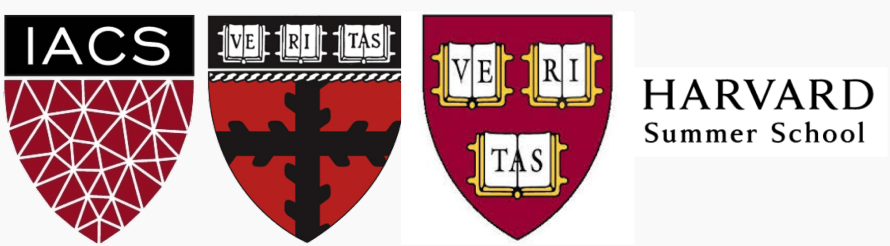
# Lecture #2: Getting our hands dirty: pandas and web scraping

## CS-S109A: Introduction to Data Science

Kevin Rader



# ANNOUNCEMENTS

- **HW0** is due tomorrow at 11:59pm. It will be checked for 'near completion' and 'reasonable effort'.
- **HW1** is posted and due Tuesday at 11:59pm.
- **HWs on Canvas. The rest of course content is on ~~Canvas~~ (and Ed)**
- **Ed is where the discussions (HW questions) and quizzes reside.**
  - Quizzes are under the 'Sway' tab.
  - If you can't connect to Ed, try logging out of Canvas, then back into Canvas.

# ANNOUNCEMENTS #2

- **OH Schedule (the rest of this week):**

  Wed: 3-4pm (Chris), 8-9:30pm (Rylan)
  Thurs: 7-8:30am (Tessa & Nabib), 9-10:30am (Sol)
  Fri: 2-3:30pm (Dominique)
  Sat: 5-6:30pm (Joel)
  Sun: 12-1:30pm (Arpit)

*Note: times may change after this week.

# From last time.

So far, we've learned:

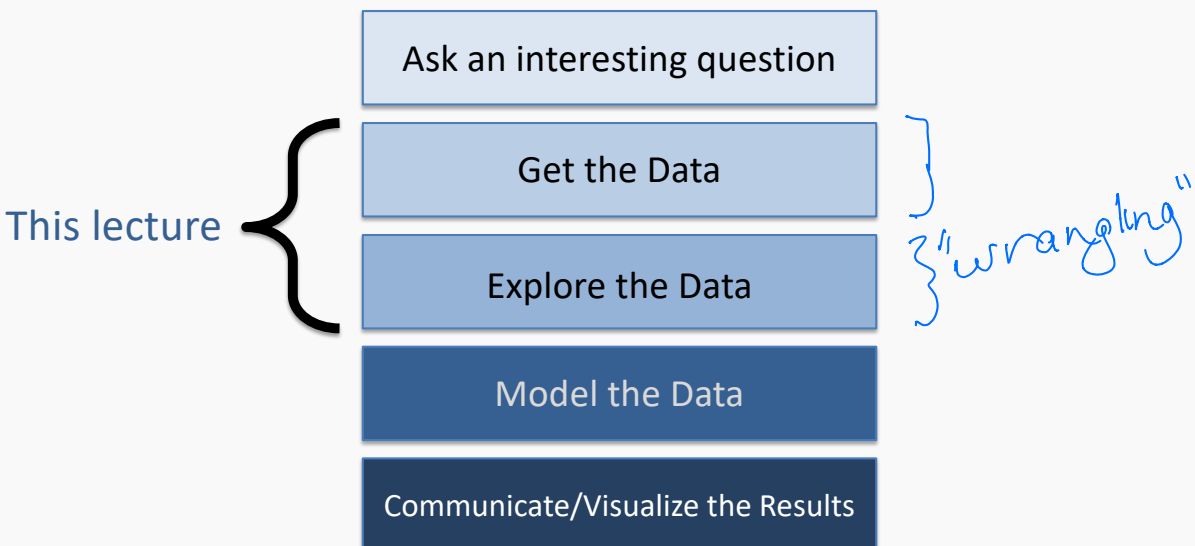
| | |
|---|---|
| Lecture 1 | What is Data Science? |
| | The Data Science Process |
| | Data: types, formats, issues, etc. |
| | Basic Summaries & Visualizations |
| Lecture 2 | How to quickly prepare data and scrape the web |
| Future lectures | How to model data |

python pandas

beautiful soup

# Background

The Data Science Process:



# Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas ← base python
  - With Pandas ← package
- Data concerns
- Collecting Data from the Web
  - Web Scraping with Beautiful Soup

# Exploratory Data Analysis (EDA)

## Why?

- EDA encompasses the "*explore* data" part of the data science process
- EDA is crucial but often overlooked:
  - If your data is bad, your results will be bad
  - Conversely, understanding your data well can help you create smart, appropriate models

# Exploratory Data Analysis (EDA)

## What?

1. Store data in data structure(s) that will be convenient for exploring/processing (Memory is fast. Storage is slow)
2. Clean/format the data so that:
   - Each row represents a single object/observation/entry
   - Each column represents an attribute/property/feature of that entry
   - Values are numeric whenever possible
   - Columns contain atomic properties that cannot be further decomposed*

"Tabular"

* Unlike food waste, which can be composted. Please consider composting food scraps.

# Exploratory Data Analysis (EDA)

## What? (continued)

summary statistics

3. Explore **global** properties: use histograms, scatter plots, and aggregation functions to summarize the data
4. Explore **group** properties: group like-items together to compare subsets of the data (are the comparison results reasonable/expected?)

"groupby" ← subgroup analysis.

This process transforms your data into a format which is easier to work with, gives you a basic overview of the data's properties, and likely generates several questions for you to follow-up in subsequent analysis.

# EDA: without Pandas

Say we have a small dataset of the top 50 most-streamed Spotify songs, globally, for 2019. (comes from Kaggle)

**NOTE:** The following music data are used purely for illustrative, educational purposes. The data, including song titles, may include explicit language. Harvard, including myself and the rest of the CS109 staff, does not endorse any of the entailed contents or the songs themselves, and we apologize if it is offensive to anyone in anyway.

CSV: comma-separated values/variables

# EDA: without Pandas

top50.csv

Each row represents a distinct song. The columns are:

- **ID:** a unique ID (i.e., 1-50)
- **TrackName:** Name of the Track
- **ArtistName:** Name of the Artist
- **Genre:** the genre of the track
- **BeatsPerMinute:** The tempo of the song.
- **Energy:** The energy of a song - the higher the value, the more energetic.
- **Danceability**: The higher the value, the easier it is to dance to this song.
- **Loudness**: The higher the value, the louder the song.
- **Liveness**: The higher the value, the more likely the song is a live recording.
- **Valence**: The higher the value, the more positive mood for the song.
- **Length**: The duration of the song (in seconds).
- **Acousticness**: The higher the value, the more acoustic the song is.
- **Speechiness**: The higher the value, the more spoken words the song contains.
- **Popularity**: The higher the value, the more popular the song is.

# EDA: without Pandas

csv as an "excel spreadsheet"

top50.csv

| | TrackName | ArtistName | Genre | BeatsPer Minute | Energy | Danceability | Loudness | Live ness | Valence | Length | Acousticness | Speechi ness | Popularity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Senorita | Shawn Menc | canadian pop | 117 | 55 | 76 | -6 | 8 | 75 | 191 | 4 | 3 | 79 |
| 2 | China | Anuel AA | reggaeton flow | 105 | 81 | 79 | -4 | 8 | 61 | 302 | 8 | 9 | 92 |
| 3 | boyfriend (w | Ariana Granc | dance pop | 190 | 80 | 40 | -4 | 16 | 70 | 186 | 12 | 46 | 85 |
| 4 | Beautiful Pec | Ed Sheeran | pop | 93 | 65 | 64 | -8 | 8 | 55 | 198 | 12 | 19 | 86 |

- 
- 
- 

**Q1:** What are some ways we can store this file into data structure(s) using regular Python (not the Pandas library).

# EDA: without Pandas

top50.csv

| | TrackName | ArtistName | Genre | BeatsPer Minute | Energy | Danceability | Loudness | Live ness | Valence | Length | Acousticness | Speechi ness | Popularity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Senorita | Shawn Menc | canadian pop | 117 | 55 | 76 | -6 | 8 | 75 | 191 | 4 | 3 | 79 |
| 2 | China | Anuel AA | reggaeton flow | 105 | 81 | 79 | -4 | 8 | 61 | 302 | 8 | 9 | 92 |
| 3 | boyfriend (w | Ariana Granc | dance pop | 190 | 80 | 40 | -4 | 16 | 70 | 186 | 12 | 46 | 85 |
| 4 | Beautiful Pec | Ed Sheeran | pop | 93 | 65 | 64 | -8 | 8 | 55 | 198 | 12 | 19 | 86 |

**Possible Solution #1:** A 2D array (i.e., matrix)

**Weaknesses:**

- What are the row and column names? Need separate lists for them – clumsy.
- Lists are O(N). We'd need 2 dictionaries just for column names

```
data = [][]
col_name -> index
index -> col_name
```

# EDA: without Pandas

top50.csv

| | TrackName | ArtistName | Genre | BeatsPer Minute | Energy | Danceability | Loudness | Live ness | Valence | Length | Acousticness | Speechi ness | Popularity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Senorita | Shawn Menc | canadian pop | 117 | 55 | 76 | -6 | 8 | 75 | 191 | 4 | 3 | 79 |
| 2 | China | Anuel AA | reggaeton flow | 105 | 81 | 79 | -4 | 8 | 61 | 302 | 8 | 9 | 92 |
| 3 | boyfriend (w | Ariana Granc | dance pop | 190 | 80 | 40 | -4 | 16 | 70 | 186 | 12 | 46 | 85 |
| 4 | Beautiful Pec | Ed Sheeran | pop | 93 | 65 | 64 | -8 | 8 | 55 | 198 | 12 | 19 | 86 |

- 
- 
- 

**Possible Solution #2:** A list of dictionaries

list

| | | |
|---|---|---|
| Item 1 | = | {“Track.Name”: “Senorita”, “Artist.Name”: “Shawn Mendes”, “Genre”: “Canadian pop”, …} |
| Item 2 | = | {“Track.Name”: “China”, “Artist.Name”: “Anuel AA”, “Genre”: “reggaetón flow”, … } |
| Item 3 | = | {“Track.Name”: “Ariana Grande”, “Artist.Name”: “boyfriend”, “Genre”: “dance pop”, … } |

# EDA: list of dictionaries

## **Possible Solution #2:** A list of dictionaries

```
f = open("../data/top50.csv", encoding = "ISO-8859-1")
column_names = f.readline().strip().split(",")[1:] # puts names in a list
cleaned_column_names = [name[1:-1] for name in column_names]
cleaned_column_names.insert(0, "ID")

dataset = []

# iterates through each line of the .csv file
for line in f:
    attributes = line.strip().split(",")

    # constructs a new dictionary for each line
    dataset.append(dict(zip(cleaned_column_names, attributes)))
```

read in the file

variable names

blank list

every line (starting at the 2nd line)

From Lecture2_Notebook.ipynb

# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q2:** Write code to print all songs (Artist and Track name) that are longer than 4 minutes (240 seconds):

```
for song in dataset:
    if int(song["Length."]) > 240:
        print(song["Artist.Name"], "-", song["Track.Name"], "is", song["Length."],"seconds long")
```

lecture2_Notebook

From lecture3.ipynb

# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q3:** Write code to print the most popular song (artist and track) – if ties, show all ties.

```
max_score = -1
most_populars = set()
for song in dataset:
    if int(song["Popularity"]) > max_score:
        most_populars = set([str(song["Artist.Name"] + "-" + song["Track.Name"])])
        max_score = int(song["Popularity"])
    elif int(song["Popularity"]) == max_score:
        most_populars.add(str(song["Artist.Name"] + "-" + song["Track.Name"]))
print(most_populars)
```

From lecture3.ipynb

# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q4:** Write code to print the songs (and their attributes), if we sorted by their popularity (highest scoring ones first).

1. first sort the list of dictionaries based on the key popularity
2. print out the sorted list (the attributes you want)

# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q4:** Write code to print the songs (and their attributes), if we sorted by their popularity (highest scoring ones first).

list

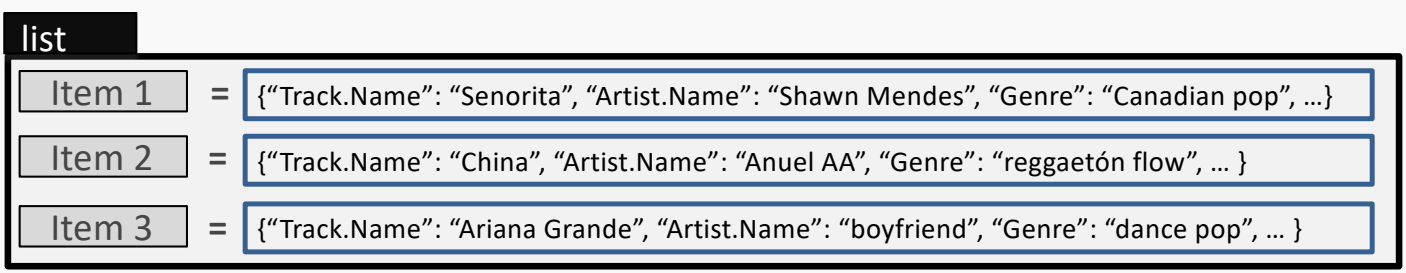
| | | |
|---|---|---|
| Item 1 | = | {"Track.Name": "Senorita", "Artist.Name": "Shawn Mendes", "Genre": "Canadian pop", …} |
| Item 2 | = | {"Track.Name": "China", "Artist.Name": "Anuel AA", "Genre": "reggaetón flow", … } |
| Item 3 | = | {"Track.Name": "Ariana Grande", "Artist.Name": "boyfriend", "Genre": "dance pop", … } |

Cumbersome to move dictionaries around in a list.
Problematic even if we don't move the dictionaries.

# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q5:** How could you check for null/empty entries? This is only 50 entries. Imagine if we had 500,000.



# EDA: list of dictionaries

**Possible Solution #2:** A list of dictionaries

**Q6:** Imagine we had another table* below (i.e., .csv file). How could we combine its data with our already-existing *dataset*?

spotify_aux.csv

| | TrackName | ArtistName | ExplicitLanguage |
|---|---|---|---|
| 1 | Senorita | Shawn Menc | TRUE |
| 2 | China | Anuel AA | FALSE |
| 3 | boyfriend (w | Ariana Granc | TRUE |
| 4 | Beautiful Pec | Ed Sheeran | FALSE |

* 3rd column is made-up (randomly created values). Pretend they're accurate.

1. Iterate through our list of dictionaries (songs)
2. Lookup if there's a match in the new file
3. add an attribute to the existing list of songs.

# EDA: with Pandas!

Kung Fu Panda is property of DreamWorks and Paramount Pictures

# Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas
  - **With Pandas**
- Data concerns
- Reading Data from the Web
  - Web Scraping with Beautiful Soup

# EDA: with Pandas

## What / Why?

- Pandas is an *open-source* Python library (anyone can contribute)
- Allows for high-performance, easy-to-use data structures and data analysis
- Unlike NumPy library which provides multi-dimensional arrays, Pandas provides 2D table object called **DataFrame** (akin to a spreadsheet with column names and row labels).
- Used by *a lot* of people

# EDA: with Pandas

## How

- import **pandas** library (convenient to rename it)
- Use **read_csv()** function

```
import pandas as pd
dataframe = pd.read_csv("yourfile.csv")
```

## Common Panda functions

**High-level viewing:**

- head() – first N observations
- tail() – last N observations
- columns() – names of the columns
- describe() – statistics of the quantitative data
- dtypes() – the data types of the columns

## Common Panda functions

### Accessing/processing:

- df["column_name"]
- Df.column_name
- .max(), .min(), .idxmax(), .idxmin()
- <dataframe> <conditional statement>
- .loc[] – label-based accessing
- .iloc[] – index-based accessing
- .sort_values()
- .isnull(), .notnull()

labels (might not be 0:n)

0:n

## Common Panda functions

**Grouping/Splitting/Aggregating:**

- groupby(), .get_groups()
- .merge()
- .concat()
- .aggegate()
- .append()

# EDA: with Pandas

Now, let's open the Lecture2_Notebook.ipynb notebook for some real-time practice.

# Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas
  - With Pandas
- **Data concerns**
- Collecting Data from the Web
  - Web Scraping with Beautiful Soup

# Data Concerns

When determining if a dataset is sound to use, it can be useful to think about these four questions:

- Did it come from a trustworthy, authoritative source?
- Is the data a complete sample?
- Does the data seem correct?
- **(optional)** Is the data stored efficiently or does it have redundancies?

# Data Concerns: the format

- Often times, there may not exist a single dataset that contains all of the information we are interested in.
- May need to merge existing datasets
- Important to do so in a sound and efficient format

# Data Concerns: the format

For example, say we have two datasets:

current top50.csv

## Dataset 1

Top 200 most-frequent streams per day (for June 2020)

| | SpotifySongID, | # of Streams, | Date |
|---|---|---|---|
| 200 | 2789179, | 42003, | 06-01 |
| | • | | |
| | • | | |
| | 3819390, | 89103, | 06-01 |
| 200 | 4492014, | 52923, | 06-02 |
| | • | | |
| | • | | |
| | 8593013, | 189145, | 06-02 |

6,000 x 3

200 most-frequent streams daily over 30 days.

## Dataset 2

Top 50 most streamed in 2019

| | SpotifySongID, | Artist, Track, [10 acoustic features] |
|---|---|---|
| 50 | 2789179, | Billie Eilish, bad guy, 3.2, 5.9, … |
| | • | |
| | • | |
| | 3901829, | Outkast, Elevators, 9.3, 5,1, … |

50 x 13

# Data Concerns: the format

For example, say we have two datasets:

## Dataset 1

Top 200 most-frequent streams per day (for June 2019)

| SpotifySongID, | # of Streams, | Date |
|---|---|---|
| 2789179, | 42003, | 06-01 |
| • | | |
| • | | |
| 3819390, | 89103, | 06-01 |
| 4492014, | 52923, | 06-02 |
| • | | |
| • | | |
| 8593013, | 189145, | 06-02 |

200 (rows 2789179 … 3819390)

200 (rows 4492014 … 8593013)

**6,000 x 3**

## Dataset 2

Top 50 most streamed in 2019, so far

| SpotifySongID, | Artist, | Track, [10 acoustic features] |
|---|---|---|
| 2789179, | Billie Eilish, | bad guy, 3.2, 5.9, … |
| • | | |
| • | | |
| 3901829, | Outkast, | Elevators, 9.3, 5,1, … |

50 (rows 2789179 … 3901829)

**50 x 13**

merge based on song ID

danceability

We are interested in determining if songs with high danceability are more popular during the weekends of June than weekdays in June. What should our merged table look like? Concerns?

popularity across days in this dataset.

# Data Concerns: the format

This is wasteful, as it has 10 acoustic features, artist, and track repeated many times for each unique song.

Datasets Merged (poorly)

| | SpotifySongID, | # of Streams, | Date, | Artist, Track, [10 acoustic features] |
|---|---|---|---|---|
| 200 | 2789179, | 42003, | 06-01 | Billie Eilish, bad guy, 3.2, 5.9, … |
| | • | | | |
| | • | | | |
| | 3819390, | 89103, | 06-01 | Outkast, Elevators, 9.3, 5,1, … |
| 200 | 4492014, | 52923, | 06-02 | |
| | • | | | |
| | • | | | |
| | 8593013, | 189145, | 06-02 | |

**6,000 x 15 ➔ 90,000 cells**

# Data Concerns: the format

Some rows may have null values for # of Streams (if the song wasn't popular in June)

## Datasets Merged (better)

**SpotifySongID, Artist, Track, [10 acoustic features], 06-01 Streams, 06-02 Streams**

50

2789179, Billie Eilish, bad guy, 3.2, 5.9, …, 42003, 42831, 43919

•
•

3901829, Outkast, Elevators, 9.3, 5,1, … 29109, 27193, 25982

**50 x 70 ➔ 3,500 cells**

# Data Concerns: the format

- Is the data correctly constructed (or are there any wrong values)?
- Is there redundant data in our merged table?
- Are there any missing values?

↑ handling & imputing missing values.

# Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas
  - With Pandas
- Data concerns
- **Collecting Data from the Web**
  - **Web Scraping with Beautiful Soup**

# Sources of data

- Data can come from several sources:
  - You curate it
  - Someone else provides it, all pre-packaged for you (possibly in .CSV)
    - Kaggle: https://www.kaggle.com/datasets
    - Johns Hopkins (COVID): https://github.com/CSSEGISandData/COVID-19
  - Someone else provides an API (Application programming interface)
    - Twitter: https://developer.twitter.com/en/docs
  - Someone else has available content, and you try to take it (ex: web scraping).
    - This part of the lecture notes addresses this approach.

# What is a website?

harvard.edu

Faculty Staff Students Alumni Parents Visitors Media What can we help you find?

About Harvard Admissions & Aid HARVARD UNIVERSITY Schools On Campus

Gazette News Events Visit Give

⚠ Latest information about COVID-19 >>

# Removing Confederate statues

Legal scholar and historian Annette Gordon-Reed puts the push to remove Confederate statues in context

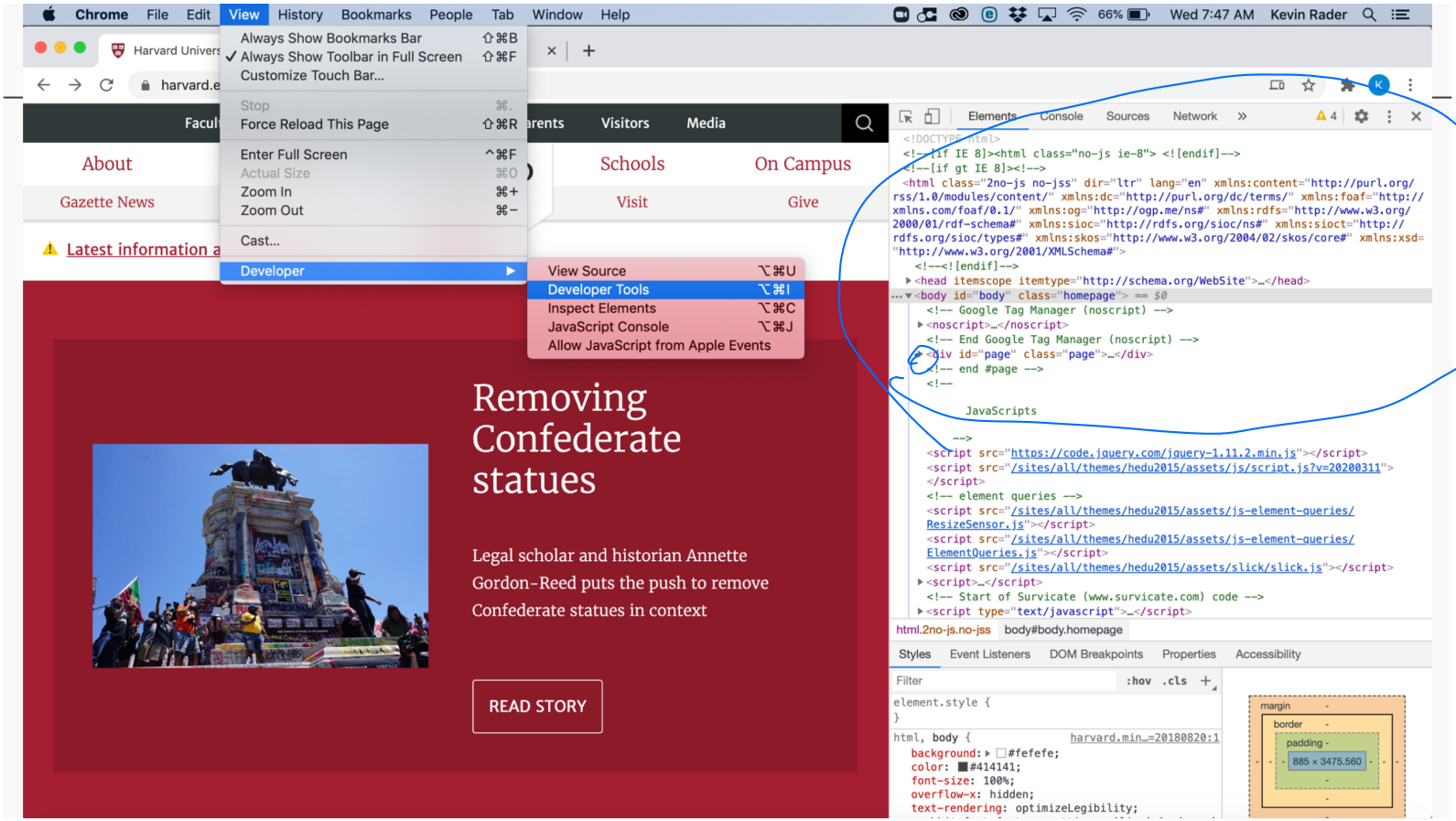
READ STORY

# Web servers

- Website servers
  - A server is a long-running process (also called a daemon) which listens on pre-specified port(s).
  - It responds to requests (from a client), which is sent using a protocol called HTTP (HTTPS is secure).
  - Our browser (like Chrome) sends these requests and downloads the content, then displays it.
  - The server communicates back to the client the following codes depending on how the connection went:
    - 2– request was successful, 4– client error, often `page not found`; 5– server error (often that your request was incorrectly formed)
      - https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

# HTML: the language of the web

- Most websites are really just snippets of **Hypertext Markup Language** (HTML) code.
- As per Wiki: "Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript and VBScript."
- We do not need to know much about HTML to scrape the web, but getting a little glimpse into the format is helpful.
- Google's Chrome can help with this task (in the menu: View -> Developer -> Developer Tools).


Chrome File Edit View History Bookmarks People Tab Window Help
Wed 7:47 AM Kevin Rader
Always Show Bookmarks Bar ⇧⌘B
Always Show Toolbar in Full Screen ⇧⌘F
Customize Touch Bar...
Stop ⌘.
Force Reload This Page ⇧⌘R
Enter Full Screen ^⌘F
Actual Size ⌘0
Zoom In ⌘+
Zoom Out ⌘−
Cast...
Developer
View Source ⌥⌘U
Developer Tools ⌥⌘I
Inspect Elements ⌥⌘C
JavaScript Console ⌥⌘J
Allow JavaScript from Apple Events
Visitors
Media
About
Schools
On Campus
Gazette News
Visit
Give
Latest information a
Removing Confederate statues
Legal scholar and historian Annette Gordon-Reed puts the push to remove Confederate statues in context
READ STORY
Elements
Console
Sources
Network
Styles
Event Listeners
DOM Breakpoints
Properties
Accessibility
Filter
element.style {
}
html, body {
background: #fefefe;
color: #414141;
font-size: 100%;
overflow-x: hidden;
text-rendering: optimizeLegibility;
margin
border
padding
885 × 3475.560

# Web scraping

- Scraping the web is then a process of using programs to download or otherwise get data from these online servers.
    - Typically, processing (HTML) code that is being sent from server to client.
- Often much faster than manually copying data!
- Part of the process is transferring the relevant data into a form that is compatible with your code.
- Always remember: legal and ethical issues should be considered (see Lecture 1).

# Web scraping in Python

- `Requests` (Python library): gets a webpage for you
  - `Requests.get(url)`
- The `BeautifulSoup` library helps to make parsing webpages (.html content) much easier!
- Use BeautifulSoup to quickly find, save, and organize all the text, or all the links, or all the tables, etc. on a webpage
- Documentation: http://crummy.com/software/BeautifulSoup

# Web scraping

- Why scrape the web?
  - Vast source of information; often useful to combine with other data sets (or across web-based sources)
  - Companies have not provided APIs or otherwise publicly available data
  - It can automate tasks (rather than using the oldf *copy-and-paste* many, many times).
  - Keep up with real-time updates on websites (think election night data).
  - Because its fun, and allows us to answer interesting questions we would otherwise be unable to answer!

# Web scraping

- A few words of advice while scraping:
  - Be careful and polite
  - Give credit where credit is due (like proper citations)
  - Be aware of media law!
  - Don't be evil (no spamming, overloading sites, etc.).

# Robots

- Be aware of the Robots.txt (like a readme) page on many websites:
  - Specified by website owners
  - Gives instructions to web robots (aka web crawlers, like our scripts).
  - Is located at the top-level of the web server's directory
    - Ex: https://www.google.com/robots.txt

# A little HTML

- Contains many angle brackets for 'tags' (define structure of the content)
- Often in pairs, ex: `<p>Hello</p>`
- Lines are often separated by unpaired breaks: `<br>`
- Tables are typically defined by `<table>`, with `<tr>` for table rows, `<th>` for table headers and `<td>` table cells/data.

# HTML examples

## A simple page:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

## A simple table:

```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

# Web scraping with Beautiful Soup

- We highly suggest using BeautifulSoup (within the library `bs4`) to do html scraping and handling for you.
  - it will quickly organize and parse dirty html for you.
  - it is sometimes easier using regular expression instead, see Friday's lab).
- Basic usage:

```
import bs4
# get bs4 object
soup = bs4.BeautifulSoup(source)
# all <a> tags, which define hyperlinks
soup.findAll('a')
# first <a>
soup.find('a')
# get all links in the page
link_list = [l.get('href') for l in soup.findAll('a')]
```

because now we can use this to iterate through relevant pages.

# BeautifulSoup structure

- HTML is structured like a tree when read in via BeautifulSoup:

```
tree = bs4.BeautifulSoup(source)

# get html root node
root_node = tree.html

# get head from root using contents
head = root_node.contents[0]

# get body from root using contents
body = root_node.contents[1]

# body can be accessed directly
tree.body
```

## Some thinking questions

- Question: how can we get a list of all image URLs?
- Question: how can we navigate through subsequent pages (i.e., crawler) recursively.
- Question: could we crawl the entire web?

**Note: we will go through much more details of scraping in Friday's lab.**