**What is Web Scraping?**

“The data we have is too less to build a machine learning model. We need more data!”

If this sounds familiar, you’re not alone! It’s the eternal problem of wanting more data to train our [machine learning](https://courses.analyticsvidhya.com/courses/applied-machine-learning-beginner-to-professional/?utm_source=course&utm_medium=web-scraping-hands-on-introduction-python) models. We don’t get cleaned and ready-for-use Excel or .csv files in data science projects, right?

So how do we deal with the obstacle of the paucity of data?

One of the most effective and simple ways to do this is through web scraping. I have personally found web scraping a very helpful technique to gather data from multiple websites. Some websites these days also provide APIs for many different types of data you might want to use, such as Tweets or LinkedIn posts.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/10/web-scraping_2.jpg)

But there might be occasions when you need to collect data from a website that does not provide a specific API. This is where having the ability to perform web scraping comes in handy. As a data scientist, you can code a simple Python script and extract the data you’re looking for.

**Web Scraping** is a technique employed to extract large amounts of data from websites whereby the data is extracted and saved to a local file in your computer or to a database in table format.

So in this course, we will learn the different components of web scraping and then dive straight into [Python](https://courses.analyticsvidhya.com/courses/introduction-to-data-science-2/?utm_source=course&utm_medium=web-scraping-hands-on-introduction-python) to see how to perform web scraping using the popular and highly effective *BeautifulSoup* library.

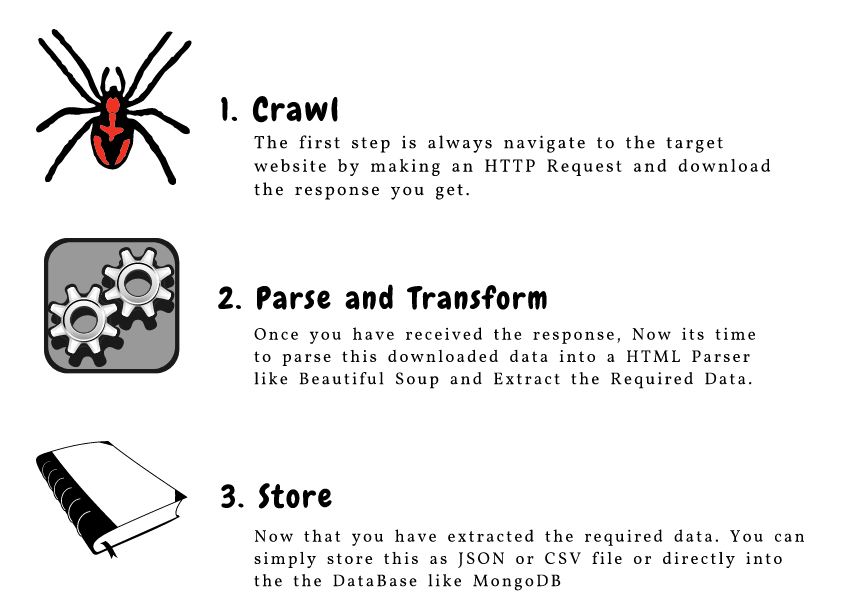
**Caution**

*A note of caution here – web scraping is subject to a lot of guidelines and rules. Not every website allows the user to scrape content so there are certain legal restrictions at play. Always ensure you read the website’s terms and conditions on web scraping before you attempt to do it.*

## Components of Web Scraping

Enable fullscreen

Here’s a brilliant illustration of the three main components that make up web scraping:

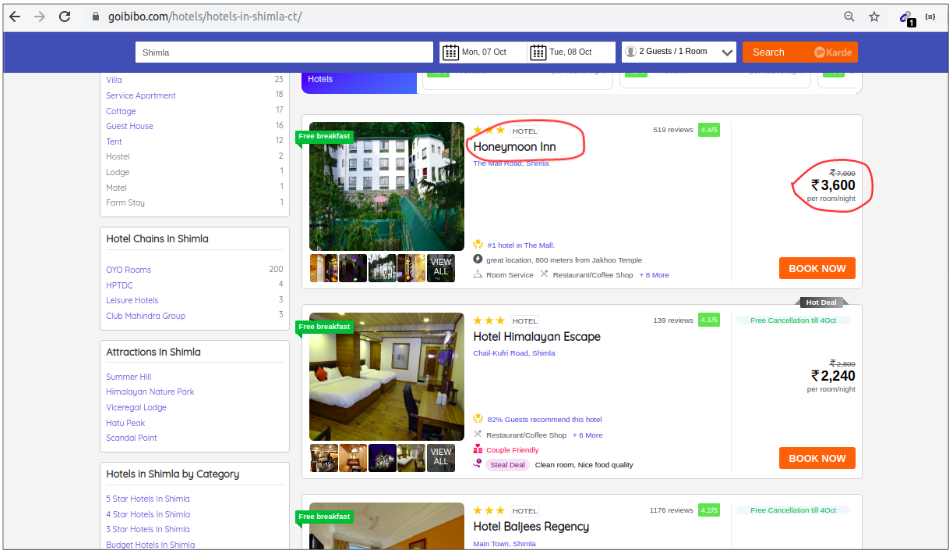
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/components-of-web-scraping.png)

Let us understand each of these components in the next module.

## Problem Setup

Enable fullscreen

We’ll understand the components involved by scraping hotel details like the name of the hotel and price per room from the goibibo website:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/target-url.png)

***Note:*** Always follow the [***robots.txt***](https://www.goibibo.com/robots.txt) file of the target website which is also known as the robot exclusion protocol. This tells web robots which pages not to crawl.

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/updated_robots_goibibo.png)

So, looks like we are allowed to scrape the data from our targeted URL. We are good to go and write the script of our web robot. Let’s begin!

## Step 1: Crawl

Enable fullscreen

The first step is to navigate to the target website and download the source code of the web page. We are going to use the [**requests**](https://pypi.org/project/requests/) library to do this. A couple of other libraries to make requests and download the source code are [*http.client*](https://docs.python.org/3/library/http.client.html#module-http.client) and [*urlib2*](https://docs.python.org/2/library/urllib2.html).

Once we have downloaded the source code of the webpage, we need to filter the contents that we need:

"""

Web Scraping - Beautiful Soup

"""

# importing required libraries

**import** requests

**from** bs4 **import** BeautifulSoup

**import** pandas **as** pd

# target URL to scrap

url = "https://www.goibibo.com/hotels/hotels-in-shimla-ct/"

# headers

headers = {

'User-Agent': "Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36"

}

# send request to download the data

response = requests.request("GET", url, headers=headers)

# parse the downloaded data

data = BeautifulSoup(response.text, 'html.parser')

print(data)

## Step 2: Parse and Transform

Enable fullscreen

The next step is to parse this data into an HTML Parser and for that, we will use the BeautifulSoup library. Now, if you have noticed our target web page, the details of a particular hotel are on a different card like most of the web pages.

So the next step would be to filter this card data from the complete source code. Next, we will select the card and click on the ‘Inspect Element’ option to get the source code of that particular card. You will get something like this:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/source-cord-card-mapping.png)

The class name of all the cards would be the same and we can get a list of those cards by just passing the tag name and attributes like the <class> tag with its name like I’ve shown below:

# find all the sections with specifiedd class name

cards\_data = data.find\_all('div', attrs={'class', 'width100 fl htlListSeo hotel-tile-srp-container hotel-tile-srp-container-template new-htl-design-tile-main-block'})

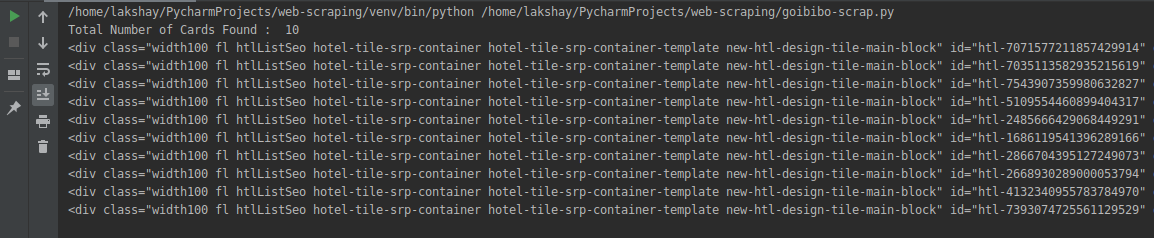
# total number of cards

**print**('Total Number of Cards Found : ', len(cards\_data))

# source code of hotel cards

**for** card **in** cards\_data:

**print**(card)



We have filtered the cards data from the complete source code of the web page and each card here contains the information about a separate hotel. Select only the Hotel Name, perform the Inspect Element step, and do the same with the Room Price:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/inspect-element-nested1.png)

Now, for each card, we have to find the above Hotel Name which can be extracted from the <p> tag only. **This is because there is only one <p> tag for each card and Room Price by <li> tag along with the <class> tag and class name:**

# extract the hotel name and price per room

**for** card **in** cards\_data:

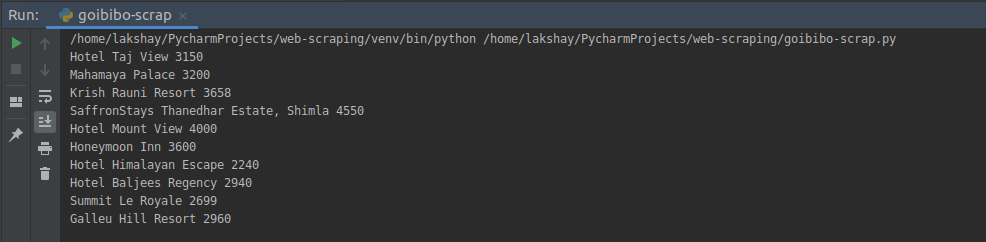
    # get the hotel name

hotel\_name = card.find('p')

    # get the room price

room\_price = card.find('li', attrs={'class': 'htl-tile-discount-prc'})

**print**(hotel\_name.text, room\_price.text)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/scaped-names-prices.png)

## Step 3: Store the Data

Enable fullscreen

The final step is to store the extracted data in the CSV file. Here, for each card, we will extract the Hotel Name and Price and store it in a Python dictionary. We will then finally append it to a list.

Next, let’s go ahead and transform this list to a Pandas data frame as it allows us to convert the data frame into CSV or JSON files:

# create a list to store the data

scraped\_data = []

**for** card **in** cards\_data:

    # initialize the dictionary

card\_details = {}

    # get the hotel name

hotel\_name = card.find('p')

    # get the room price

room\_price = card.find('li', attrs={'class': 'htl-tile-discount-prc'})

    # add data to the dictionary

card\_details['hotel\_name'] = hotel\_name.text

card\_details['room\_price'] = room\_price.text

    # append the scraped data to the list

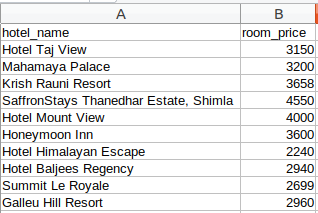
scraped\_data.append(card\_details)

# create a data frame from the list of dictionaries

dataFrame = pd.DataFrame.from\_dict(scraped\_data)

# save the scraped data as CSV file

dataFrame.to\_csv('hotels\_data.csv', index=False)

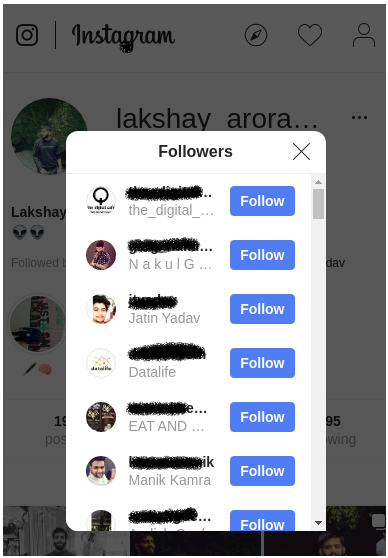
[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/csv-view.png)

Congrats! We have successfully created a basic web scraper. I want you to try out these steps and try to get more data like ratings and address of the hotel. Now let’s see how to perform some common tasks like scraping URLs, Email IDs, Images, and Scrape Data on Page Loads.

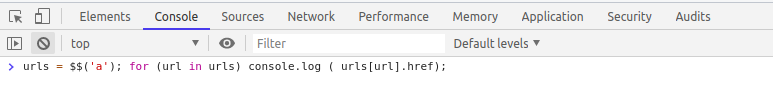
## Single Webpage Scraping

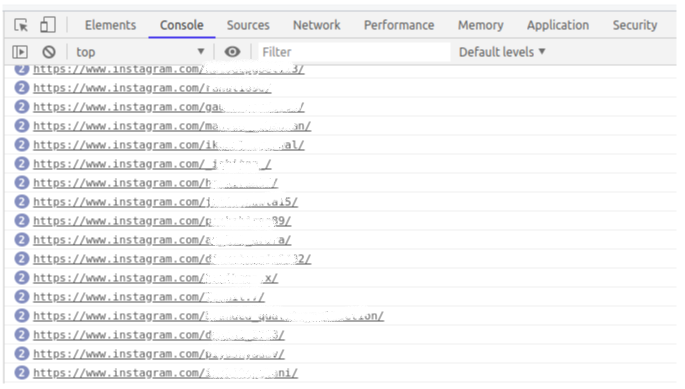
Two of the most common features we try to scrape are website URLs and email IDs. I'm sure you've worked on projects or challenges where extracting email IDs in bulk was required (see marketing teams!). So let's see how to scrape these aspects in Python.

### Using the Console of the Web Browser

* Let's say we want to keep track of our Instagram followers and want to know the username of the person who unfollowed our account. First, log in to your Instagram account and click on followers to check the list:                                            [](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/instagram_blurred.png)
* Scroll down all the way so that we have all the usernames loaded in the background in our browser's memory
* Right-click on the browser's window and click 'Inspect Element'
* In the Console Window, type this command:

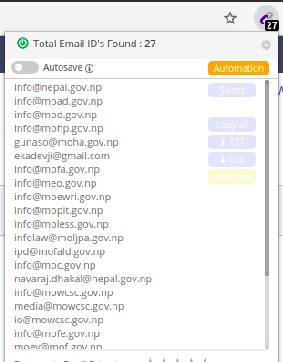
***urls = $$('a'); for (url in urls) console.log ( urls[url].href);***

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/console.png)

* With just one line of code, we can find out all the URLs present on that particular page:[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/usernames-blurred.png)
* Next, save this list at two different time stamps and a simple Python program will let you know the difference between the two. We would be able to know the username of who unfollowed our account!
* There can be multiple ways we can use this hack to simplify our tasks. The main idea is that with a single line of code we can get all the URLs in one go

### Using the Chrome Extension Email Extractor

* [Email Extractor](https://chrome.google.com/webstore/detail/email-extractor/jdianbbpnakhcmfkcckaboohfgnngfcc?hl=en) is a Chrome plugin that captures the Email IDs present on the page that we are currently browsing
* It even allows us to download the list of Email IDs in CSV or Text file:



## Multiple Webpage Scraping(BeautifulSoup and Regex)

The above solutions are efficient only when we want to scrape data from just one page. But what if we want the same steps to be done on multiple webpages?

There are many websites that can do that for us at some price. But here’s the good news – we can also write our own web scraper using Python! You can use the code below:

'''

Web Scraping - URLs and Email IDs

'''

# importing required libraries

**import** urllib.request

**from** bs4 **import** BeautifulSoup

# URL TO SCRAP

wiki = "https://dlca.logcluster.org/display/public/DLCA/4.1+Nepal+Government+Contact+List"

#Query the website and return the html to the variable 'page'

#For python 3 use urllib.request.urlopen(wiki)

page = urllib.request.urlopen(wiki)

#Parse the html in the 'page' variable, and store it in Beautiful Soup format

soup = BeautifulSoup(page,features='html.parser')

**print**('\n\nPage Scrapped !!!\n\n')

**print**('\n\nTITLE OF THE PAGE\n\n')

**print**(soup.title.string)

**print**('\n\nALL THE URLs IN THE WEB PAGE\n\n')

all\_links = soup.find\_all('a')

**print**('Total number of URLs present = ',len(all\_links))

**print**('\n\nLast 5 URLs in the page are : \n')

**if** **len**(all\_links) > 5 :

  last\_5 = all\_links[len(all\_links)-5:]

**for** url **in** last\_5 :

**print**(url.get('href'))

emails = []

**for** url **in** all\_links :

**if**(str(url.get('href')).find('@') > 0):

        emails.append(url.get('href'))

**print**('\n\nTotal Number of Email IDs Present: ', len(emails))

**print**('\n\nSome of the emails are: \n\n')

**for** email **in** emails[:5]:

**print**(email)

## Scrape Images

In this section, we will scrape all the images from the same goibibo webpage. The first step would be the same to navigate to the target website and download the source code. Next, we will find all the images using the **<img>** tag:

"""

Web Scraping - Scrap Images

"""

# importing required libraries

**import** requests

**from** bs4 **import** BeautifulSoup

# target URL

url = "https://www.goibibo.com/hotels/hotels-in-shimla-ct/"

headers = {

'User-Agent': "Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/77.0.3865.90 Safari/537.36"

}

response = requests.request("GET", url, headers=headers)

data = BeautifulSoup(response.text, 'html.parser')

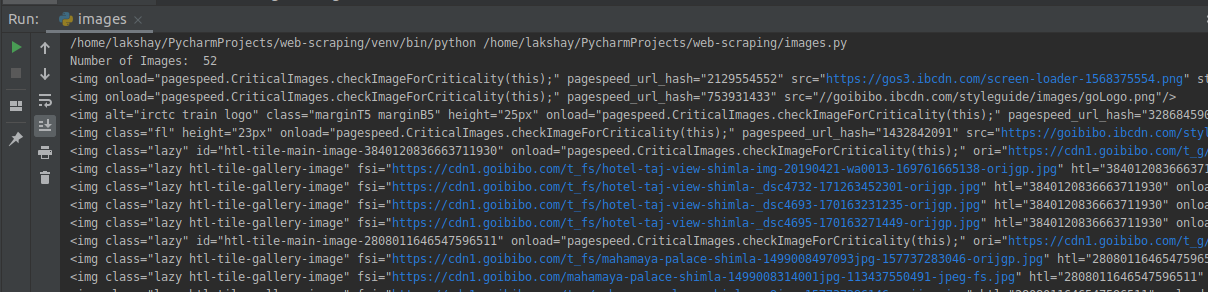
# find all with the image tag

images = data.find\_all('img', src=True)

**print**('Number of Images: ', len(images))

**for** image **in** images:

**print**(image)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/image_part_1.png)

From all the image tags, select only the ***src*** part. Also, notice that the hotel images are available in ***jpg*** format. So we will select only those:

# select src tag

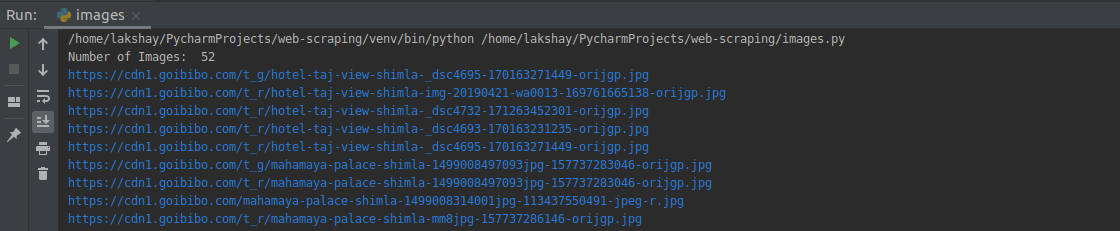
image\_src = [x['src'] for x in images]

# select only jp format images

image\_src = [x for x in image\_src if x.endswith('.jpg')]

**for** image **in** image\_src:

**print**(image)

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/image_part_2.png)

Now that we have a list of image URLs, all we have to do is request the image content and write it in a file. Make sure that you open the file ***‘wb’ (***write binary) form:

image\_count = 1

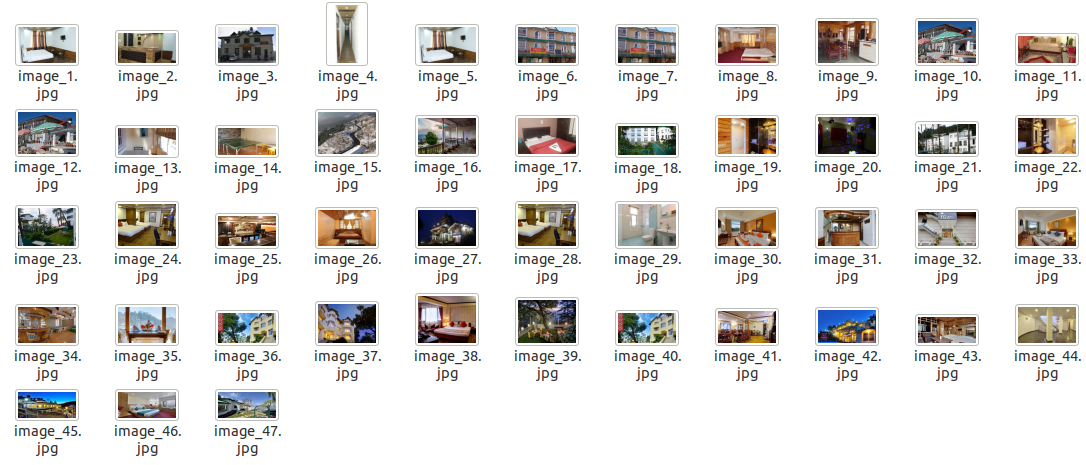
**for** image **in** image\_src:

**with** open('image\_'+str(image\_count)+'.jpg', 'wb') **as** f:

res = requests.get(image)

f.write(res.content)

    image\_count = image\_count+1

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/09/images_output.png)

You can also update the initial page URL by page number and request them iteratively to gather data in a large amount.

## Scarpe Data on Page Load

Enable fullscreen

Let’s have a look at the web page of the steam community [Grant Theft Auto V Reviews](https://steamcommunity.com/app/271590/reviews/?browsefilter=toprated&snr=1_5_100010_). You will notice that the complete content of the webpage will not get loaded in one go.

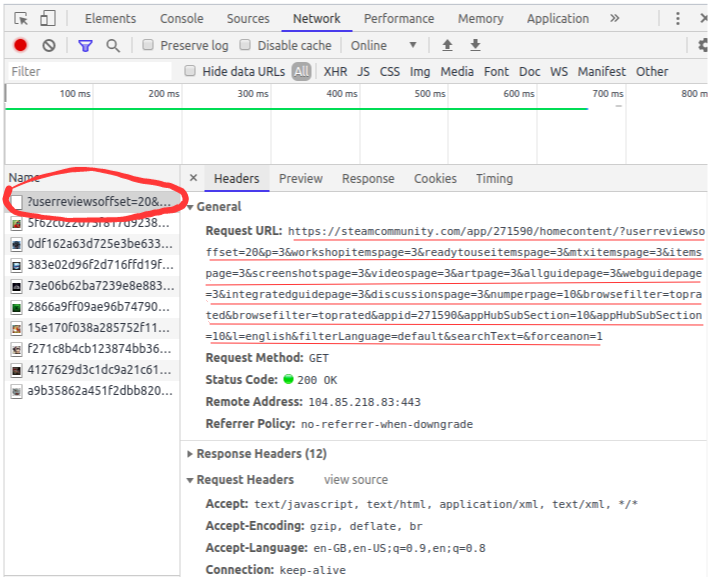
We need to scroll down to load more content on the web page (the age of endless scrolling!). This is an optimization technique called Lazy Loading used by the backend developers of the website.

But the problem for us is when we try to scrape the data from this page, we will only get a limited content of the webpage:

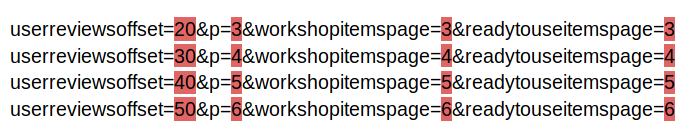
Some websites also create a ‘Load More’ button instead of the endless scrolling idea. This will load more content only when you click that button. The problem of limited content still remains. So let’s see how to scrape these kinds of web pages.

Navigate to the target URL and open the ‘Inspect Element Network’ window. Next, click on the reload button and it will record the network for you like the order of image loads, API requests, POST requests, etc.:

Clear the current records and scroll down. You will notice that as you scroll down, the webpage is sending requests for more data:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/10/edited_network_analysis.png)

Scroll further and you will see the pattern in which the website is making requests. Look at the following URLs – only some of the parameter values are changing and you can easily generate these URLs through a simple Python code:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2019/10/urls_analysis.png)

You need to follow the same steps to crawl and store the data by sending requests to each of the pages one by one.