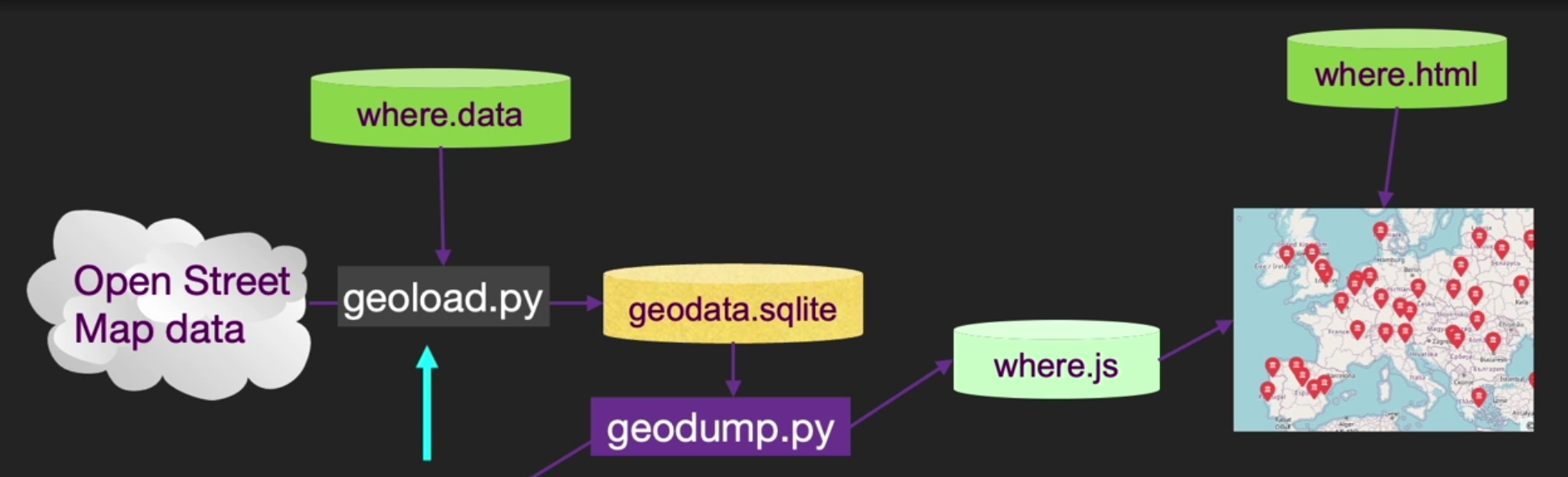
**Multi-Step Data Analysis**

- Data analysis is done as a multi-step process:  
  
1. Gather data from data source   
 - Process needs to be restart and stoppable  
 - Can take a long time to get data  
2. Clean data to make it useful  
 - Relatively quick since data is already on computer/local cache  
3. Conduct analysis on clean data  
 - Turn into charts, histograms, dashboards, etc..  


**Textbook:**

**OpenStreetMap:**

- This program will read the input lines in *where.data* and for each line check to see if it is already in the database. If we don’t have the data for the location, it will call the geocoding API to retrieve the data and store it in the database, essentially functioning as a local cache  
A screen shot of a computer

Description automatically generated  
- You can visualize the data using the *geodump.py* program. This program reads the database and writes the file where.js with the location, latitude, and longitude in the form of executable JavaScript code  
A screenshot of a computer

Description automatically generated

- The file *where.html* consists of HTML and JavaScript to visualize a Google map. It reads the most recent data in where.js to get the data to be visualized.  
- This is a JavaScript variable that contains a list of lists. The syntax for JavaScript list constants is very similar to Python, so the syntax should be familiar to you.  
A computer screen shot of a computer code

Description automatically generated

**Visualizing Networks and Interconnections:**

- In this application, we will perform some of the functions of a search engine.   
1. We will first spider a small subset of the web   
2. Then run a simplified version of the Google page rank algorithm to determine which pages are most highly connected  
3. Lastly, we will visualize the page rank and connectivity of our small corner of the web.

- The first program *spider.p*y program crawls a web site and pulls a series of pages into the database *spider.sqlite*, recording the links between pages  
A computer screen shot of a computer code

Description automatically generated  
  
- If you restart the program and tell it to crawl more pages, it will not re-crawl any pages already in the database. Upon restart it goes to a random non-crawled page and starts there.  
A screen shot of a computer

Description automatically generated

- If you want to dump the contents of the *spider.sqlite* file, you can run *spdump.py*.   
- This shows the number of incoming links, the old page rank, the new page rank, the id of the page, and the url of the page. The *spdump.py* program only shows pages that have at least one incoming link to them.  
A group of white text on a black background

Description automatically generated

- Once you have a few pages in the database, you can run page rank on the pages using the *sprank.py* program. You simply tell it how many page rank iterations to run.  
A black screen with white text

Description automatically generated  
- You can dump the database again to see that page rank has been updated  
A screenshot of a computer

Description automatically generated

- A search engine usually runs both the crawling and ranking programs all the time.  
- You can run *sprank.py* as many times as you like and it will simply refine the page rank each time you run it. You can even run *sprank.py* a few times and then go spider a few more pages with *spider.py* and then run *sprank.py* to reconverge the page rank values.  
- For each iteration of the page rank algorithm it prints the average change in page rank per page.  
- The network initially is quite unbalanced and so the individual page rank values change wildly between iterations. But in a few short iterations, the page rank converges. You should run *sprank.py* long enough that the page rank values converge.

- To visualize the current top pages in terms of page rank, run *spjson.py* to read the database and write the data for the most highly linked pages in JSON format to be viewed in a web browser.  
A black screen with white text

Description automatically generated

- You can view this data by opening the file *force.html* in your web browser. This shows an automatic layout of the nodes and links. You can click and drag any node and you can also double-click on a node to find the URL that is represented by the node.  
A network of lines and dots

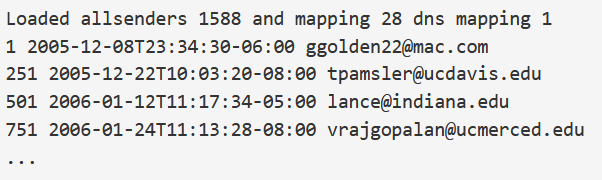
Description automatically generated

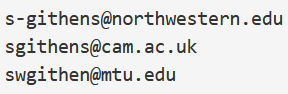
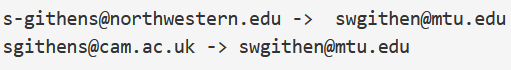
**Visualizing Mail Data:**

- The file *README.txt* in the above ZIP may have instructions as to how you can download a pre-spidered copy of the content.sqlite file for a majority of the Sakai email corpus so you don’t have to spider for five days just to run the programs.   
- If you download the pre-spidered content, you should still run the spidering process to catch up with more recent messages.  
- The base URL is hard-coded in the gmane.py and is hard-coded to the Sakai developer list  
- The program scans content.sqlite from one up to the first message number not already spidered and starts spidering at that message. It continues spidering until it has spidered the desired number of messages or it reaches a page that does not appear to be a properly formatted message  
  
(- If your spider stops, and it seems it has hit a missing message, go into the SQLite Manager and add a row with the missing id leaving all the other fields blank and restart gmane.py. This will unstick the spidering process and allow it to continue. These empty messages will be ignored in the next phase of the process.)

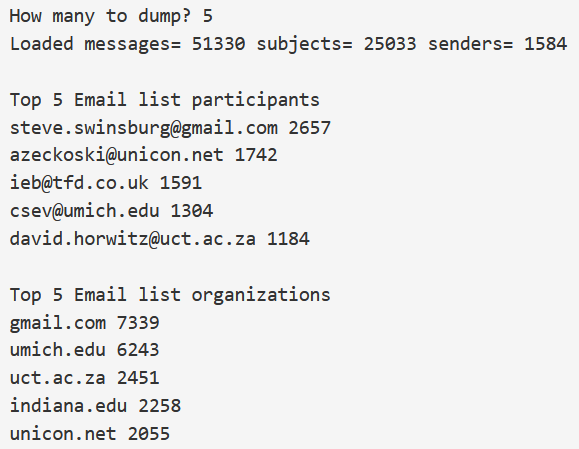
- The second process is to run the program *gmodel.py*. This program reads the raw data from *content.sqlite* and produces a cleaned-up and well-modeled version of the data in the file *index.sqlite*. This file will be much smaller (often 10X smaller) than *content.sqlite* because it also compresses the header and body text.  
- Each time *gmodel.py* runs it deletes and rebuilds *index.sqlite*, allowing you to adjust its parameters and edit the mapping tables in *content.sqlite* to tweak the data cleaning process.

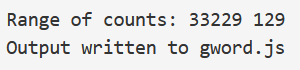
- This is a sample run of gmodel.py. It prints a line out each time 250 mail messages are processed so you can see some progress happening



- In the mapping.sqlite database there are two tables that allow you to map both domain names and individual email addresses that change over the lifetime of the email list  
- We can add two entries to the Mapping table in mapping.sqlite so gmodel.py will map all three to one address:  
 

- You can rerun the gmodel.py over and over as you look at the data, and add mappings to make the data cleaner and cleaner allowing for quick data analysis   
- The first, simplest data analysis is to determine “who sent the most mail?” and “which organization sent the most mail”? This is done using gbasic.py



- You can produce a simple visualization of the word frequency in the subject lines in the file gword.py  
- This produces the file gword.js which you can visualize using gword.htm to produce a word cloud similar to the one at the beginning of this section.  
 

- A second visualization is produced by gline.py. It computes email participation by organizations over time.  
- Its output is written to gline.js which is visualized using gline.htm  
  
