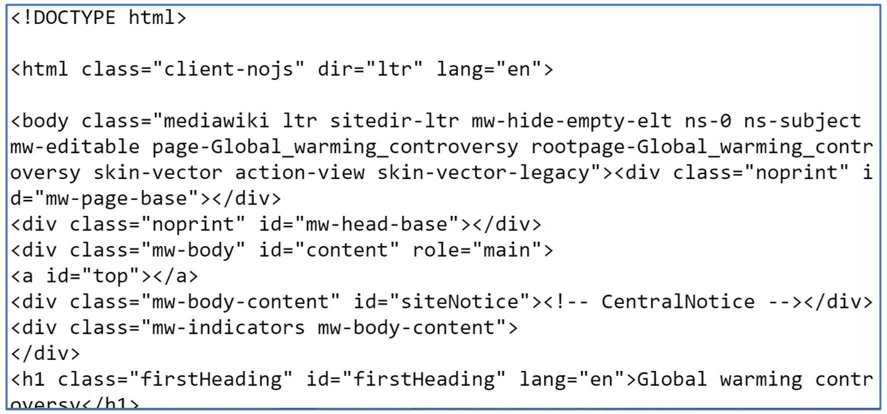
- The extract method removes from the html tree the elements specified;
- For instance, assuming the original soup, we want to remove elements whose text are not part of the main article :

tags_to_remove = ['figcaption', 'link', 'input', 'script', 'title', 'figure', 'head', 'form', 'meta', 'noscript', 'iframe','nav', 'footer', 'img', 'button', 'style']

- The list can be expanded: check out the html documentation to see what elements are not relevant for the task at hand;

[s.extract() for s in soup(tags_to_remove)]

- Now, when we print the soup, we see that those elements are no longer there.



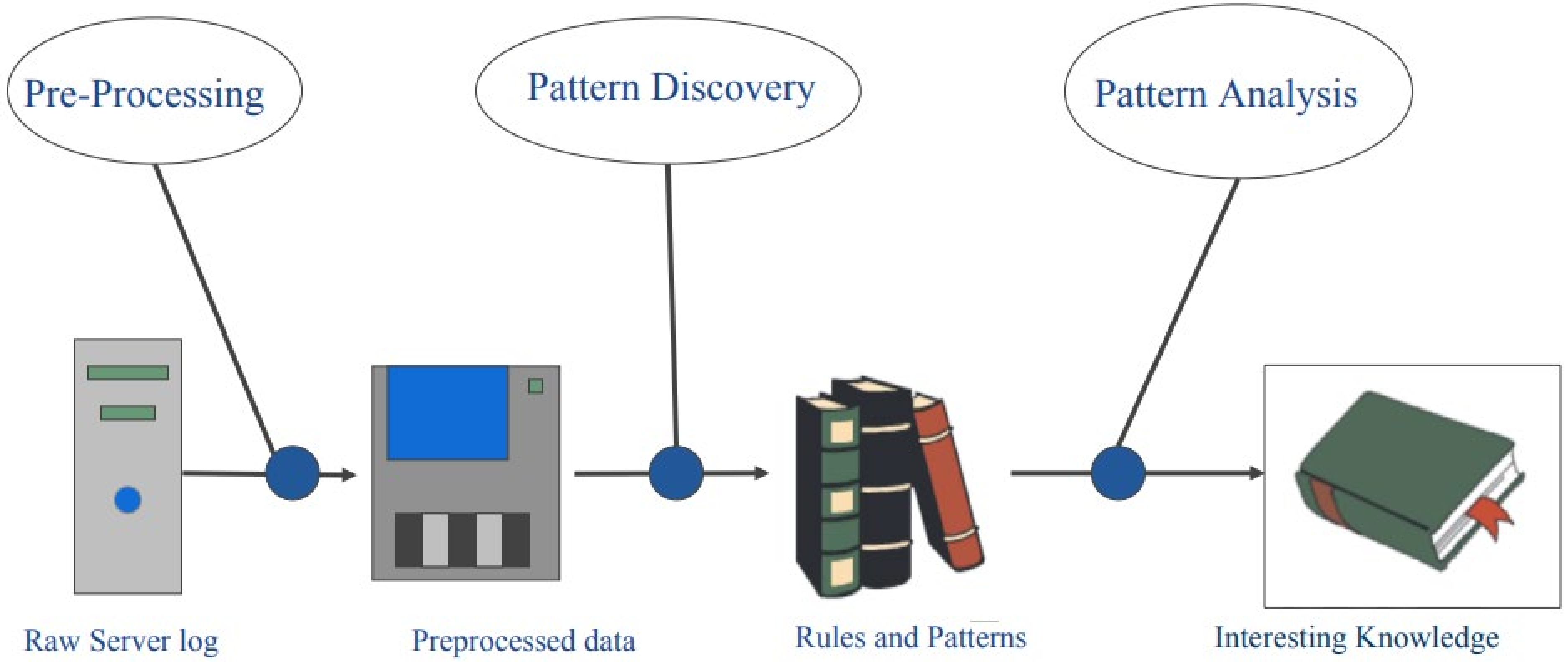
```
<!DOCTYPE html>

<html class="client-nojs" dir="ltr" lang="en">

<body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject
mw-editable page-Global_warming_controversy rootpage-Global_warming_contr
oversy skin-vector action-view skin-vector-legacy"><div class="noprint" i
d="mw-page-base"></div>
<div class="noprint" id="mw-head-base"></div>
<div class="mw-body" id="content" role="main">
<a id="top"></a>
<div class="mw-body-content" id="siteNotice"><!-- CentralNotice --></div>
<div class="mw-indicators mw-body-content">
</div>
<h1 class="firstHeading" id="firstHeading" lang="en">Global warming contr
oversy</h1>
```

NOTE: We can also combine the 2 approaches;
For instance, if we want to remove the nested script, styles and img elements completely (not only the tags) from any paragraph.

Web Usage Mining

- Web usage Mining is the automated discovery and analysis of patterns in click stream or other user access data.
- Aims at the discovery of user access patterns from web usage logs.
  - Web server logs data
- Generally, every click is recorded (known as **clickstream**).
- Often additional information of a user is known such as
  - Personal details or
  - General demographic information (e.g. population/regions average salary, average age, ethnicity)
  - A lot of work (as usual) is put into the pre-processing of the data to get it into the correct format.

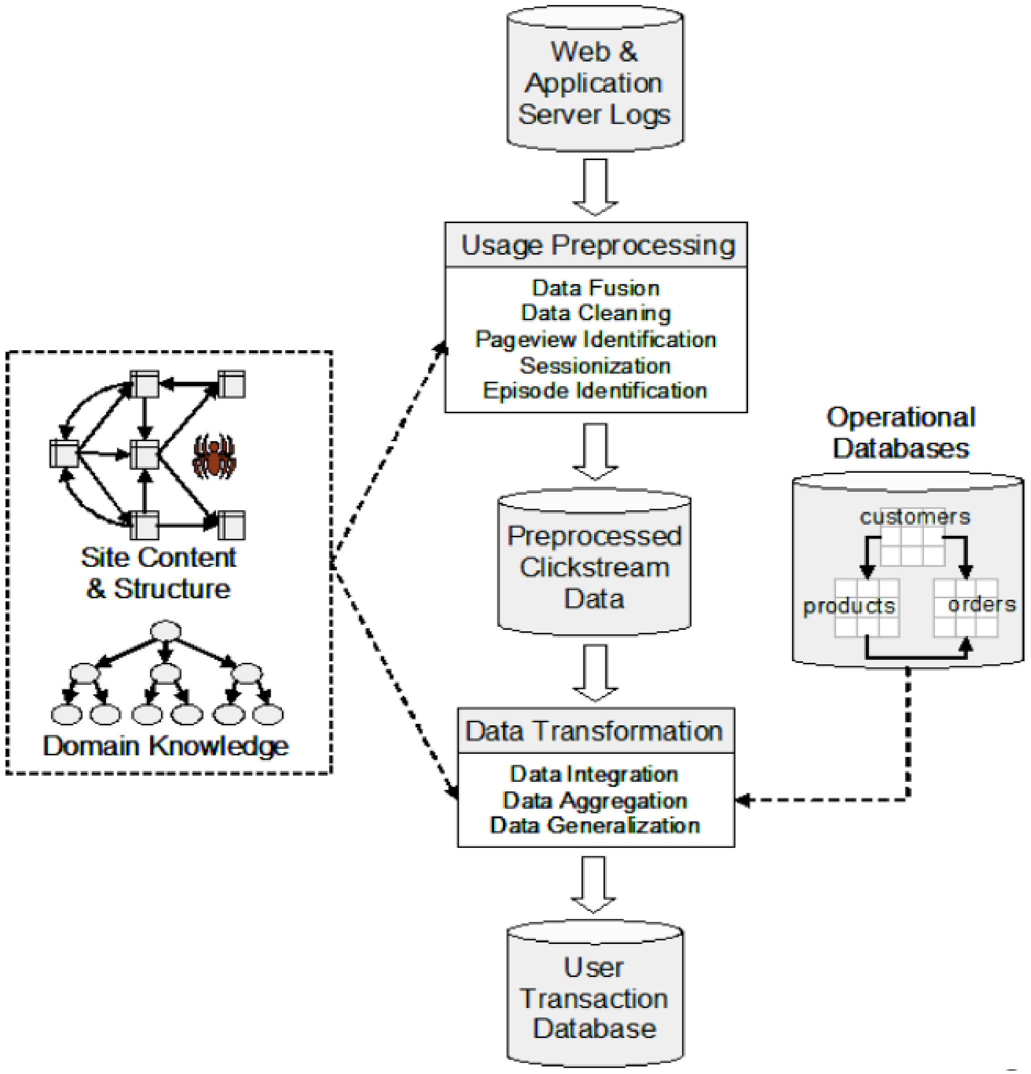
Web Usage Mining



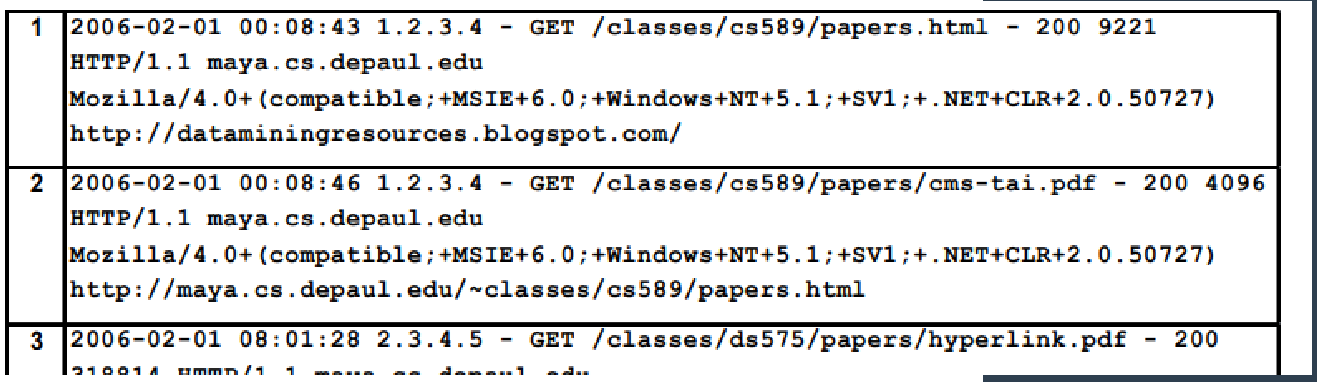
Three Main Phases:

1. **Data Collection and Preparation**
2. Pattern Discovery
3. Pattern Analysis

Web Usage Mining Process – Data collection and preparation



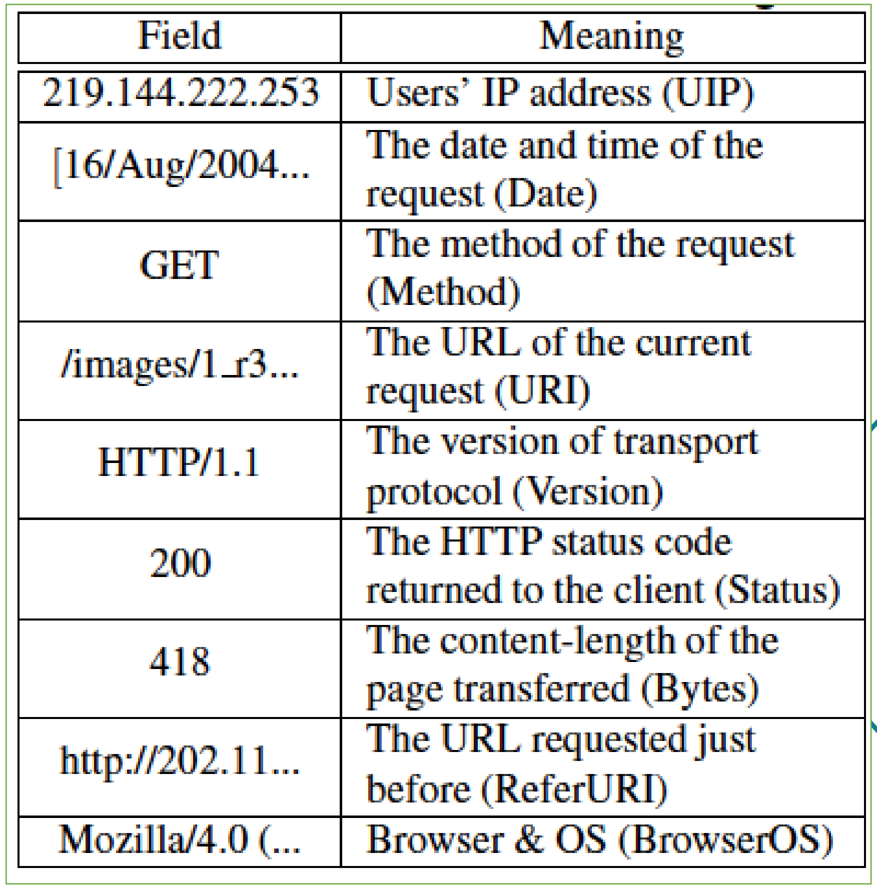
- A large part of web usage is about processing usage (clickstream data) retrieved from web server logs:



```
1  2006-02-01 00:08:43 1.2.3.4 - GET /classes/cs589/papers.html - 200 9221
   HTTP/1.1 maya.cs.depaul.edu
   Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1;+.NET+CLR+2.0.50727)
   http://dataminingresources.blogspot.com/
2  2006-02-01 00:08:46 1.2.3.4 - GET /classes/cs589/papers/cms-tai.pdf - 200 4096
   HTTP/1.1 maya.cs.depaul.edu
   Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1;+.NET+CLR+2.0.50727)
   http://maya.cs.depaul.edu/~classes/cs589/papers.html
3  2006-02-01 08:01:28 2.3.4.5 - GET /classes/ds575/papers/hyperlink.pdf - 200
   318814 HTTP/1.1 maya.cs.depaul.edu
```

Web Usage Mining - Attributes

- In general, log files include:
  - IP address
  - ClientID
  - UserID
  - Date, Time & Time zone
  - Request
  - Status
  - Bytes
- And may also include:
  - Protocol
  - Method
  - Referrer
  - ClientBrowser/Agent



Field	Meaning
219.144.222.253	Users' IP address (UIP)
[16/Aug/2004...	The date and time of the request (Date)
GET	The method of the request (Method)
/images/1_r3...	The URL of the current request (URI)
HTTP/1.1	The version of transport protocol (Version)
200	The HTTP status code returned to the client (Status)
418	The content-length of the page transferred (Bytes)
http://202.11...	The URL requested just before (ReferURI)
Mozilla/4.0 (...	Browser & OS (BrowserOS)

There are a number of log formats:

Common Log Format – supported by all web servers.

Extended Log Format – provides flexibility on what fields are recorded (used by IIS and others).

Other proprietary formats

Information Available from Server Log

- **Click-through rate or CTR** = (Clicks / Impressions) * 100; so, it divides the number of clicks on a link by the number of impressions or views it receives in the web server logs ➔ provides insights into how often users click on specific links or pages after being exposed to them
- **Conversion Rate** = (Conversions / Unique Visitors) * 100; it measures the percentage of website visitors who complete a desired action or goal, often referred to as a conversion (e.g., buying a product after viewing a product) ➔ high conversion rates suggest the website is using an effective motivating strategy for users to convert.
- **New Acquisition Rate** = (New User Registrations / Total Visitors) * 100; it measures the effectiveness of acquiring and attracting new users to a website or service.
  - Needs additional information from company database to distinguish between new and existing users.
- **Acquisition Cost** = (total cost of marketing efforts during a set period of time)/(number of newly acquired users during the same period of time); lower acquisition cost indicates that the marketing efforts are more cost-effective, as it requires fewer resources to acquire each new user.
- **Click Streams**: typical paths taken by users through the website; or typical paths taken by users who make a purchase versus users who don't.
- **Entry and Exit pages**.
- **Referrer pages**.

Key elements of pre-processing

1. **Data Fusion**
   - In distributed environments, logs are often kept on different web servers. This stage merges these log files.
     - Must ensure that timestamps include time zone
2. **Data Cleaning**
   - Cleaning is site specific, but could include removal of unimportant objects (e.g., images, css), identification and removal of crawler related traffic, removing records with failed HTTP status codes.
3. **Page view Identification**
   - Identification of page views depends on the structure winthin each each page generally consists of multiple files.
     - Identify the collection of Web files/objects/resources representing a specific “user event” corresponding to a clickthrough (e.g. viewing a product page, adding a product to a shopping cart)
     - In some cases, it may be appropriate to consider pageviews at a higher level of aggregation (e.g. they may correspond to many user event related to the same concept category, like the purchase of a product on an online ecommerce site).

Key elements of pre-processing

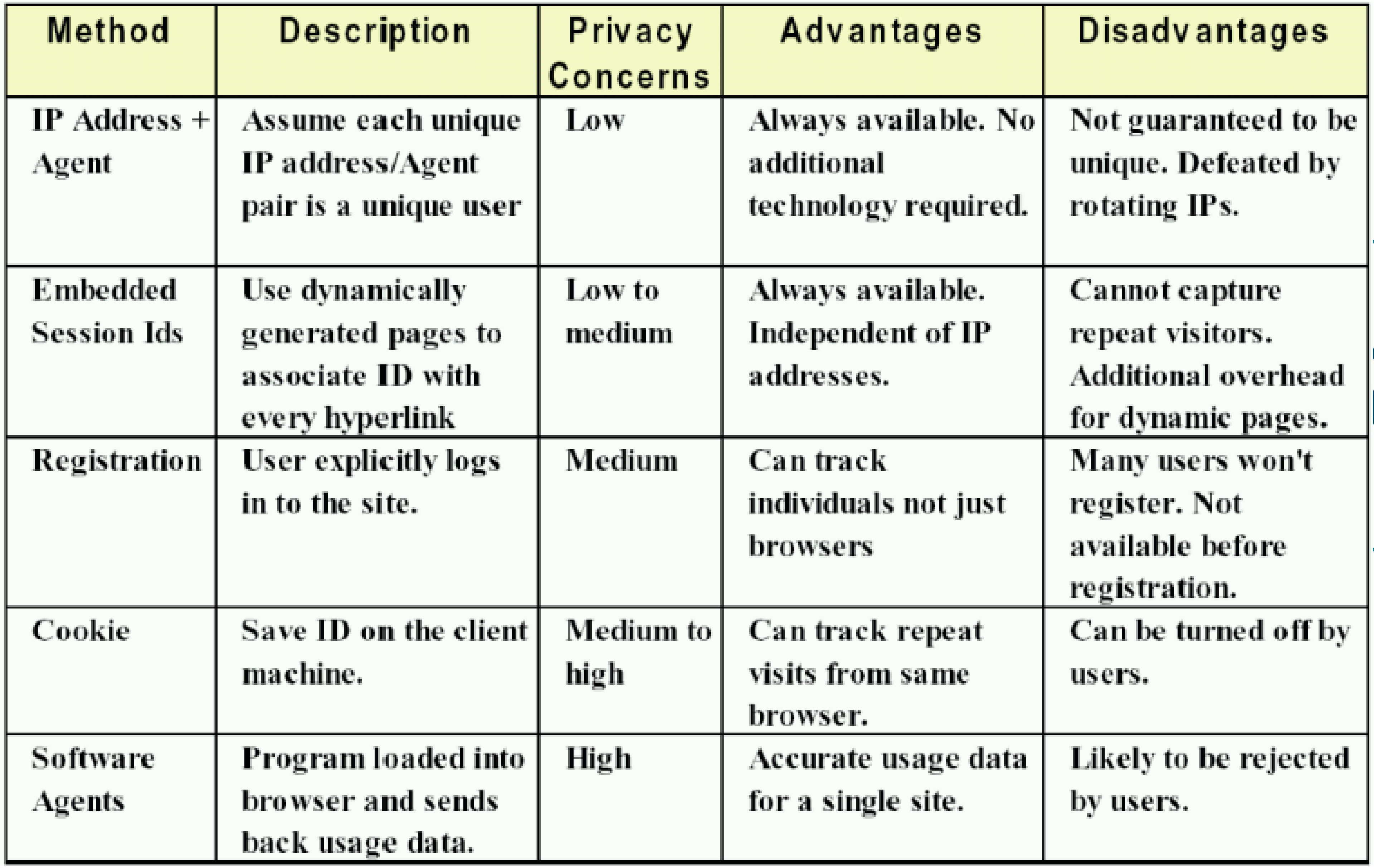
4. User Identification

- User activity record refers to the sequence of logged activities belonging to the same user.
- Cookies are often used to separate users.
- IP addresses alone are often not sufficient, but can be used in combination with other data such as the Agent who accessed the web server (e.g., IE8;Win7;SP1), the operating systems, etc.

5. Sessionisation

- Segmenting the user activity record of each user into sessions based on predefined criteria.
- It involves identifying and organising user actions performed within a specific timeframe into individual sessions, allowing for a better understanding of user behaviour and engagement patterns.
- Example: identifying navigation patterns, i.e., common paths taken by users through the website, popular entry points, and frequently visited pages.

User Identification options

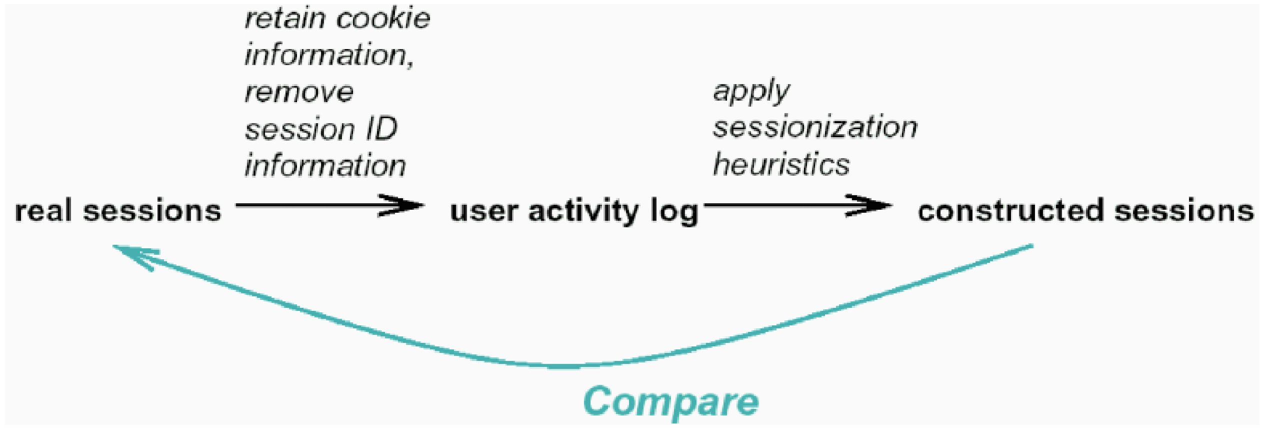


Method	Description	Privacy Concerns	Advantages	Disadvantages
IP Address + Agent	Assume each unique IP address/Agent pair is a unique user	Low	Always available. No additional technology required.	Not guaranteed to be unique. Defeated by rotating IPs.
Embedded Session Ids	Use dynamically generated pages to associate ID with every hyperlink	Low to medium	Always available. Independent of IP addresses.	Cannot capture repeat visitors. Additional overhead for dynamic pages.
Registration	User explicitly logs in to the site.	Medium	Can track individuals not just browsers	Many users won't register. Not available before registration.
Cookie	Save ID on the client machine.	Medium to high	Can track repeat visits from same browser.	Can be turned off by users.
Software Agents	Program loaded into browser and sends back usage data.	High	Accurate usage data for a single site.	Likely to be rejected by users.

Examples of rules for session identification

1. The different IP addresses distinguish different users;
2. If the IP addresses are same, the different browsers and operation systems indicate different users;
3. If all of the IP address, browsers and operating systems are same, the referer information should be taken into account. The ReferURI field is checked, and a new user session is identified if the URL in the ReferURI field hasn't been accessed previously, or there is a large interval (usually more than 10 seconds [4]) between the accessing time of this record and the previous one if the ReferURI field is empty;

OR . . .
Use cookies to store sessionID/customerID.


retain cookie information, remove session ID information
apply sessionization heuristics
real sessions
user activity log
constructed sessions
Compare

Key elements of pre-processing

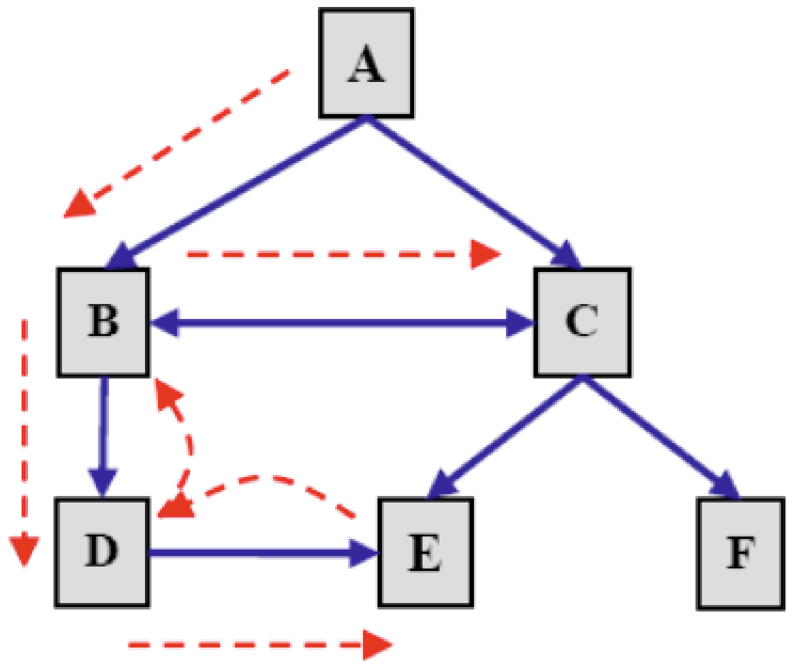
6. Path Completion

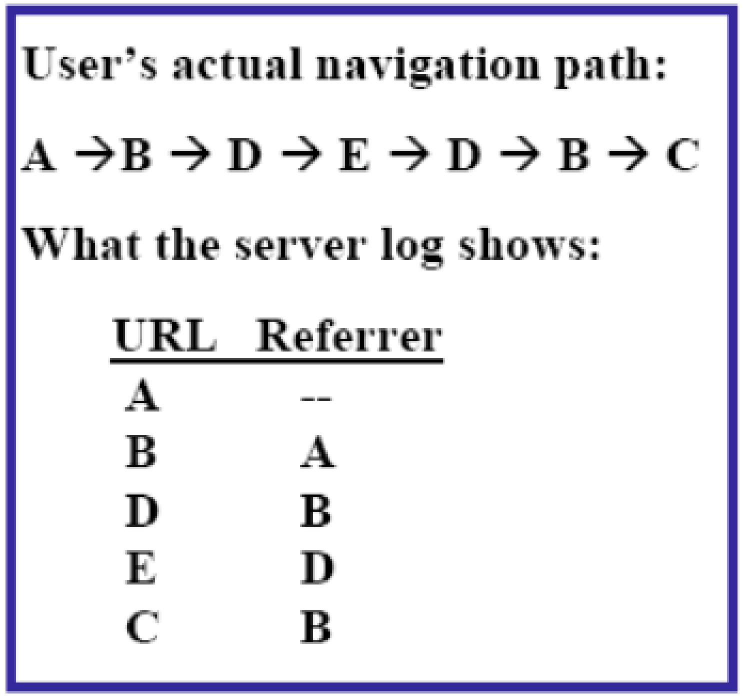
- Client or proxy-side cashing can result in missing access references to previously cached objects
- Knowing the structure of the site it is possible to fill in the missing values.
- Simple heuristic can be applied to fill in the gaps, e.g.
  - Look at the referrer URI
  - Look at the time interval, and on average how long that user spends on content pages.
  - Fill missing pages using the shortest path (via viewed pages) to next new page requested.

7. Data Integration

- Integration of pre-processed data with data coming from other sources
  - integrate e-commerce and application server data
  - integrate demographic / registration data

Path completion example . . .




User's actual navigation path:

A →B → D → E → D → B → C

What the server log shows:

URL	Referrer
A	--
B	A
D	B
E	D
C	B

- Knowledge of site structure is required to infer missing pages.
- Many paths are possible: usually, the selected path is the one requiring the fewest number of "back" reference.

Results

- When the preprocessing steps are complete, the following matrices can be generated from the log files of a web application for further analysis:

- **User page view matrix**
  - Lists users and pages they've viewed.
- **Term page view matrix**
  - Terms are displayed on the vertical axis and pages on the horizontal axis
  - The transpose of this will be the page view feature matrix
- **Content Enhanced transaction matrix**
  - Displays users on the vertical axis and terms on the horizontal axis

- Subsequently, modelling algorithms can be applied, e.g., association mining, clustering, classification.

U

	page A	page B	page C	page D	page E
user 0	15	4	1	0	0
user 1	2	0	25	0	0
user 2	200	1	0	0	3
user 3	56	0	0	4	4
user 4	0	0	23	50	0
user 5	0	0	5	3	0

- **U** = user page views matrix
- **P** = term page views matrix
- **U X P** = content enhanced transaction matrix; combines U and P to determine user interests

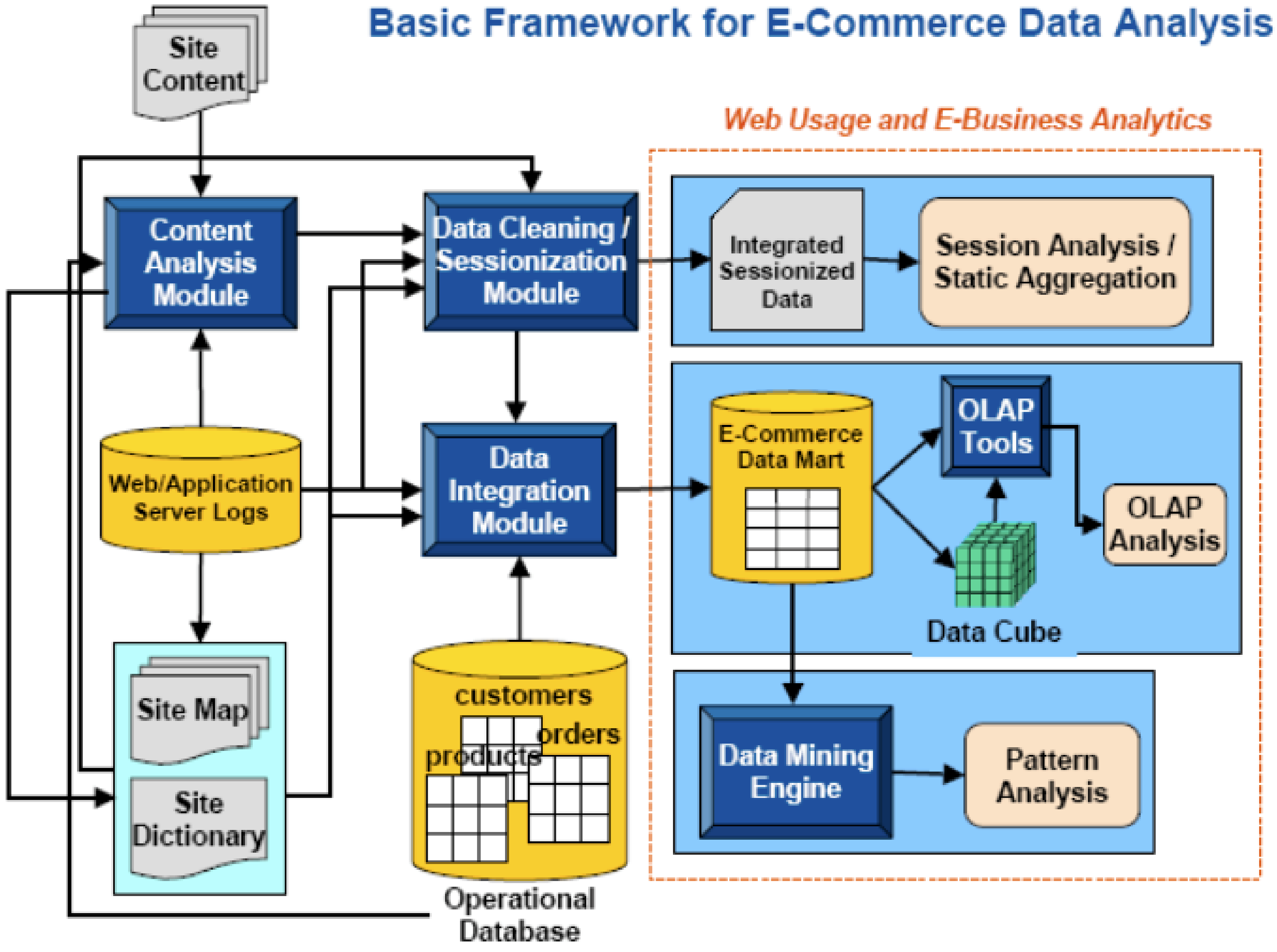
P

	food	news	car	house	party	sky
page A	0	1	1	0	0	0
page B	1	0	0	1	0	0
page C	1	1	0	0	0	0
page D	0	0	1	0	0	1
page E	0	0	0	1	1	0

U X P

	food	news	car	house	party	sky
user 0	1	1	1	1	0	0
user 1	1	1	0	0	0	0
user 2	0	1	1	1	1	0
user 3	0	1	2	1	1	1
user 4	1	1	1	0	0	1
user 5	1	1	0	0	0	0

Basic Framework for E-Commerce Data Analysis



Web Structure Mining: summary…

- Web structure mining discovers useful information and knowledge by analysing hyperlinks which represent the structure of the web.

- Analysing links lets us derive information about most **important web pages**, **gateway pages**, **communities of users** who share common interest, etc.

- We are not covering this topic any further. . .

Text corpus

- **Corpus** = a collection of related documents/articles/books/tweets/posts/messages, etc.
  - We will typically use **documents** to refer to any of the above.
- Corpora can be annotated (labelled) for supervised learning.
- They can also be unlabelled typically used for unsupervised learning.
- A corpus can be broken into individual documents or, using a more general approach, into categories with one or more documents.
- Depending on the task, a document could be broken down into paragraphs, sentences, words or, even characters.

Text Corpus

2 types of corpora:

1. **Generic Corpora** typically used for initial testing.

   Examples: Wikipedia Corpus, Brown Corpus, etc.

2. **Domain Specific Corpora** used for building application specific language models.

- Models trained using domain specific corpora perform better because …
  - Different domains use different language (vocabulary, acronyms, common phrases, etc.)
  - By fitting models in a narrower context, the prediction space is smaller and more specific, and therefore better able to handle the flexible aspects of language.

Corpus Reading or Corpus Building?

- ➢ **Generic Corpora** typically are read in, but **Domain Specific ones** can also be read in if they are publicly available.
- ➢ Nltk offers many functionalities for corpus reading
- ➢ They expose corpora via **CorpusReader** objects; some shown below:

PlaintextCorpusReader: A reader for corpora that consist of plain-text documents, where paragraphs are assumed to be split using blank lines.
TaggedCorpusReader: A reader for simple part-of-speech tagged corpora, where sentences are on their own line and tokens are delimited with their tag.
TwitterCorpusReader: A reader for corpora that consist of tweets that have been serialized into line-delimited JSON.
WordListCorpusReader: List of words, one per line. Blank lines are ignored.
XMLCorpusReader: A reader for corpora whose documents are XML files.
CategorizedCorpusReader: A mixin for corpus readers whose documents are organized by category.

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Corpus Reading or Corpus Building?

- However, many **Domain Specific Corpora** need to be built, after the data has been scraped from the web (or by other means).
- For instance, in Python, if we want to build a corpus for binary classification, where we need one label for the positive class and another for the negative class, we can map each document to the corresponding label, using a dictionary structure:

[{ **‘label’:** ‘positive class’, **‘text’:** ‘…’},

{ **‘label’:** ‘negative class’, **‘text’:** ‘…’}]

- **‘label’** and **‘text’** are the **keys** in the dictionary structure
- NOTE: the labels of the texts themselves must be known in advance.

Corpus Reading or Corpus Building?

- It is also important to randomise the order of the documents using the **sample** method from pandas.
  - This is to avoid mining algorithms learning based on the order of rows/documents.
- You can then use the numpy array and dataframe structures to build a labelled corpus.
- Once the corpus is built, it should be written to a csv file, using the **to_csv** function from pandas.
- **NOTE:** csv cells can take a limited number of characters- around 32k- and, depending on the parameters passed to the to_csv function, we need to ensure that we limit each of the documents to this, otherwise, the remaining characters will be written to subsequent cells and we may end up with many rows being unlabelled or with missing data.