Digital Technologies and Value Creation

Dr. Philippe Blaettchen
Bayes Business School (formerly Cass)

Overview – subject to change

Overarching theme	Week	
Introduction	1	Introduction to analytics applications and coding basics
Gathering data	2	Scraping web data
Gathering data / descriptive analytics	3	More on scraping, data pre-processing and descriptive analytics
Gathering data / descriptive analytics	4	Descriptives in marketing analytics, and using social media APIs
Descriptive analytics	5	Descriptives in people analytics
NO LECTURE	6	NO LECTURE
Predictive analytics	7	Retaining employees and customers
Predictive analytics	8	Valuing a (social media) customer base
Predictive analytics	9	Segmenting customers and positioning products
Prescriptive analytics	10	Optimizing products and organizations
Prescriptive analytics	11	A/B-testing in practice

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

Learning objectives of today

Goals: Learn to use some of the key tools used for web scraping in Python

- Which tools should be used for what type of project?
- How tor read information from an HTML?
- How to deal with JavaScript-based dynamic sites?

How will we do this?

- We start with BeuatifulSoup and Requests, then see how we can add Selenium into the mix
- We then move to Scrapy, usually used for bigger projects and building crawlers. We add Splash into the mix next week
- We will use the tutorial and a part of the lecture next week to have enough time with the different tools

An overview of scraping

Scraping, crawling, and spiders (don’t worry, they don’t have 8 legs)

(Web) scraping:

- Take any publicly available data and import it (or statistics based on it) to your hardware
- Web scraping (vs. data scraping): data comes from the web

- Taking and downloading data
- Can be adjusted manually

Web crawling:

- Like web scraping, but we keep going from page to page
- → Finding and following links

- Process that goes through targets and finds new ones
- Needs a “bot” for scraping, due to volume → web crawler, crawler, spider, or spiderbot

BAYES
BUSINESS SCHOOL
CITY, UNIVERSITY OF LONDON

An overview of the technologies we will use

- Get specific elements from web pages
- Simplifies web scraping
- Additional tools needed to retrieve pages (e.g., Requests)

- Automates browsers
- Originally a testing suite for running applications on different browsers

Scrapy

- Create spiders (asynchronous, fast)
- Inbuilt functionality to deal crawling/scraping difficulties
- Easily adjust use of server resources

- “Render” JavaScript used for dynamic websites

- Simple to get started
- One-off scraping tasks

- Navigate complex page behavior
- One-off, speed not important

- Complex scraping tasks
- Build once, run frequently

- Deal with JavaScript in Scrapy projects

The legal parts

- The legal situation of scraping can be complicated
- What is never allowed is to republish copyrighted information (unless there is some license granting you the right to, but this usually involves citation)
- It is generally forbidden to violate terms of service
    - When you create an account, you explicitly agree to the terms of service of a site
    - Even without an account, you implicitly agree
    - And even when not illegal, providers may block your account or IP!
- A good first place to check: robots.txt
- Generally, be reasonable: even if there is no problem scraping a site in principle, remember that server resources are finite!

Scraping for People Analytics with BeautifulSoup

Requests and BeautifulSoup

- Each time the user clicks on a link (in HTML: an anchor tag with an href= value), the browser makes a connection to the web server and issues a "GET" request
  - This signals that the browser wants to GET the document at the specified URL
- The server returns the (HTML) document to the browser, which formats and displays the document to the user
- We can create a GET request using the requests package (without having to worry too much about the details), and return the relevant HTML document
- BeautifulSoup allows us to parse through the returned HTML more easily

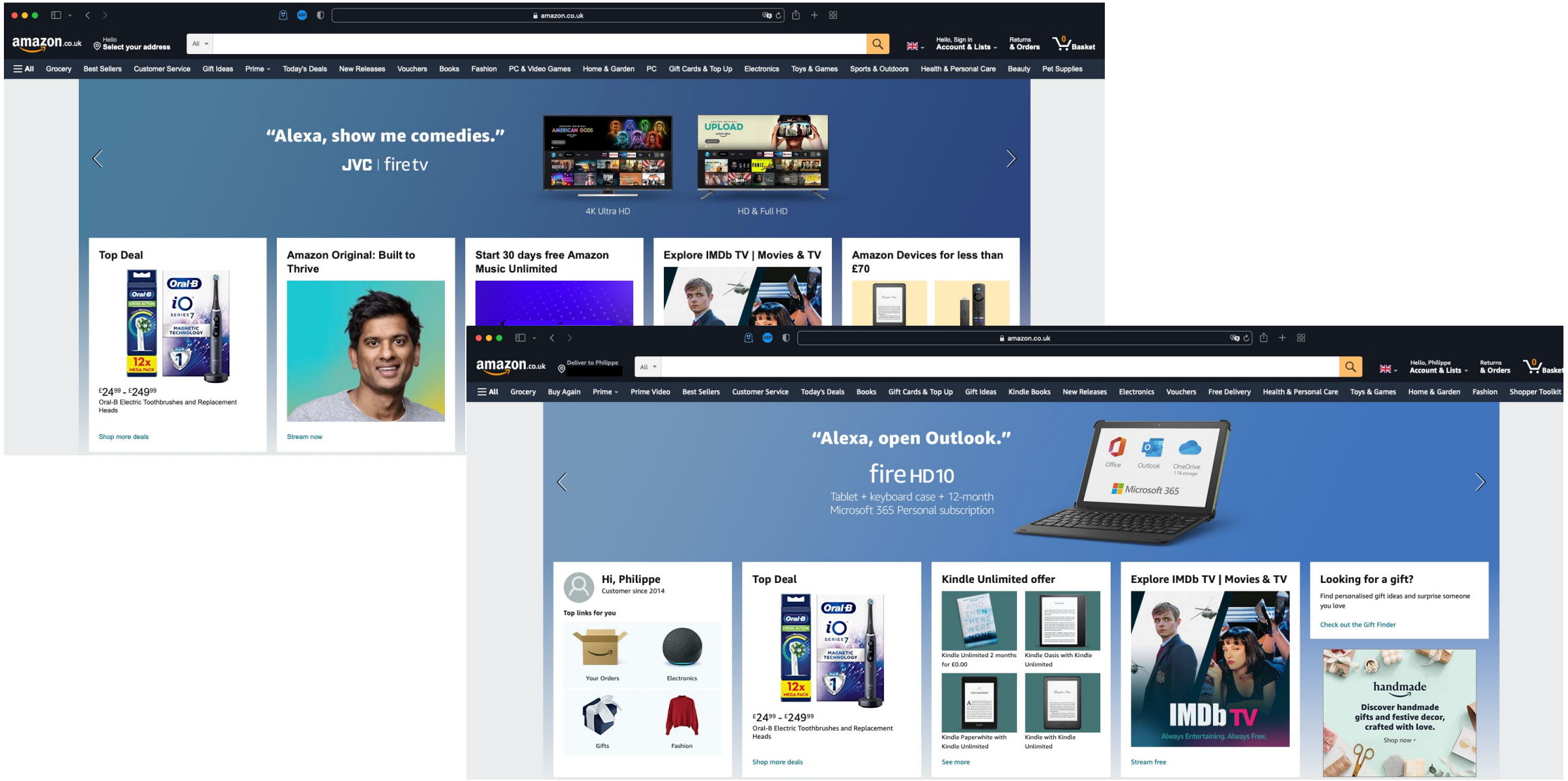
BeautifulSoup in pratice

Dynamic content and Selenium

Dynamic content

- Recall that XML documents can be treated as a tree
- The same is true for HTML → the Document Object Model (DOM) does just that
- Many web pages require your browser to run JavaScript to query the DOM and also modify it
  - E.g., you haven't accepted cookies yet and the content won't be loaded until you do
  - E.g., you tick something or click a button and the page changes – the web server will not have a completely specified HTML document for each combination of clicks!
  - E.g., an online shop displays different items to you, depending on your past behavior – no shop stores an HTML document for each of its users!

Dynamic content – an example



The problem with scraping dynamic web sites

- When making a request, the return value doesn't specify the full HTML document – what we get back in Python will look different than the HTML we inspect
- To extract information based on navigating an HTML, we first need to render the website (as if we are actually using a browser to look at it)
- One way to do this is to simulate a browser

Continuing our example with Selenium

Scraping with Scrapy

Before we start: an intro to object-oriented programming

- Programming around objects rather than procedures
- Instead of thinking about “what happens in which sequence”, we think about “what does X do, and how should Y react to it”
- Allows us to simplify complex programming problems and more easily represent the real world in our models
- Allows us to make modular programs

A program to simulate a chess game

Procedural programming:

1. White makes a move
2. Black makes a move (if white has done X, then do Y)
3. White makes a move (if black has done X, then do Y)
4. …

Object-oriented programming:

- There is one game “object” with a 64-dimensional state
- There are two player “objects” that belong to the game “object”. A player “object” either uses the black or the white pieces
- The players can perform certain moves (“functions”) whenever it is their turn

Classes and objects

- An object represents an entity in our program – with its specific data, and functions/methods that it can perform (e.g., the specific player moving the black pieces)
- A class is a blueprint for creating an object – it defines the “abilities” of our objects (e.g., any player more generally), as well as the way in which objects are created

Let’s see how to build a class (and objects) in Python

Large projects with Scrapy

- A tool to create spiders
- Works in an asynchronous way (multiple requests in parallel!)
  → extremely fast!
- Contains many procedures for post-processing data and dealing with the messiness that is the internet
- Handles errors “with ease”
- Handles log-ins when necessary
- Can fine-tune the rate with which you access server resources to avoid being banned

→ Dealing with many websites or subpages
Complex and large scraping tasks where runtime can become an issue

Scrapy: the limits of Jupyter Notebooks

Reading HTMLs using XPath and CSS

XPath

- A query language for selecting nodes from an XML (or HTML) document
- XPath engines differ, which can lead to inconsistent results – but this is only really relevant for Internet Explorer users
- More versatile and powerful than CSS
  - E.g., can also go up in the XML tree (../)

CSS Selectors

- Cascading Style Sheets (CSS) is a language that describes how an HTML should look in the browser
- CSS Selectors are the basis for defining which rules to apply (e.g., p, h1, span)
- Runs faster than XPath – but marginally so with current hardware

In many cases, the difference is marginal:

- response.xpath('//div[contains(@class, "button")/text()')
- response.css('div[class~=button]::text')

Exploring websites with the scrapy shell

Until next week!