Data Science for Social Good

April 2, Session 2

questions?

- Please be patient
  - You have diverse backgrounds & interests
  - Data science is interdisciplinary in nature
- Each team is required to deliver only one presentation and submit one report
  - All members must actively participate in both the preparation and delivery of the presentation

today we cover

- Digital traces
- Characteristics of big data
- Data collection

Chapter 2, Bit by Bit: Social Research in the Digital Age, by Matthew Salganik (2018)
Chapter 2, Big Data and Social Science (Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences), 2nd Edition, by Ian Foster, Rayid Ghani, Frauke Kreuter (Editors)
Available at: https://textbook.coleridgeinitiative.org/

observing behavior

“any data that results from observing a social system without intervening in any way.”

- Collecting data about behavior in the analog era,
  - as “who does what, when,”
  - was expensive,
  - and rare.

observing behavior in the digital age

- The behaviors of billions of people are recorded
  - Every time you click on a website, make a call on your mobile phone, or pay for something with your credit card, a digital record of your behavior is created and stored by a business.
  - Smartphone data
  - Smartwatch data
  - Electronic health record
  - Every content we produce, e.g., news article

- They are often called “digital traces,”
  - a by-product of people’s everyday actions

other data sources

Survey Data

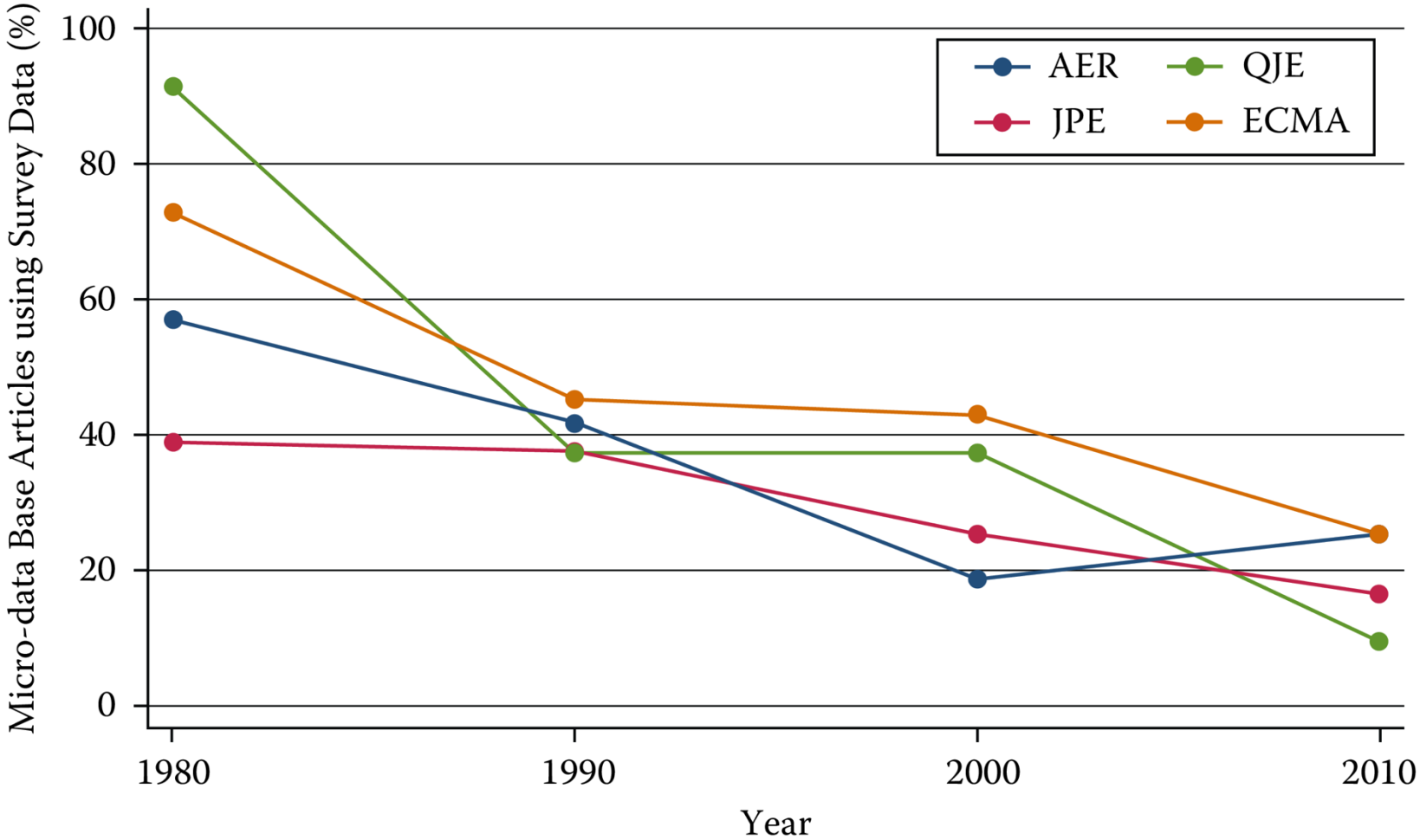
- Involves the systematic recruitment of a large number of participants
- Uses highly structured questionnaires
- Employs statistical methods to generalize findings from the sample to a larger population

In-Depth Interviews

- Involve a small number of participants
- Use semistructured conversations
- Produce rich, qualitative descriptions of participants' experiences and perspectives

Running Experiments

- Involves systematically intervening in the world
- Generates data ideally suited for answering questions about cause-and-effect relationships



Note: "Pre-existing survey" data sets refer to micro surveys such as the CPS or SIPP and do not include surveys designed by researchers for their study. Sample excludes studies whose primary data source is from developing countries.

Use of pre-existing survey data in publications in leading journals, 1980–2010 (Chetty 2012)

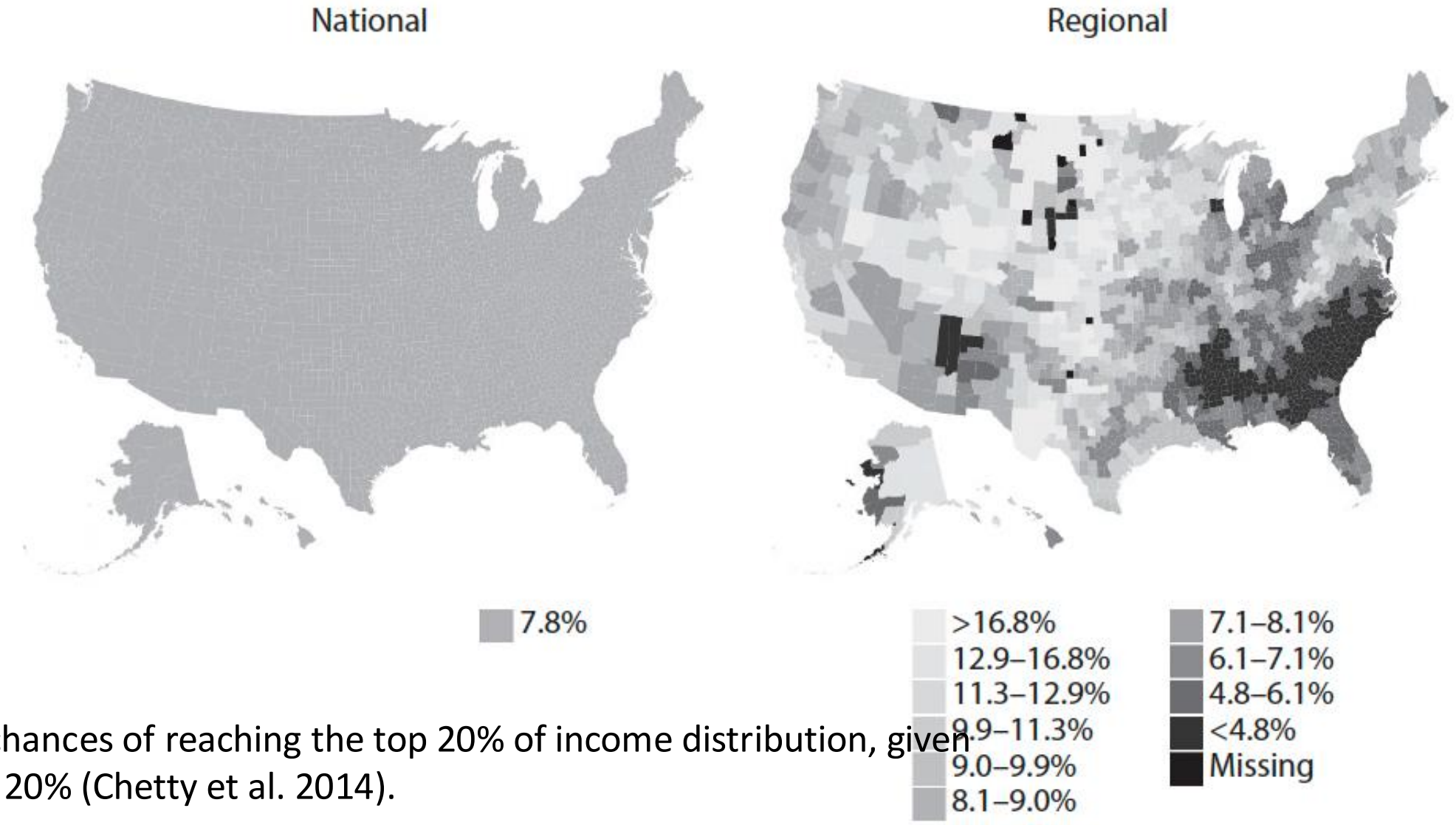
Chetty, Raj. 2012. "The Transformative Potential of Administrative Data for Microeconometric Research." http://conference.nber.org/confer/2012/SI2012/LS/ChettySlides.pdf. Accessed February 1, 2016.

some characteristics of such data

1. Volume – Large amounts of data
2. Always-On – Continuous data collection
3. Nonreactive – Less observer effect
4. Incomplete – Missing essential variables
5. Inaccessible – Data owned by corporations or governments
6. Nonrepresentative – Biased sample of population
7. Drifting – Changes over time
8. Algorithmically Confounded – Influenced by AI/algorithms
9. Dirty – Contains errors or inconsistencies
10. Sensitive – Raises privacy concerns

volume

- The scale of data collected is unprecedented
- Millions of records per second from diverse sources
- Enables study of
  - rare events,
  - capturing heterogeneity,
  - detection small differences.



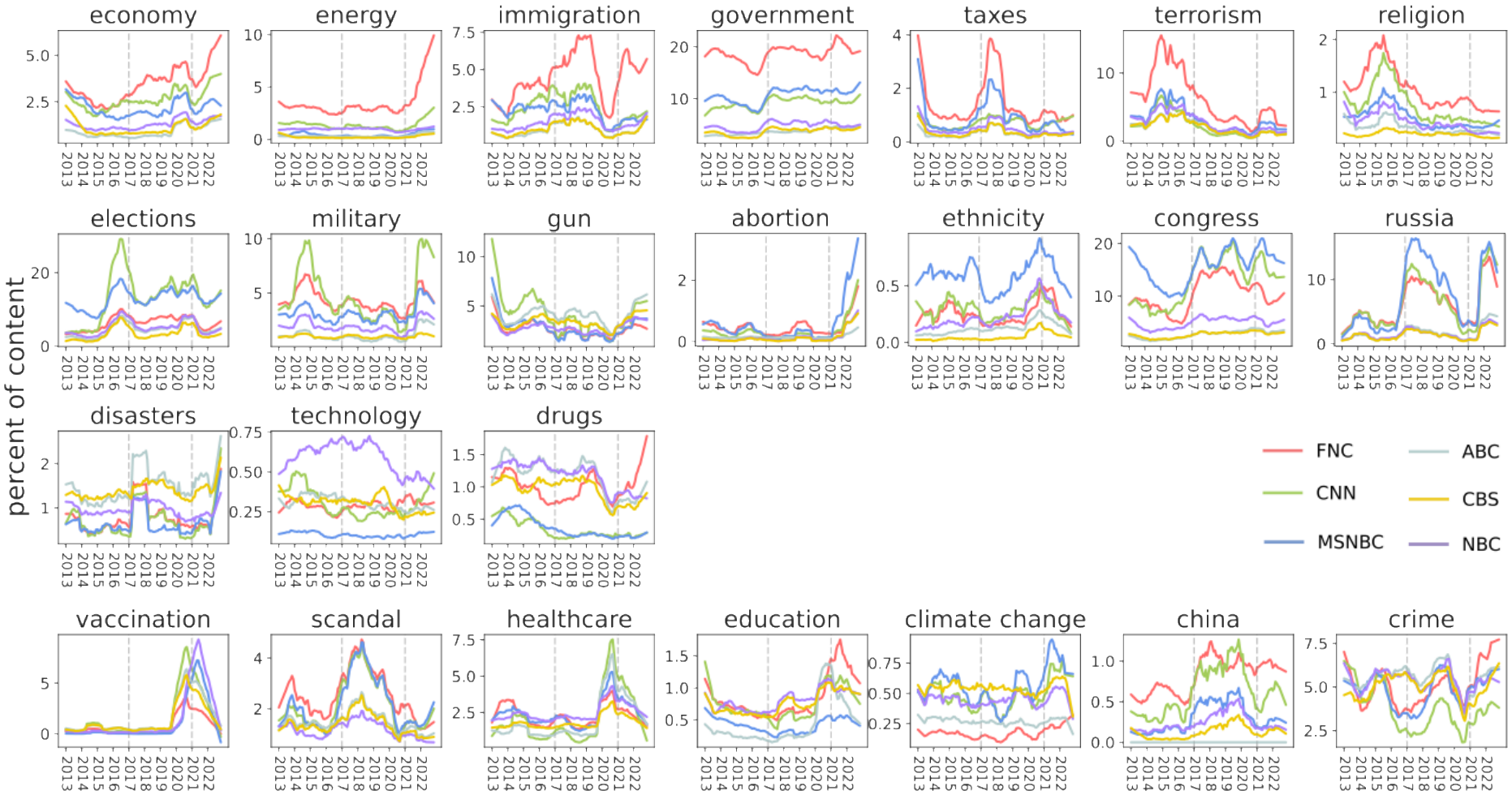
- Estimates of a child's chances of reaching the top 20% of income distribution, given parents in the bottom 20% (Chetty et al. 2014).

- The probability that a child reaches the top quintile of the national income distribution starting from a family in the bottom quintile is about 13% in San Jose, California, but only about 4% in Charlotte, North Carolina.

always-On

- Digital data collection never stops
  - Constantly collecting data
  - Longitudinal data (i.e., data over time)
- Captures real-time changes and unexpected events
- Useful for monitoring social phenomena and forecasting trends

a decade of selection bias on TV



nonreactive

- “Reactivity” is defined as people changing their behavior when being observed

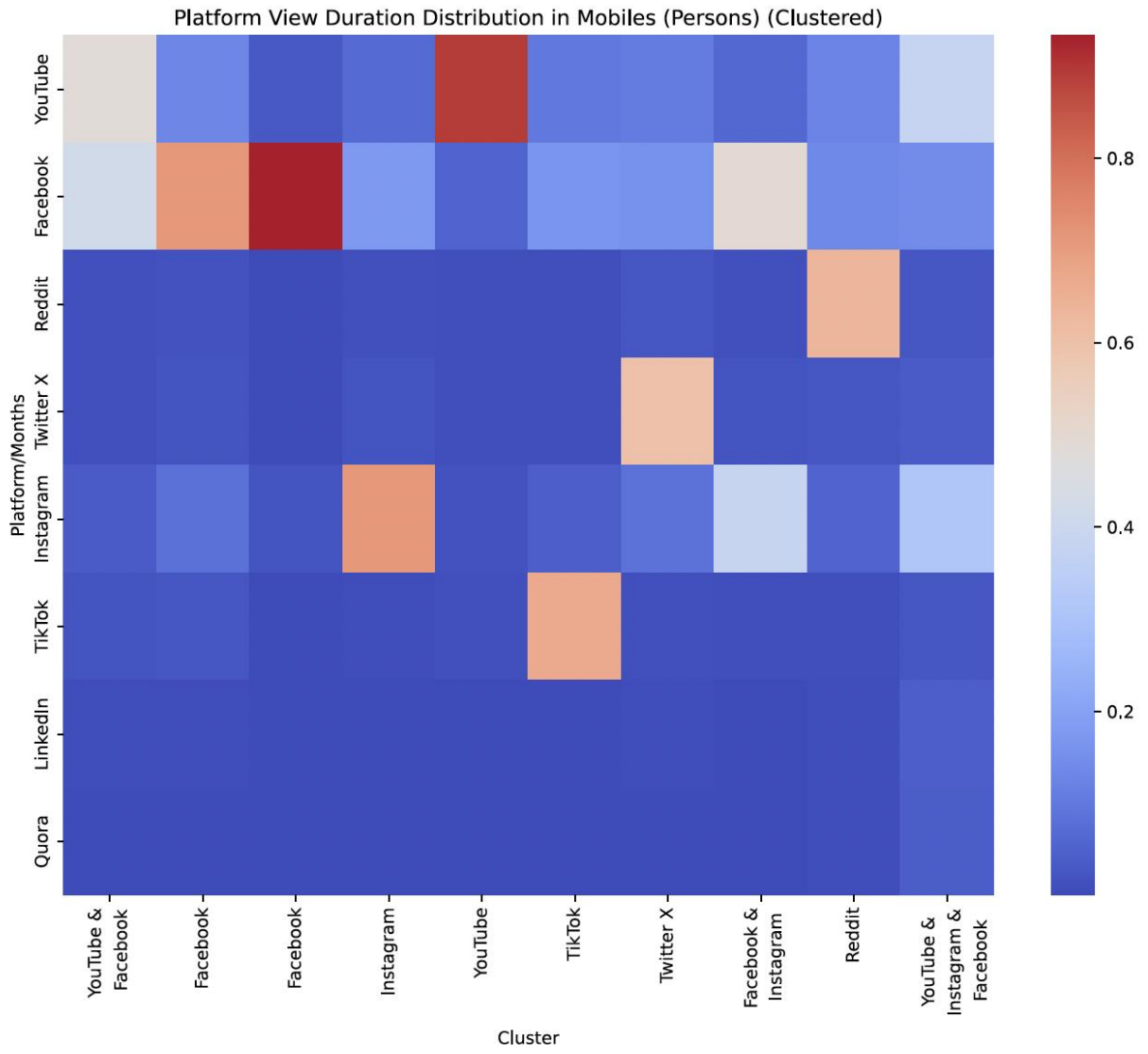
- No change in behavior, as participants are
  - either not aware that their data are being captured,
  - or they have become so accustomed to it!

- Not quite, for example
  - social desirability biases

incomplete

- Big does not mean complete
  - Lack of key demographic information
    - e.g., social media data lacks age or income
  - Limited view
    - e.g., only see behavior within one platform
  - Need to measure things
    - e.g., how to measure constructs such as intelligence or problematic content

the archetypes of app consumption on mobile

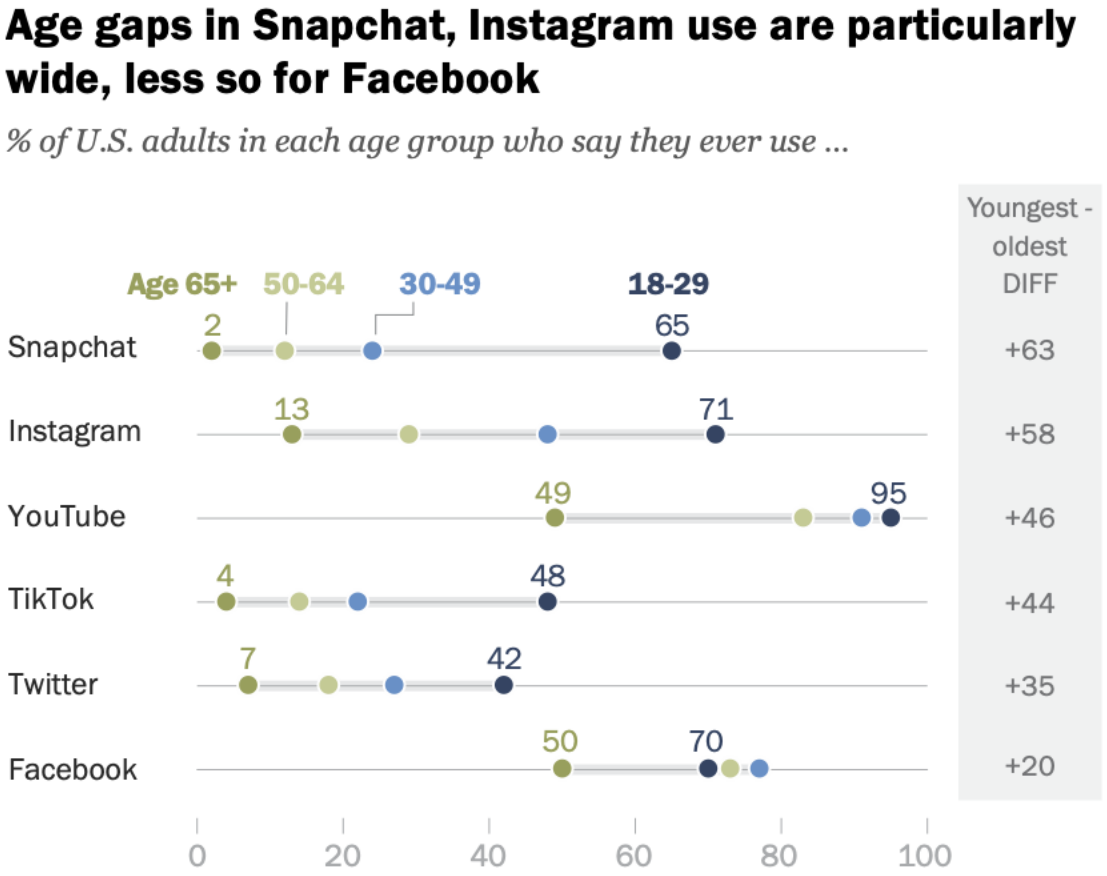


inaccessible

- Much of big data is proprietary
- Researchers rely on partnerships or API access
  - Facebook and Google control access to vast user interaction data

nonrepresentative

- Digital traces do not represent the full population
- Biases in who generates and consumes data



https://www.pewresearch.org/internet/2021/04/07/social-media-use-in-2021/

data drift

- Platforms and behaviors change over time
  - YouTube changed its platform's features in 2019
- Data collected today may not reflect past or future patterns
  - Facebook’s newsfeed algorithm changes affect user engagement

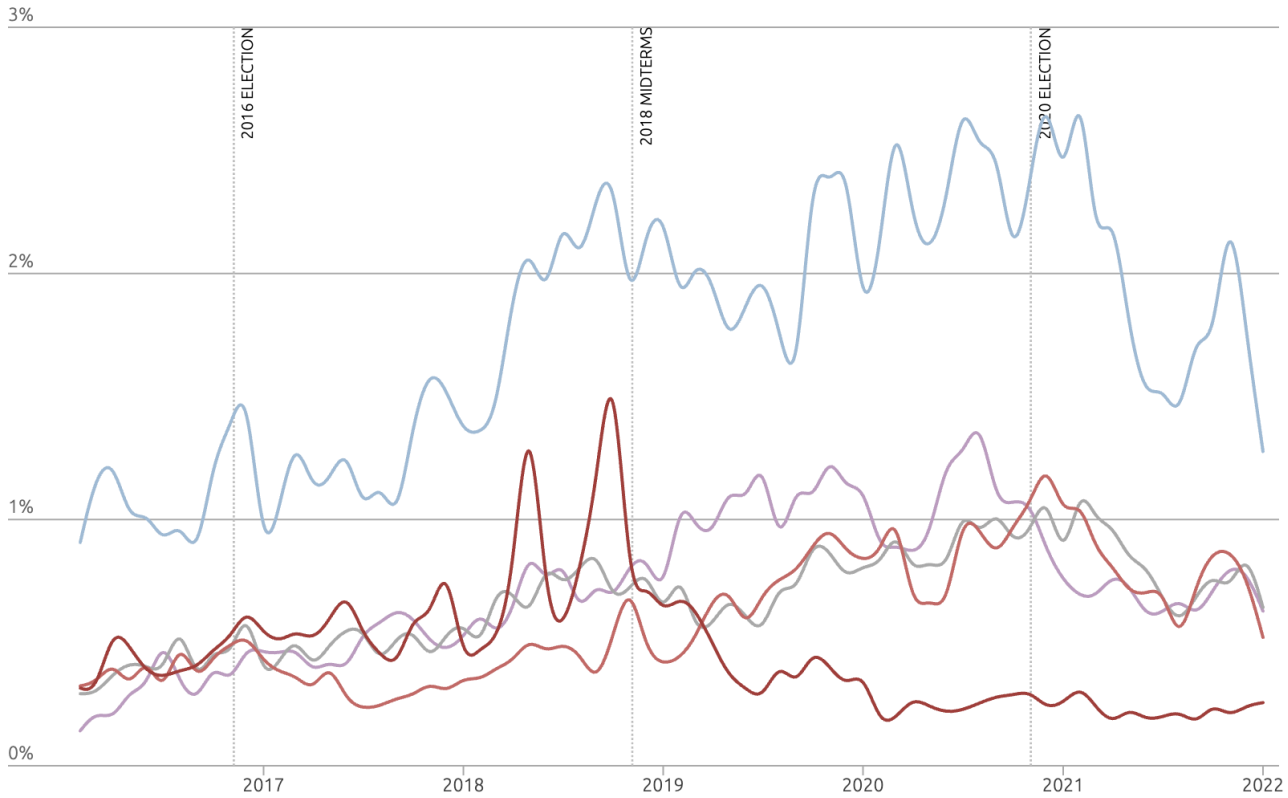
watts-lab.github.io/LivingJournal-YoutubeDashboard/

PodCast | Learning From Dat... | How to Prepare for... | Top 30 Programmi... | Video | Introductio... | http://www.naftali...

Community engagement

Total consumption share

Left | Center | Anti-woke | Right | Far Right



Consumption share of the six political channel categories. For example, out of the total minutes of content that Americans viewed across the platform in October 2018, 1.76% of those minutes were far-right content. These statistics shed light on the amount of political content being consumed, and not necessarily the communities that consume it.

Download data

YouTube Official Blog | News & Events | Creator & Artist Stories | Culture & Trends | Inside YouTube | Made On YouTube

NEWS AND EVENTS

Continuing our work to improve recommendations on YouTube

TECH | ALL | CHEAT SHEET | MEDIA | OBSESSED | ROYALS | POLITICS

YouTube Tweaks Algorithm to Fight 9/11 Tr
Flat Earthers, Miracle Cures

WHAT TOOK THEM SO LONG?

Videos about impossible conspiracy theories show up high in YouTube's recommendations. That's going company says.

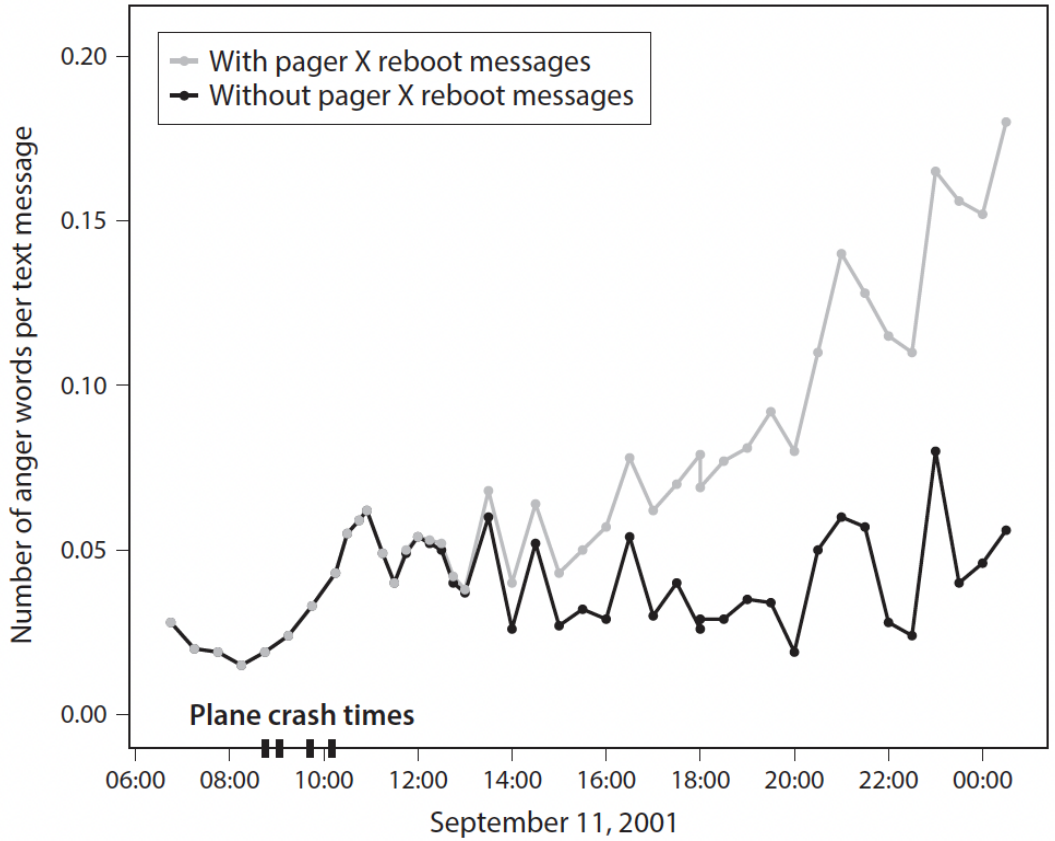
Kelly Weill | Published Jan. 25 2019 12:41PM EST

algorithmic confounding

- Data collection and visibility shaped by algorithms
- Difficult to separate human behavior from algorithmic influence
  - Google autocomplete suggestions influence search behavior
- On Facebook there are an anomalously high number of users with approximately 20 friends
  - Facebook encouraged people with few connections on Facebook to make more friends until they reached 20 friends

dirty data

- Large datasets contain errors, duplicates, and inconsistencies
- Requires extensive cleaning and validation
  - Luv u, Looooooove YOUUU, etc
- Example: Bots and spam accounts skew social media analysis
  - words related to (1) sadness (e.g., “crying” and “grief”), (2) anxiety (e.g., “worried” and “fearful”), and (3) anger (e.g., “hate”and “critical”).

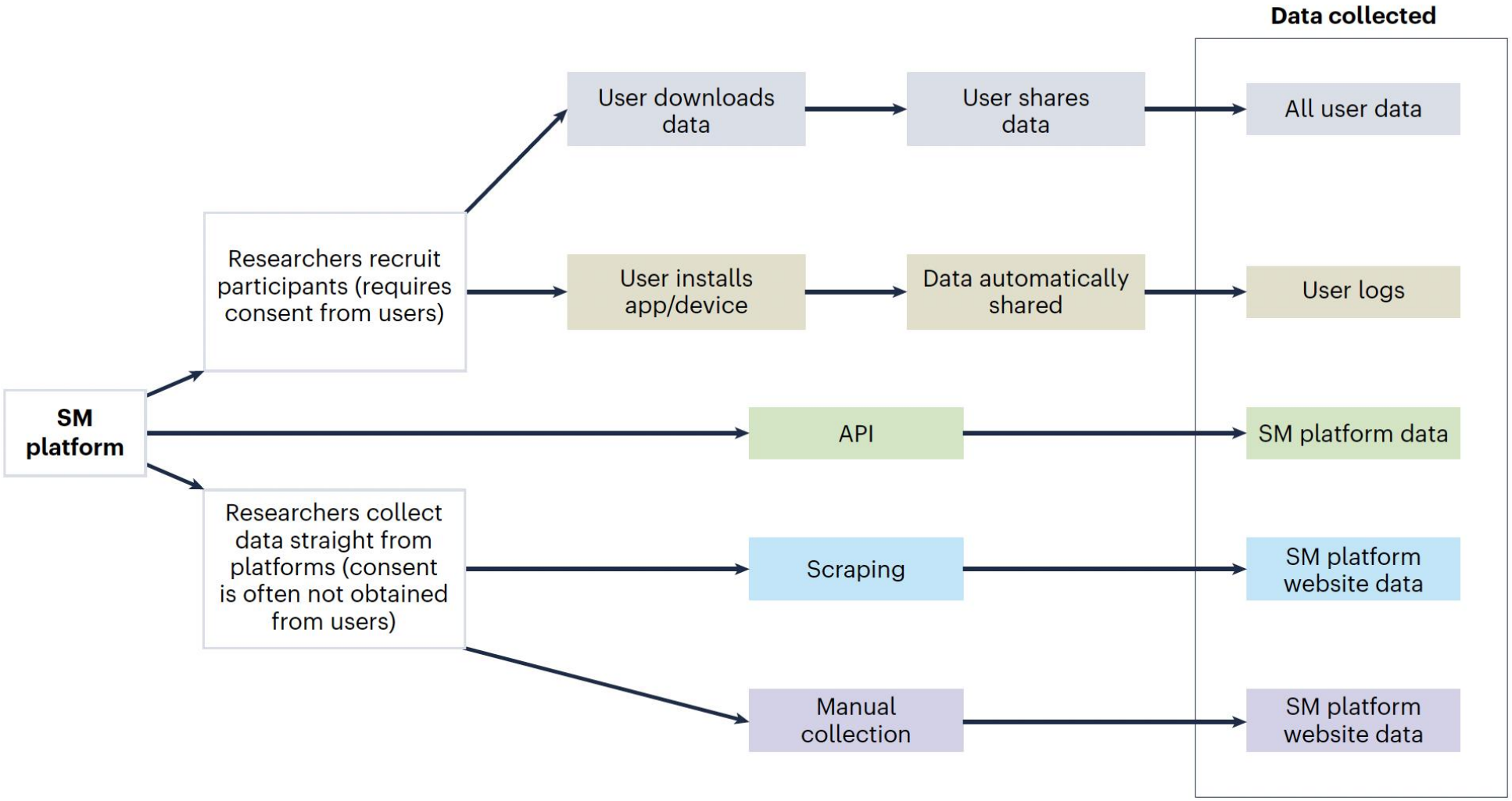


sensitive data

- Digital data contains personally identifiable information
- Raises ethical and legal concerns about privacy
- Example: Location tracking from mobile devices

introduction to data collection

common routes for social media data access

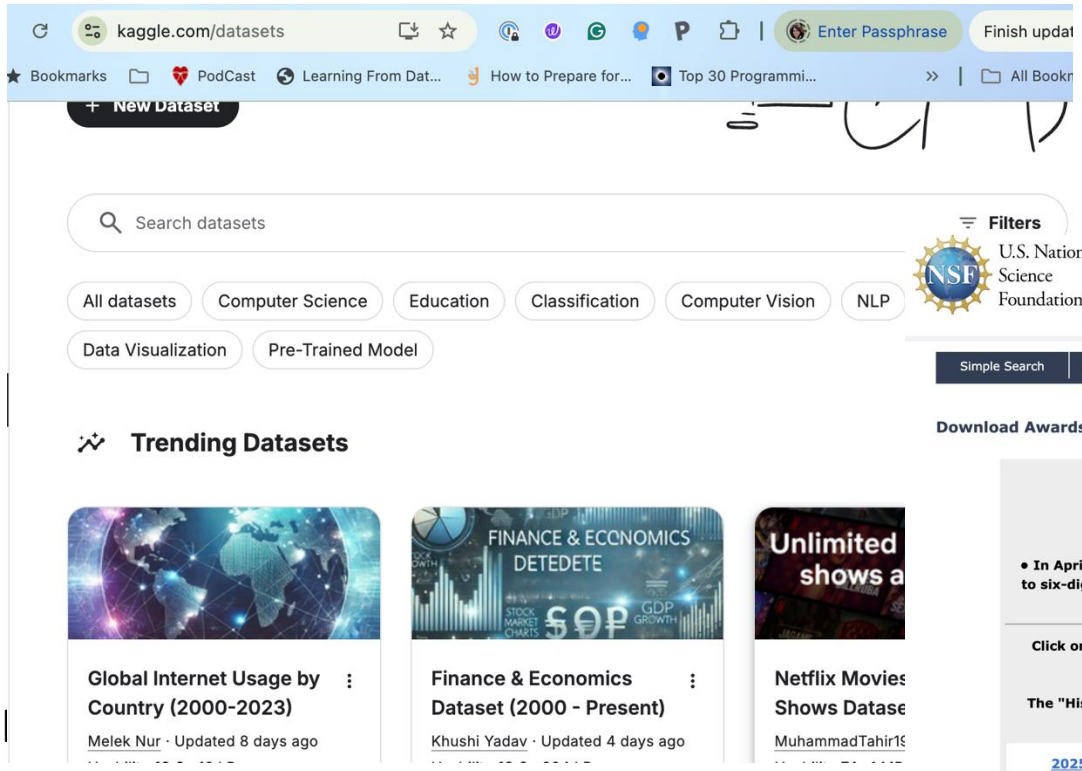


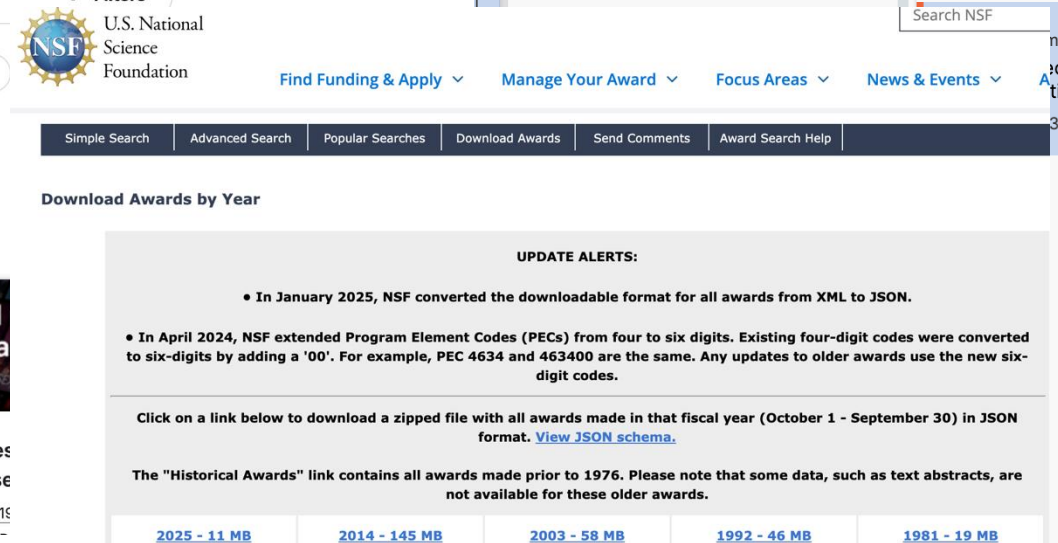
Ayravainen, L., and A. G. Cork. "Platform-controlled social media APIs threaten open science." Nature (2023).

downloading prepared data

- The simplest approach is often to manually go to the web and look for data files or other information.

the American Community Survey (ACS)





scraping information from the web

- Automated collection of web data using scripts

- Used for collecting news articles, job postings, and online discussions.
  - Collecting news articles to analyze media bias.
  - Gathering product prices from e-commerce websites.
  - Scraping social media posts for sentiment analysis on social issues.

Requests Library

- Facilitates sending HTTP requests to interact with web resources.

- Enables actions such as retrieving a webpage.

```
import requests

response = requests.get('https://example.com')

print(response.text)  # Outputs the HTML content of the page
```

Beautiful Soup Library

- Assists in parsing HTML and XML documents to extract specific data.
- Simplifies navigation and searching within parsed documents.

```
from bs4 import BeautifulSoup

html_doc = '<html><head><title>Sample Title</title></head><body></body></html>'

soup = BeautifulSoup(html_doc, 'html.parser')

print(soup.title.string)  # Outputs: Sample Title
```

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:

• The requests.get(url) function fetches the HTML content of the specified URL.

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:
• The requests.get(url) function fetches the HTML content of the specified URL.

2. Parse the HTML:
• BeautifulSoup parses the HTML content, allowing for easy navigation and data extraction.

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:
• The requests.get(url) function fetches the HTML content of the specified URL.

2. Parse the HTML:
• BeautifulSoup parses the HTML content, allowing for easy navigation and data extraction.

3. Extract the Title:
• The <h1> tag typically contains the article's title. The get_text(strip=True) method retrieves the text without leading or trailing whitespace.

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:
• The requests.get(url) function fetches the HTML content of the specified URL.

2. Parse the HTML:
• BeautifulSoup parses the HTML content, allowing for easy navigation and data extraction.

3. Extract the Title:
• The <h1> tag typically contains the article's title. The get_text(strip=True) method retrieves the text without leading or trailing whitespace.

4. Extract the Author:
• The author's name is located within an <a> tag with the attribute data-testid set to 'author-name'.

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:
• The requests.get(url) function fetches the HTML content of the specified URL.

2. Parse the HTML:
• BeautifulSoup parses the HTML content, allowing for easy navigation and data extraction.

3. Extract the Title:
• The <h1> tag typically contains the article's title. The get_text(strip=True) method retrieves the text without leading or trailing whitespace.

4. Extract the Author:
• The author's name is located within an <a> tag with the attribute data-testid set to 'author-name'.

5. Extract the Publication Date:
• The <time> tag contains the publication date in its datetime attribute.

```python
import requests
from bs4 import BeautifulSoup

# URL of the article
url = 'https://www.theverge.com/2024/9/18/24247995/youtube-hype-creators'

# Send a GET request to the URL
response = requests.get(url)
response.raise_for_status()  # Raise an exception for HTTP errors

# Parse the HTML content using BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')

# Extract the article title
title_tag = soup.find('h1')
title = title_tag.get_text(strip=True) if title_tag else 'Title not found'

# Extract the author name
author_tag = soup.find('a', {'data-testid': 'author-name'})
author = author_tag.get_text(strip=True) if author_tag else 'Author not found'

# Extract the publication date
date_tag = soup.find('time')
date = date_tag['datetime'] if date_tag else 'Date not found'

# Extract the article content
article_body = soup.find('div', {'class': 'duet--article--article-body-component'})
if article_body:
    paragraphs = article_body.find_all('p')
    content = '\n'.join([para.get_text(strip=True) for para in paragraphs])
else:
    content = 'Content not found'

# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

1. Send a GET Request:
• The requests.get(url) function fetches the HTML content of the specified URL.

2. Parse the HTML:
• BeautifulSoup parses the HTML content, allowing for easy navigation and data extraction.

3. Extract the Title:
• The <h1> tag typically contains the article's title. The get_text(strip=True) method retrieves the text without leading or trailing whitespace.

4. Extract the Author:
• The author's name is located within an <a> tag with the attribute data-testid set to 'author-name'.

5. Extract the Publication Date:
• The <time> tag contains the publication date in its datetime attribute.

6. Extract the Article Content:
• The article's body is within a tag with the class 'duet--article--article-body-component'. All <p> tags within this div represent paragraphs of the article.

soup

title 13/506


```
<!DOCTYPE html>
<html lang="en-US"><meta content="nocache" name="bingbot"/><head><meta charset="utf-8"/><meta content="summary_la
rge_image" name="twitter:card"/><meta content="@verge" name="twitter:site"/><meta content="549923288395304" prope
rty="fb:app_id"/><meta content="The Verge" property="og:site_name"/><meta content="width=device-width, initial-sc
ale=1, shrink-to-fit=no" name="viewport"/><meta content="Verge" name="apple-mobile-web-app-title"/><meta content
="IucFf_TKtbFFH8_YeFyEteQIwYPdANM1R46_U9DpAr4" name="google-site-verification"/><link href="/rss/index.xml" rel
="alternate" title="The Verge" type="application/rss+xml"/><title>YouTube's new Hype feature is a way to promote
and discover smaller creators | The Verge</title><meta content="index,follow,max-image-preview:large" name="robot
s"/><meta content="Any creator with fewer than 500,000 subscribers can now make it into a Hype leaderboard, which
is designed to help fans help their favorite creators get discovered." name="description"/><meta content="YouTube
Hype gives smaller creators a place to shine" property="og:title"/><meta content="It's a whole new trending syste
m, and a whole new kind of "like" button." property="og:description"/><meta content="https://www.theverge.com/202
4/9/18/24247995/youtube-hype-creators" property="og:url"/><meta content="article" property="og:type"/><meta conte
nt="2024-09-18T14:30:00+00:00" property="article:published_time"/><meta content="2024-09-18T14:30:00+00:00" prope
rty="article:modified_time"/><meta content="https://platform.theverge.com/wp-content/uploads/sites/2/chorus/uploa
ds/chorus_asset/file/25627848/YouTube_Hype.jpg?quality=90&amp;strip=all&amp;crop=0,3.4613147178592,100,93.0773705
64282" property="og:image"/><meta content="A phone running YouTube's app, showing the Hype leaderboard." property
="og:image:alt"/><meta content="image/jpeg" property="og:image:type"/><meta content="1200" property="og:image:wid
th"/><meta content="628" property="og:image:height"/><link href="https://www.theverge.com/2024/9/18/24247995/yout
ube-hype-creators" rel="canonical"/><meta content="David Pierce" name="author"/><meta content="https://www.thever
ge.com/2024/9/18/24247995/youtube-hyp
eators a place to shine" name="twitte
ind of "like" button." name="twitter:
s/sites/2/chorus/uploads/chorus_asset
7178592,100,93.077370564282" name="tw
leaderboard." name="twitter:image:alt
es smaller creators a place to shine"
47995/youtube-hype-creators" name="pa
s/sites/2/chorus/uploads/chorus_asset
7178592,100,93.077370564282" name="pa
b-date"/><meta content="youtube" name
creators,tech,youtube" name="parsely-
```


```python
# Display the extracted information
print(f'Title: {title}\n')
print(f'Author: {author}\n')
print(f'Date: {date}\n')
print('Content:')
print(content)
```

```
Title: YouTube Hype gives smaller creators a place to shine

Author: Author not found

Date: 2024-09-18T14:30:00+00:00

Content:
When a YouTuber hits 500,000 subscribers, the company has found that things tend to change. That half-million mile
stone is something of a tipping point in terms of both growth and revenue, says Bangaly Kaba, a director of produc
t management at YouTube. "We just saw a disproportionate growth in earnings," he says, "even though most of our cr
eators are smaller than this size." Big channels also tend to get more views, which gets them recommended more, wh
ich gets them more views, which gets them more revenue, and around it goes.
```

challenges

- Many websites deliberately make this difficult to prevent easy access to their underlying data
- Some websites explicitly prohibit this type of activity in their terms of use.
  - Legal restrictions (robots.Txt), privacy issues, ethical concerns

- The structure of the websites changes often, requiring researchers to keep updating their code.
  - While the code accurately captures the data from the website at the time of this writing, it may not be valid in the future as the structure and content of the website changes.

API-Based data collection

- Increasingly, organizations are providing Application Programming Interfaces (APIs) to enable scripted and programmatic access to the data they hold.
  - much easier and generally more effective to work with.
- Simply a tool that allows a program to interface with a service.
- APIs can take many different forms and be of varying quality and usefulness.

examples

- Social media platforms
  - TikTok
  - Reddit
  - YouTube

- Science publications
  - Elsevier Article Retrieval API
  - Querying MAG-Like Data from OpenAlex (MAG Successor)
    - Microsoft Academic Graph (MAG) closed as of 2021

examples

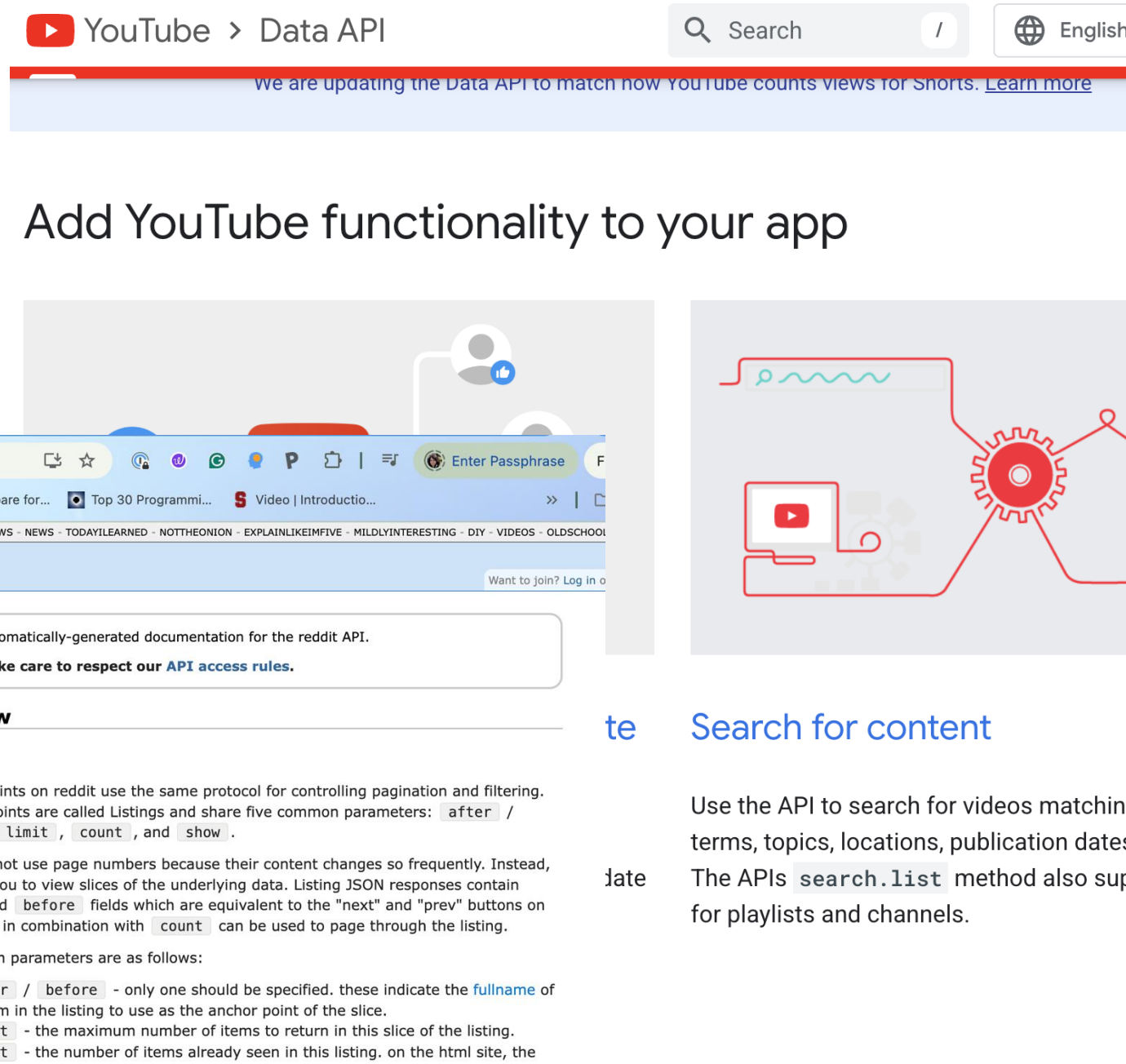
Source	Description	API	Free
	Bibliographic Data		
PubMed	An online index that combines bibliographic data from Medline and PubMed Central. PubMed Central and Europe PubMed Central also provide information.	Y	Y
Web of Science	The bibliographic database provided by Thomson Reuters. The ISI Citation Index is also available.	Y	N
Scopus	The bibliographic database provided by Elsevier. It also provides citation information.	Y	N
Crossref	Provides a range of bibliographic metadata and information obtained from members registering DOIs.	Y	Y
Google Scholar	Provides a search index for scholarly objects and aggregates citation information.	N	Y
Microsoft Academic Search	Provides a search index for scholarly objects and aggregates citation information. Not as complete as Google Scholar, but has an API.	Y	Y

using publication APIs: Science of Science!

- Gender inequality and self-publication are common among academic editors
  - A dataset of 220 million publications and 240 million scientists collected via the Microsoft Academic Graph (MAG) API
  - Editor data from Elsevier article retrieval API

platform-specific features and terms

- Always read the policy and follow the rules!

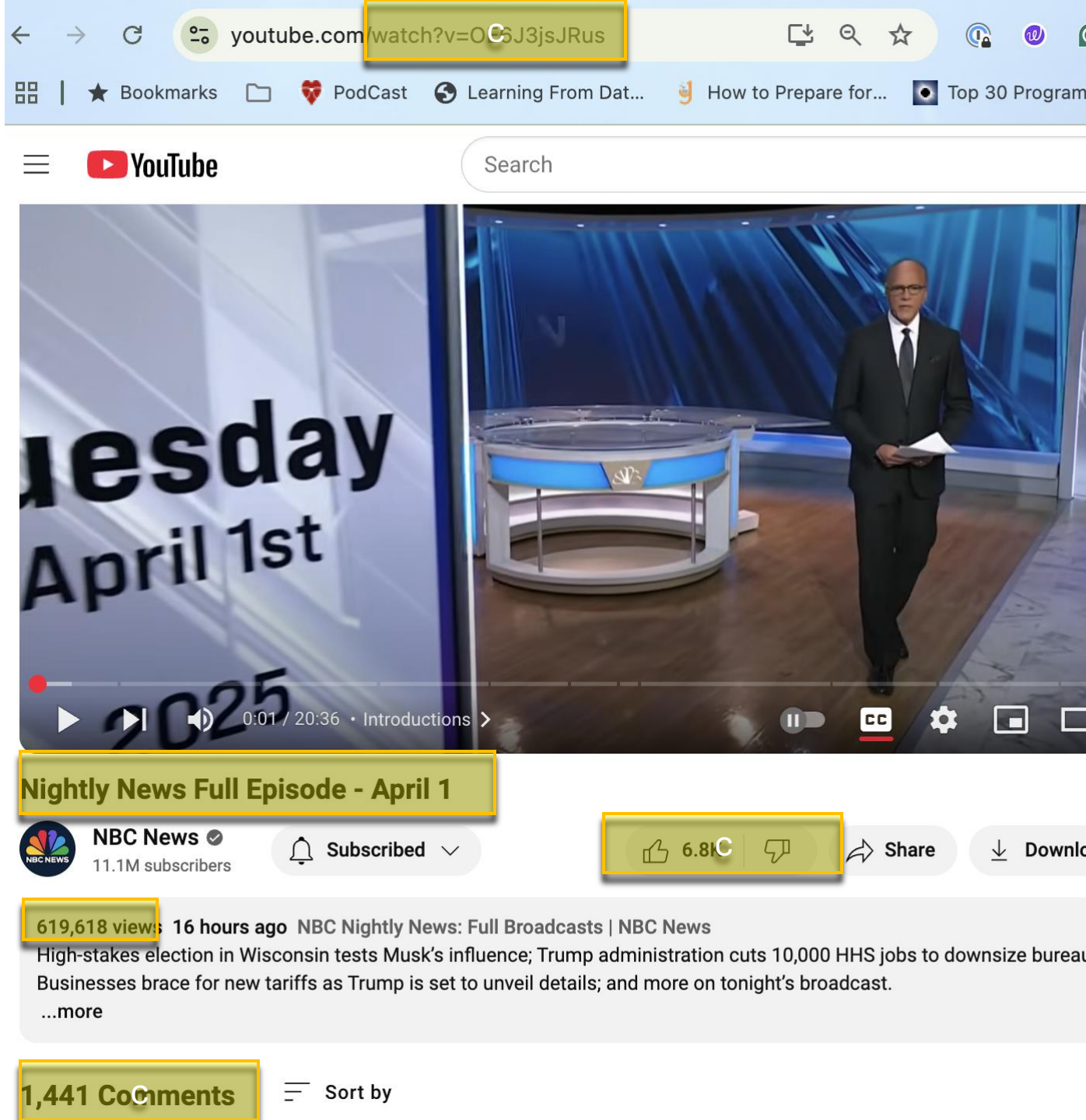


common functions

- GET: used to retrieve data from the server
- POST: send data to the server to create a new resource
- PUT: update an existing resource on the server
- DELETE: delete an existing resource on the server

case study: YouTube

- Title
- Statistics
- Channel
  - Description
- Transcript
- Topic
  - Content type
- Comments



case study: YouTube

Obtain an API Key:

- Visit the Google Cloud Console.
- Create a new project or select an existing one.
- Navigate to “APIs & Services” > “Library”, search for “YouTube Data API v3”, and enable it.
- Go to “APIs & Services” > “Credentials”, click on “Create Credentials”, and choose “API key”.

Install the Required Library:

- Ensure you have the google-api-python-client library installed:

```
pip install google-api-python-client
```

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS                                    #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
   - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS                                    #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
    - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
    - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS                                    #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
    - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
    - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.
3. The build() function initializes the YouTube service object using the API key, specifying API name 'youtube' and version 'v3'.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS[redacted] #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
    - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
    - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.
3. The build() function initializes the YouTube service object using the API key, specifying API name 'youtube' and version 'v3'.
4. Specify the target video
    - The video_id variable (e.g., 'OE6J3jsJRus') holds the unique ID of the YouTube video whose data you want to retrieve.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS[redacted]"  #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired

# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
   - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
   - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.
3. The build() function initializes the YouTube service object using the API key, specifying API name 'youtube' and version 'v3'.
4. Specify the target video
   - The video_id variable (e.g., 'OE6J3jsJRus') holds the unique ID of the YouTube video whose data you want to retrieve.
5. Construct the API request
   - part='snippet,statistics' requests both descriptive metadata (title, channel info, etc.) and numerical data (view counts, likes).
   - id=video_id indicates the specific video to query.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS                                    #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
    - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
    - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.
3. The build() function initializes the YouTube service object using the API key, specifying API name 'youtube' and version 'v3'.
4. Specify the target video
    - The video_id variable (e.g., 'OE6J3jsJRus') holds the unique ID of the YouTube video whose data you want to retrieve.
5. Construct the API request
    - part='snippet,statistics' requests both descriptive metadata (title, channel info, etc.) and numerical data (view counts, likes).
    - id=video_id indicates the specific video to query.
6. Send the request to the API
    - execute() sends the request and retrieves the video data in a structured JSON response.

```python
from googleapiclient.discovery import build

# Replace 'YOUR_API_KEY' with your actual API key
api_key = "AIzaS[redacted]"  #

# Initialize the YouTube service object
youtube = build('youtube', 'v3', developerKey=api_key)

# Input video
video_id = 'OE6J3jsJRus'  # Replace with your desired


# Set the parameters for information you wanna extract
request = youtube.videos().list(
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()

# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']

print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

1. Import necessary module
   - from googleapiclient.discovery import build loads the Google API client library, which allows Python to communicate with YouTube's Data API.
2. Set up authentication
   - An API key (e.g., api_key = "...") is required to access the YouTube Data API and must be kept secure.
3. The build() function initializes the YouTube service object using the API key, specifying API name 'youtube' and version 'v3'.
4. Specify the target video
   - The video_id variable (e.g., 'OE6J3jsJRus') holds the unique ID of the YouTube video whose data you want to retrieve.
5. Construct the API request
   - part='snippet,statistics' requests both descriptive metadata (title, channel info, etc.) and numerical data (view counts, likes).
   - id=video_id indicates the specific video to query.
6. Send the request to the API
   - execute() sends the request and retrieves the video data in a structured JSON response.
7. Extract relevant data from the response
   - response['items'][0] accesses the video object from the list of results.

case study: YouTube

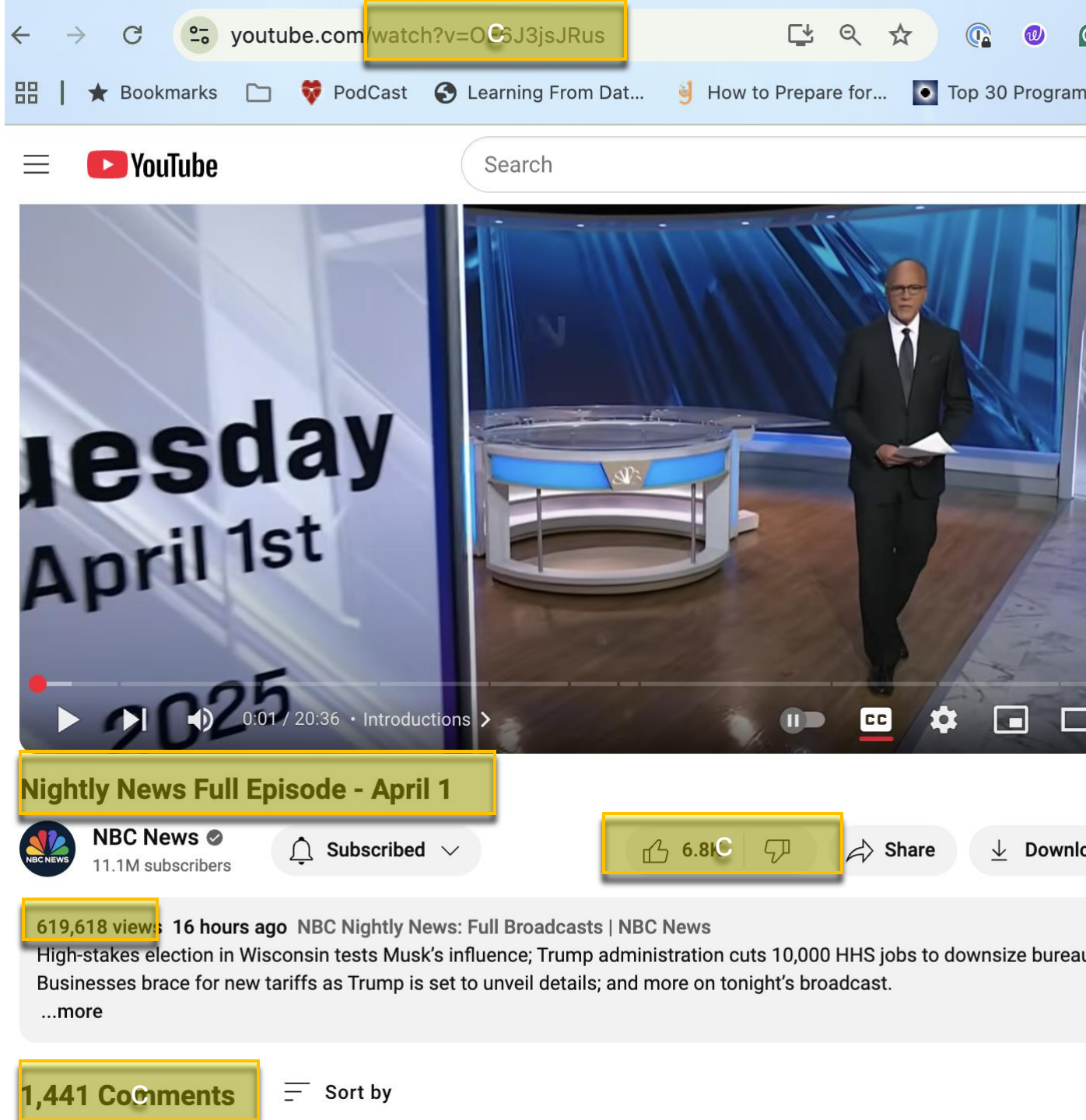
```
    part='snippet,statistics',
    id=video_id
)

# Send the request
response = request.execute()


# Extract information
video = response['items'][0]
title = video['snippet']['title']
statistics = video['statistics']


print(f"Title: {title}")
print("Statistics:")
for key, value in statistics.items():
    print(f"  {key}: {value}")
```

```
Title: Nightly News Full Episode - April 1
Statistics:
  viewCount: 617169
  likeCount: 6798
  favoriteCount: 0
  commentCount: 1432
```



case study: Reddit

python

```python
import praw

# Initialize the Reddit instance
reddit = praw.Reddit(
    client_id='YOUR_CLIENT_ID',         # Replace with your Client ID
    client_secret='YOUR_CLIENT_SECRET',# Replace with your Client Secret
    user_agent='YOUR_APP_NAME'          # Replace with your application's name
)

# Choose the subreddit you want to interact with
subreddit = reddit.subreddit('learnpython')

# Retrieve the top 5 hot posts from the subreddit
for post in subreddit.hot(limit=5):
    print(f"Title: {post.title}")
    print(f"Score: {post.score}")
    print(f"URL: {post.url}\n")
```

limitations in access

- Each part requested has an associated *quota* cost.
  - quota is a specific limit or allocation of a resource that is officially allowed

- You have to be mindful of this when selecting parts to avoid exceeding your quota limits.

using automated users

- Programmatic users who mimic real users browsing behavior
- Very popular on auditing applications:
  - Systematic statistical probing of an online platform to uncover societally problematic behavior underlying its algorithms
- Examples of studies include:
  - Presence of partisan bias in search engine components
  - Investigated representativeness issues, such as racial and gender bias in online freelance marketplaces

examples

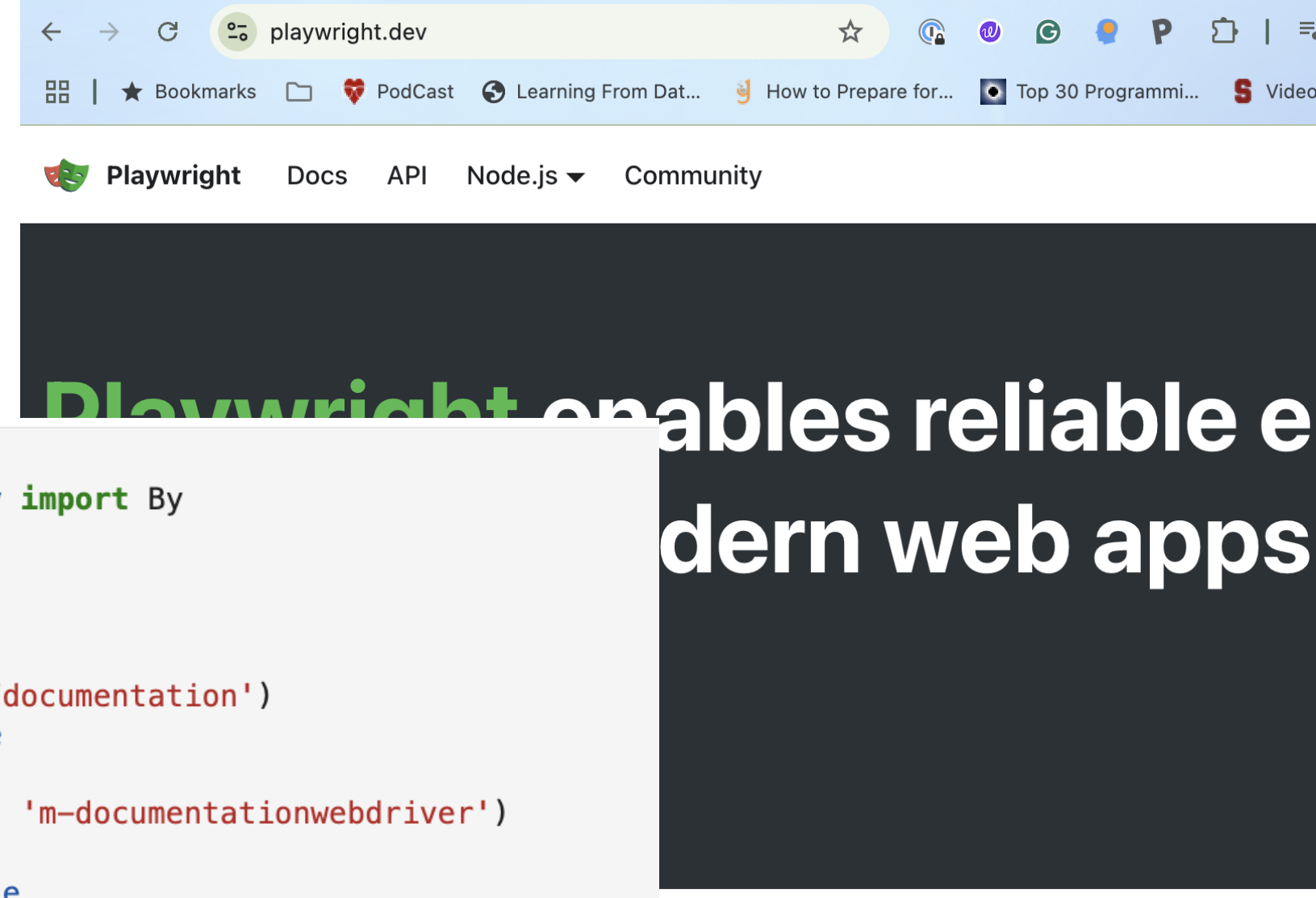
```python
from selenium import webdriver
from selenium.webdriver.common.by import By


driver = webdriver.Chrome()

driver.get('https://selenium.dev/documentation')
assert 'Selenium' in driver.title

elem = driver.find_element(By.ID, 'm-documentationwebdriver')
elem.click()
assert 'WebDriver' in driver.title

time.sleep(4)
driver.quit()
```



- Auditing E-Commerce Platforms for Algorithmically Curated Vaccine Misinformation [1]

  Data available: https://social-comp.github.io/AmazonAudit-data/

- Measuring Misinformation in Video Search Platforms: An Audit Study on YouTube [2]
  - Created 150 Google accounts
  - Tested over four topics: Chemtrails conspiracy, Flat Earth, Moon landing, and Vaccine misinformative

[1] Juneja, Prerna, and Tanushree Mitra. "Auditing e-commerce platforms for algorithmically curated vaccine misinformation." Proceedings of the 2021 chi conference on human factors in computing systems. 2021.
[2] Juneja, Prerna, and Tanushree Mitra. "Auditing e-commerce platforms for algorithmically curated vaccine misinformation." In Proceedings of the 2021 chi conference on human factors in computing systems, pp. 1-27. 2021.

data donation

- Individuals voluntarily share their personal data for research.
  - User Engagement with TikTok's Short Format Video Recommendations using Data Donations [1]
  - Instagram Data Donation: A Case Study on Collecting Ecologically Valid Social Media Data for the Purpose of Adolescent Online Risk Detection [2]

[1] Zannettou, Savvas, et al. "Analyzing User Engagement with TikTok's Short Format Video Recommendations using Data Donations." Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. 2024.
[2] Razi, Afsaneh, et al. "Instagram data donation: a case study on collecting ecologically valid social media data for the purpose of adolescent online risk detection." CHI Conference on Human Factors in Computing Systems Extended Abstracts. 2022.

Survey Measures:

- Social Media Use
- Negative Online Experiences
- Personal Experiences and Demographics

2.1 Data Collection Design and Approach

Figure 1: Instagram Data Donation Main Page

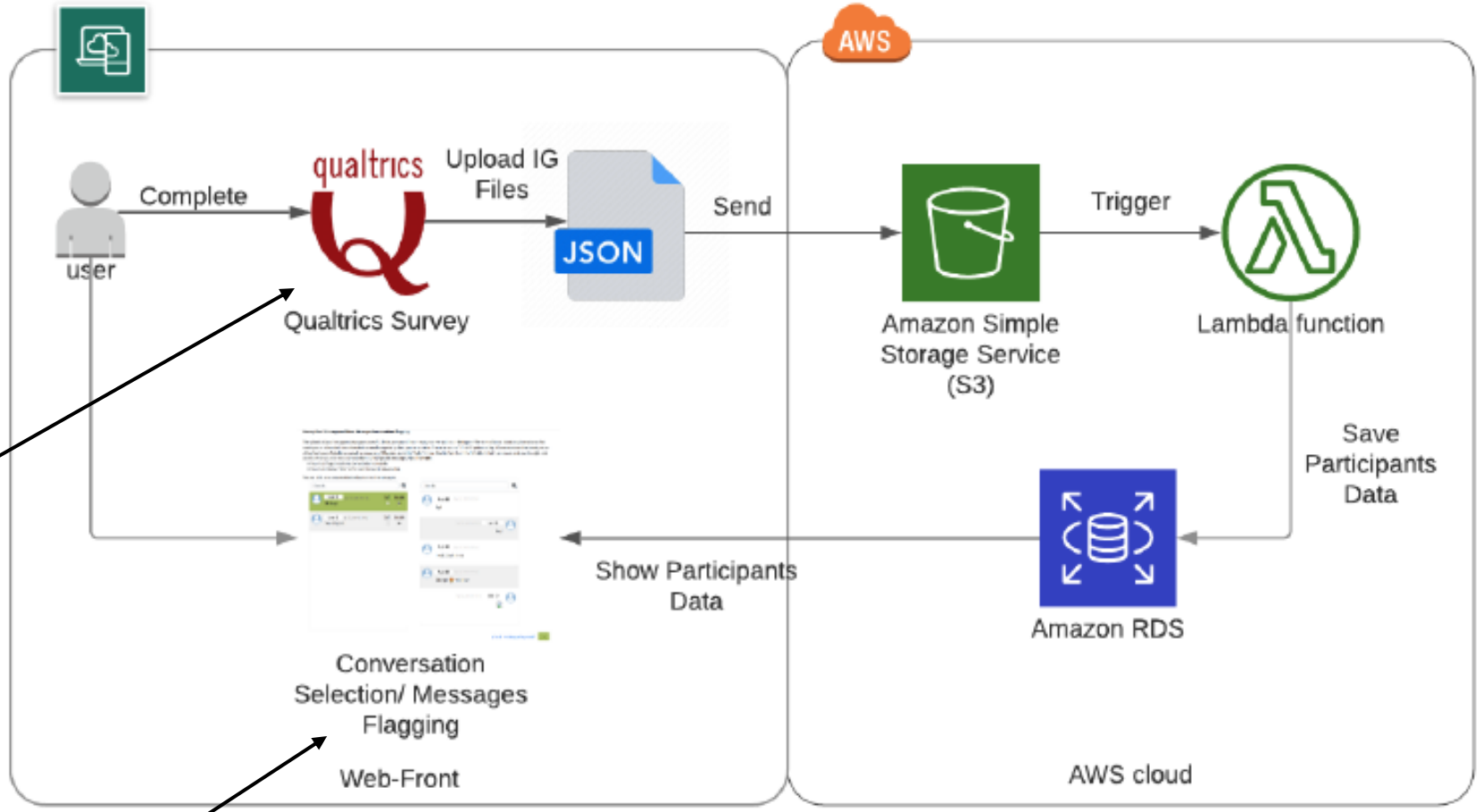


Figure 3: Instagram Data Donation System Architecture.

Ground Truth Annotations by Participants

data characteristics

- Small-scale but rich data
  - 347 TikTok users & collected 9.2M TikTok video recommendations
  - 150 Instagram users & over 5 million messages on Instagram

- User engagement & algorithm role on TikTok:
  - TikTok shows more videos from non-followed accounts
  - Only 10% of viewed videos came from followed accounts
  - Users followed more accounts over time, but their exposure to followed content remained flat
  - This likely reflects algorithmic design to prioritize popular, attention-grabbing videos, not personal networks.

- Online safety on Instagram:
  - Over 5 million messages on Instagram
  - 60% of the messages were flagged as low, 26% as medium, and 14% as high risk levels
  - Survey of mental health, social media use, and personal risk experiences
  - Annotate direct message (DM) conversations as “safe” or “unsafe”

panel data

- Longitudinal data tracking the same individuals over time.
- Helps understand trends, causality, and behavioral changes.
- Example: Studying how media consumption habits change over a decade.

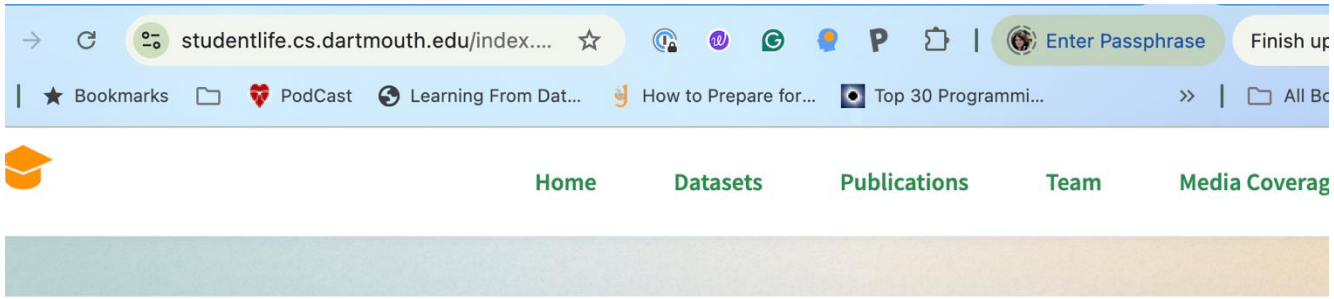
Horta Ribeiro, Manoel, Homa Hosseinmardi, Robert West, and Duncan J. Watts. "Deplatforming did not decrease Parler users' activity on fringe social media." PNAS nexus 2, no. 3 (2023): pgad035.

more on sensor data

- GPS and Mobile Sensor Data
  - Smartphones collect GPS location, accelerometer, and movement data.
  - Used in mobility studies, public health tracking, and disaster response.
  - Example: Tracking human mobility patterns during COVID-19.

- Wearable Data
  - Devices like Apple Watch and Fitbit collect health and activity metrics.
  - Used for sleep analysis, heart rate monitoring, and fitness studies.
  - Example: Wearable data predicting early signs of heart disease.

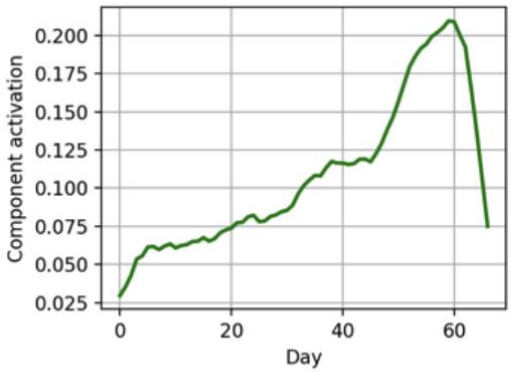
sensory data



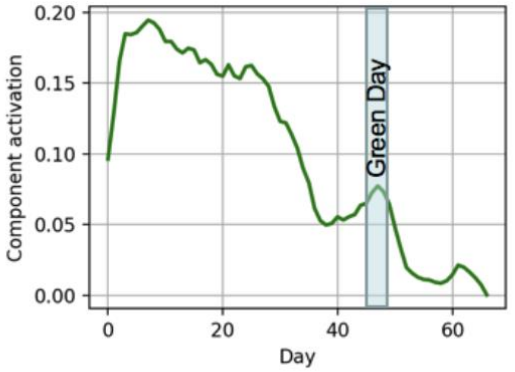
StudentLife Study

The StudentLife study at Dartmouth College is driven by the idea that smartphones, wearables, machine learning, and AI technology will bridge the gap between the skyrocketing demand for mental health support on our campuses and the limited access to healthcare professionals. Moreover, we believe AI will revolutionize diagnosing, managing, and treating mental illness at scale, on our campus and worldwide.

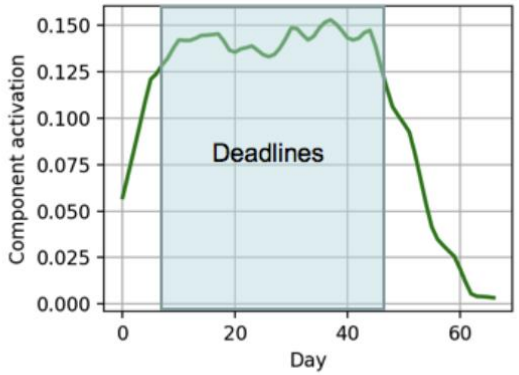
- bed time, wake up time and sleep duration
- the number of conversations and duration of each conversation per day
- physical activity (**walking, sitting, running, standing**)
- where they were located and how long they stayed there (i.e., dorm, class, party, gym)
- the number of people around a student through the day
- outdoor and indoor (in campus buildings) mobility
- **stress level** through the day, across the week and term
- **positive affect** (how good they felt about themselves)
- eating habits (where and when they ate)
- app usage



(a) 1^{st} component: Studying pattern.

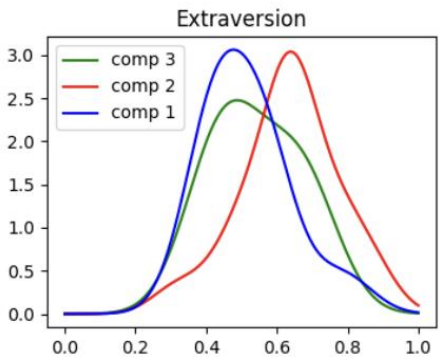


(b) 2^{nd} component: Partying pattern.

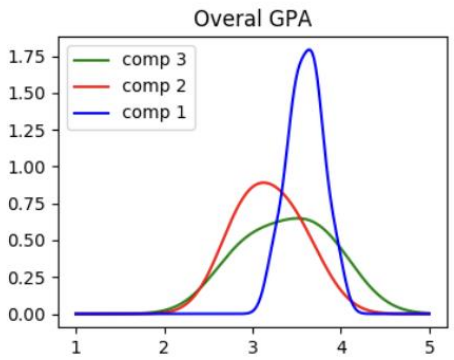


(c) 3^{rd} component: Deadline pattern.

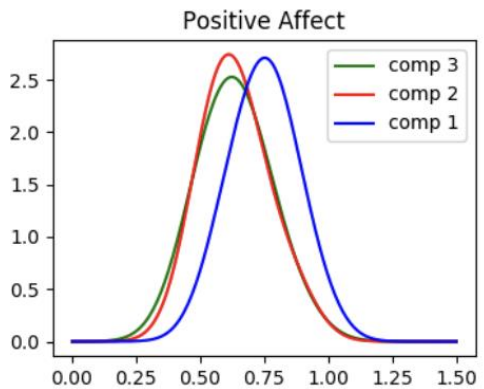
Patterns of behavior from physical activity



(a)



(b)



(c)

Joining metadata with activity data

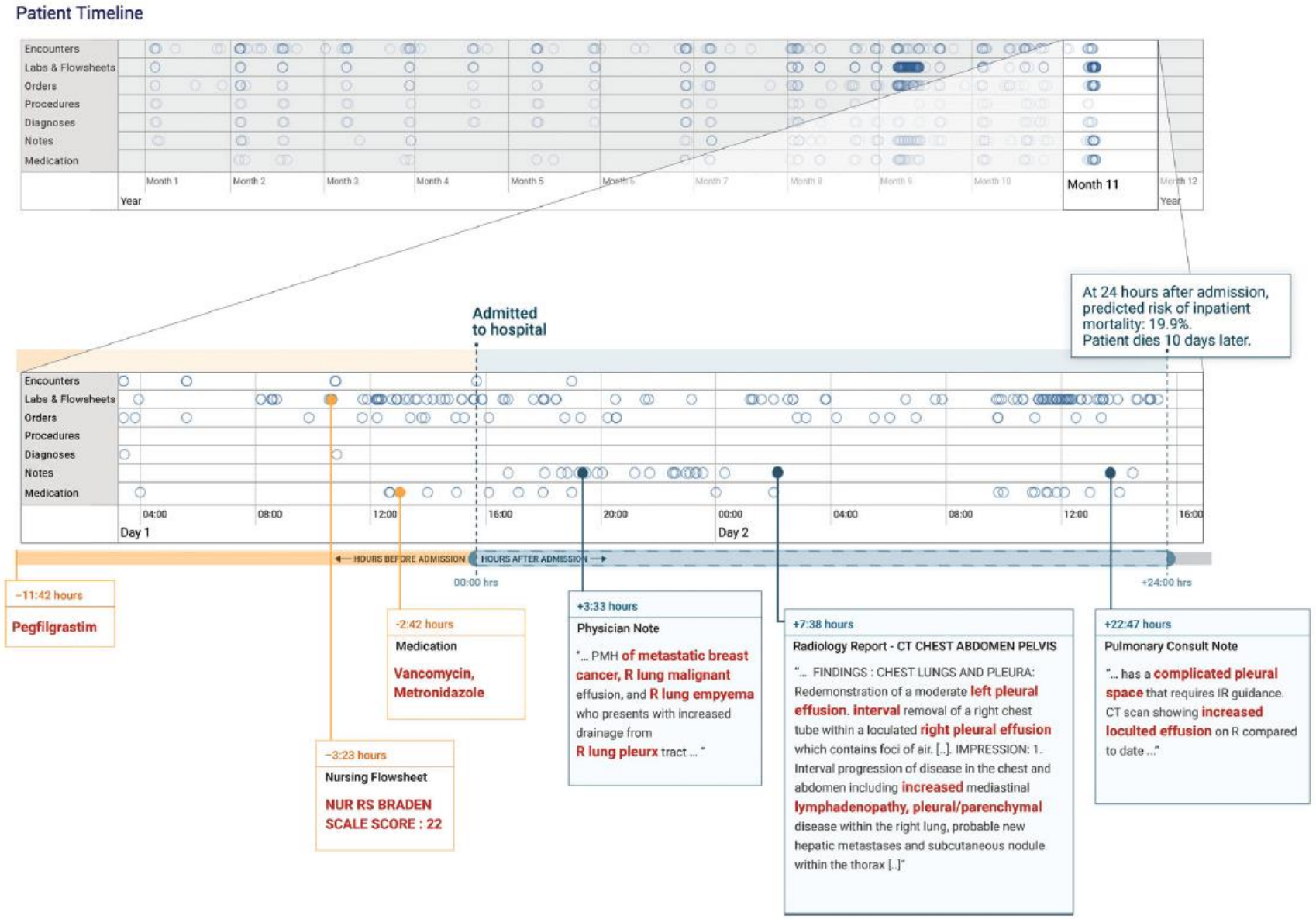
societal issues: inequality & public health

- Data: smartphone accelerometers
  - 717,527 people
  - 111 countries across the globe
  - 68 million days of physical activity
- Gender Gap:
  - In countries with high inequality, women are disproportionately less active, accounting for much of the inequality.
- Environment Matters:
  - Cities with higher walkability scores (e.g., San Francisco) had lower activity inequality
- Policy Implication:
  - Simulations show that targeting the most inactive people (inequality-focused interventions) would reduce obesity up to 4x more than population-wide efforts.

Althoff, Tim, Rok Sosič, Jennifer L. Hicks, Abby C. King, Scott L. Delp, and Jure Leskovec. "Large-scale physical activity data reveal worldwide activity inequality." Nature 547, no. 7663 (2017): 336-339.

Electronic Health Records (EHRs)

- Digital patient records from hospitals and clinics.
- Contains medical history, lab results, and prescriptions.



Rajkomar, Alvin, Eyal Oren, Kai Chen, Andrew M. Dai, Nissan Hajaj, Michaela Hardt, Peter J. Liu et al. "Scalable and accurate deep learning with electronic health records." NPJ digital medicine 1, no. 1 (2018): 18.

And many more

- Survey data
  - Data collected via structured questionnaires
  - Can be conducted via phone, online, or in-person interviews
- Government data: public datasets released by governments.
  - Includes census data, crime reports, and economic indicators.
    - Using open government data to study income inequality.
- Transaction & financial data
  - Identity theft, payment fraud
- Satellite & remote sensing data
  - Agricultural productivity using satellite images, climate change, traffic control

Conclusion: Choosing the Right Data Source

- Each data source has strengths and weaknesses.
- The choice depends on the research question, ethical considerations, and data access.
- Combining multiple data sources often leads to better insights.

transaction & financial data

- Banking and credit card transaction records.
- Used in economic behavior research, fraud detection, and financial inclusion studies.
- Example:
  - Identity theft, Payment fraud (e.g., card-not-present, wire transfer scams), Phishing, malware

Udeh, Ezekiel Onyekachukwu, et al. "The role of big data in detecting and preventing financial fraud in digital transactions." World Journal of Advanced Research and Reviews 22.2 (2024): 1746-1760.

satellite & remote sensing data

- Satellite imagery and geospatial data for environmental monitoring.
- Used in deforestation tracking, urban development studies, and disaster response.
- Example:
  - Monitoring agricultural productivity using satellite images.
  - Climate change
  - Traffic control

biometric data

- Collected through fingerprint scanners, facial recognition, and retinal scans.
- Used in security, personal identification, and health monitoring.
- Example: Facial recognition for access control and law enforcement applications.