Adding Websites for Retail Assistant to Scrape

Vadim Torgashov

2020

1 Requirements

This project requires the use of these Python libraries: Beautiful Soup, Requests, psutil, PyInstaller, FreeProxy, and LXML. This was also made with Python 3.7, so if you use Python 2 your mileage may vary.

2 Checking the robots.txt

You should always scrape only when given expressed permission by the website owner. The easiest way to check for that is by going on the robots.txt of the website. Simply take the default URL like "www.ebay.com" and add "/robots.txt" on the end. The section you want to check is under "User-Agent", under it will be a list of routes labeled as "Allow" or "Disallow". If the only thing under "User-Agent" is "Disallow: /" or "Disallow: /*" then that means absolutely no crawling or scraping of the website is permitted and you should respect their wishes. However if crawling is allowed on some directories then look at what directory a normal search goes to. If your website has a way to order the results by newest items added you should use that ordering to make sure that the route does not change. If the route does change then make sure that route is allowed.

3 How to add the website to the GUI

In the file "RetailerAssistant.py" go to variable "retailer_list" at line 19 and add the name of the website as a string to that list. Be aware of exactly how you write the site's name as it will need to match what the scraper uses.

4 How to add the website's webstyle

A webstyle is the database entry that informs the scraper how to find necessary information on a webpage. The file "webstyle_generator.py" is prepared to add, test, and finalize the webstyle entry for your desired website. To begin with you want to make sure variable "testing" at line 10 is set to True to assure you run the test and your database entry will be added to a separate file from the default webstyle database. If you need to reset the default webstyle database delete "webstyles.db" and run "restore_webstyles.py". The main area of concern is the collection of variables going from line 21 to line 44. Firstly set the "retailer_name" variable at line 22 to the same string as you added to the "retailer_list" in the GUI. This includes identical spacing, capitalization, and punctuation. You must now assign the link to variable "website" at line 21.

4.1 Setting the link

To get the best link you should search for an item, set the department/category to "all" (if the website categorizes by departments like Ebay and Amazon), sort by newest items added, and set the items per page to the largest number, and put it on page 2 (some websites have a different URL once you go past page 1). Once you have your link you should isolate where the search term is in the link and replace it with "%s". Then find where the page number is and replace it with "%d". Here is a valid link for Ebay as an example: "https://www.ebay.com/sch/i.html?_nkw=%s&_sop=10&_ipg=200&_pgn=%d". When you take that link you can replace the "%s" with whatever you want to search for and replace the "%d" with the page number and the URL should take you to the correct page. One thing to note is that the program simply increases the page number that is inside the link. A lack of a page number in the link causes the program to simply scrape one page and nothing else. Which may or may not work for your needs. If you still want to progress through the result pages without a "%d" in the link then its up to you to create code to traverse the website's result pages. The URL can change if you click past the first page of results and the page number tends to be there.

4.2 Setting up the HTML elements

The remaining variables are all for HTML element names and for how far the program has to look. If the HTML element for something like the price has a name or id then assign that name as the "*_html" variable. The format to assign any HTML element is "element,id_type:name". With the element being something like a 'span', the id_type is an 'id','class', etc., and the name is what the id_type is assigned as. One example would be: "span,class:s-item__price". The 'extra_html' variable is used for any unique elements a item listing on that website can have. Such as Amazon items that are labeled as Prime. The "*_depth" variables are to only be used if the desired element has no name or id. You can select a parent element that is named and designate how many subelements the program must traverse to find the desired element. The parent element is not counted in the depth count and neither is the desired element. The count is done linearly so if a subelement has its own sub elements they will also be counted. Leave the depth count at 0 if the desired element is the one assigned to its "_html" variable. If an element is not present or not to be included simply leave the variable as "ex". There are far too many variables on purpose. It provides all the options someone would want to search for, but likely won't use all for any given website.

4.3 Running the test

Once you have filled out all the fields you want to scrape for you need to switch the variable "test_search" on line 19 to an item to search for. Run the program. The first thing that is printed out should be the link used to get to the website. If the link is correct copy and paste it in your browser. The remaining items printed

to the screen will be items that the program scraped from that link. Check to make sure what has been scraped is accurate to the real page. If any changes need to be made simply modify the necessary variables. If any special parsing has to be done to some elements that will be covered in the next section.

4.4 Finalizing the webstyle

Once a successful test has been ran the "testing" variable on line 10 must be switched to False and the program reran again. At this point you can delete the file "data/webstyles_test.db".

5 How to parse incoming data

Some websites might have unique methods of showing their data and coding in every website's formatting is impossible. The current way data is parsed assumes links are an href, any links that start with "/" will be appended to the website's default URL (such as appending "/itm..." to "www.ebay.com"), getting most data simply involves looking in between the ">" and "<" of any given element, and if the space between ">" and "<" is blank the program will go to the next subelement until it gets data. On top of any unique formatting a website may have some items may have unique formatting different than whats normally on the site. Ebay for instance will change the structure of the title element when an item has the label "New Listing". In the file "RetailAssistantScraper.py" you must subclass the class "CustomFilter". The name of your class must be "custom_" + retailer + "_filter", with the retailer being written the exact same way as how it was added to the GUI and webstyle. If the name of the retailer has spaces replace those with underscores. The subclass is to only have one static method: "filter". The method is to receive the element list formatted by the existing program and the original unmodified list (in that order). This way you can look at how the program formatted the elements and reformat them yourself using the original list. Make sure to return the modified list. I would suggest simply taking the reformatted element and overwriting it in the "filtered_listing" list, rather than reformatting the entire "unfiltered_listing" list (unless everything is to be reparsed). The data and the order it appears in for both lists in a comment on the first line of function "special_site_formatting_controller". There is examples below of how the class and method looks like.

```
class custom_Ebay_filter(CustomFilter):
    @staticmethod
    def filter(filtered_listing, unfiltered_listing):
```

6 How to edit the listing before adding to database

Once all the data has been correctly parsed and put into a list you can do last minute modifications before it is stored. Such as adding characters you want to be displayed in the GUI, moving certain data to a different field, etc. In the file "RetailAssistantScraper.py" you must subclass the class "CustomFormatting". The name of you class must be "custom_" + retailer + "_formatting", with the retailer being written the exact same way as how it was added to the GUI and webstyle. If the name of the retailer has spaces replace those with underscores. The subclass is to only have one static method: "format". The method is to receive 1 variable which will be a listing. The data a listing contains is in the same order as used in the previous section. Make sure to return the modified listing. Below are two examples of how the class and method looks like.

```
class custom_Property_Room_formatting(CustomFormatting):
    @staticmethod
    def format(listing):
```

```
class custom_Ebay_formatting(CustomFormatting):
    @staticmethod
```

7 Putting the whole thing together

Go into the folder where all the .py files are saved using command line. Run these commands to produce .exe files needed: "pyinstaller –onefile –windowed RetailAssistantScraper.py" and "pyinstaller –onefile –windowed RetailerAssistant.py".