Data Collection Flipkart

September 12, 2024

1 1. Importing Modules

[1]:
```
import requests
from bs4 import BeautifulSoup
import re
import numpy as np
import pandas as pd
import time
import random
```

1.1 Explanation for Modules

- `requests` module is to send a request to the URL to fetch the data.
- `BeautifulSoup` is a class using the object of which we will deal with the scraped HTML data.
- `re` is for using regex patterns to filter out data and create our dataframe in an organized format.
- `numpy` and `pandas` module if for manipulating data values and handling the data overall.
- `time` module is used to create a time delay during scraping.
- `random` module is used to generate random numbers to be used during scraping time delays.

2 2. Scraping All The Webpages Of Flipkart For Laptop Data

Click here for the Initial Site

- As highlighted in the red box of the above image we have access to 68 pages of flipkart to scrape the data from.
- So based on that we will write the code to scrape the data

[2]:
```
# Defining Request Headers to scrape the data
request_header = {
    'Content-Type': 'text/html; charset=UTF-8',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/
 ↪20100101 Firefox/119.0',
    'Accept-Encoding': 'gzip, deflate, br',
    'Referer': 'https://www.flipkart.com/',
    'Origin': 'https://www.flipkart.com',
    'Accept-Language': 'en-US,en;q=0.9'
}
```

2.0.1 Understanding The Request Headers

- **Content-Type**: This tells the server what kind of data you're sending. In this case, it's HTML text with a specific character set (UTF-8).
- **User-Agent**: This identifies the browser and operating system making the request. It helps the server understand how to format the response. For example, we are using Firefox on a Windows 10 machine.
- **Accept-Encoding**: This tells the server which compression methods your client can handle. Here, it indicates that the client can accept responses compressed with gzip, deflate, or br (Brotli).
- **Referer**: This indicates the URL from which the request originated. It helps the server understand the context of the request. In this scenario, it shows that the request is coming from Flipkart's website.
- **Origin**: Similar to Referer, this specifies the origin of the request, which is also Flipkart in this case. It's used for security purposes, particularly with cross-origin requests.
- **Accept-Language**: This tells the server which languages your client prefers. Here, it indicates a preference for US English, but can also accept other forms of English.

[3]:
```
# Storing the URL as a f-string
page = 1
URL = f"https://www.flipkart.com/search?
 ↪q=laptop&otracker=search&otracker1=search&marketplace=FLIPKART&as-show=on&as=off&page={page
```

- The URL is stored as f-string becuase, by changing the page number in the URL, we can access the next page data, this can be utilized in conjunction with for loop to scrape all the available data.

[4]:
```
# Scraping Code

total_pages = 68 # Total number of pages being scraped
i = 1 # Counter to self verify the pages being scraped successfully
raw_text = [] # List to store all the raw html code

# Loop to iterate over all the pages by changing the f-string URL
for page in range (1, total_pages+1):

    # Fetching the data from URL based on the above request headers
    response = requests.get(URL, headers=request_header)

    # Random number to be used as time delay in order to make the script␣
 ↪behaviour more human like
    delay = random.randint(5,10)
    print("Time Delay:",delay,end=" seconds    : ")

    # While Loop: covers the edge case wherein the first attempt to fetch the␣
 ↪data failed,
    # by continuously requesting the data at irregular time intervals in order␣
 ↪to mimic human behavior
```

```
    while response.status_code!=200:
        time.sleep(delay)
        response = requests.get(URL,headers=request_header)

    # Confirmation Message of Successful Scrape
    print("Page",i," status:",response)

    # Incrementing Page Counter
    i+=1

    # Appending the raw HTML code in the list
    raw_text.append(response.text)

    # A random delay before requesting the data from next page
    time.sleep(delay)
```

```
Time Delay: 8 seconds    : Page 1  status: <Response [200]>
Time Delay: 10 seconds    : Page 2  status: <Response [200]>
Time Delay: 9 seconds    : Page 3  status: <Response [200]>
Time Delay: 10 seconds    : Page 4  status: <Response [200]>
Time Delay: 9 seconds    : Page 5  status: <Response [200]>
Time Delay: 8 seconds    : Page 6  status: <Response [200]>
Time Delay: 5 seconds    : Page 7  status: <Response [200]>
Time Delay: 7 seconds    : Page 8  status: <Response [200]>
Time Delay: 8 seconds    : Page 9  status: <Response [200]>
Time Delay: 5 seconds    : Page 10  status: <Response [200]>
Time Delay: 10 seconds    : Page 11  status: <Response [200]>
Time Delay: 9 seconds    : Page 12  status: <Response [200]>
Time Delay: 8 seconds    : Page 13  status: <Response [200]>
Time Delay: 6 seconds    : Page 14  status: <Response [200]>
Time Delay: 9 seconds    : Page 15  status: <Response [200]>
Time Delay: 6 seconds    : Page 16  status: <Response [200]>
Time Delay: 9 seconds    : Page 17  status: <Response [200]>
Time Delay: 9 seconds    : Page 18  status: <Response [200]>
Time Delay: 9 seconds    : Page 19  status: <Response [200]>
Time Delay: 10 seconds    : Page 20  status: <Response [200]>
Time Delay: 10 seconds    : Page 21  status: <Response [200]>
Time Delay: 8 seconds    : Page 22  status: <Response [200]>
Time Delay: 6 seconds    : Page 23  status: <Response [200]>
Time Delay: 8 seconds    : Page 24  status: <Response [200]>
Time Delay: 10 seconds    : Page 25  status: <Response [200]>
Time Delay: 6 seconds    : Page 26  status: <Response [200]>
Time Delay: 6 seconds    : Page 27  status: <Response [200]>
Time Delay: 9 seconds    : Page 28  status: <Response [200]>
Time Delay: 8 seconds    : Page 29  status: <Response [200]>
Time Delay: 7 seconds    : Page 30  status: <Response [200]>
Time Delay: 7 seconds    : Page 31  status: <Response [200]>
Time Delay: 10 seconds    : Page 32  status: <Response [200]>
```

```
Time Delay: 10 seconds    : Page 33  status: <Response [200]>
Time Delay: 8 seconds    : Page 34  status: <Response [200]>
Time Delay: 7 seconds    : Page 35  status: <Response [200]>
Time Delay: 8 seconds    : Page 36  status: <Response [200]>
Time Delay: 5 seconds    : Page 37  status: <Response [200]>
Time Delay: 7 seconds    : Page 38  status: <Response [200]>
Time Delay: 5 seconds    : Page 39  status: <Response [200]>
Time Delay: 7 seconds    : Page 40  status: <Response [200]>
Time Delay: 5 seconds    : Page 41  status: <Response [200]>
Time Delay: 6 seconds    : Page 42  status: <Response [200]>
Time Delay: 6 seconds    : Page 43  status: <Response [200]>
Time Delay: 8 seconds    : Page 44  status: <Response [200]>
Time Delay: 5 seconds    : Page 45  status: <Response [200]>
Time Delay: 9 seconds    : Page 46  status: <Response [200]>
Time Delay: 10 seconds    : Page 47  status: <Response [200]>
Time Delay: 10 seconds    : Page 48  status: <Response [200]>
Time Delay: 5 seconds    : Page 49  status: <Response [200]>
Time Delay: 9 seconds    : Page 50  status: <Response [200]>
Time Delay: 5 seconds    : Page 51  status: <Response [200]>
Time Delay: 5 seconds    : Page 52  status: <Response [200]>
Time Delay: 9 seconds    : Page 53  status: <Response [200]>
Time Delay: 8 seconds    : Page 54  status: <Response [200]>
Time Delay: 8 seconds    : Page 55  status: <Response [200]>
Time Delay: 7 seconds    : Page 56  status: <Response [200]>
Time Delay: 8 seconds    : Page 57  status: <Response [200]>
Time Delay: 5 seconds    : Page 58  status: <Response [200]>
Time Delay: 6 seconds    : Page 59  status: <Response [200]>
Time Delay: 9 seconds    : Page 60  status: <Response [200]>
Time Delay: 9 seconds    : Page 61  status: <Response [200]>
Time Delay: 6 seconds    : Page 62  status: <Response [200]>
Time Delay: 9 seconds    : Page 63  status: <Response [200]>
Time Delay: 8 seconds    : Page 64  status: <Response [200]>
Time Delay: 7 seconds    : Page 65  status: <Response [200]>
Time Delay: 5 seconds    : Page 66  status: <Response [200]>
Time Delay: 9 seconds    : Page 67  status: <Response [200]>
Time Delay: 5 seconds    : Page 68  status: <Response [200]>
```

3 3. Saving The Raw HTML Data in CSV

- Now we will save the raw HTML code for each page in a CSV by converting the list into a dataframe.
- Saving in CSV will ensure that we don't have to scrape the entire data everytime we want to work on the data as scraping itself is a time consuming process.

[5]:
```
# Converting the list to Data Frame
df = pd.DataFrame(raw_text,columns=["Raw Data"])
```

```
# Printing a sample to ensure correct data format
df.head()
```

[5]:

	Raw Data
0	<!doctype html><html lang="en"><head><link hre…
1	<!doctype html><html lang="en"><head><link hre…
2	<!doctype html><html lang="en"><head><link hre…
3	<!doctype html><html lang="en"><head><link hre…
4	<!doctype html><html lang="en"><head><link hre…

[6]:
```
# Dataframe has been created successfully and can now be saved in a CSV file
df.to_csv(r"data\raw.csv")
```

4 References

1. HTML Error Codes: Used for refrencing response codes and their meaning.
2. Request Module of Python: Used for creating custom headers during scraping