Getting our hands dirty: pandas and web scraping

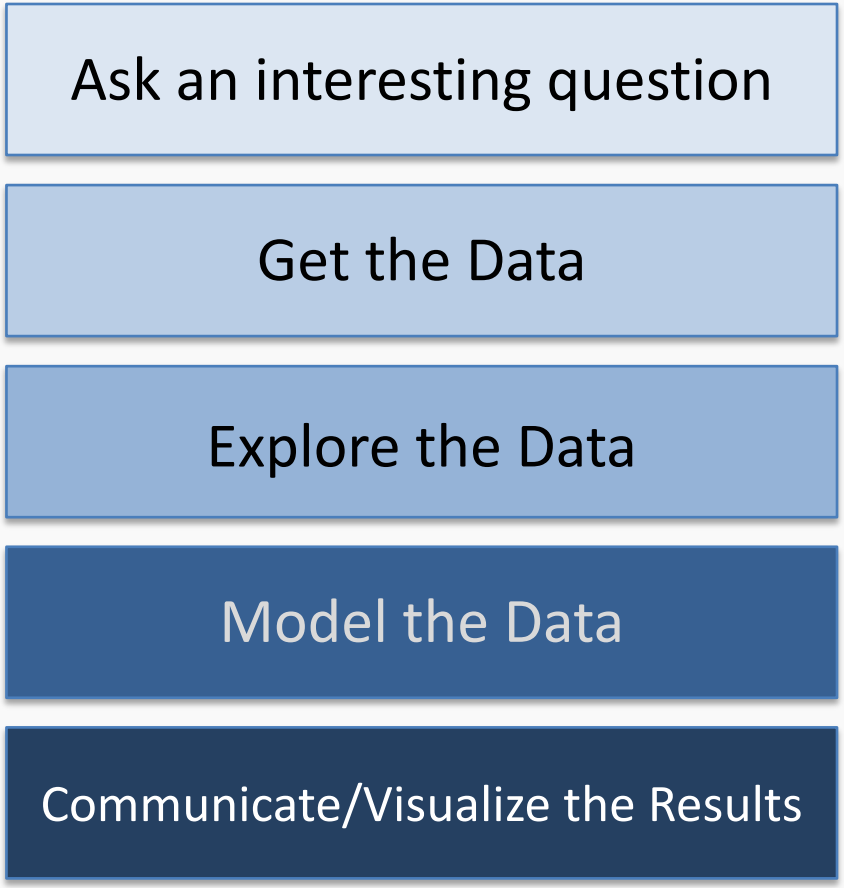
Background

So far, we've learned:

- What is Data Science?
- The Data Science Process
- Data: types, formats, issues, etc.
- Visualization (briefly)
- How to quickly prepare data and scrape the web
- How to model data

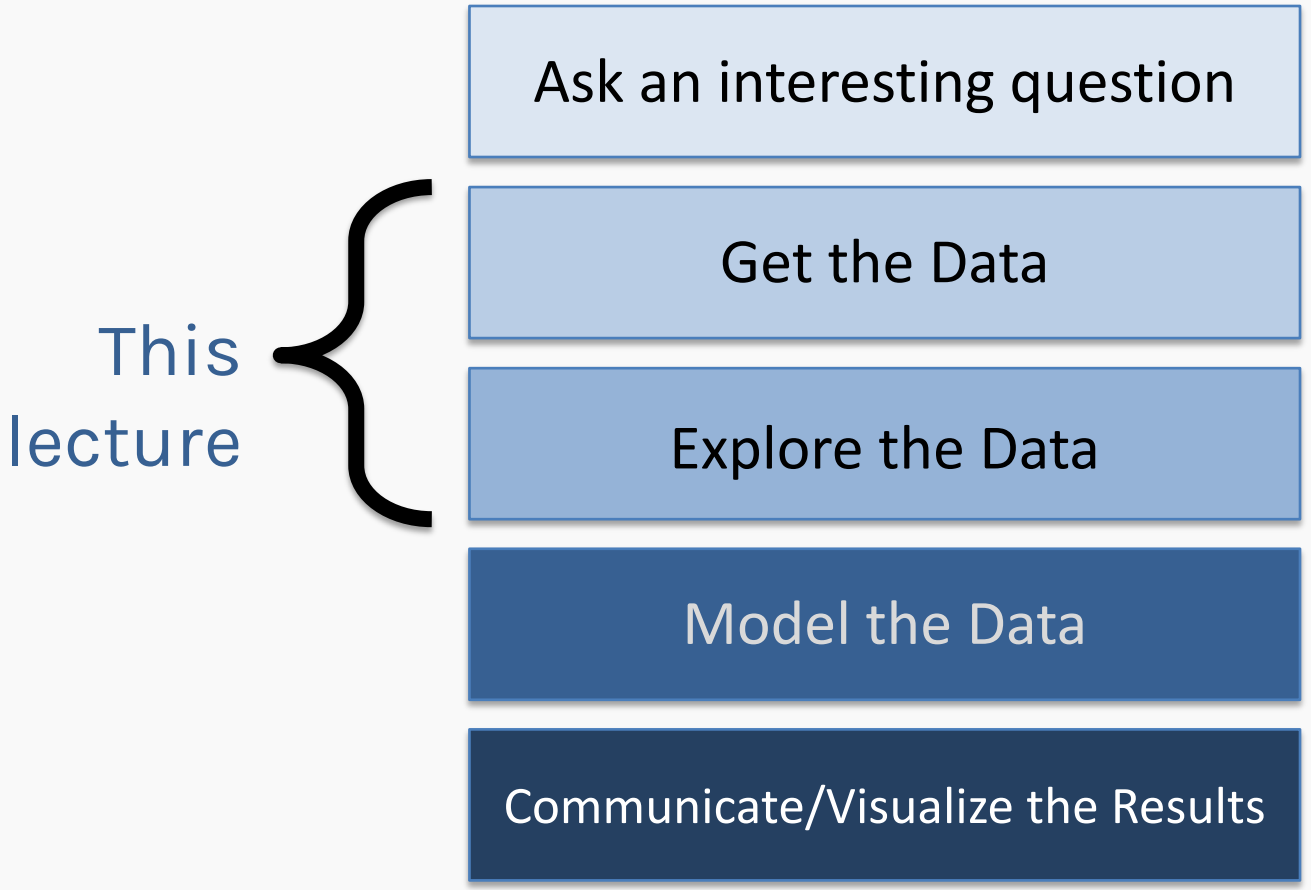
Background

The Data Science Process:



Background

The Data Science Process:



Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas (part 1) – These slides
  - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

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*

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

What? (continued)

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?)

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.

EDA: without Pandas

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

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.

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

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)))
```

From Day3.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")
```

From Day3.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 Day3.ipynb

EDA: list of dictionaries

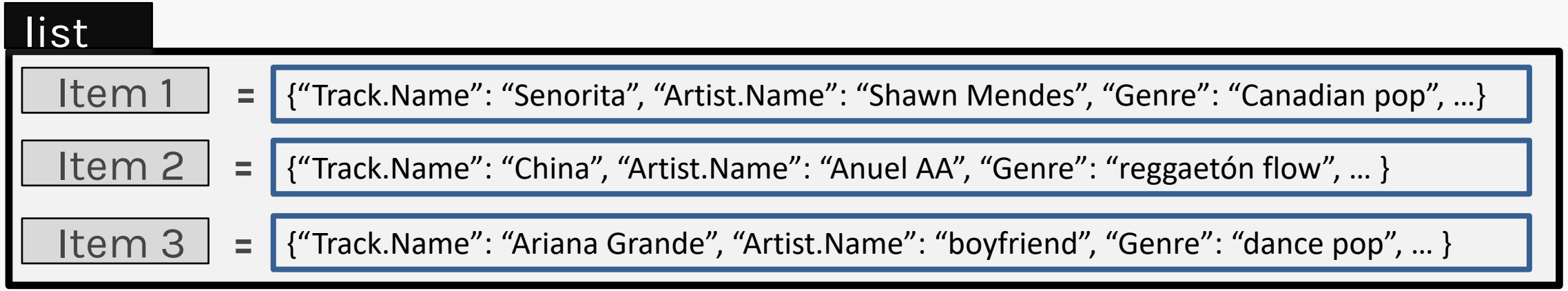
Possible Solution #2: A list of dictionaries

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

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).

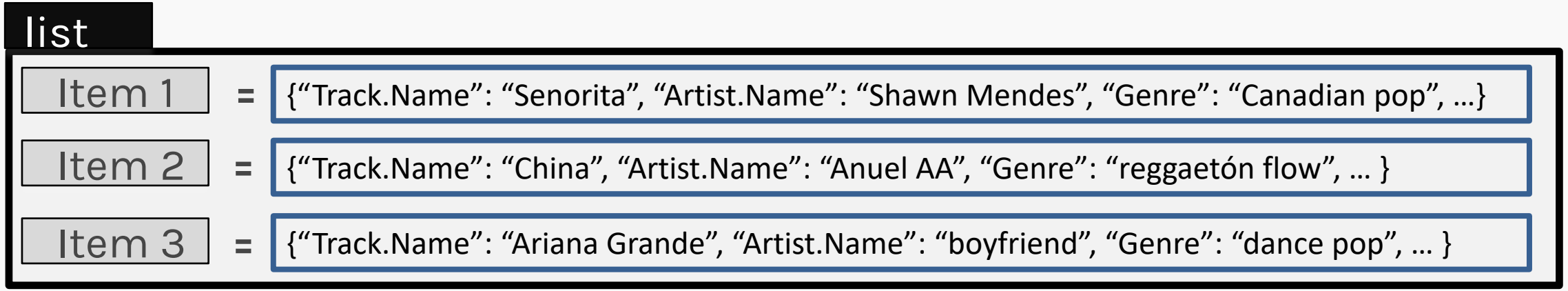


Cumbersome to move dictionaries around in a list. Problematic even if we don’t move the dictionaries.

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 by me. Random values. Pretend they're accurate.

EDA: with Pandas!

Kung Fu Panda is property of DreamWorks and Paramount Pictures

Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas (part 1) – These slides
  - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

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()

Common Panda functions

Grouping/Splitting/Aggregating:

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

EDA: with Pandas

Now, let’s open the Day3.ipynb. notebook for some real-time practice.

Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas (part 1) – These slides
  - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

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:

Dataset 1

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

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

Dataset 2

Top 50 most streamed in 2019, so far

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

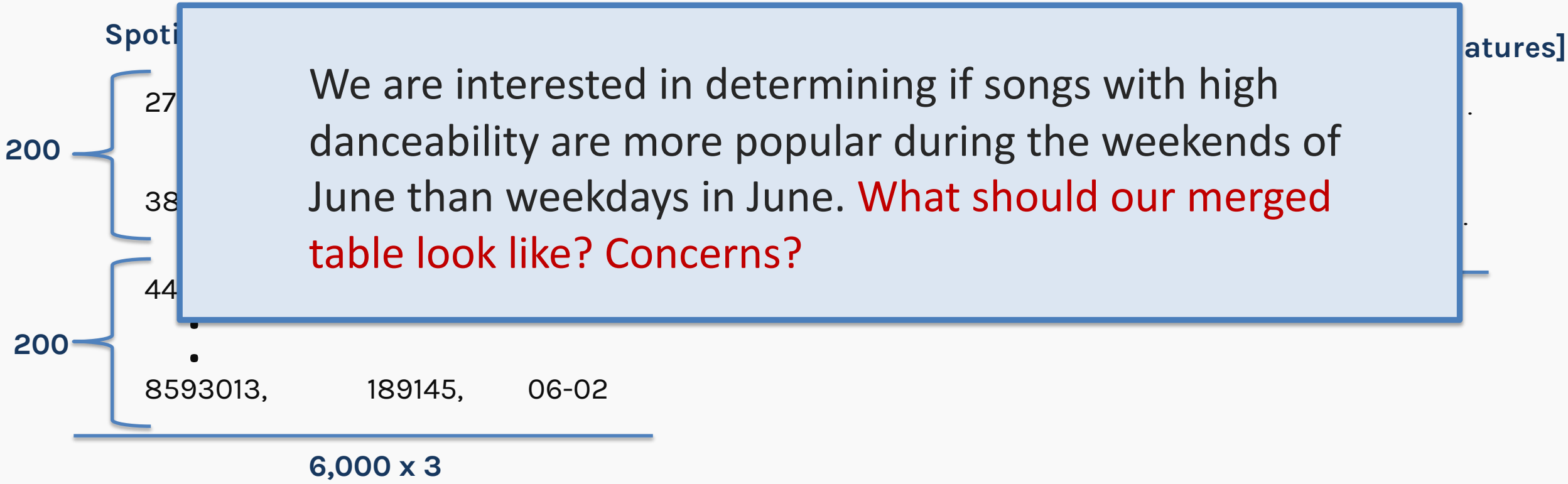
For example, say we have two datasets:

Dataset 1

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

Dataset 2

Top 50 most streamed in 2019, so far



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?

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 values wrong)?
- Is there redundant data in our merged table?
- Missing values?

Lecture Outline

- Exploratory Data Analysis (EDA):
  - Without Pandas (part 1) – These slides
  - With Pandas (part 2) – Mostly Jupyter Notebook
- Data concerns (part 3) – These slides
- Web Scraping with Beautiful Soup (part 4) – Mix

Sources of data

- Data can come from:
  - You curate it
  - Someone else provides it, all pre-packaged for you
  - Someone else provides an API
  - Someone else has available content, and you try to take it (web scraping)

Web scraping

- Web servers
  - A server is a long-running process (also called a daemon) which listens on a pre-specified port(s)
  - It responds to requests, which is sent using a protocol called HTTP (HTTPS is secure)
  - Our browser sends these requests and downloads the content, then displays it
  - 2– request was successful, 4– client error, often `page not found`; 5– server error (often that your request was incorrectly formed)

Web scraping

- Using programs to download or otherwise get data from online
- Often much faster than manually copying data!
- Transfer the data into a form that is compatible with your code
- Legal and moral issues (per Lecture 2)

Web scraping

- Requests (Python library): gets a webpage for you
- Requests.get(url)
- BeautifulSoup library parses webpages (.html content) for you!
- Use BeautifulSoup to find all the text or all the links on a page
- Documentation: http://crummy.com/software/BeautifulSoup

Web scraping

- Why scrape the web?
- vast source of information, combine with other data sets
- companies have not provided APIs
- automate tasks
- keep up with sites
- fun!

Web scraping

- be careful and polite
- give credit
- care about media law
- don't be evil (no spam, overloading sites, etc.)

Robots.txt

- specified by web site owner
- gives instructions to web robots (aka your script)
- is located at the top-level directory of the web server

e.g.: http://google.com/robots.txt

Web scraping

HTML

- angle brackets
- should be in pairs, eg Hello
- maybe in implicit bears, such as

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ttle</title>
  </head>
  <body>
    <h1>Body Title</h1>
    <p>Body Content</p>
  </body>
</html>
```

Beautiful Soup

- will normalize dirty html
- basic usage

```
import bs4
## get bs4 object
soup = bs4.BeautifulSoup(source)
## all a tags
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')]
```

HTML is a tree

```
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
body = root_node.contents[1]

## could directly access body
tree.body
```

Web scraping

- 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?