Web Mining and Recommender Systems

Tools: Collecting and parsing Web data with urllib and BeautifulSoup

Learning Goals

- Show how to crawl and parse web data

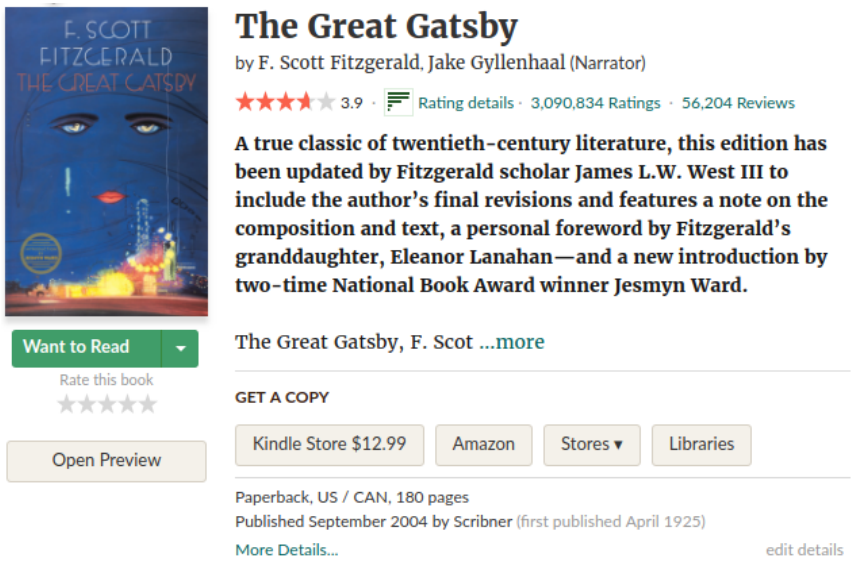
Collecting our own datasets

Suppose that we wanted to collect data from a website, but didn't yet have CSV or JSON formatted data

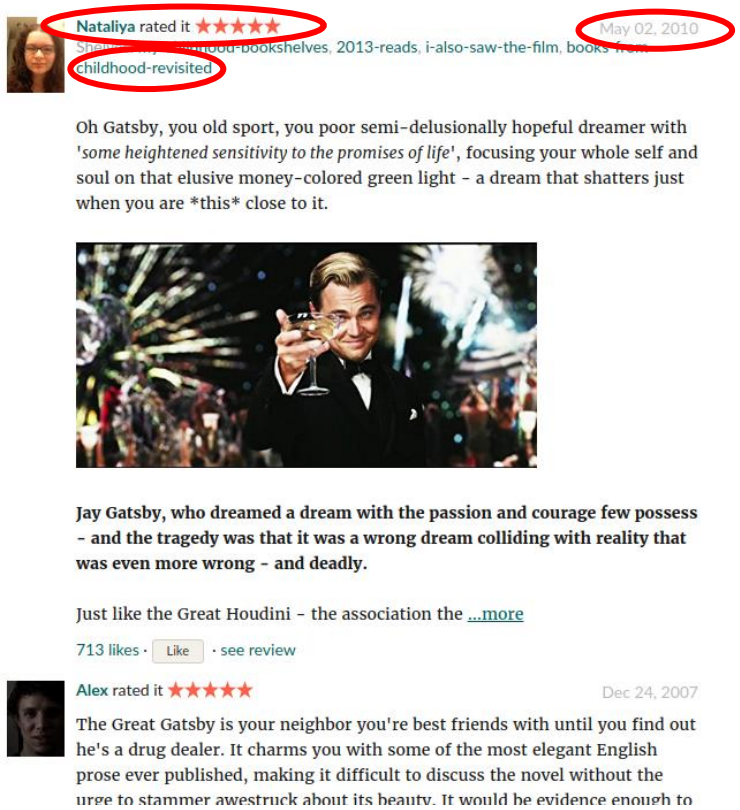
- How could we collect new datasets in machine-readable format?
- What Python libraries could we use to collect data from webpages?
- Once we'd collected (e.g.) raw html data, how could we extract structured information from it?

Collecting our own datasets

E.g. suppose we wanted to collect reviews of "The Great Gatsby" from goodreads.com:
(https://www.goodreads.com/book/show/4671.The_Great_Gatsby)



Collecting our own datasets



How could we extract fields including

- The *ID* of the user,
- The *date* of the review
- The *star rating*
- The *text* of the review itself?
- The *shelves* the book belongs to

Code: urllib

Our first step is to extract the html code of the webpage into a python string. This can be done using **urllib:**

```
In [1]: from urllib.request import urlopen

In [2]: f = urlopen("https://www.goodreads.com/book/show/4671.The_Great_Gatsby")

In [3]: html = str(f.read())

In [4]: html

Out[4]: 'b\'<!DOCTYPE html>\\n<html class="desktop\\n">\\n\\n\\n<head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# good_reads: http://ogp.me/ns/fb/good_reads#">\\n  <title>\\nThe Great Gatsby by F. Scott Fitzgerald\\n</title>\\n\\n\\n    <script type="text/javascript"> var ue_t0=window.ue_t0||+new Date();\\n </script>\\n  <script type="text/javascript">\\n    (function(e){var c=e;var a=c.ue||{};a.main_scope="mainscopecsm";a.q=[];a.t0=c.ue_t0||+new Date();a.d=g;function g(h){return +new Date()-(h?0:a.t0)}function d(h){return function(){a.q.push({n:h,a:arguments,t:a.d()})}}function b(m,l,h,j,i){var k={m:m,f:l,l:h,c:""+j,err:i,fromOnError:1,args:arguments};c.ueLogError(k);return false}b.skipTrace=1;e.onerror=b;function f(){c.uex("ld")}if(e.addEventListener){e.addEventListener("load",f,false)}else{if(e.attachEvent){e.attachEvent("onload",f)}}a.tag=d("tag");a.log=d("log");a.reset=d("rst");c.ue_csm=c;c.ue=a;c.ueLogError=d("err");c.ues=d("ues");c.uet=d("uet");c.uex=d("uex");c.uet("ue")})(window);(function(e,d){var a=e.ue||{};function c(g){if(!g){return}var f=d.head||d.getElementsByTagName("head")[0]||d.documentElement,h=d.createElement("script");h.async="async";h.src=g;f.insertBefore(h,f.firstChild)}function b(){var k=e.ue_cdn||"z-ecx.images-amazon.com",g=e.ue_cdns||"images-na.ssl-images-amazon.com",j="/images/G/01/csminstrumentation/",h=e.ue_file|
```

Note: acts like a file object once opened

Note: url of "The Great Gatsby" reviews

Reading the html data

This isn't very nice to look at, it can be easier to read in a browser or a text editor (which preserves formatting):

```
<!DOCTYPE html>
<html class="desktop
">


<head prefix="og: http://ogp.me/ns# fb: http://ogp.me/ns/fb# good_reads: http://ogp.me/ns/fb/good_reads#">
  <title>
The Great Gatsby by F. Scott Fitzgerald
</title>


    <script type="text/javascript"> var ue_t0=window.ue_t0||+new Date();
 </script>
  <script type="text/javascript">
    (function(e){var c=e;var a=c.ue||{};a.main_scope="mainscopecsm";a.q=[];a.t0=c.ue_t0||+new Date();a.d=g;fu
{m:m,f:l,l:h,c:""+j,err:i,fromOnError:1,args:arguments};c.ueLogError(k);return false}b.skipTrace=1;e.onerror=
{e.attachEvent("onload",f)}}a.tag=d("tag");a.log=d("log");a.reset=d("rst");c.ue_csm=c;c.ue=a;c.ueLogError=d("
f=d.head||d.getElementsByTagName("head")[0]||d.documentElement,h=d.createElement("script");h.async="async";h.
amazon.com",j="/images/G/01/csminstrumentation/",h=e.ue_file||"ue-full-11e51f253e8ad9d145f4ed644b40f692._V1_.
1:0}i=f?"https://":"http://";i+=f?g:k;i+=j;i+=h;c(i)}if(!e.ue_inline){if(a.loadUEFull){a.loadUEFull()}else{b(
```

Reading the html data

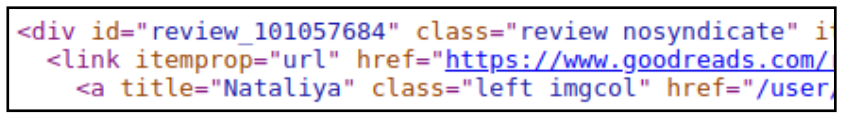
To extract review data, we'll need to look for the part of the html code which contains the reviews:

```
1227 <div id="bookReviews">
1228
1229
1230
1231
1232
1233   <div class="friendReviews elementListBrown" >
1234     <div class="section firstReview">
1235
1236
1237 <div id="review_101057684" class="review nosyndicate" itemprop="reviews" itemscope itemtype="http://schema.org/Review">
1238   <link itemprop="url" href="https://www.goodreads.com/review/show/101057684" />
1239     <a title="Nataliya" class="left imgcol" href="/user/show/3672777-nataliya"><img alt="Nataliya" src="https://images.gr-asse
1240
1241   <div class="left bodycol">
1242     <div class="reviewHeader uitext stacked">
1243         <a class="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>
1244
1245       <span itemprop="author" itemscope itemtype="http://schema.org/Person">
1246         <a title="Nataliya" class="user" itemprop="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>
1247       </span>
1248
```

Here it is (over 1000 lines into the page!)

Reading the html data

To extract review data, we'll need to look for the part of the html code which contains the reviews:



- Note that each individual review starts with a block containing the text "<div id="review_..."
- We can collect all reviews by looking for instances of this text

Code: string.split()

To split the page into individual reviews, we can use the string.split() operator. Recall that we saw this earlier when reading csv files:

```
In [5]: reviews = html.split('<div id="review_')[1:]

In [6]: len(reviews)
Out[6]: 30

In [7]: reviews[0]
Out[7]: '101057684" class="review hosyndicate" itemprop="reviews" itemscope itemtype="http://schema.org/Review">\\n  <link i
temprop="url" href="https://www.goodreads.com/review/show/101057684" />\\n    <a title="Nataliya" class="left imgco
l" href="/user/show/3672777-nataliya"><img alt="Nataliya" src="https://images.gr-assets.com/users/1395089173p2/36727
77.jpg" /></a>\\n\\n  <div class="left bodycol">\\n    <div class="reviewHeader uitext stacked">\\n        <a class
="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>\\n\\n      <span
itemprop="author" itemscope itemtype="http://schema.org/Person">\\n        <a title="Nataliya" class="user" itemprop
="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>\\n      </span>\\n\\n        rated it\\n
   <span class=" staticStars" title="it was amazing"><span size="15x15" class="staticStar p10">it was amazing</span>
<span size="15x15" class="staticStar p10"></span><span size="15x15" class="staticStar p10"></span><span size="15x15"
class="staticStar p10"></span><span size="15x15" class="staticStar p10"></span></span>\\n\\n\\n\\n        \\n\\n\\n
```

Note: the page contains 30 reviews total

Note: Ignore the first block, which contains everything *before* the first review

Code: parsing the review contents

Next we have to write a method to parse individual reviews (i.e., given the text of one review, extract formatted fields into a dictionary)

In [8]:
```
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
        for s in shelfBlock.split('shelf=')[1:]:
            shelves.append(s.split('"')[0])
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">')[1].split('</div')[0]
    d['reviewBlock'] = reviewBlock
    return d
```

Code: parsing the review contents

Let's look at it line-by-line:

```
In [8]: def parseReview(review):
            d = {}
```

- We start by building an empty dictionary
- We'll use this to build a *structured* version of the review

Code: parsing the review contents

Let's look at it line-by-line:

Note: Two splits: everything *after* the first quote, and *before* the second quote

- The next line is more complex:

```
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
```

- We made this line by noticing that the stars appear in the html inside a span with class " staticStars":

```
1248
1249          rated it
1250          <span class=" staticStars" title="it was amazing"><span size="15x15" class="staticStar p10">it was amazing<
p10"></span><span size="15x15" class="staticStar p10"></span></span>
1251
```

- Our "split" command then extracts everything inside the "title" quotes

Code: parsing the review contents

Let's look at it line-by-line:

- The following two lines operate in the same way:

Note: Everything between the two brackets of this "<a" element

```
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
```

- Again we did this by noting that the "date" and "user" fields appear inside certain html elements:

```
1240
1241  <div class="left bodycol">
1242    <div class="reviewHeader uitext stacked">
1243        <a class="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>
1244
1245      <span itemprop="author" itemscope itemtype="http://schema.org/Person">
1246        <a title="Nataliya" class="user" itemprop="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>
1247      </span>
1248
```

Code: parsing the review contents

Let's look at it line-by-line:

- Next we extract the "shelves" the book belongs to
- This follows the same idea, but in a "for" loop since there can be many shelves per book:

```
shelves = []
try:
    shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
    for s in shelfBlock.split('shelf=')[1:]:
        shelves.append(s.split('"')[0])
    d['shelves'] = shelves
except Exception as e:
    pass
```

Note: Everything inside a particular <div

- Here we use a try/except block since this text will be missing for users who didn't add the book to any shelves

Code: parsing the review contents

Next let's extract the review contents:

In [8]:
```
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class=" staticStars" title="')[1].split('"')[0]
    d['date'] = review.split('<a class="reviewDate')[1].split('>')[1].split('<')[0]
    d['user'] = review.split('<a title="')[1].split('"')[0]
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div')[0]
        for s in shelfBlock.split('shelf=')[1:]:
            shelves.append(s.split('"')[0])
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">')[1].split('</div')[0]
    d['reviewBlock'] = reviewBlock
    return d
```

Code: parsing the review contents

Now let's look at the results:

```
In [9]: reviewDict = [parseReview(r) for r in reviews]

In [10]: reviewDict[0]

Out[10]: {'date': 'May 02, 2010',
 'reviewBlock': '\\n            <span id="reviewTextContainer101057684" class="readable"\\n              >\\n
      \\n<span id="freeTextContainer7513160808421149349"><br>Oh Gatsby, you old sport, you poor semi-delusionally h
opeful dreamer with \\\'<                                                    i>\\\', focusing your whole self and
soul on that elusive mone                                                    you are *this* close to it. <br><br
><img src="https://i.gr-a                                                    dimages/1380334543i/693798._SX540_.j
pg" width="400" class="gr                                                    eam with the passion and courage few
possess - and the tragedy                                                    hat was even more wrong - and deadl
y.</b> <br><br>Just like                                                     h id="freeText7513160808421149349" s
tyle="display:none"><br>O                                                    eful dreamer with \\\'<i>some height
ened sensitivity to the p                                                    oul on that elusive money-colored gr
een light - a dream that                                                     <img src="https://i.gr-assets.com/im
ages/S/compressed.photo.g                                                    g" width="400" class="gr-hostedUserI
mg"><br><br><b>Jay Gatsby                                                    possess - and the tragedy was that i
t was a wrong dream colli                                                    .</b> <br><br>Just like the Great Ho
udini - the association t                                                    ed in illusions and escape. Except e
ven the power of most cou                                                    be the world, our past, and ourselve
s, giving rise to one of the most famous closing lines of a novel.<blockquote>\\n  <i>\\\'Gatsby believed in the gre
en light, the orgastic future that year by year recedes before us. It eluded us then, but that\\xe2\\x80\\x99s no ma
```

- Looks okay, but the review block itself still contains embedded html (e.g. images etc.)
- How can we extract just the text part of the review?

The BeautifulSoup library

Extracting the text contents from the html review block would be extremely difficult, as we'd essentially have to write a html parser to capture all of the edge cases

Instead, we can use an existing library to parse the html contents: **BeautifulSoup**

Code: parsing with BeautifulSoup

BeautifulSoup will build an element tree from the html passed to it. For the moment, we'll just use it to extract the text from a html block

```
In [11]: from bs4 import BeautifulSoup

In [12]: soup = BeautifulSoup(reviewDict[0]['reviewBlock'])

In [13]: soup.text

Out[13]: "\\n              \\n            \\nOh Gatsby, you old sport, you poor semi-delusionally hopeful dreamer with \\'som
         e heightened sensitivity to the promises of life\\', focusing your whole self and soul on that elusive money-colored
         green light - a dream that shatters just when you are *this* close to it. Jay Gatsby, who dreamed a dream with the p
         assion and courage few possess - and the tragedy was that it was a wrong dream colliding with reality that was even
         more wrong - and deadly. Just like the Great Houdini - the association the\\n  Oh Gatsby, you old sport, you poor se
         mi-delusionally hopeful dreamer with \\'some heightened sensitivity to the promises of life\\', focusing your whole
         self and soul on that elusive money-colored green light - a dream that shatters just when you are *this* close to i
         t. Jay Gatsby, who dreamed a dream with the passion and courage few possess - and the tragedy was that it was a wron
         g dream colliding with reality that was even more wrong - and deadly. Just like the Great Houdini - the association
         the title of this book so easily invokes - you specialized in illusions and escape. Except even the power of most co
         urageous dreamers can be quite helpless to allow us escape the world, our past, and ourselves, giving rise to one of
```

The BeautifulSoup library

In principle we could have used BeautifulSoup to extract *all* of the elements from the webpage

However, for simple page structures, navigating the html elements is not (necessarily) easier than using primitive string operations

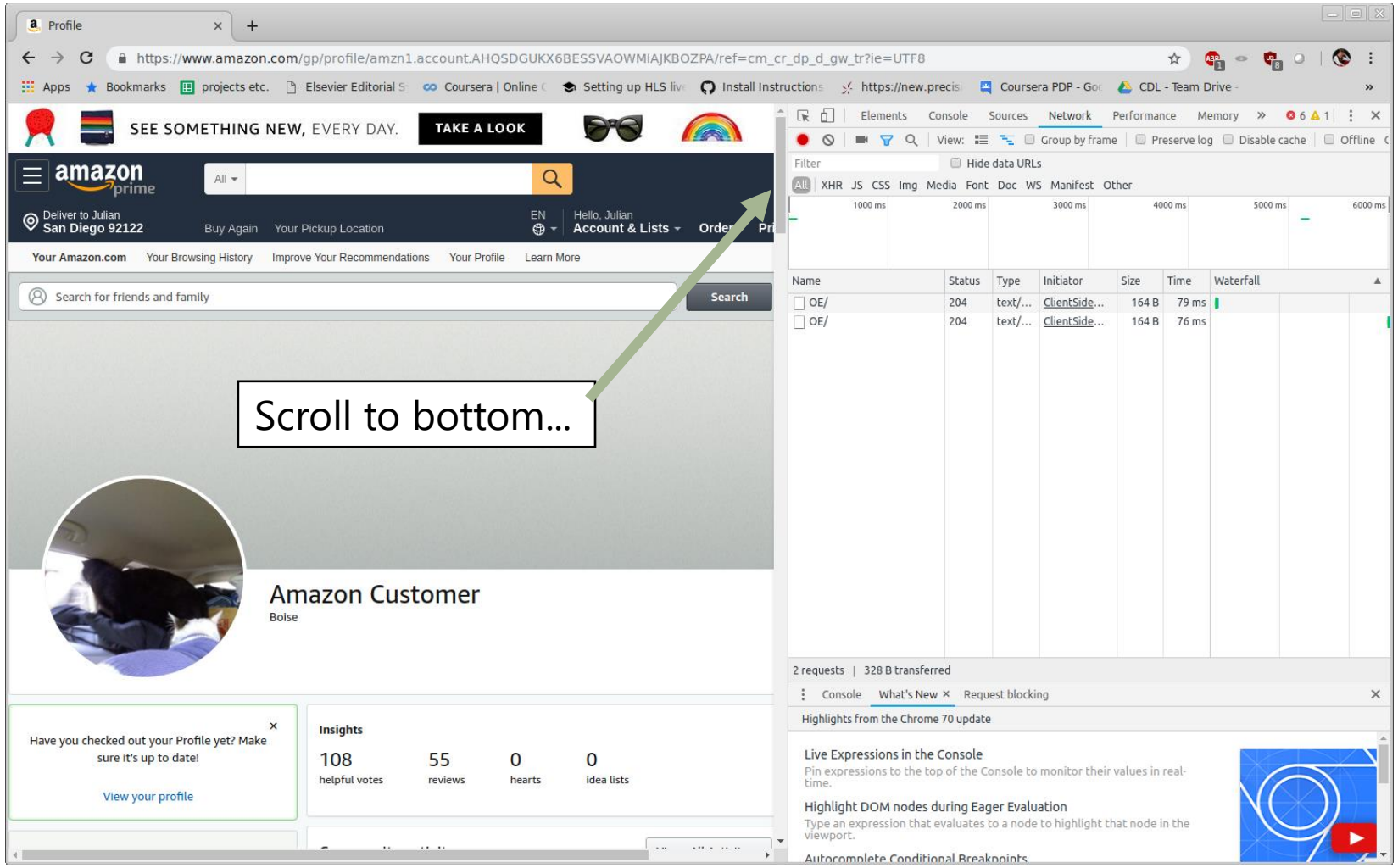
Advanced concepts…

1. What if we have a webpage that loads content **dynamically?**

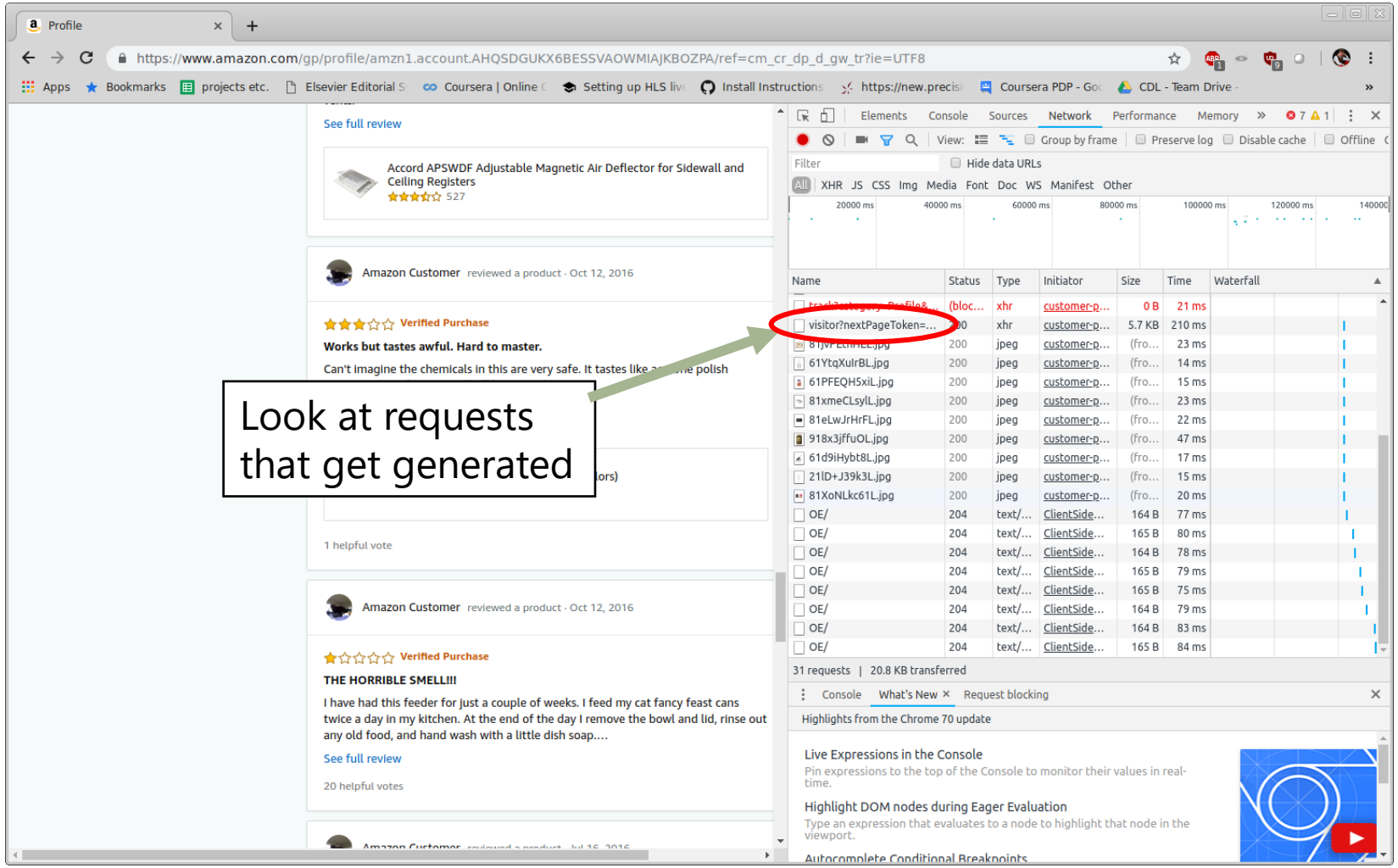
(e.g. https://www.amazon.com/gp/profile/amzn1.account.AHQSDGUKX6BESSVAOWMIAJKBOZPA/ref=cm_cr_dp_d_gw_tr?ie=UTF8)

- The page (probably) uses javascript to generate requests for new content
- By monitoring network traffic, perhaps we can view and reproduce those requests
- This can be done (e.g.) by using the Developer Tools in chrome

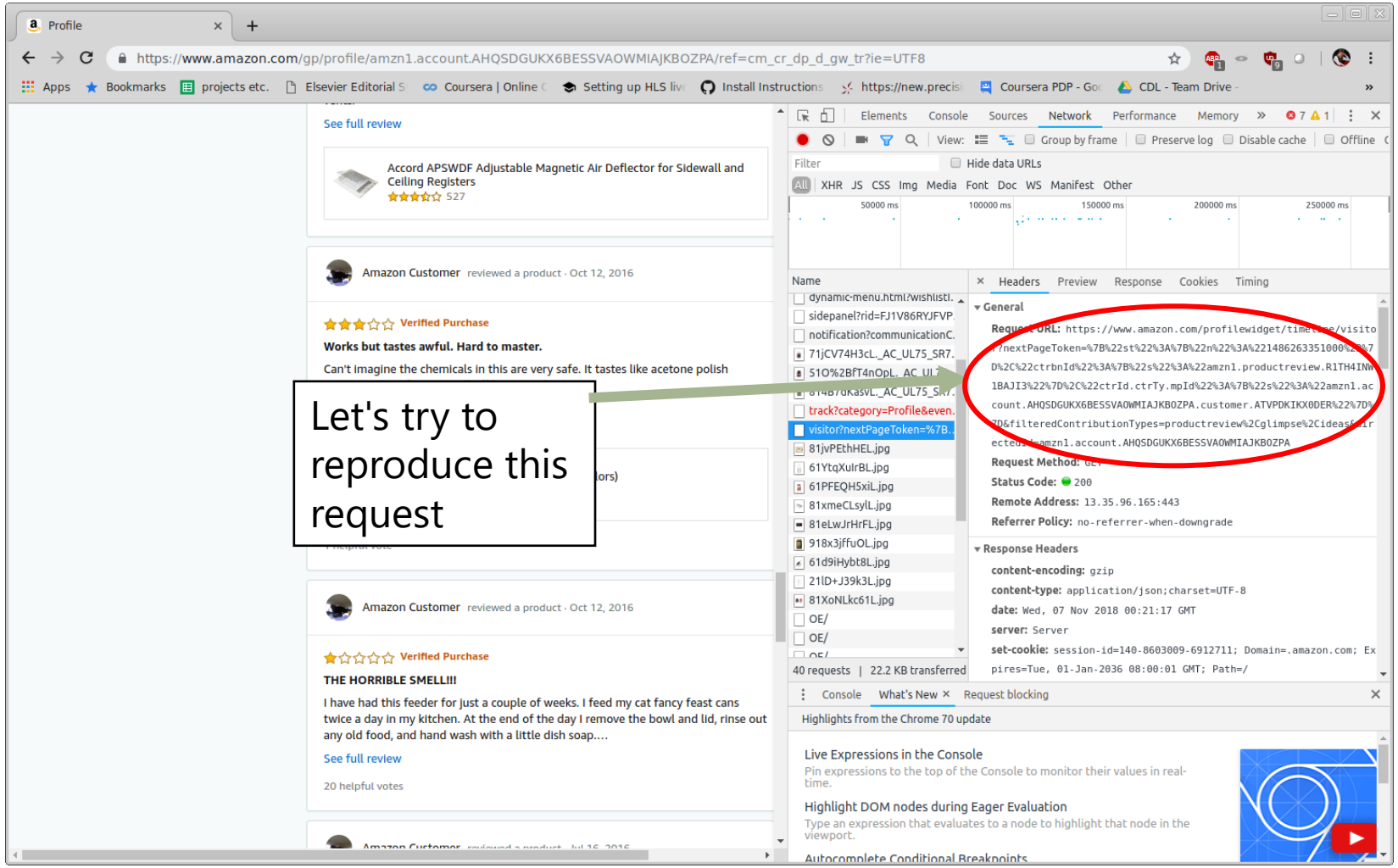
Pages that load dynamically…



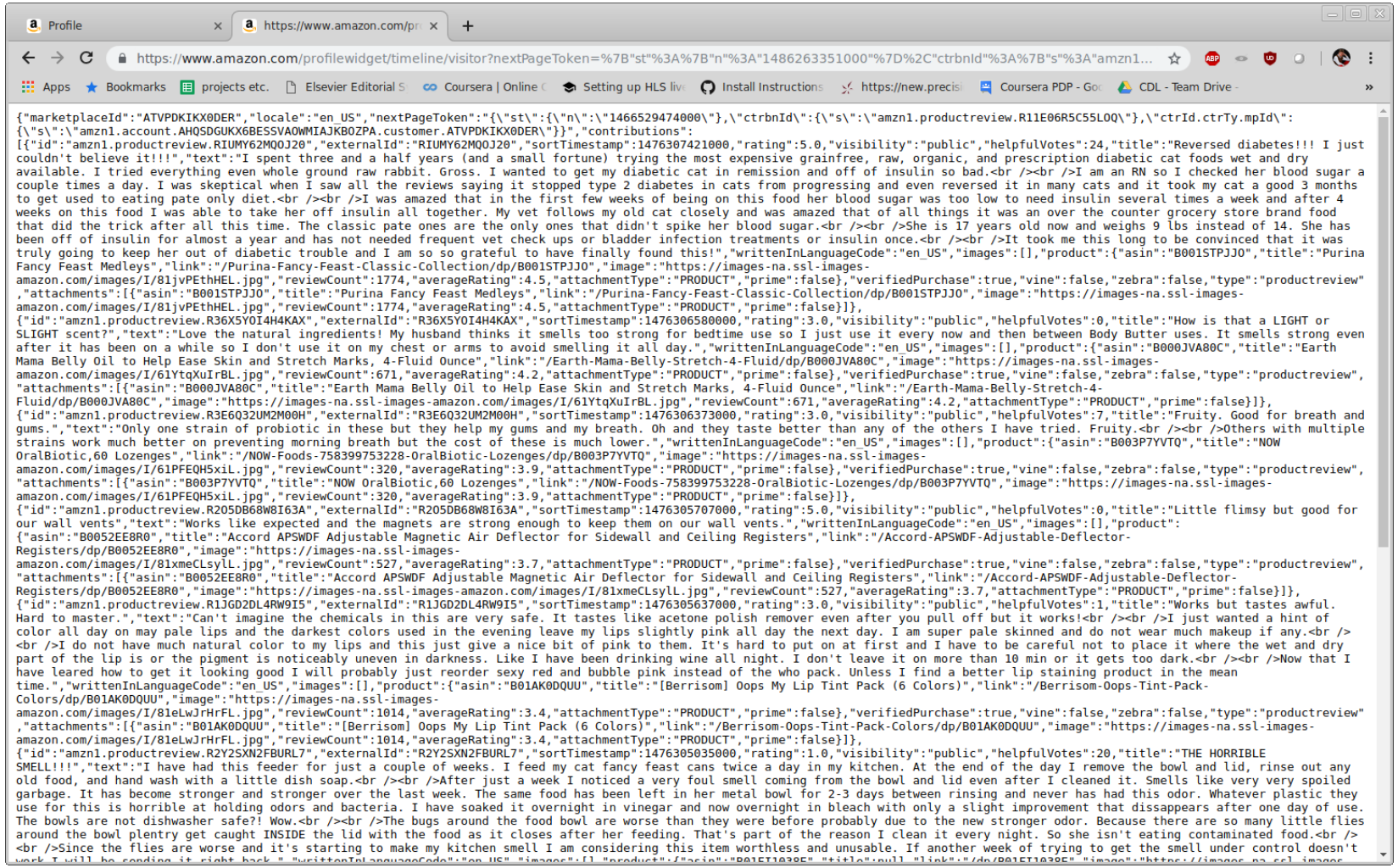
Pages that load dynamically…



Pages that load dynamically…



Pages that load dynamically…



Advanced concepts…

2. What if we require **passwords, captchas,** or **cookies?**

- You'll probably need to load an actual browser
- This can be done using a **headless browser,** i.e., a browser that is controlled via Python
  - I usually use **splinter** (https://splinter.readthedocs.io/en/latest/)

- Note that once you've entered the password, solved the captcha, or obtained the cookies, you can normally continue crawling using the *requests* library

Summary

- The **urllib** library can be used to request data from the web as if it is a file, whereas **BeautifulSoup** can be used to convert the data to structured objects
- Parsing can also be achieved using primitive string processing routines

- Make sure to check the page's **terms of service** first!

Learning Outcomes

- Introduced programmatic approaches to collect datasets from the web

Web Mining and Recommender Systems

Parsing time and date data

Learning Goals

- Show how to parse time and date data

Time and date data

Dealing with time and date data can be difficult as string-formatted data doesn't admit easy comparison or feature representation:

- Which date occurs first, 4/7/2003 or 3/8/2003?
- How many days between 4/5/2003 - 7/15/2018?
- e.g. how many hours between 2/6/2013 23:02:38 - 2/7/2013 08:32:35?

Time and date data

Most of the data we've seen so far include plain-text time data, that we need to carefully manipulate:


```
{'business_id': 'FYWN1wneV18bWNgQjJ2GNg', 'attributes':
{'BusinessAcceptsCreditCards': True, 'AcceptsInsurance':
True, 'ByAppointmentOnly': True}, 'longitude': -111.9785992,
'state': 'AZ', 'address': '4855 E Warner Rd, Ste B9',
'neighborhood': '', 'city': 'Ahwatukee', 'hours': {'Tuesday':
'7:30-17:00', 'Wednesday': '7:30-17:00', 'Thursday': '7:30-
17:00', 'Friday': '7:30-17:00', 'Monday': '7:30-17:00'},
'postal_code': '85044', 'review_count': 22, 'stars': 4.0,
'categories': ['Dentists', 'General Dentistry', 'Health &
Medical', 'Oral Surgeons', 'Cosmetic Dentists',
'Orthodontists'], 'is_open': 1, 'name': 'Dental by Design',
'latitude': 33.3306902}
```

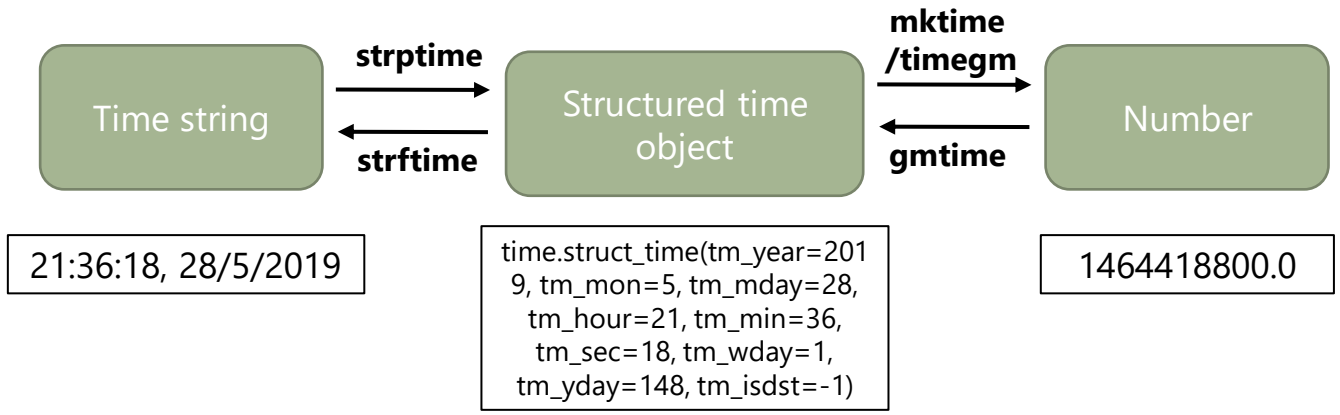
Time and date data

Here we'll cover a few functions:

- Time.strptime: convert a time **string** to a structured time **object**
- Time.strftime: convert a time **object** to a **string**
- Time.mktime / calendar.timegm: convert a time **object** to a **number**
- Time.gmtime: convert a **number** to a time **object**

Time and date data

Here we'll cover a few functions:



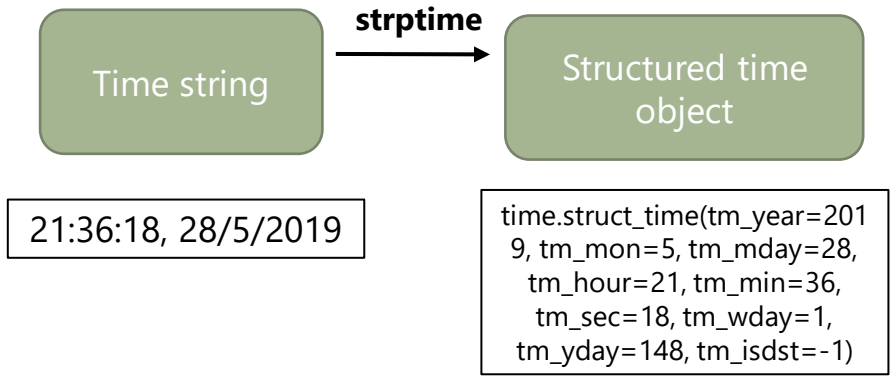
Concept: Unix time

Internally, time is often represented as a number, which allows for easy manipulation and arithmetic

- The value (Unix time) is the **number of seconds since Jan 1, 1970 in the UTC timezone**
- so I made this slide at 1532568962 = 2018-07-26 01:36:02 UTC (or 18:36:02 in my timezone)
- But real datasets generally have time as a "human readable" string
- Our goal here is to convert between these two formats

strptime

First, let's look at converting a string to a structured object (strptime)



Code: time.strptime()

```
In [1]: import time
        import calendar
```

String-formatted time data

```
In [2]: timeString = "2018-07-26 01:36:02"

In [3]: timeStruct = time.strptime(timeString, "%Y-%m-%d %H:%M:%S")

In [4]: timeStruct

Out[4]: time.struct_time(tm_year=2018, tm_mon=7, tm_mday=26, tm_hour=1, tm_min=36, tm_sec=2, tm_wday=3, tm_yday=207, tm_isds
        t=-1)

In [5]: timeStruct.tm_wday

Out[5]: 3
```

Note: this day is a Wednesday!

```
In [6]: help(time.strptime)

    Help on built-in function strptime in module time:

    strptime(...)
        strptime(string, format) -> struct_time

        Parse a string to a time tuple according to a format specification.
```

Note: different time formatting options in the help page

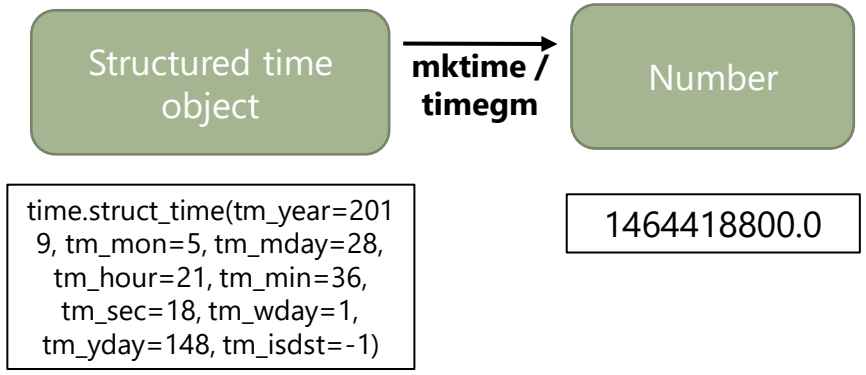
strptime

Strptime is convenient when we want to extract **features** from data

- E.g. does a date correspond to a weekday or a weekend?
- Converting month names or abbreviations (e.g. "Jan") to month numbers
- Dealing with mixed-format data by converting it to a common format
- But if we want to perform arithmetic on timestamps, converting to a number may be easier

time.mktime and calendar.timegm

For this we'll use mktime to convert our structured time object to a number:



Code: time.mktime() and calendar.timegm()

Structured time data from previous slide

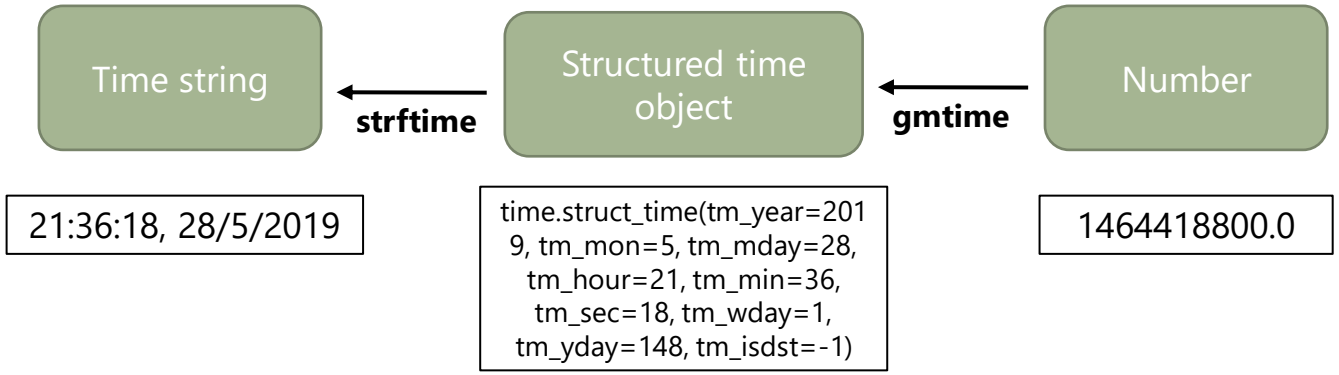
```
In [7]: t1 = calendar.timegm(timeStruct)

In [8]: t2 = time.mktime(timeStruct)

In [9]: t1, t2
Out[9]: (1532568962, 1532594162.0)

In [10]: t1 + 60*60*24*5
Out[10]: 1533000962
```

Five days later

- time.mktime() allows us to convert our structured time object to a number
- **NOTE:** mktime assumes the structure is a *local* time whereas timegm assumes the structure is a *UTC* time
- This allows for easy manipulation, arithmetic, and comparison (e.g. sorting) of time data

time.strftime and time.gmtime

Finally, both of these operations can be *reversed*, should we wish to format time data as a string or structure



Code: time.strftime() and time.gmtime()

```
In [11]: time.gmtime(t1 + 60*60*24*5)
```

Five days later than the previous time

```
Out[11]: time.struct_time(tm_year=2018, tm_mon=7, tm_mday=31, tm_hour=1, tm_min=36, tm_sec=2, tm_wday=1, tm_yday=212, tm_isds
         t=0)

In [12]: time.strftime("%Y-%m-%d %H:%M:%S", time.gmtime(t1 + 60*60*24*5))

Out[12]: '2018-07-31 01:36:02'
```

- These methods can be used to put adjusted times back into string format

Learning Outcomes

- Introduced various methods to parse time and date data

Web Mining and Recommender Systems

Introduction to Matplotlib

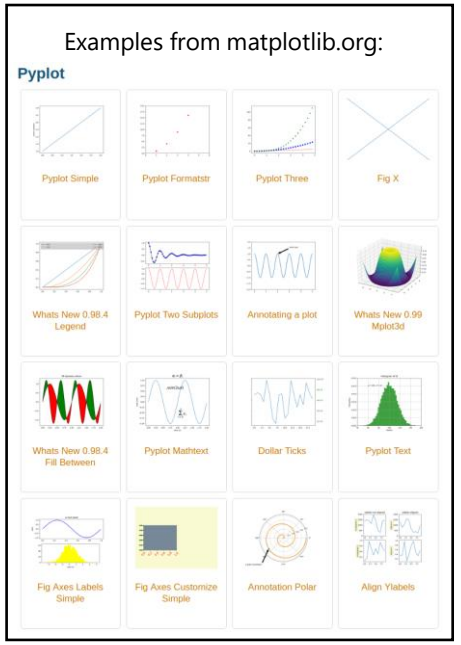
Learning Goals

- Introduce Matplotlib for plotting and visualizing data

Matplotlib

Matplotlib is a powerful library that can be used to generate both quick visualizations, as well as publication-quality graphics

- We'll introduce some of its most basic functionality (via pyplot), such as bar and line plots
- Examples (with code) of the types of plots that can be generated are available on https://matplotlib.org/



Code: generating some simple statistics

First, let's quickly compile some statistics from (e.g.) Yelp's review data

In [1]:
```
import json
import time
path = "datasets/yelp_data/review.json"
f = open(path, 'r')
```

In [2]:
```
dataset = []
for i in range(50000):
    dataset.append(json.loads(f.readline()))
```

In [3]:
```
datasetWithTimeValues = []
```

In [4]:
```
for d in dataset:
    d['date']
    d['timeStruct'] = time.strptime(d['date'], "%Y-%m-%d")
    d['timeInt'] = time.mktime(d['timeStruct'])
    datasetWithTimeValues.append(d)
```

Code: generating some simple statistics

```
In [5]: from collections import defaultdict

In [6]: weekRatings = defaultdict(list)

In [7]: for d in datasetWithTimeValues:
            day = d['timeStruct'].tm_wday
            weekRatings[day].append(d['stars'])

In [8]: weekAverages = {}

In [9]: for d in weekRatings:
            weekAverages[d] = sum(weekRatings[d]) * 1.0 / len(weekRatings[d])

In [10]: weekAverages

Out[10]: {0: 3.7094594594594597,
          1: 3.715375187253166,
          2: 3.750551876379691,
          3: 3.763665361751486,
          4: 3.7551891653172382,
          5: 3.7231843981953134,
          6: 3.7072147651006713}
```

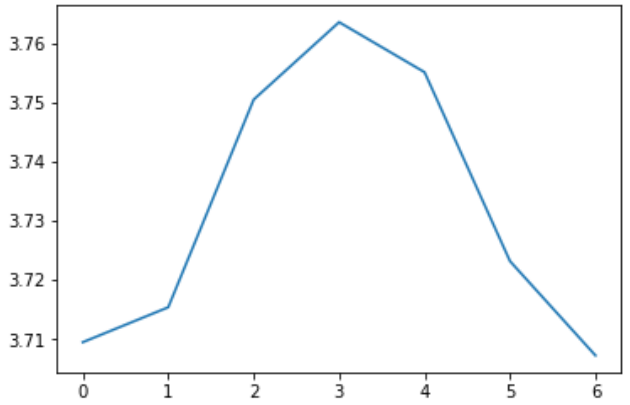
Average ratings per day of week

Code: drawing a simple plot

```
In [11]: X = list(weekAverages.keys())
```

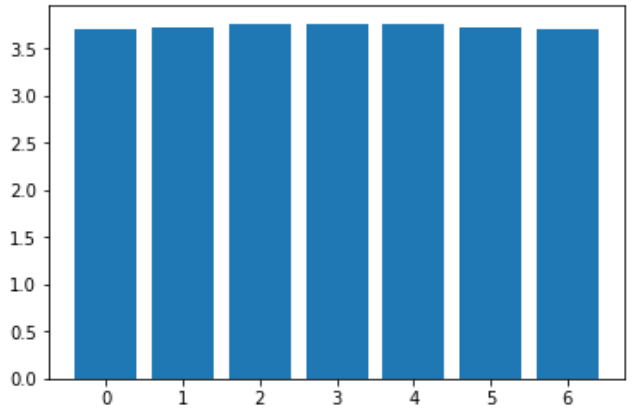
[0,1,2,3,4,5,6]

```
In [12]: Y = [weekAverages[x] for x in X]

In [13]: import matplotlib.pyplot as plt

In [14]: plt.plot(X, Y)
Out[14]: [<matplotlib.lines.Line2D at 0x7fc15a615a20>]
```



Code: bar plots

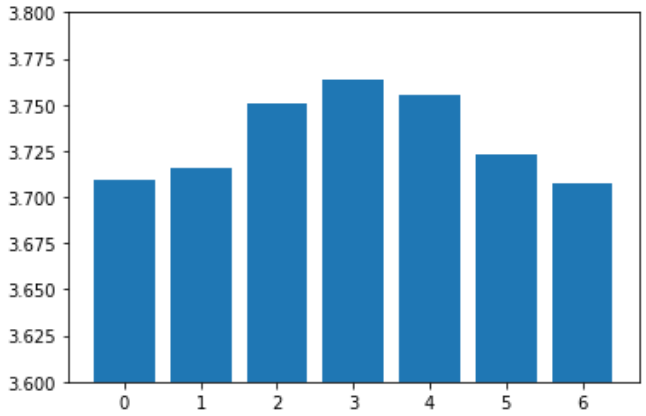
```
In [15]: plt.bar(X, Y)
Out[15]: <Container object of 7 artists>
```



- Looks right, but need to zoom in more to see the detail

Code: bar plots

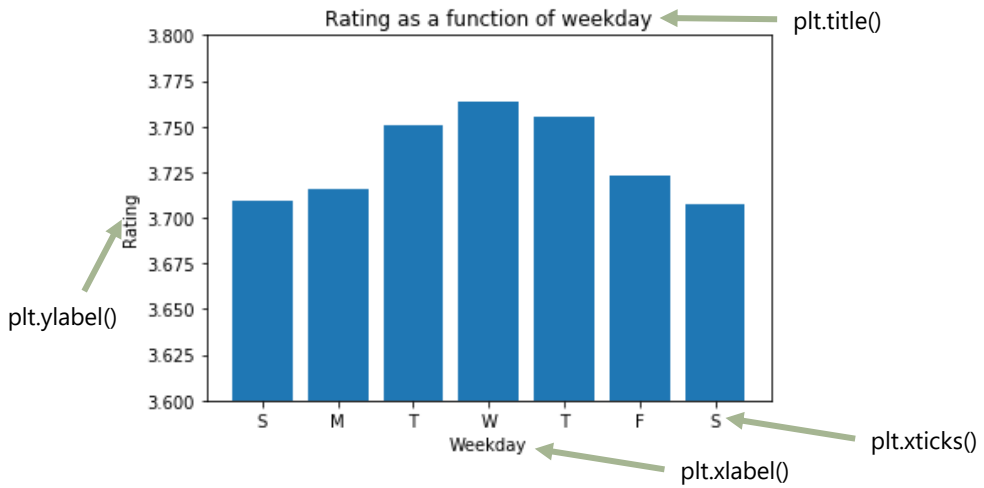
```
In [16]: plt.ylim(3.6, 3.8)
         plt.bar(X, Y)

Out[16]: <Container object of 7 artists>
```



- Next let's add some details

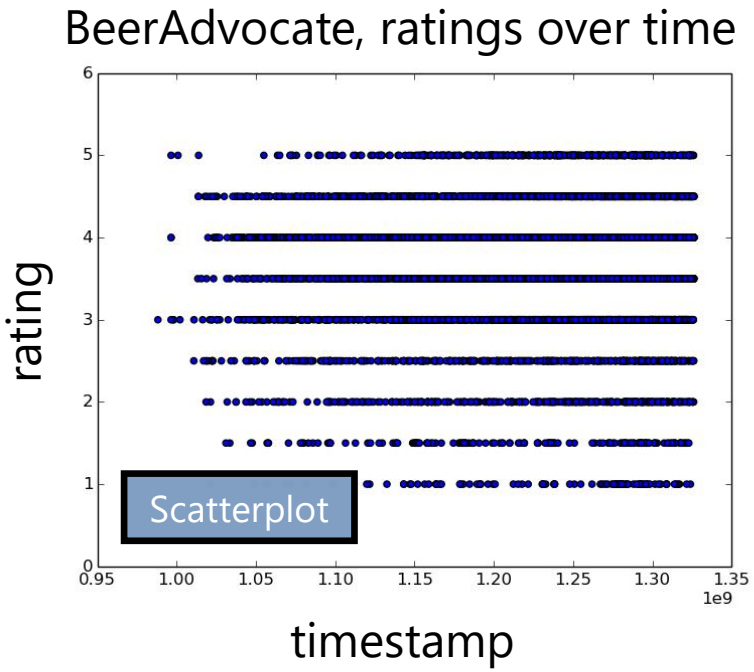
Code: bar plots

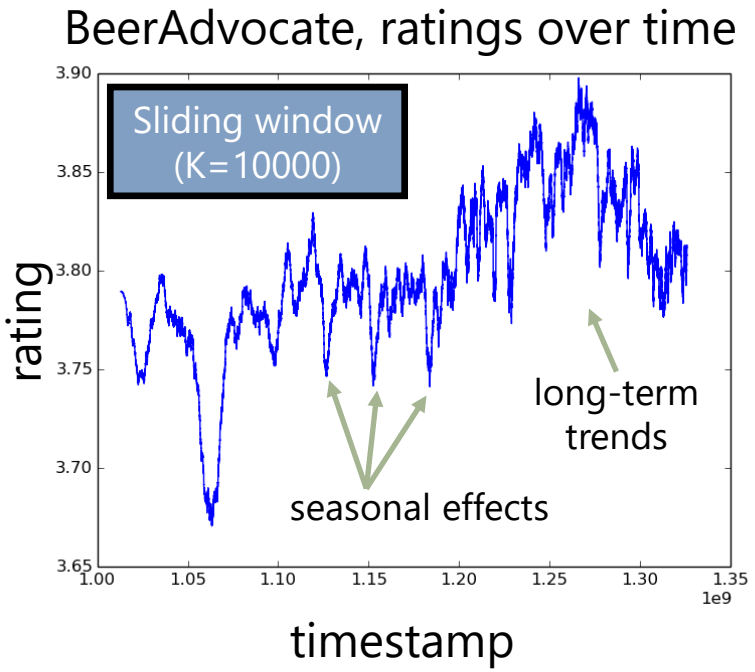
```
In [17]: plt.ylim(3.6, 3.8)
         plt.xlabel("Weekday")
         plt.ylabel("Rating")
         plt.xticks([0,1,2,3,4,5,6],['S', 'M', 'T', 'W', 'T', 'F', 'S'])
         plt.title("Rating as a function of weekday")
         plt.bar(X, Y)

Out[17]: <Container object of 7 artists>
```



Example: sliding windows

Also useful to plot data:





Code on course webpage

Learning Outcomes

- Briefly introduced Matplotlib

Web Mining and Recommender Systems

Gradient descent in tensorflow

Learning Goals

- Introduce Tensorflow
- Show how high-level libraries can help to automate gradient-based optimization

Tensorflow

Tensorflow, though often associated with deep learning, is really just a library that simplifies gradient descent and optimization problems, like those we've already implemented

Most critically, it computes gradients **symbolically,** so that you can just specify the objective, and Tensorflow can run gradient descent

Here we'll reimplement some of our previous gradient descent code in tensorflow

Code: Gradient Descent in Tensorflow

Reading the data is much the same as before (except that we first import the tensorflow library)

In [1]:
```
import tensorflow as tf
```

In [2]:
```
path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')
```

In [3]:
```
dataset = []
header = f.readline().strip().split(',')
for line in f:
    line = line.split(',')
    dataset.append(line)
```

In [4]:
```
header.index('pm2.5')
```

Out[4]: 5

In [5]:
```
dataset = [d for d in dataset if d[5] != 'NA']
```

Code: Gradient Descent in Tensorflow

Next we extract features from the data

In [6]:
```
def feature(datum):
    feat = [1, float(datum[7]), float(datum[8]), float(datum[10])] # Temperature, pressure, and wind speed
    return feat
```

In [7]:
```
X = [feature(d) for d in dataset]
y = [float(d[5]) for d in dataset]
```

In [8]:
```
y = tf.constant(y, shape=[len(y),1])
```

In [9]:
```
K = len(X[0])
```

Note that we convert *y* to a native tensorflow vector. In particular we convert it to **column** vector. We have to be careful about getting our matrix dimensions correct or we may (accidentally) apply the wrong matrix operations.

Code: Gradient Descent in Tensorflow

Next we write down the objective – note that we use native tensorflow operations to do so

```
In [10]: def MSE(X, y, theta):
           return tf.reduce_mean((tf.matmul(X,theta) - y)**2)
```

Next we setup the variables we want to optimize – note that we explicitly indicate that these are **variables** to be optimized (rather than constants)

```
In [11]: theta = tf.Variable(tf.constant([0.0]*K, shape=[K,1]))

In [12]: optimizer = tf.train.AdamOptimizer(0.01)

In [13]: objective = MSE(X,y,theta)
```

Initialized to zero

Stochastic gradient descent optimizer with learning rate of 0.01

Specify the objective we want to optimize – note that no computation is performed (yet) when we run this function

Code: Gradient Descent in Tensorflow

Boilerplate for initializing the optimizer...

```
In [14]: train = optimizer.minimize(objective)
```

We want to **minimize** the objective

```
In [15]: init = tf.global_variables_initializer()
```

```
In [16]: sess = tf.Session()
         sess.run(init)
```

Code: Gradient Descent in Tensorflow

Run 1,000 iterations of gradient descent:

```
In [17]: for iteration in range(1000):
           cvalues = sess.run([train, objective])
           print("objective = " + str(cvalues[1]))
```

```
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5103
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
```

Code: Gradient Descent in Tensorflow

Print out the results:

In [18]:
```
with sess.as_default():
  print(MSE(X, y, theta).eval())
  print(theta.eval())
```

```
7836.5093
[[ 0.23223479]
 [-0.89481604]
 [ 0.11925128]
 [-0.4959688 ]]
```

Summary

Note that in contrast to our "manual" implementation of gradient descent, many of the most difficult issues were taken care of for us:

- No need to compute the gradients – tensorflow does this for us!
- Easy to experiment with different models
- Very fast to run 1,000 iterations, especially with GPU acceleration!

Other libraries

Tensorflow is just one example of a library that can be used for this type of optimization. Alternatives include:

- Theano - http://deeplearning.net/software/theano/
- Keras - https://keras.io/
- Torch - http://torch.ch/
- Etc.

Each has fairly similar functionality, but some differences in interface

Learning Outcomes

- Introduced Tensorflow