Web-Scraping:

How Charlotte Is A Life Saver Again

Fayang Pan

Advisor: Dr. Eric Nordmoe,

Department of Mathematics & Computer Science

A paper submitted in partial fulfillment of the requirements

for the degree of Bachelor of Arts at Kalamazoo College.

Department of Mathematics & Computer Science

Kalamazoo College

MI, USA

2014

Preface

Charlotte, a spider, was a life saver in E.B. White's novel, *Charlotte's Web*. I have never been a fan of spiders. But in the summer of 2013, a spider creeped into my life and saved me.

This is not an ordinary spider. The Internet is its web, and information is its prey. It crawls into every corner of a webpage and fetches whatever information it needs. What is this? You will find out in this paper.

I spent 10 weeks in Pinnacle Solutions, Inc as a summer intern, playing with spiders. Nobody in the company knew anything about spiders, so I had to learn mostly by myself. Through the internship, with help from online resources and colleagues, I gained more understanding of these spiders, so I am now writing a paper for them. You will hardly find any reference to books or journals, because there are not many of them. The topic is relatively new, and the ways to build spiders are still evolving. I spent a lot of time asking for help from online forums, as most of the knowledge about spiders actually resides in those forums.

I would like to express my very great appreciation to Mr. Donald Penix Jr., for offering me the internship and the opportunity to develop web-scrapers. He enlightened me with the idea of this paper, trusted me and rendered all the support I needed. I would like to extend my deep gratitude to Dr. Eric Nordmoe, my advisor, for his patient guidance and useful critiques. I would like to thank my parents, Ping Pan and Ling Yang, for their understanding and unwavering support. A heartfelt thanks

goes to my girlfriend, Yin Cai, for her caring presence when I lost my way in the web. Finally, I wish to thank my colleagues at Pinnacle Solutions, Inc. and my friends for their encouragement and support.

Abstract

The demand for information is ever increasing. While some have the privilege of accessing private data, others look for ways to collect information from where anyone could visit: the Internet. On the Internet, there is much information available just on the webpages, and there is a way to possibly extract certain pieces from webpages and organize those pieces efficiently: web-scraping. This paper describes what is web-scraping, its controversy, application, implementation, and potential.

The content of this paper is mostly developed from my firsthand experience during an internship. The analysis of web-scraping is based on my encounter with the technology, and the examples are mainly from online sources. The paper covers some suggestions on building a web-scraper, in parallel with existing online tutorials. In the end, the paper discusses the possible future of web-scraping, and some challenges it faces.

Contents

List of Figures

Chapter 1

An Evaluation of Web-Scraping

Introduction

Web-scraping is a way to collect existing information from the Internet. It is a nascent, powerful, but somewhat controversial subject in the arena of modern technology. Using tools to simulate viewing and downloading of a webpage through a browser, the 'spider' (a metaphoric name for the part that goes into each web structure) goes into the webpage and 'crawls' (a metaphor for copy-and-paste) desirable contents into a formatted data structure. Since this technology only requires access to the webpages, theoretically, anything that can be downloaded can be scraped off the Internet.

In the past summer, I worked with Pinnacle Solutions, Inc.,[1] a company in Indianapolis, Indiana. My job was solely to develop web-scrapers. In this paper, I will provide an overview and discussion of web-scraping, followed by a description of my internship experience.

[1]For information, visit `http://psiconsultants.com/`

1.1 An Overview of Web-Scraping

In general, there are three basic ways to collect data: from primary sources, e.g., conducting surveys and studies; from compiled data in secondary sources, e.g., databases like the U.S. Census; and from existing but uncompiled information, e.g., product reviews from *Amazon.com*. Web-scraping focuses on the last way: to harvest data from a structured webpage, e.g., an HTML table, to a structured format, e.g., json and csv. It provides an amalgamation of the deluge of data spread over websites.

1.2 Usage of Web-Scraping

As web-scraping does not generate new data or results, it seems useless. However, people from various backgrounds may find web-scraping very useful. Here are a few examples:

If a student were to conduct a study on what lessons people learn from documentaries such as *Food, Inc.*, he/she could go to `http://imdb.com`, `http://amazon.com` and `http://rottentomatoes.com` to scrape all the viewers' reviews. As there are thousands of reviews for the movie, web-scraping will help the student save much time trying to copy and paste every review. From the integrated results, the students might do a word frequency count to guess what kind of impression the majority have.

If a company were to have launched a product, they could use web-scraping to monitor their product, their competitors' products, and their partners' products. If people from Amazon.com want to know how successful is the new Kindle Fire HDX, they can web-scrape all the reviews about Kindle Fire HDX, about Nexus 7, and about Kindle Fire HD. Through analysis, Amazon.com is able to collect customers' feedback promptly.

If a statistician were to monitor the price change of thousands of commodities,

and if all these commodities are sold online, the statistician can use web-scraping to collect the prices. The scraper will just copy the prices from the product page of an online retailer website, and paste all of them into a spreadsheet. Shifts in prices may reflect inflation/deflation, and the data can be used in economic research.

If a data analyst from Apple were to study how the public responds whenever a new product is released, he/she could collect the review dates from review websites and plot a frequency graph showing number of reviews per day. The number of reviews generated per day may reflect how receptive the customers are to a new product.

1.3 Advantages of Web-Scraping

1.3.1 Speed

Web-scraping allows people to copy and paste from webpages faster. For instance, someone wants to harvest 10,000 pieces of Amazon.com product reviews. Doing the job manually by copying and pasting every little piece of information into an excel spreadsheet would take a very long time and be prone to mistakes. Web-scraping technology, on the other hand, makes the process much faster. By specifying certain paths in the webpage structure, the spiders go into specific attributes and fields to crawl information. With the help of web-scraping, 10,000 reviews can be copied and pasted into a csv file within 15 minutes.[2]

1.3.2 Convenience for Data Analysis

Web-scraping improves the efficiency of data analysis. The tables of data that web-scraping produces are more easily processed than plain text. As web-scraping grabs from a structured format, it is able to copy and paste all the information needed into another structured format. For instance, after a csv file is generated by scraping

[2]The result is based on running of my own scraper.

data from Amazon product reviews, all rows in the *ratings* column have a range of 1 to 5 for number of stars, all rows in the *review body* column will contain texts of the review bodies. A well-structured database will then provide foundation for further data Data Mining and other modeling techniques.

1.3.3 Availability

Web-scraping tools are becoming more widely available today. Popular tools for web-scraping include browser extensions such as *firebug* and *iMacros*, programs such as *Wget*, programming languages such as *Perl*, *Hadoop* and *Java* which have libraries to support sending HTTP requests, and programming language extensions such as *Scrapy* and *Beautiful Soup* for *Python*. It is becoming more feasible for anyone to learn web-scraping, and to harvest data beyond traditional methods such as using web APIs.

1.4 Challenges of Web-Scraping

1.4.1 Legality

Web-scraping raises legal and ethical issues. While it is free to download many webpages, usage of the data within might not be in compliance with the terms and conditions of the source.

There exists a fine line between using and stealing data. Currently, there is no law forbidding web-scraping all together, nor is there any way to ban downloading of webpages. However, as web-scraping becomes more versatile and powerful, companies are becoming more vigilant on data protection.

In 2000, Bidder's Edge, a website that collects auction information from various websites, collected and displayed auction information from eBay. In response, eBay

sued Bidder's Edge for "trespass to chattels",[3] and Bidder's Edge soon went out of business.[4]

In March 2013, Associated Press (AP) won a lawsuit against Meltwater, a Norwegian group, over copyright infringement. Before the lawsuit, if a customer wanted to know how frequently a certain word appeared in the news, Meltwater could provide the information through scraping from various news websites. Meltwater used to be able to collect the news, put them into databases, and render them to customers on demand, but not any more. The court ruled that the usage of news resources went beyond the scope of "fair usage",[5] forbidding Meltwater from continuing to scrape from news agencies.[6]

In general, there are few things any scraper should watch out for. First, terms of use of the websites. Even though the terms are usually long and tedious, "I did not see those" is no excuse for violating the terms. Second, the scraper should also consider the purpose of the scraped information. If the scraper or customer were to use the scraped information for malicious purposes such as spamming and libel, it is likely to be illegal. Third, the scraper should consider the effect of scraping on others. If the number of requests for a certain scraping activity is too large for the website server to handle, other visitors to this website may experience malfunctions of the website. It is thus always wise to consult an attorney before beginning any commercial scraping activity.

[3]For more information, visit `http://definitions.uslegal.com/t/trespass-to-chattels/`

[4]For information on the ruling, visit `http://www.tomwbell.com/NetLaw/Ch06/eBay.html`

[5]For more information about fair use, visit `http://fairuse.stanford.edu/overview/fair-use/four-factors/`

[6]For more information on the ruling, visit `http://www.scribd.com/doc/131847330/Meltwater-AP-Ruling`

1.4.2 Intricate and Dynamic Web Structures

In addition to legal concerns about scraping, prospective scrapers face increasingly complex web structures. AJAX, for instance, sends out an XMLHTTPRequest only when the user performs certain maneuvers in the browser. Also, embedded JavaScript contents may not be downloaded locally. The advent of these dynamic ways of loading web contents brings new challenges to web-scraping.

1.4.3 Instability

Another underlying problem comes from the fact that web-scraping is based on webpage structure. If the website adds a few toolbars, the locations of HTML attributes will change, resulting in a crash of the scraper. There also exists the problem of encoding and decoding. A scraper crash may also be caused by a website using unusual encoding of text.

If the structure is dependent on a specific web browser, the content might appear different as well. A typical web-scraper downloads webpages with their HTML structures through the default browser, and a typical website offers the same HTML to any browser. However, some websites sometimes return different HTML structures to different web browsers for enhanced user experience, causing ambiguity of downloaded data. These potential problems manifest web-scraping's intrinsic dependence on the web interface.

In the following two pictures, the same webpage is opened using two different browsers, Google Chrome and Mozilla Firefox, under the same environment at the same time. Rendering of the page by the two browsers differ in locations of choice of flavor, availability of customer image, font sizes and layouts, sites available for sharing, etc.

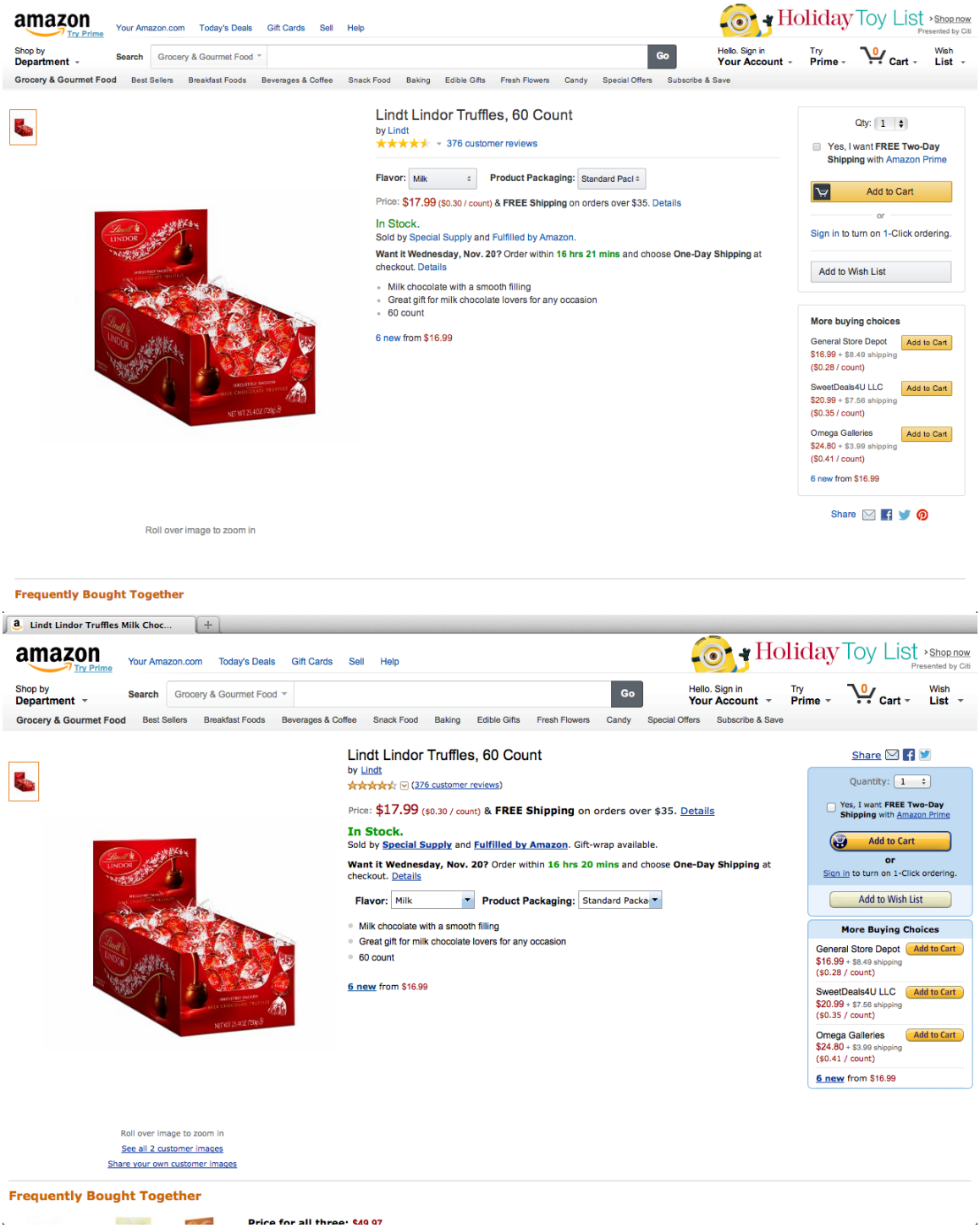


Figure 1.1: What Google Chrome displays versus what Firefox displays with identical urls.

Chapter 2

My Internship

2.1 Overview of My Internship at Pinnacle Solutions, Inc.

Pinnacle Solutions, Inc.(PSI) is a start-up which helps clients to analyze and interpret their databases. Founded in 1996 by Kalamazoo College alumnus Donald Penix, Jr., the company, located in downtown Indianapolis, Indiana, currently has over 20 employees. PSI primarily uses *SAS* for processing data, and *Amazon Web Services* for server management.

I was a summer intern at PSI in 2011, and my job title was Business Intelligence Content Developer. At that time, Mr. Penix was very interested in sentiment analysis, a technology that aims to discover and analyze sentiment reflected through comments, reviews, and blog posts alike. For instance, a comment such as "I love this iPad!" indicates a positive sentiment, and one such as "I am sick of the iPad!" suggests a negative sentiment. Sentiment analysis will consolidate related comments and analyze them using Data Mining and NLP. After research, we found sentiment analysis a tough job for PSI, as none of the employees had the needed expertise. So we decided to start from scratch, with data collection being the first step towards sentiment

analysis.

2.2 How Web-Scraping Became the Best Option

Data analysis begins with data collection. As previously discussed, there are three ways to collect data. PSI is not interested in conducting primary research, because the research can be costly and lengthy. PSI would love to always have secondary databases available for analysis, but that is hardly possible, as PSI usually needs information beyond the scope of existing databases. Therefore, it becomes evident that PSI need to collect data from existing but uncompiled data from the Internet. Since a large amount of data is needed, the process should be automated. In order to harvest usable data from the Internet, there are many ways other than web-scraping.

First, one can extract necessary information using a web API. Many websites offer APIs for developers to use to build applications. Twitter, for instance, published a new version of its API in August, 2013. Through an API, users can request information directly from the website's databases, and the data will be clean and formatted. Twitter, for instance, offers exportable data in json format. However, to obtain access to data in greater breadth, amount or detail, users may need to pay for them. Moreover, many websites may have limited information provided in their web APIs, or may not provide APIs at all. In that case, even though users can view the data from the browser, they cannot download it through APIs. Therefore, for a small company, it might be expensive and insufficient to rely on APIs for data harvesting.

Second, there are available third-party scraping and data harvesting services. There are applications like *Mozenda*,[1] and highly customizable services such as *scrapinghub*,[2] *GNIP*,[3] and *Open Amplify*.[4] However, these services are costly and, per-

[1] `http://mozenda.com/`
[2] `http://scrapinghub.com/`
[3] `http://gnip.com/`
[4] `http://www.openamplify.com/`

haps, beyond the means of small companies. Moreover, in the long run, as PSI needs more information from the Internet, its dependence on these services would be ever-increasing. Therefore, it would be a wise choice if PSI could come up with a low cost, independent system to harvest information.

Third, as they are a certified *SAS* reseller, they could use the software *SAS Sentiment Analysis Studio*.[5] However, this application is not only expensive for clients to buy, but also dependent on the web API.

After many thorough discussion sessions, a free and open software solution capable of fetching data from the Internet via web-scraping was deemed desirable. Many web-scraping resources are free, so both the start-up and ongoing costs of doing web-scraping are low. There are many existing online communities discussing and developing web-scraping tools, so the technical support is active and extensive.

After some research, we chose Scrapy, a Python web-scraping framework, for the job.

2.3 Scrapy, a Python Framework

To quote from its website, "Scrapy is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing."[6]

Scrapy depends not only on Python, but also on a few other libraries. Scrapy uses Twisted,[7] an event-driven networking engine, and Zope,[8] a web application server framework. The installation guide can be found on Scrapy's website.[9]

[5]For more information, visit `http://www.sas.com/text-analytics/sentiment-analysis/`

[6]`http://scrapy.org/`

[7]`http://twistedmatrix.com/trac/`

[8]`http://zope2.zope.org/about-zope-2/what-is-zope-2`

[9]`http://doc.scrapy.org/en/latest/intro/install.html`

There are four basic classes in the Scrapy framework. These classes represent four main mechanisms of Scrapy.

The spider. *Spider.py* is responsible for downloading webpages, going into their structures and extracting data from them. At the very least, the user needs to specify the domain, the starting url, and the location in the webpage from which to crawl. If the user wants to scrape from multiple tables or multiple pages, or even conditionally scrape from certain pages, he/she can define those in *Spider.py*.

The items. *Items.py* is a class specifying the column names in the result. In the end, the result would be a table consisting of all the scraped data, and the items class defines what they are. Items in the same class will be in the same table.

The settings. *Settings.py* includes instructions about what information will be written into the log file, limits on the frequency of requests sent, and other customizable features. In the event of a user login is needed, the user can pre-fill that in the login settings to avoid access denial.

The pipeline. *Pipelines.py* decides how the scraped data are to be processed. By default, all data will be dumped into a single file. However, if the user wants to add a filter to remove some data, pipeline can handle that. If the user wants to split the results from one spider into two separate files, he/she can do that through coding in *pipelines.py*.

Scrapy has more advanced features, which would fall outside of the scope of this paper. In the next chapter, I offer a high level approach to web-scraping using Scrapy.

Chapter 3

Scrapy Explained in Greater Detail

3.1 Test How Much You Can See

Web-scraping depends on whatever the browser can "see," and there are times when the browser sees less than a person does. To check, one can use the following line of code in the command-line:[1]

```
scrapy shell http://amazon.com
```

After Scrapy finish analyzing the web address, it goes into its prompt and asks for further instructions. Here, the user can key in

```
view(response)
```

A browser window opens and shows the structure of the website from Scrapy's perspective. The following two screenshots demonstrate the differences in downloading a page through Scrapy and viewing the page through the browser.

[1] cmd.exe in Windows, Terminal in Mac OS/Linux

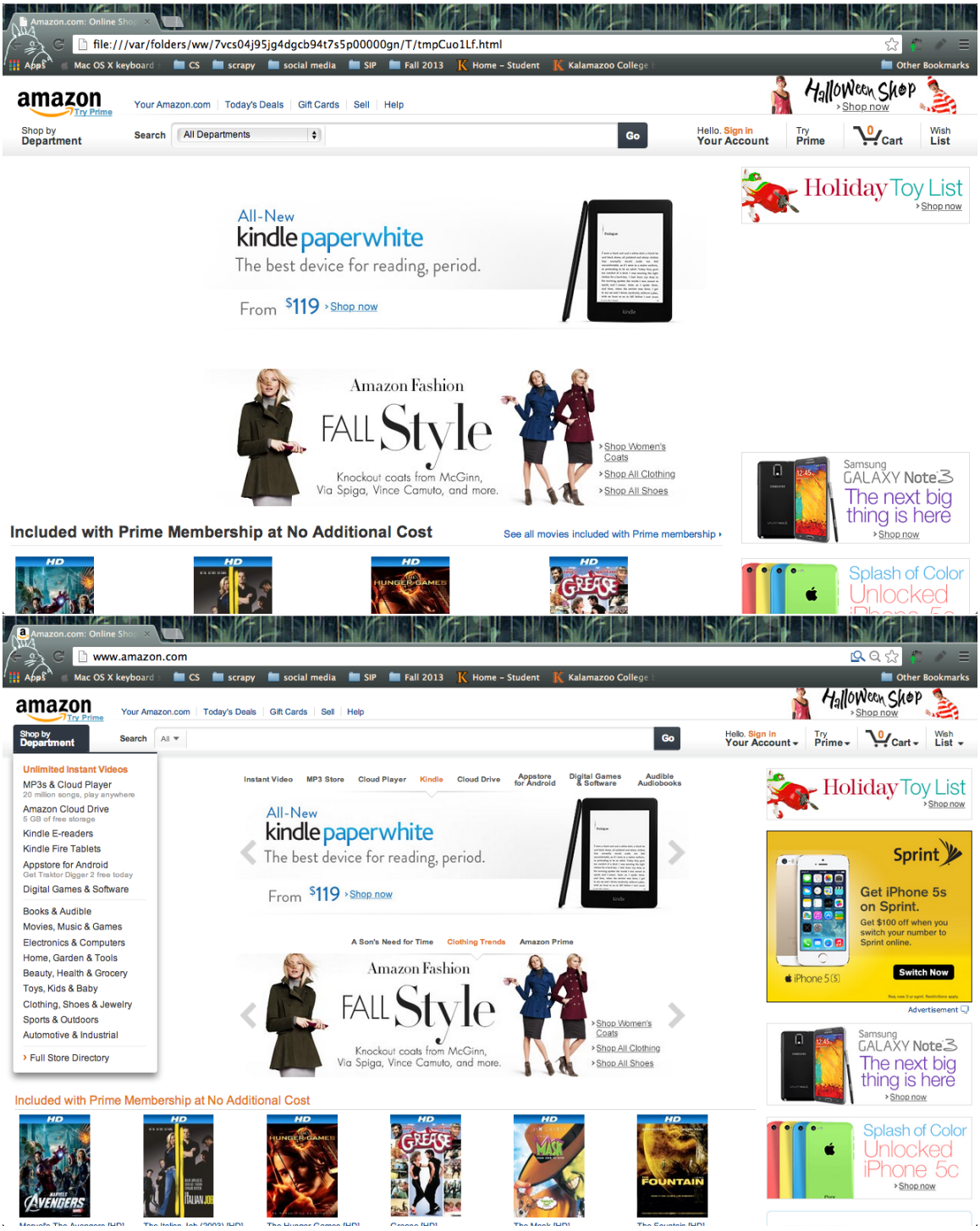


Figure 3.1: What Scrapy is able to download locally (above) versus what the user can see (below).

3.2 Test If the Data is Scrapable

In theory, any data which can be seen can be downloaded, and any downloaded data can be scraped. In practice, however, it is not that simple. Even if Scrapy can see the data the user needs, the difficulty of scraping varies widely across websites. For instance, in the event of embedded JavaScript code, scraping the result of a function call will be more difficult.

After setting up a project with few commands,[2] it is wise to test if certain data are scrapable. Before knowing if the data are scrapable, one needs to specify where exactly the data are located. To establish a standard way of referring to a certain field in the webpage, XPath[3]is used.

To retrieve a certain XPath, one can use a built-in *"Inspect Element"* tool, or an extension such as *firebug*,[4] available for *Firefox* and *Google Chrome*. Figure 3.2 provides an example using *"Inspect Element"* in *Google Chrome* to extract an XPath from `http://www.nbcnews.com`.

Given a specific XPath, one can write related code to test if Scrapy can recognize the XPath and scrape the data. If one piece of data is scrapable, it is likely that the entire table which contains that piece of data can be scraped. If the sample table is scrapable, it is also likely that similar tables can be scraped, too.

[2]For more information, visit `http://doc.scrapy.org/en/latest/intro/tutorial.html`. The website has a simple example on how to start a project

[3]Abbreviation of "XML Path Language".For more information, see `http://www.w3.org/TR/xpath/`

[4]Visit `http://getfirebug.com/formoreinformation`

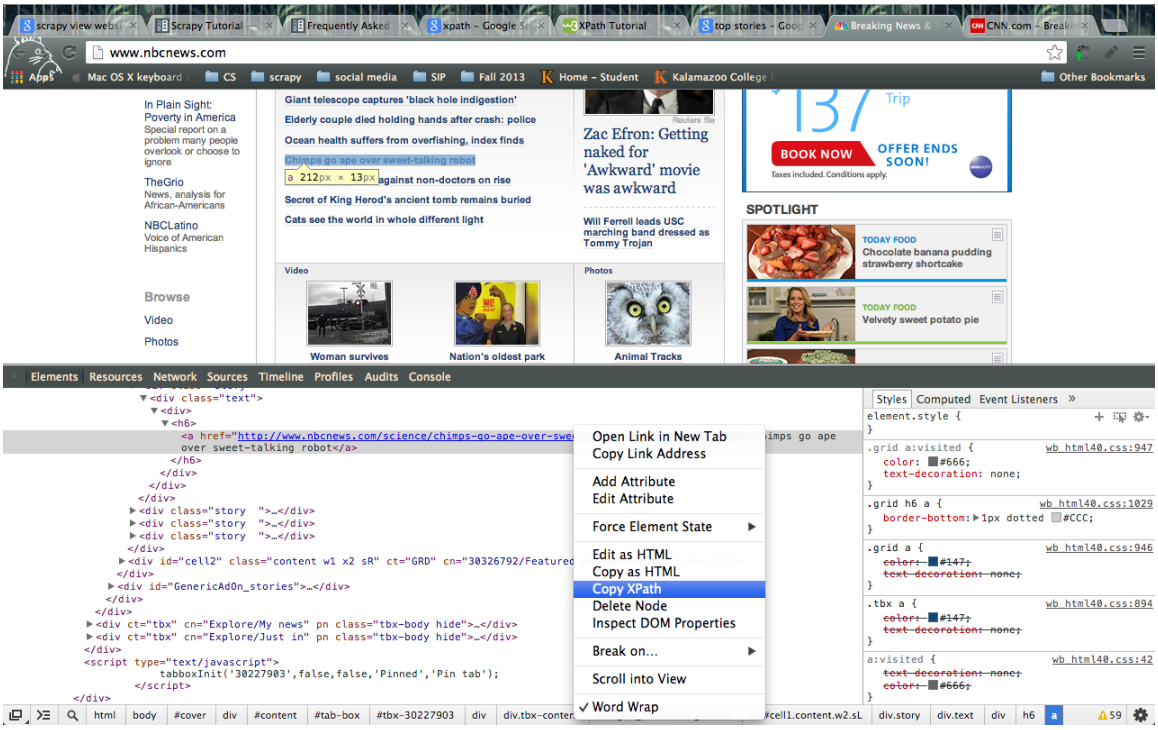


Figure 3.2: A screenshot of how to obtain an XPath from a webpage, and the XPath of the link is *//*[@id="cell1"]/div[8]/div/div/h6/a*.

3.3 Iterators

Web-scraping relies heavily on the structure of the webpage. If the webpage is very organized, it is possible for the user to use XPath to find all entries in a table, and send out a request to scrape the next page. In Scrapy, the request is in the form of a callback function. The spider will scrape the data, yield the results from the page, and then excute the callback function. Here is a snippet of that concept:

```
class Spider(BaseSpider):
    def parse(self,response):
    ##some code to extract data from this current
      page
        yield results_from_this_page
        if next_page:
```

```
yield Request(url_of_the_next_page,
    callback=self.parse)
```

In the code above, *BaseSpider* is one of the spider classes the user has to inherit to build a user-defined spider. In any spider class, a method like *parse(self,response)* is necessary to communicate with the computer and instruct which fields in the html to scrape from. The *parse* method yields a generator, but the code does not necessarily end there. In the event that there is a next page with the same html format, the parse method will be able to go into the new html webpage and collect the new data in the same way it did for the previous page. Therefore, the method *parse* can also yield a *Request*. *Request* represents an HTTP Request in many cases. It sends a signal to the scraper and asks for a *Response*, which will be handed to the spider and the spider will scrape it.

3.4 Specification

After sketching and testing the basic functionalities, we proceed to consider the types of information clients want and the types of information needed to make database management easier.

The clients usually ask for specific data that are useful to their businesses. For instance, an author may want to read all of the reviewers of her latest book. To collect book reviews from websites such as `http://amazon.com` and `http://goodreads.com`, we need different spiders, one for each website. Nevertheless, it may be possible to instruct the spiders to generate tables with identical column names so that the author can view everything in one sheet. That data sheet may contain the book title, edition number, publisher, review website, user name of the reviewer, date of the review, ratings, the body of the review, and other information specified by the column names.

On the developer's end, however, the structure will be very different. To track every single review, the developer may need a column with the url from which that particular review was pulled. To iterate through all reviews of the same book, the developer may need to track that book's identification number on the website to generate the correct url to scrape from. To deal with different encodings and web structures of different websites, the developer needs to customize every spider.

A piece of pseudo-code for extracting a certain text could be:

```
item['some_field']=HtmlXPathSelector(response).select('
    SOME/XPATH/IN/HTML/text()').extract()
```

In the code above, 'some_field' is the user defined name for the column. The HtmlXPathSelector will find the specified XPath, and extract the text contained in the XPath.

3.5 Produce sample output

After studying what and how the spider should scrape from webpages, we are ready to produce sample output. The sample output is a small subset of potentially large actual output. For instance, if a product has a million reviews, the sample output may include the first 100 of them. We want to make sure the sample output is desirable before scraping from the entire database. The sample output serves three purposes.

Formatting and encoding. It is not unusual when the ideal is far from reality. Although XPath offers a specific location of all scrapable information, it is highly possible that the formatting is not desirable. Also, most websites are written in utf-8, but because Python's csv module defaults encoding to ascii, a paragraph often has a $\backslash u$ in front of it. For instance, a review "I like this!" will be translated into \u" , , , , I like this!" through Scrapy.

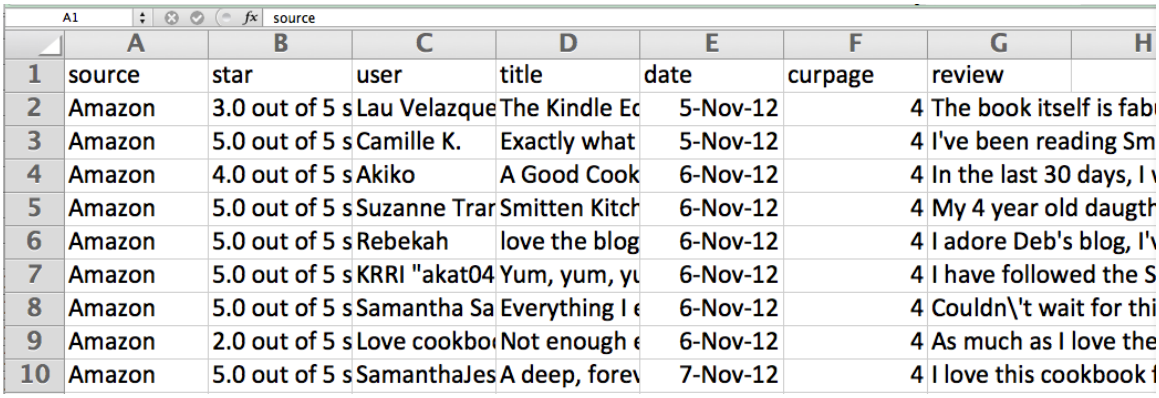
```
2013-12-02 22:26:23-0600 [amazon] DEBUG: Scraped from <200 http://www.amazon.com/Kindle-Pape
rwhite-Touch-light/product-reviews/B007OZNZG0/?ie=UTF8&sortBy=bySubmissionDateAscending&show
Viewpoints=0&pageNumber=50.html>
        {'date': [u'October 14, 2012'],
         'review': [u'\n    \n      ',
                    u'\n      ',
                    u'\n      ',
                    u'\n      ',
                    u"\n\nvery disappointed that Amazon took out the text to speech, I sold
my kindle touch in anticipation for this paperwhite and found out that the new kindle doesn'
t have text-to-speech. I would consider myself an auditory learner when I read more difficul
t books I like being able to read along with the voice.",
                    u'Please bring this back in the next generation\n      ',
                    u'\n      ',
                    u'\n\n'],
         'star': [u'3.0 out of 5 stars'],
         'title': [u'Bring back text to speech']}
```

Figure 3.3: Example of extracted data in Terminal.

Communication with clients and manager. Only when the sample data sheet has been produced, can the results of scraping be evaluated. There are limitations to scraping, so it is essential to make clients and managers aware of those. For instance, the time it takes to produce sample output influences the frequency of running the scraper. Sometimes, to keep off web-scrapers and protect data, the target website restricts the IP address from visiting it too fast. The simplest way to work around the problem it to limit the number of pages downloaded in a time interval. However, the time taken to scrape data will be longer than expected.

Testing. As web-scraping is based on web structure and HTTP requests, it is inherently unstable. Although the code may work perfectly in theory, it may encounter problems in practice. These problems may be relevant to Internet connection, bandwidth, geographic location, configuration of the computer, and other factors which may influence the scraping process. As the spider sends out requests and receives responses at a fast pace, it might encounter concurrency issues, causing some data to be skipped over or scraped repeatedly. All these issues are instrumental to the performance of a certain spider and Scrapy project, so the developer needs to read the sample output in great detail and make amendments to the code.

After the sample output is deemed satisfactory, the developer may consider adding



	A	B	C	D	E	F	G	H
1	source	star	user	title	date	curpage	review	
2	Amazon	3.0 out of 5 s	Lau Velazque	The Kindle Ec	5-Nov-12	4	The book itself is fab	
3	Amazon	5.0 out of 5 s	Camille K.	Exactly what	5-Nov-12	4	I've been reading Sm	
4	Amazon	4.0 out of 5 s	Akiko	A Good Cook	6-Nov-12	4	In the last 30 days, I	
5	Amazon	5.0 out of 5 s	Suzanne Trar	Smitten Kitch	6-Nov-12	4	My 4 year old daugth	
6	Amazon	5.0 out of 5 s	Rebekah	love the blog	6-Nov-12	4	I adore Deb's blog, I'	
7	Amazon	5.0 out of 5 s	KRRI "akat04	Yum, yum, yu	6-Nov-12	4	I have followed the S	
8	Amazon	5.0 out of 5 s	Samantha Sa	Everything I e	6-Nov-12	4	Couldn\'t wait for thi	
9	Amazon	2.0 out of 5 s	Love cookbo	Not enough e	6-Nov-12	4	As much as I love the	
10	Amazon	5.0 out of 5 s	SamanthaJes	A deep, forev	7-Nov-12	4	I love this cookbook t	

Figure 3.4: Example of extracted data in csv format.

other functionalities which may enhance the utility of a particular scraper.

For instance, to efficiently scrape updated reviews from a website, one can build a "last page tracker." If a product has 50 pages of reviews on day 1, one can view the reviews by ascending date order so that the latest review appears on the last page. After the scraper scrapes the 50 pages on day 1, it stores the last page as 50, so that when there are 60 pages on day 2, the scraper can start scraping from page 50 to 60, without re-scraping the first 49 pages.

For another example of an enhancement to web-scraping, one can make the scraper more interactive with the user. One can use Pipeline to save the output data into a user specified directory. For some scraping jobs like product reviews, it is also possible to ask for an identifier of the product, and the scraping process will start from an auto-generated url.

Chapter 4

Future Development of the Project

After harvesting data from online sources, there are various ways to enjoy the fruit of one's efforts. This chapter will illustrate some of the possible ways to use the data.

4.1 Natural Language Toolkit

Natural Language ToolKit, or NLTK, is a toolkit for natural language processing in Python. The toolkit has many functions, and one of them is tokenization. Tokenization is the process of breaking up a paragraph into sentences, a sentence into phrases, a phrase into words, or even a word into its components. These subsets are called tokens, and more study can be done through analyzing these tokens.

For instance, if a product manager would like to monitor the sentiment of customers regarding a product, he/she could use web-scraping to collect review texts from online rating websites. Although those texts are available, the feedback from the reviews is not quantitative. With NLTK, however, one can tokenize each review into sentences, and sentences into phrases. For example, one can assign a score of "1" if the review says "I hate this product!", and a score of "10" if the review reads "I love this product!". There are databases such as WordNet available for download

as a package for NLTK, and there are algorithms for assigning scores to words.

However, there are few challenges in the process of detecting sentiment.

The ambiguity of language. A single word may have different meanings in different contexts. For instance, in the review of a product, "I am sick of it" and "This product is sick!" have opposite sentiments, but they both use the word "sick" to express the sentiment. As the internet language becomes increasingly versatile, with slang, emoticons, puns, and other forms of language in the scraped text, the difficulty of guessing the right sentiment of an expression increases drastically.

The complexity of language. Short statements such as "This product is great!" and "I hate it!" are generally easier for sentiment detection. As the statement becomes longer, extending into a complex sentence, paragraph, or even multiple paragraphs, the evaluation of the sentiment of the review also becomes more complicated. For instance, for a review such as, "Though this product may not be the worst in the world, please choose other products if possible.", it seems obvious to a human that the sentence contains negative sentiment against the product. To the natural language processor, however, when it sees "not" followed by "worst", it may assign a very positive sentiment to this review.

4.2 Data Mining

Web-scraping has the potential to collect data from multiple dimensions. When a review website displays reviews on its webpage, information such as date, ratings, review texts and other relevant details will be available as well.

With multi-faceted data, it is possible to conduct further research on relationships between columns of data. For instance, some publishers want to find how likely a book is to get a a five-star review. Using web-scraping, we can collect data from few book review websites such as *goodreads.com* and *amazon.com*. A five-star review might

be related to a book's genre, language, country, length, availability, historic value, publishing house, awards received, etc., and those pieces of information should be available on the webpages. Using web-scraping, the publishers could collect possible factors leading to a five-star review, and thus developing better strategies to publish books in the future.

While it seems feasible to operate data mining under many collectable variables, the actual practice will not be easy. There are two main limitations: parsing and accessible data. The default format of scraped data is string, but more often than not, numbers are preferred in data mining. The process of parsing numbers from their string representations may incur errors. Moreover, websites are not likely to display some data, such as demographic data, to the public. To investigate possible correlations, insufficient data will be available for scraping.

4.3 Graphic User Interface

For future development of web-scraping, it is possible to build user interface from the code. Softwares such as Mozenda[1] have similar features.

To scrape information related to a particular item, the user can choose a website to scrape from, and input necessary fields such as the unique identifier of that item in the website. Then, the user can choose from a checklist of what areas to scrape from, and the number of results shown. After clicking a button, the Python script at the back end will use Scrapy to retrieve the input information, create starting and ending urls, and scrape selected results from relevant pages.

However, there are few limitations of establishing an interface. As the back end Python script retrieves information from the user interface, invalid input may cause the program to stop functioning. In addition, since web-scraping depends heavily on web structures, in the event of a change in web interfaces, the spider will probably fail

[1]For more information, visit http://www.mozenda.com/

to crawl information. Therefore, both the programmer and the user need to update frequently. The stability of the software will be very fragile.

Appendices

Glossary

AJAX Asynchronous JavaScript and XML. To quote from w3schools, "AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page." For more information, visit `http://www.w3schools.com/ajax/ajax\_intro.asp` 11

API Application Programming Interface. It defines and explains how users can use components of a program. A web's API usually guides users into permitted ways of extracting information from its database. 10, 15

Business Intelligence technologies to congregate data for business analysis. 14

crawl the process of downloading webpages and copy certain information from them. 9

csv Comma-Separated Values, a plain text form of tabulated data. Entries in the same row are separated by commas,(sometimes pipelines or other characters) and columns are separated by newlines. 8, 9

Data Mining The practice of searching through large amounts of computerized data to find useful patterns or trends. (Definition from Merriam-Webster) 5, 10, 14, 28

encoding and decoding The conversion between elements of a certain language and numbers and texts. For instance, the character "A" is encoded as "0X41" in UTF-8 hex format, and "01110001" in UTF-8 binary format decodes to the character "q". 12

harvest Data Harvesting, the automated process of gathering and organizing data. 8, 10, 15

HTTP Request A message sent from the client to the server through actions such as clicking a button on a webpage, asking for information. The server will examine the message, and send a response message back to the client. 22

JavaScript A computer programming language that allows users to interact with contents on webpages. 11, 20

json JavaScript Object Notation, a human-readable format for storing data. It consists mostly of name/value pairs, so different parts of a data table will be easily recognized by their names. 8, 15

NLP Natural Language Processing, the study of using computers to interpret natural languages through linguistic analysis. 14

NLTK Natural Language ToolKit, a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.(Extracted from `http://nltk.org/`) 27, 28

utf-8 Universal Character Set Transformation Format–8-bit is a variable-width encoding that can represent every character in the Unicode character set. It was designed for backward compatibility with ASCII. More information can be

found at http://en.wikipedia.org/wiki/UTF-8 and http://www.unicode.org/versions/Unicode6.0.0/ 24

XMLHTTPRequest A JavaScript object. It provides an easy way to retrieve data from a URL without having to do a full page refresh, allowing the response to be loaded within the script.
(From https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpReques
11

XPath XML Path Language, a query language for selecting nodes from an XML document, which can also be used with HTML. For more information, visit http://doc.scrapy.org/en/latest/topics/selectors.html 20, 23

Bibliography

Herman, M., "Recursively Scraping Web Pages With Scrapy", blog, 8 Nov. 2012; http://mherman.org/blog/2012/11/08/recursively-scraping-web-pages-with-scrapy/.

Mitchell, L., et al., "The Geography of Happiness: Connecting Twitter Sentiment and Expression, Demographics, and Objective Characteristics of Place," *Plos One*, 29 May 2013; DOI: 10.1371/journal.pone.0064417.

Paratey, P., "Writing a spider in 10 mins using Scrapy", blog, 21 Jan. 2010; http://pravin.paratey.com/posts/writing-a-spider-in-10-mins-using-scrapy.

Scrapy 0.21 documentation, Scrapy developers; http://doc.scrapy.org/en/latest/.

Smolar, C., "How legal is content scraping?", *venturebeat*, 30 May 2011; http://venturebeat.com/2011/05/30/how-legal-is-content-scraping/.

Stim, R., "Measuring Fair Use: The Four Factors," *Stanford Copyright and Fair*

Use Center, 2013; http://fairuse.stanford.edu/overview/fair-use/four-factors/.

Wagnon, J., "Web Scraping - Data Collection or Illegal Activity?", *F5 DevCentral* , 16 May 2013; https://devcentral.f5.com/articles/web-scraping-data-collection-or-illegal-activity#.UsKk-2RDtc_.

XPath Tutorial, W3Shools; http://www.w3schools.com/xpath/default.asp.