##### Web Scraping using Python

##### A PROJECT REPORT

###### *Submitted by*

###### Mr. Pratyush Bajaj



***in partial fulfillment for the award of the degree***

***of***

##### B. Tech(Bachelor of Technology)

IN

**Graphic Era (Deemed to be University)**

Clement Town, Dehradun -248001

**Introduction:**

Web Scraping, also known as web harvesting or web data extraction, is used to extract data from websites. Although it can be done manually by a user, it is generally done by a bot or an AI controlled web crawler. In this data extraction information of similar types is extracted from a web site and is stored in a spreadsheet or a central local database, which is later used for analysis.

Webpages are build using text-based mark-up language, i.e. HTML and XHTML, which generally contains useful data in text format. However, as we see these pages do not display the HTML directly and are made for the end user and not made for automated use. As a result, the web scrapers come into play and make our lives easier.

**Motivation:**

There have been times when scrolling through websites is much more time consuming and there are thing which are rather easier if we had a list to go through rather than scrolling though webpages. I personally faced this problem when I was myself searching to buy a mobile phone for myself. I spent endless times checking different products. With the help of web scraping I can easily get the products I wish along with it’s its rating and URL available to us the very same instant on a single page.

**Techniques:**

Web Scraping is the process of automatically mining data or collecting information from the World Wide Web. There are methods that some websites use to prevent web scraping, such as detecting and disallowing bots from crawling (viewing) their pages. In response, there are web scraping systems that rely on using techniques such as DOM (Document Object Model), computer vision and natural language processing to simulate human browsing to enable gathering web page content for offline parsing. Current web scraping solutions range from the ad-hoc, requiring human effort, to fully automated systems that can convert entire websites into structured information, with limitations.

* **Human copy-and-paste**
* **Text pattern matching**
* **HTTP programming**
* **HTML parsing**
* **DOM parsing**

**Procedure:**

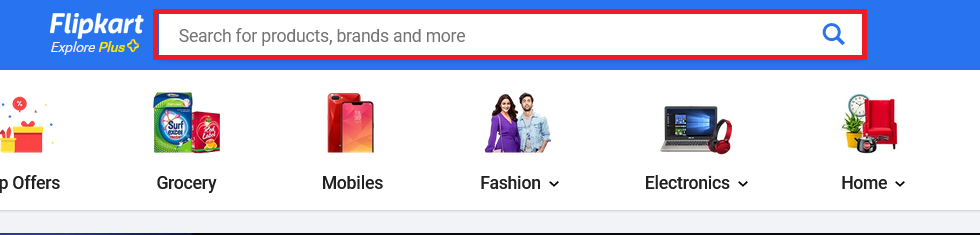
The library of codes we are going to use for this project are:

* Requests Library
* Beautiful Soup Library
* Pandas

The use of each of these will be explained with the code.

I have made two different projects to demonstrate the working of a web scraper. Later in this report I have explained the working of one of them and code of the other.

**Topic 1: Web Scraping from a Searched Flipkart product**

The most important and the first step to web scraping is to examine closely the website from which we want to extract the data from. In this topic I have chosen to scrape the webpage when we search for a product on the Flipkart search bar.

(**Fig 1.1** Flipkart Search Bar)

As seen in **Fig 1.1** when we go to the homepage of Flipkart, we see a page similar to what we see above. On the very top center of the page, we can see the search handle as highlighted in the above image.

Graphical user interface, text, application, email

Description automatically generatedNext, we search for the products we wish to scrape the data for, in this case we need variety, so I search for mobile phones under 20000, which gives us a plenty of data to be analyzed and scraped from. After searching the said thing, we get to a page like this.

(**Fig 1.2** Searched page on Flipkart)

Here we see that there are similar types of data such as the name of the product, the price of the product, number of reviews and ratings and the number of stars received by the product. Using the inspector, we will see if the similar type of data has similar HTML classes too, by which we can easily select the classes we want to extract from this webpage. We will check the classes of the names of the two products displayed above.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence(Fig 1.3 Inspector view of the first product)

(Fig 1.3 Inspector view of the second product)

If we notice the highlighted area, we can see that the classes under which the names of both the product’s name is given, is the same. Similarly, we check all the other data that we wish to extract and then proceed to building the web scraper.

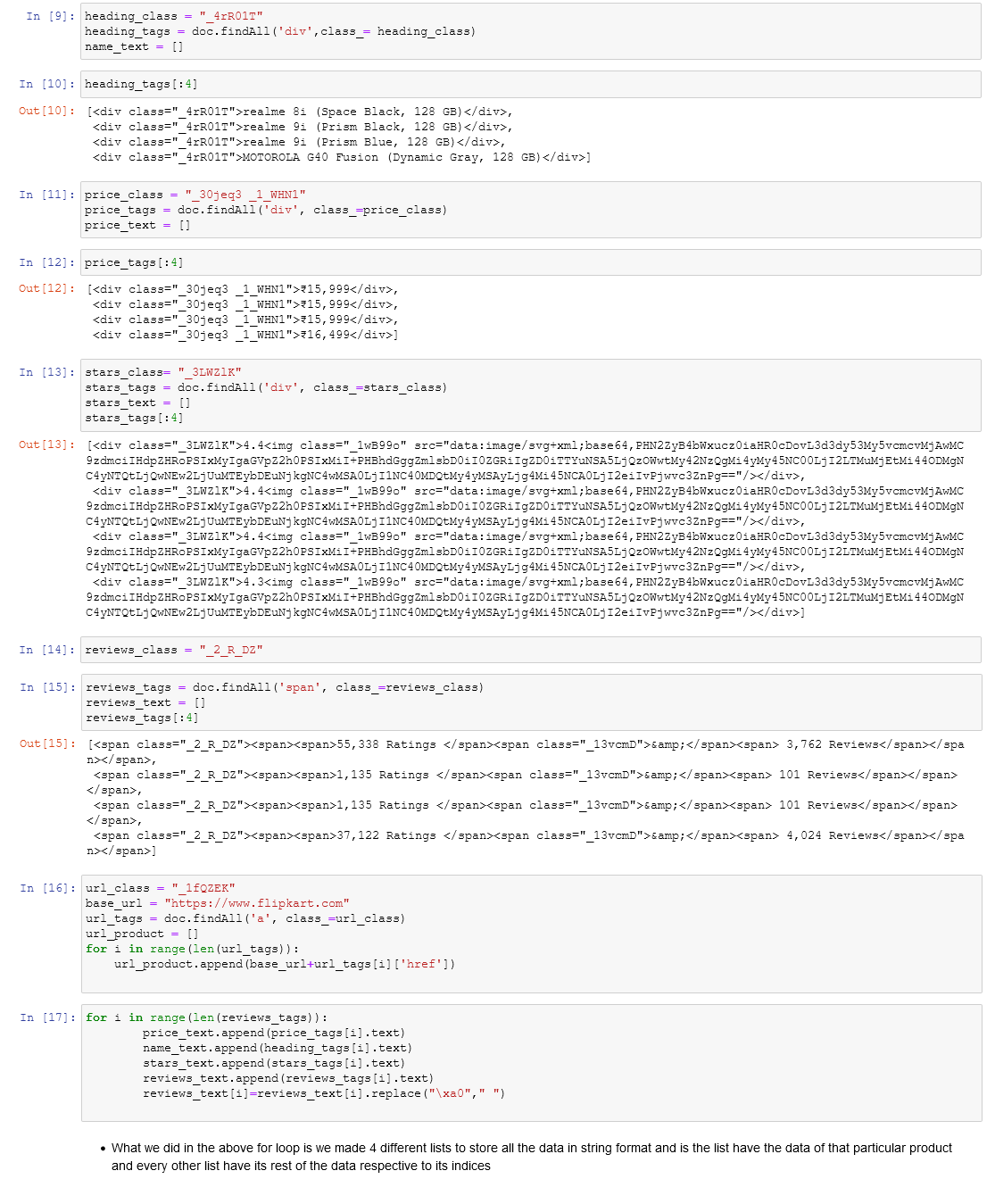
For this project I’ve used Jupyter Notebook because data because they are great for showcasing my work. We can see the code and its result together. We can run cell by cell for better understanding and we can also insert markdown which is better than normal comments because we can use things like bullet points and makes the project look organized.

Graphical user interface, text, application, email

Description automatically generatedFirst, we install the Requests and BeautifulSoup library and store the URL of the webpage in a variable ‘products\_url’ and requests we GET from the webpage discussed above.

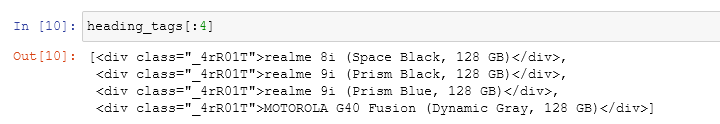
(Fig 1.4 Installing the libraries and parsing webpage content)

Then we store the content and of the page and using Beautiful Soup we parse the html and sore it in a variable named ‘doc’.

Next as discussed in Fig 1.3 we use all the classes and store them in a variable corresponding to the class of what we’re looking for i.e., name, price, stars, etc.

(Fig 1.5 Using the classes extracting information)

In the Fig 1.5, after examining the HTML of the webpage we choose the classes of the data we wish to extract that is name of the product, price of the product, the stars received by the product, the number of ratings and reviews received by the product and the URL of the product (the URL is not a separate entity that is displayed on the webpage like the other data that is extracted). In the name on inspecting, we see that it is stored in a <div> so using BeautifulSoup we use the doc we created above and use the findAll function. The findAll function takes 2 arguments i.e., the tag the data is stored under (in the case of the name of the product it is stored under the <div>) and the name of the class of the data we wish to extract (in this case which we have stored under ‘heading\_class’) and store the data received in a list called ‘heading\_tags’.

After getting all the heading tags we analyze the data we got (since there is a lot of data under the same class, we only check the first four because the structure of the rest will be same).

(Fig 1.6 The data stored in the ‘heading\_tags’)

We see we do not get the exact name of the product rather we get the entire <div> tag i.e., the tag under which the name of the product is stored.

Similarly, we do the same thing for every class and the entire tags that hold the data we want and store it into their corresponding variables.

In the code with the extraction of every tag we have also initialized n list with identifiers like ‘name\_text’, ‘price\_text’ and so on. These lists will be used later to hold the exact information we want, that is without the <div> tags and all the unnecessary things.

If we notice, I did something extra for the ‘url\_tags’. We did that because on inspecting we saw that the <a> tags didn’t store the complete URL of the webpage of the product and was without the – ‘https://flipkart.com’. So, I created a variable called ‘base\_url’ using which stored the missing part of the URL. Next, we iterate through every URL tag and using append function and append the ‘base\_url’ with the ‘href’ part of the ‘url\_tags’ and store it in ‘url\_products’ list. After coming out of the for loop this list will have the exact URL to the product.

In a similar manner to what we did to extract the text part from the ‘url\_tags’ we extract the data for name, price, stars, reviews and ratings. In the reviews and ratings, we were getting some part in the i.e., “\xa0” so we replace it with “ ”.

With this we have extracted all the data we wanted from the webpage. Now, we use store all the data in a dictionary. We store the lists under their respective headings.

We then import the Pandas library to convert the dictionary to a dataframe. Then we display the first 4 items of data extracted to the dataframe to verify if we the data extracted is correct or not.

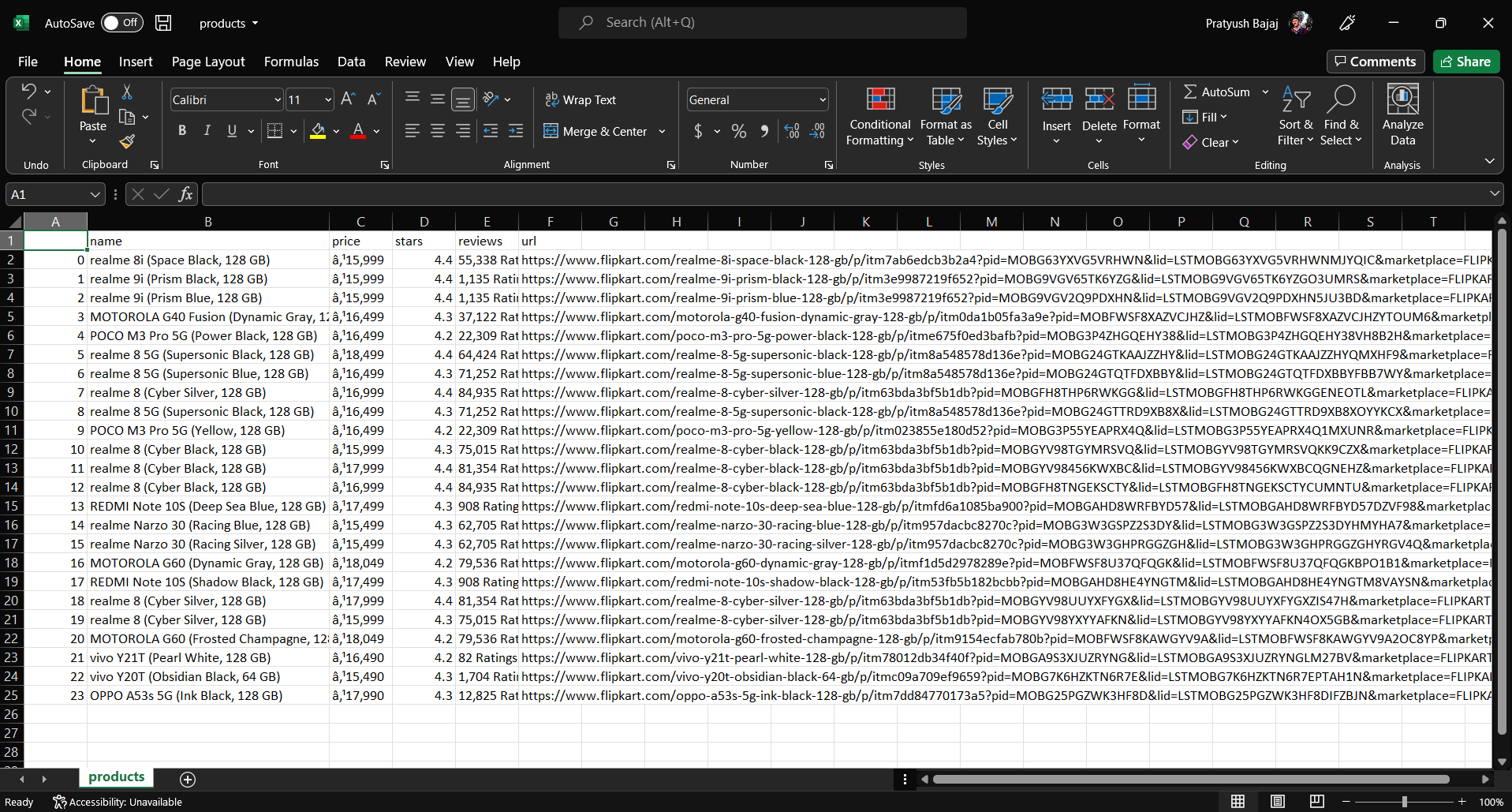
Table

Description automatically generated

(Fig 1.7 Verifying the extracted data)

Then we export the data in csv file which is created in the same directory as the project.

(Fig 1.8 Exporting to a CSV file)



(Fig 1.9 The exported CSV file)

**Summary:**

Web Scraping is an interesting and an extremely popular technique which proves itself to be quite handy to learn. There are several other libraries apart from Beautiful Soup. Scrappy is a very popular open source web crawling framework that is also written in Python. It’s ideal for web scraping and extracting data using API’s. Beautiful Soup is used to create a parse tree and extract data from the HTML of a webpage.

Web Scraping has multiple applications across various industries. Some of these are mentioned below:

* **Price Monitoring**
* **Market Research**
* **News Monitoring**
* **Sentiment Analysis**
* **Email Marketing**

This code can also be accessed from my GitHub Profile:

<https://github.com/BajajPratyush/web_scraping>

**Bibliography**

* <https://www.flipkart.com/>
* <https://github.com/topics>
* <https://en.wikipedia.org/wiki/Web_scraping>
* <https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/>
* <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>