# Web Data Sources

The entity we chose was **Books**, and we extracted books from the **Fantasy genre** from -

1. Amazon.com
2. Goodreads.com

Our main motivation to do this was that both amazon.com and goodreads.com are owned by the same company (Amazon). We thought that it would be interesting to see what kind of books were cross-listed, and if listings on one site influence listings on the other.

## Description of Amazon

We scraped books from the fantasy genre on Amazon. This was a two step process:

**Step I**

- First, we perform a query for “fantasy novels” and Amazon returns 100 pages, each containing 11 results to the query. We might also occasionally see a sponsored result and hence the number of results per page varies between 11 to 12. Thus we could scrape roughly 1100 results this way, per query. For each search result, we scrape the name and the ID of that novel. (ID is an alphanumeric string used by Amazon to uniquely identify each book. This ID is specific to Amazon only and carries no significance elsewhere)
- Since fantasy is a very broad genre, we used three facets to refine the search results. They were “Mystery, Thriller & Suspense”, “Fantasy Action & Adventure” and “Military Fantasy”. Scraping approximately 1100 results from each facet gave us a total of about 3400 tuples.
- The code for the first step can be found here: AmazonScraper.py

**Step II**

- In the second step, for each of the IDs obtained in the previous step, we make a request to the page where the book can be purchased. This page contains the product description and details such as publisher, publishing date, ISBN number, rating and other attributes outlined below.
- We obtained roughly 3400 tuples at the end of this process. A few of the tuples contained null values and were dropped.
- The code for the second step can be found here: fetchDetails.py

## Description of Goodreads

We scraped books from fantasy and fiction genre from Goodreads. We used the Scrapy framework for this task, here are the steps:

- We used the scrapy startproject command to create the project and subclassed scrapy.Spider to create the spider.
- We populated the start_urls with the weblinks from querying goodreads.com with keywords “fantasy” and “fiction”.
- Scrapy provides a callback function which receives the response body when the crawler bot reads the webpage. We used XPaths to extract text from elements that we matched using nested classes and ids. We also performed some string regex and datetime operations to extract data values for each attribute, as explained in the next section.
- Because the search results were paginated, we programmed the crawler to follow the ‘next’ link to continue crawling until we hit our ceiling of 3000 pages.
- We also used the AutoThrottle extension to ease the traffic generated from our IP. The didn’t fire concurrent requests and added a little latency between each download request, starting initially at 5 seconds and incrementally tuning it after each request.
- Lastly we used the scrapy crawl command to start the crawler and saved the results to a JSON file.
- The code for the goodreads crawler can be found here and the configuration file here.

# Extraction of Structured Data

## Amazon

We used *beautifulsoup,* an open source python module to scrape data from Amazon.

**Step I**
We scraped the book name and ID from each search result, across multiple pages. A sample URL which we used was:

*https://www.amazon.com/s/ref=sr_pg_**<pageNumber>**?rh=n%3A283155%2Cn%3A25%2Cn%3A16190%2Cn%3A14051773011%2Ck%3Afantasy%2Cp_n_feature_browse-bin%3A2656022011&page=**<pageNumber>***

By varying the page number, we were able to go through each of the 100 pages of the search results. We used a total of 4 such URLs and all were similarly formatted.

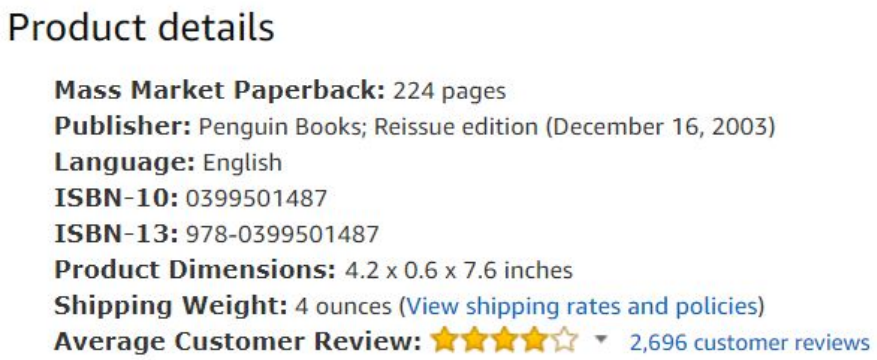
The name and ID for each result were extracted by manually constructing a wrapper and the rules. We obtained both from the following rule / CSS selector:

*'#result_**{resultNumber}** > div > div > div > div.a-fixed-left-grid-col.a-col-right > div.a-row.a-spacing-small a'*

Each page contains between 11 to 12 results and thus by varying the page number and the result number, we were able to pull the names and IDs of 3400 books.

**Step II**
The other attributes were scraped as follows:

- "Author" could be found in two different and mutually exclusive regions in the web page. Hence we handled both the scenarios. The rules corresponding to them are:
    - *a.a-link-normal.contributorNameID*
    - *#byline > span > a*
- "Rating" was scraped by the following rule:
    - *Span.arp-rating-out-of-text*
- The remaining attributes were scraped from the product description table:



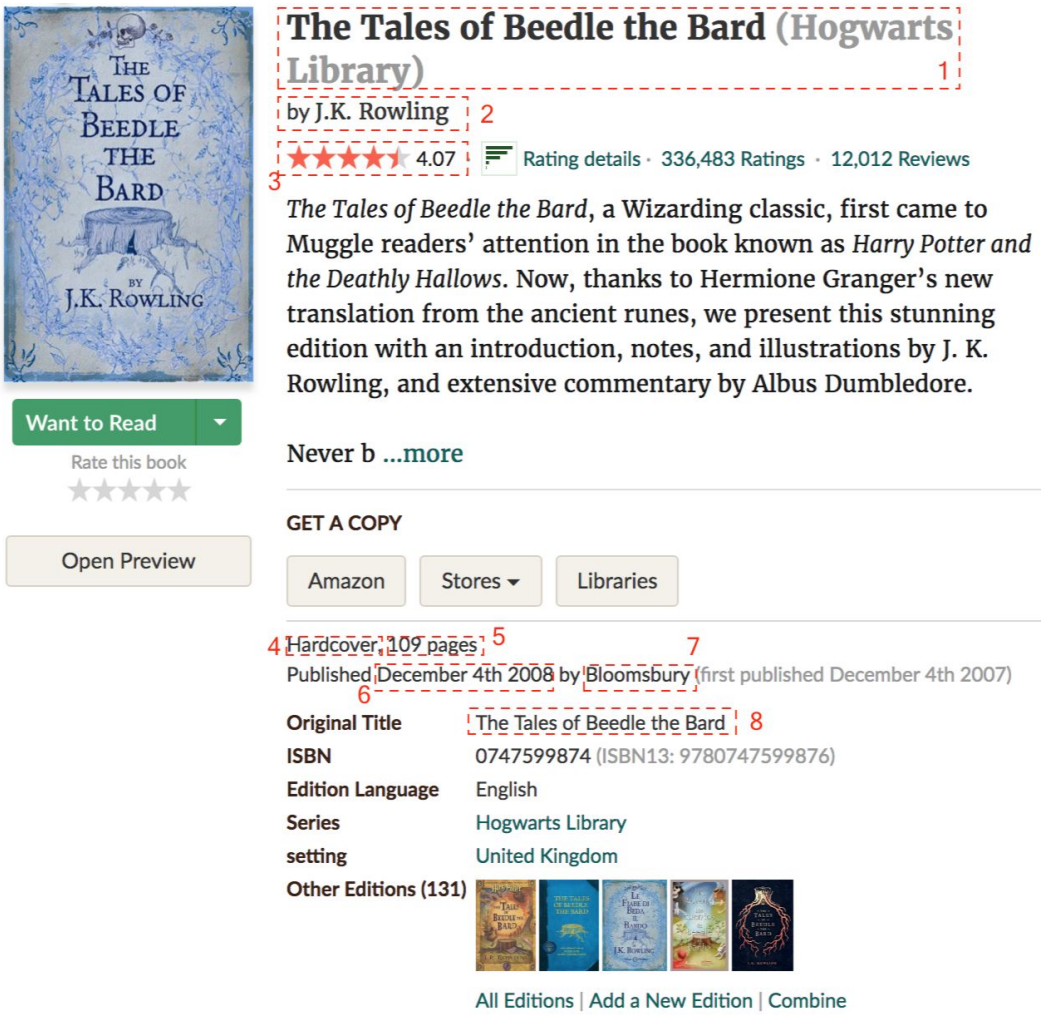

    - Rule: *table#productDetailsTable div.content ul li*
    - The bold text corresponded to the attribute name and the text following the bold text corresponded to the value of that attribute.
    - Note that while the table above contains "Rating", the text corresponding to the rating (*4 stars* in the above example) was obtained only on a mouseover. Hence we did not extract rating from this table.

## Goodreads

The id and source attributes were extracted from the web URL and the remaining data attributes were scraped from the webpage as follows.

1. Book_name: extracted using xpath with id selector.
2. Author: extracted using xpath with class selector.
3. Average rating: extracted using xpath with class selector.

4. Book format: extracted using xpath with attribute ‘itemprop’ for matching.
5. Page count: extracted using xpath with attribute ‘itemprop’ for matching.
6. Date Published: it’s xpath returned it as path of a larger sentence, so we used the structured phrasing of this sentence to extract the date phrase as a string and then used datetime.strptime to extract the formatted date.
7. Publisher: extracted as part of the same sentence as #6.
8. (Alternate) Book Name: extracted using xpath with class selector.




**Entity and Table details :**
As mentioned earlier, the entity we extracted was Books.

| Source | No. of Tuples | Schema<br>Column Name - Type |
|---|---|---|

| | | |
|---|---|---|
| Amazon | 3387 | ID - Long<br>Name - String<br>Author - String<br>Rating - Float (out of 5)<br>Format - String<br>Publisher - String<br>Pages - Int<br>Publishing Date - String<br>(format - month day, year)<br>ISBN-10 - Alphanumeric<br>ISBN-13 - Alphanumeric |
| Goodreads | 3001 | Author - String<br>Avg_rating - Float (out of 5)<br>Book_format - String<br>Book_name - String<br>Book_name_alt - String<br>Count_pages - Int<br>Date_published - String<br>(format - year-month-day<br>hours-minutes-seconds)<br>Id - Alpha numeric<br>Publisher - String<br>Source - Alpha numeric |

The final common schema that we created was -

| S. No. | Column Name | Type | Description |
|---|---|---|---|
| 1 | Name | String | Name of the book |
| 2 | Author | String | Name of the author of the book |
| 3 | Publisher | String | Name of the publisher |
| 4 | Publishing_Date | String | Date published. Format - year-month-day |
| 5 | Format | String | Format of the book - Paperback, Kindle, etc. |
| 6 | Pages | Int | Number of pages |
| 7 | Rating | Float | Book rating out of 5 |

**Open source tools used -**

1. Beautifulsoup - Beautifulsoup is a python library used to extract data out of HTML and XML files. Data is extracted into beautifulsoup objects, from which can extract multiple things, such as text belonging to a particular tag, all of the text in the HTML file, etc.
2. Scrapy - Scrapy is an open source framework used to crawl web pages and extract structured data. We write rules according to the web site we are scrapping, and scrapy asynchronously processes crawling requests and extracts data into python dictionaries. It also provides control over things like delay between each crawl request, number of concurrent requests per domain per IP, etc.