

30538 Problem Set 4: Web Scraping

Yuxin Zheng

2026-02-05

Due 02/07 at 5:00PM Central.

“This submission is my work alone and complies with the 30538 integrity policy.” Add your initials to indicate your agreement: YZ

Github Classroom Assignment Setup and Submission Instructions

1. Accepting and Setting up the PS4 Assignment Repository

- Each student must individually accept the repository for the problem set from Github Classroom (“ps4”) – <https://classroom.github.com/a/hWhcHqH>
 - You will be prompted to select your cnetid from the list in order to link your Github account to your cnetid.
 - If you can’t find your cnetid in the link above, click “continue to next step” and accept the assignment, then add your name, cnetid, and Github account to this Google Sheet and we will manually link it: <https://rb.gy/9u7fb6>
- If you authenticated and linked your Github account to your device, you should be able to clone your PS4 assignment repository locally.
- Contents of PS4 assignment repository:
 - `ps4_template.qmd`: this is the Quarto file with the template for the problem set. You will write your answers to the problem set here.

2. Submission Process:

- Knit your completed solution `ps4.qmd` as a pdf `ps4.pdf`.
 - Your submission does not need runnable code. Instead, you will tell us either what code you ran or what output you got.
- To submit, push `ps4.qmd` and `ps4.pdf` to your PS4 assignment repository. Confirm on Github.com that your work was successfully pushed.

Grading

- You will be graded on what was last pushed to your PS4 assignment repository before the assignment deadline
- Problem sets will be graded for completion as: {missing (0%); - (incomplete, 50%); + (excellent, 100%)}
 - The percent values assigned to each problem denote how long we estimate the problem will take as a share of total time spent on the problem set, not the points they are associated with.
- In order for your submission to be considered complete, you need to push both your `ps4.qmd` and `ps4.pdf` to your repository. Submissions that do not include both files will automatically receive 50% credit.

```

import pandas as pd
import altair as alt
import time

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")

```

```

RendererRegistry.enable('png')

```

Step 1: Develop initial scraper and crawler

```

import re
from urllib.parse import urljoin

import requests
import pandas as pd
from bs4 import BeautifulSoup

# URL for the first page of enforcement actions
LIST_URL = "https://oig.hhs.gov/fraud/enforcement/"
BASE_URL = "https://oig.hhs.gov"

# Add a User-Agent to avoid being blocked
HEADERS = {
    "User-Agent": "DAP30538-ps4-scraper/1.0 (contact: student@uchicago.edu)"
}

# Request the enforcement actions page
response = requests.get(LIST_URL, headers=HEADERS, timeout=30)
response.raise_for_status()

# IMPORTANT: use html.parser to avoid lxml dependency issues
soup = BeautifulSoup(response.text, "html.parser")

# Restrict scraping to the main content area
main = soup.find("main") or soup

rows = []

# Loop through potential containers on the page

```

```

for container in main.find_all(["article", "li", "div"]):

    # Extract title and link from anchor tag
    a = container.find("a", href=True)
    if not a:
        continue

    title = a.get_text(" ", strip=True)
    link = urljoin(BASE_URL, a["href"])

    # Only keep links that point to enforcement action detail pages
    if not link.startswith(LIST_URL) or link == LIST_URL:
        continue

    # Extract date (either from a <time> tag or from surrounding text)
    date = None
    time_tag = container.find("time")
    if time_tag:
        date = time_tag.get_text(" ", strip=True)
    else:
        text = container.get_text(" ", strip=True)
        match = re.search(
            r"(January|February|March|April|May|June|July|August|September|October|November|December)",
            text,
        )
        if match:
            date = match.group(0)

    # Extract category label if present on the list page
    category = None
    for cat in ["Criminal and Civil Actions", "State Enforcement Agencies"]:
        if cat in container.get_text(" ", strip=True):
            category = cat
            break

    # Append result as one observation
    rows.append({
        "title": title,
        "date": date,
        "category": category,
        "link": link
    })

```

```
# Create tidy dataframe and remove duplicates
df = (
    pd.DataFrame(rows)
    .drop_duplicates(subset=["title", "link"])
    .reset_index(drop=True)
)

# Remove non-action informational pages (no date)
df = df[df["date"].notna()].reset_index(drop=True)

# Print the head of the dataframe
df.head()
```

	title	date	category	link
0	Houston Transplant Doctor Indicted For Making ...	February 5, 2026	Criminal and Civil Actions	htt
1	MultiCare Health System to Pay Millions to Set...	February 4, 2026	Criminal and Civil Actions	htt
2	Brooklyn Banker Pleads Guilty to Laundering Pr...	February 3, 2026	None	htt
3	Delafield Man Sentenced to 18 Months' Imprison...	February 3, 2026	Criminal and Civil Actions	htt
4	Former NFL Player Convicted for \$197M Medicare...	February 3, 2026	Criminal and Civil Actions	htt

I scraped the first page of the HHS Office of Inspector General’s “Enforcement Actions” web-page and collected four variables for each enforcement action: the title of the enforcement action, the posted date, the category shown on the list page (e.g., “Criminal and Civil Actions”), and the hyperlink associated with the enforcement action.

Each enforcement action is treated as one observation. I stored the scraped results in a tidy pandas dataframe, where each row corresponds to a single enforcement action and each column corresponds to one variable. I then printed the head of the dataframe to verify that the scraper worked as intended.

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code
 1. Define the function and a run indicator
 - Define `scrape_enforcement_actions(year, month, RUN_SCRAPER)`.
 - If `RUN_SCRAPER` is `False`, exit without running the scraper.

- Rationale: this prevents the scraper from running every time the `.qmd` file is knitted.
2. Check input validity
 - If `year < 2013`:
 - Print a message reminding the user to restrict inputs to `year >= 2013` (since only actions after 2013 are listed).
 - Exit the function without scraping.
 3. Initialize key variables
 - Set `start_date` to the first day of the input month and year (e.g., `YYYY-MM-01`).
 - Set `base_url` to the HHS OIG enforcement actions page.
 - Initialize an empty list `rows = []` to store scraped enforcement actions.
 - Initialize a page counter (e.g., `page = 1`).
 - Initialize a Boolean flag (e.g., `keep_scraping = True`).
 4. Loop through pages using a while loop
 - Use a `while keep_scraping` loop rather than a fixed `for` loop, since the number of pages is unknown in advance.
 - For each iteration of the loop:
 - (a) Construct the URL for the current page (based on the site’s pagination pattern).
 - (b) Send an HTTP request and parse the HTML content.
 - (c) Identify all enforcement-action entries on the page.
 - (d) For each enforcement action on the page:
 - (i) Extract the `title`, `date`, `category`, and `link`.
 - (ii) Convert `date` into a comparable date object.
 - (iii) If the action date is **on or after** `start_date`, store it in `rows`.
 - (iv) If the action date is **earlier than** `start_date`, set `keep_scraping = False` and stop scraping.
 - (e) If no enforcement actions are found on the page, set `keep_scraping = False`.
 5. Advance pagination responsibly
 - If `keep_scraping` is still `True`:
 - Increment the page counter (e.g., `page = page + 1`).
 - Pause execution for 1 second using `time.sleep(1)` to avoid potential server-side blocking.
 6. Finalize output
 - Convert `rows` into a tidy pandas DataFrame.

- Save the DataFrame as `enforcement_actions_year_month.csv` (do not commit this file to Git).
 - Return the DataFrame for use in later parts of the problem set.
- b. Create Dynamic Scraper

```
import re
import time
from datetime import datetime, date
from urllib.parse import urljoin

import requests
import pandas as pd
from bs4 import BeautifulSoup

# Constants
BASE_URL = "https://oig.hhs.gov"
LIST_URL = "https://oig.hhs.gov/fraud/enforcement/"
OUTFILE = "enforcement_actions_year_month.csv"

HEADERS = {
    "User-Agent": "DAP30538-ps4-scraper/1.0"
}

# Helpers
def _parse_date_from_text(text: str):
    """Extract dates like 'January 3, 2024' from text."""
    m = re.search(
        ↪ r"(January|February|March|April|May|June|July|August|September|October|November|December) \d+, \d{4}"
        text,
    )
    if not m:
        return None
    return datetime.strptime(m.group(0), "%B %d, %Y").date()

def _get_page_url(page: int) -> str:
    """Page 1 is the base URL; page >= 2 uses ?page=N."""
    if page <= 1:
        return LIST_URL
    return f"{LIST_URL}?page={page}"

def _extract_actions_from_page(html: str) -> pd.DataFrame:
```

```

"""Extract enforcement actions from ONE list page."""
soup = BeautifulSoup(html, "html.parser")
main = soup.find("main") or soup

rows = []

for container in main.find_all(["article", "li", "div"]):
    a = container.find("a", href=True)
    if not a:
        continue

    title = a.get_text(" ", strip=True)
    link = urljoin(BASE_URL, a["href"].strip())

    # Only keep enforcement detail pages (exclude list page itself)
    if not link.startswith(LIST_URL) or link == LIST_URL:
        continue

    # ---- Date extraction: <time> -> list text -> detail page fallback
    ↪ ----
    dt = None
    date_str = None

    time_tag = container.find("time")
    if time_tag:
        date_str = time_tag.get_text(" ", strip=True)
        try:
            dt = datetime.strptime(date_str, "%B %d, %Y").date()
        except Exception:
            dt = _parse_date_from_text(container.get_text(" ",
↪ strip=True))
        else:
            dt = _parse_date_from_text(container.get_text(" ", strip=True))

    # FINAL fallback: if still missing, fetch the detail page
    if dt is None:
        try:
            detail_resp = requests.get(link, headers=HEADERS, timeout=30)
            detail_resp.raise_for_status()
            detail_soup = BeautifulSoup(detail_resp.text, "html.parser")
            dt = _parse_date_from_text(detail_soup.get_text(" ",
↪ strip=True))
        except Exception:

```

```

        dt = None

    # ---- Category extraction (best-effort from list card text) ----
    category = None
    card_text = container.get_text(" ", strip=True)
    for cat in ["Criminal and Civil Actions", "State Enforcement
↪ Agencies", "Exclusions"]:
        if cat in card_text:
            category = cat
            break

    # Keep only rows with a valid parsed date
    if dt is not None:
        rows.append({
            "title": title,
            "date": dt.strftime("%B %d, %Y"),
            "date_dt": dt,          # date object for comparisons
            "category": category,
            "link": link,
        })

    df = pd.DataFrame(rows).drop_duplicates(subset=["title",
↪ "link"]).reset_index(drop=True)
    return df

# Main dynamic scraper
def scrape_enforcement_actions(year: int, month: int, RUN_SCRAPER: bool =
↪ False) -> pd.DataFrame | None:
    """
    Scrape enforcement actions from (year, month) up to today.
    Saves output to OUTFILE.
    """
    if not RUN_SCRAPER:
        print("RUN_SCRAPER is False - skipping scraping.")
        return None

    if year < 2013:
        print("Please restrict to year >= 2013 (only enforcement actions
↪ after 2013 are listed).")
        return None

    start_date = date(year, month, 1)

```

```

all_rows = []
page = 1
keep_scraping = True

while keep_scraping:
    url = _get_page_url(page)
    resp = requests.get(url, headers=HEADERS, timeout=30)
    resp.raise_for_status()

    page_df = _extract_actions_from_page(resp.text)

    if page_df.empty:
        break

    # List is reverse-chronological; sort newest -> oldest
    page_df = page_df.sort_values("date_dt", ascending=False)

    for _, row in page_df.iterrows():
        if row["date_dt"] >= start_date:
            all_rows.append(row.to_dict())
        else:
            keep_scraping = False
            break

    if keep_scraping:
        page += 1
        time.sleep(1) # required delay between pages

df = pd.DataFrame(all_rows).drop_duplicates(subset=["title",
↪ "link"]).reset_index(drop=True)

# Save for reuse when knitting (do NOT commit to git)
df.to_csv(OUTFILE, index=False)
return df

# Step 2(b): Starting from January 2024
# IMPORTANT: do NOT scrape when knitting; reuse the CSV generated in Step
↪ 2(c)
OUTFILE = "enforcement_actions_year_month.csv" # this is the file you
↪ already have
df_all = pd.read_csv(OUTFILE)
df_all["date_dt"] = pd.to_datetime(df_all["date"]).dt.date

```

```
# Filter to actions since Jan 2024 for the Step 2(b) answer
start_2024 = pd.to_datetime("2024-01-01").date()
df_2024 = df_all[df_all["date_dt"] >= start_2024].copy()

n_actions_2024 = len(df_2024)
earliest_2024 = df_2024.sort_values("date_dt").iloc[0]

print(n_actions_2024)
earliest_2024["date"], earliest_2024["title"], earliest_2024["category"],
↳ earliest_2024["link"]
```

1787

```
('January 03, 2024',
 'Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In
 Georgia',
 'State Enforcement Agencies',
```

```
'https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-p
```

Running my dynamic scraper starting from January 2024 produced 1,787 enforcement actions in the final dataframe.

The earliest enforcement action scraped was dated January 3, 2024 and titled “Former Nurse Aide Indicted in Death of Clarksville Patient Arrested in Georgia.” This action falls under the category State Enforcement Agencies and links to the official HHS OIG enforcement page.

- c. Test Your Code

```
# Step 2(c): Starting from January 2022 (use this dataframe for all later
↳ questions)
OUTFILE = "enforcement_actions_year_month.csv"
df_2022 = pd.read_csv(OUTFILE)
df_2022["date_dt"] = pd.to_datetime(df_2022["date"]).dt.date

n_actions_2022 = len(df_2022)
earliest_2022 = df_2022.sort_values("date_dt").iloc[0]

print(n_actions_2022)
earliest_2022["date"], earliest_2022["title"], earliest_2022["category"],
↳ earliest_2022["link"]
```

3377

```
( 'January 04, 2022',
  'Integrated Pain Management Medical Group Agreed to Pay $10,000 for
  Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded
  Individuals',
  nan,

  'https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-10000-for-allegedly-violating-the-civil-monetary-penalties-law-by-employing-excluded-individuals/' ]
```

Running my dynamic scraper starting from January 2022 produced 3,377 enforcement actions in the final dataframe.

The earliest enforcement action scraped was dated January 4, 2022 and titled “Integrated Pain Management Medical Group Agreed to Pay \$10,000 for Allegedly Violating the Civil Monetary Penalties ...”. The link to the enforcement action is: [‘https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-10000-for-allegedly-violating-the-civil-monetary-penalties-law-by-employing-excluded-individuals/’].

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
import pandas as pd
import altair as alt

# Read the CSV generated in Step 2(c)
df = pd.read_csv("enforcement_actions_year_month.csv")

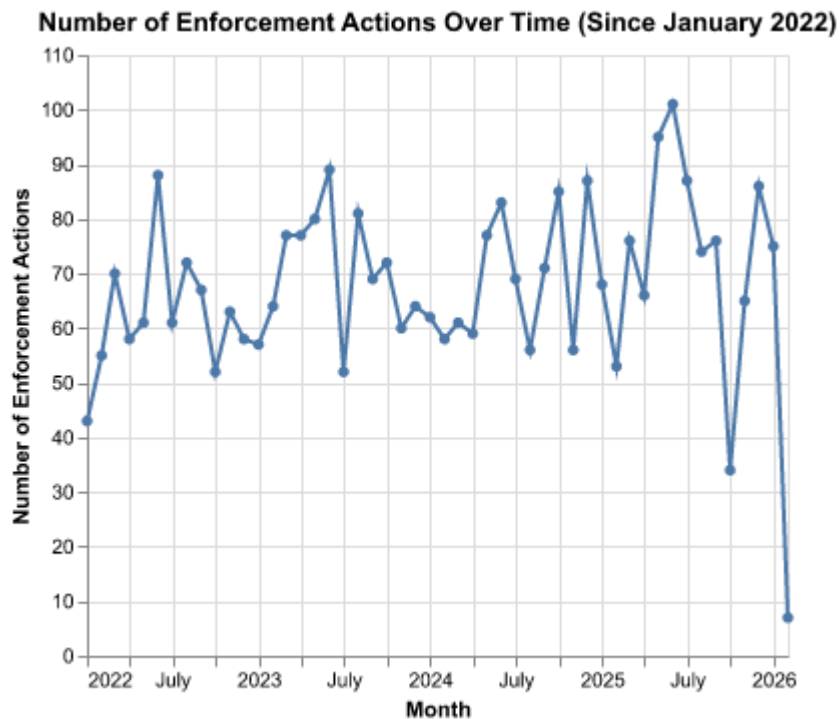
# Recreate date_dt
df["date_dt"] = pd.to_datetime(df["date"])

# Create year-month variable for aggregation
df["year_month"] = df["date_dt"].dt.to_period("M").astype(str)

# Line chart: number of enforcement actions over time (monthly)
monthly_counts = (
    df.groupby("year_month")
      .size()
      .reset_index(name="n_actions")
)

# Altair
```

```
alt.Chart(monthly_counts).mark_line(point=True).encode(
    x=alt.X("year_month:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
    tooltip=["year_month", "n_actions"]
).properties(
    title="Number of Enforcement Actions Over Time (Since January 2022)",
    width=350
)
```



2. Plot the number of enforcement actions categorized:

- based on “Criminal and Civil Actions” vs. “State Enforcement Agencies”

```
import pandas as pd
import altair as alt

# Ensure date is datetime
df["date_dt"] = pd.to_datetime(df["date"])

# Month aggregation key
```

```

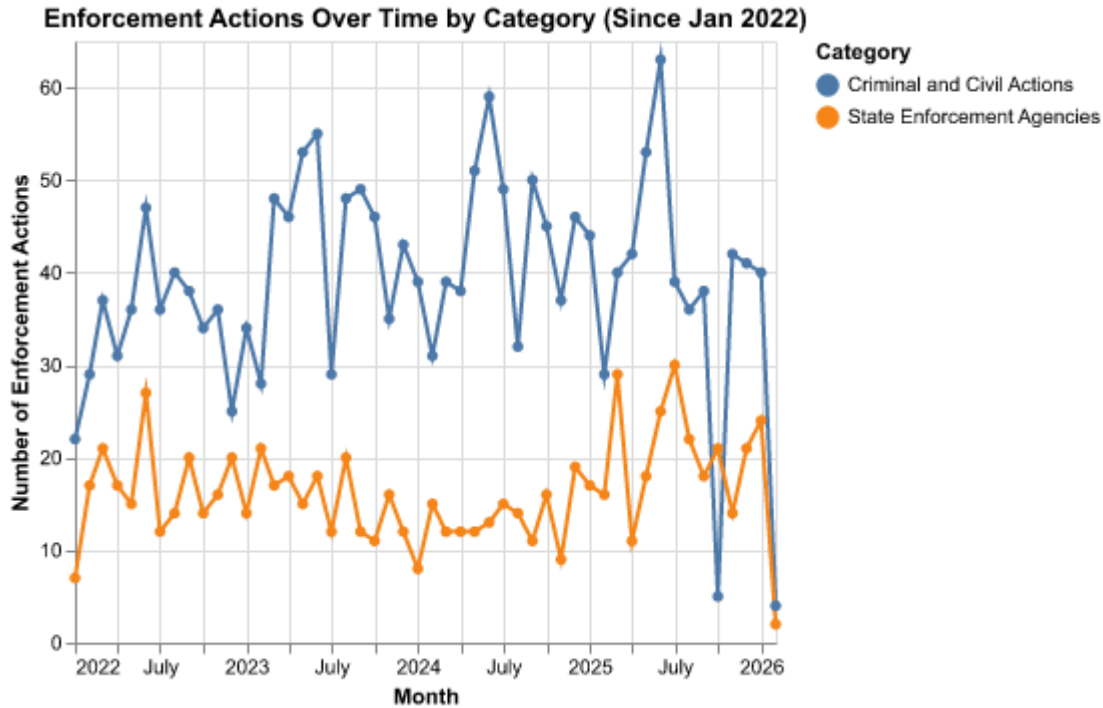
df["year_month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

# Keep only the two categories asked
cat_keep = ["Criminal and Civil Actions", "State Enforcement Agencies"]
cat_df = df[df["category"].isin(cat_keep)].copy()

cat_monthly = (
    cat_df.groupby(["year_month", "category"])
           .size()
           .reset_index(name="n_actions")
)

alt.Chart(cat_monthly).mark_line(point=True).encode(
    x=alt.X("year_month:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
    color=alt.Color("category:N", title="Category"),
    tooltip=[
        alt.Tooltip("year_month:T", title="Month"),
        alt.Tooltip("category:N", title="Category"),
        alt.Tooltip("n_actions:Q", title="# Actions")
    ]
).properties(
    title="Enforcement Actions Over Time by Category (Since Jan 2022)",
    width=350
)

```



- based on five topics

```
import pandas as pd
import altair as alt
import re

# If df isn't defined yet in your notebook/session, uncomment the next lines:
# df = pd.read_csv("enforcement_actions_year_month.csv")
# df["date_dt"] = pd.to_datetime(df["date"])

df["date_dt"] = pd.to_datetime(df["date"])
df["year_month"] = df["date_dt"].dt.to_period("M").dt.to_timestamp()

# Only Criminal and Civil Actions
cca = df[df["category"] == "Criminal and Civil Actions"].copy()

def classify_topic(title: str) -> str:
    t = (title or "").lower()

    # Health Care Fraud
    if
        ↪ re.search(r"\b(medicare|medicaid|health|hospital|clinic|nursing|physician|home
        ↪ health|dme)\b", t):
```

```

        return "Health Care Fraud"

# Financial Fraud
if re.search(r"\b(bank|financial|wire fraud|money
    ↳ laundering|loan|tax|securities|embezzl)\b", t):
    return "Financial Fraud"

# Drug Enforcement
if re.search(r"\b(drug|opioid|fentanyl|pharmacy|controlled substance|pill
    ↳ mill|prescription)\b", t):
    return "Drug Enforcement"

# Bribery/Corruption
if re.search(r"\b(bribe|bribery|kickback|corrupt|corruption|gratuit)\b",
    ↳ t):
    return "Bribery/Corruption"

return "Other"

cca["topic"] = cca["title"].apply(classify_topic)

topic_monthly = (
    cca.groupby(["year_month", "topic"])
        .size()
        .reset_index(name="n_actions")
)

alt.Chart(topic_monthly).mark_line(point=True).encode(
    x=alt.X("year_month:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
    color=alt.Color("topic:N", title="Topic"),
    tooltip=[
        alt.Tooltip("year_month:T", title="Month"),
        alt.Tooltip("topic:N", title="Topic"),
        alt.Tooltip("n_actions:Q", title="# Actions")
    ]
).properties(
    title="Criminal & Civil Actions Over Time by Topic (Since Jan 2022)",
    width=350
)

```

