30538 Problem set 4: Web Scraping

Rajat Kanti Paul

2026-02-07

Due 02/07/2026 at 5:00PM Central

"This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: Rajat Kanti Paul

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/hWhtcHqH
        - 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.

(40%) Step 1: Develop initial scraper and crawler

Scraping: Go to the first page of the HHS OIG's "Enforcement Actions" page and scrape and collect the following into a dataset: * Title of the enforcement action * Date * Category (e.g, "Criminal and Civil Actions") * Link associated with the enforcement action

Collect your output into a tidy dataframe and print its `head`.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

print("Initial scraper and crawler:")

# Downloading HTML with requests.get()
url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'lxml')

print("\nAfter Imspecting: Found li tags:", len(soup.find_all('li')))

print("\nFor Enforcement Actions: tag used is 'li' and class used is
 ↪  'usa-card card--list pep-card--minimal'")
print("Total usa-card items:", len(soup.find_all('li', class_='usa-card')))

# for first page
cards = soup.find_all('li', class_='usa-card')[:20]

data = []
for card in cards:
    title_tag = card.find('h2', class_='usa-card__heading').find('a')
    title = title_tag.text.strip() if title_tag else 'N/A'

    date_tag = card.find('span', class_='text-base-dark')
    date = date_tag.text.strip() if date_tag else 'N/A'

    tags = card.find_all('li', class_='usa-tag')
    category = ', '.join([tag.text.strip() for tag in tags])

    data.append({'title': title, 'date': date, 'category': category})

enf = pd.DataFrame(data)
print(f"\nhead of dataset: \n{enf.head()}")
```

```
Initial scraper and crawler:

After Imspecting: Found li tags: 145

For Enforcement Actions: tag used is 'li' and class used is 'usa-card
card--list pep-card--minimal'
Total usa-card items: 20

head of dataset:
                                                title              date  \
0  Houston Transplant Doctor Indicted For Making ...  February 5, 2026
1  MultiCare Health System to Pay Millions to Set...  February 4, 2026
2  Brooklyn Banker Pleads Guilty to Laundering Pr...  February 3, 2026
3  Delafield Man Sentenced to 18 Months' Imprison...  February 3, 2026
4  Former NFL Player Convicted for $197M Medicare...  February 3, 2026

                     category
0  Criminal and Civil Actions
1  Criminal and Civil Actions
2                   COVID-19
3  Criminal and Civil Actions
4  Criminal and Civil Actions
```

(40%) Step 2: Making the scraper dynamic

1. **Turning the scraper into a function**: You will write a function that takes as input a month and a year, and then pulls and formats the enforcement actions like in Step 1 starting from that month+year to today.

    - It is *very important* to make sure that you include an indicator whether or not to actually run the function. if you do not include an indicator, then each time you try to knit the qmd file, the scraper will run, and it will take a very long time to compile your pdf. Instead, run the function once to create a file called enforcement_actions_year_month.csv, which you can use in subsequent parts of the pset. Then turn the indicator off so that knitting your qmd file goes smoothly.
    - This function should first check that the year inputted >= 2013 before starting to scrape. if the year inputted < 2013, it should print a statement reminding the user to restrict to year >= 2013, since only enforcement actions after 2013 are listed.
    - It should save the dataframe output into a .csv file named as "enforcement_actions_*year_month*.csv" (do not commit this file to git)
    - if you're crawling multiple pages, always add 1 second wait before going to the next page to prevent potential server-side block. To implement this in Python, you may look up `.sleep()` function from `time` library.

a. Before writing out your function, write down pseudo-code of the steps that your function will go through. if you use a loop, discuss what kind of loop you will use and how you will define it. *Hint: Note that a simple* `for` *loop may not be sufficient for what this crawler requires. Use online resources to look into different types of loops or different ways of using* `for` *loops to see if there is something that is more appropriate for this task.*

Ans:

Pseudocode for Dynamic Scraper Function:

FUNCTION scrape_enforcement_actions(start_month, start_year, run_scraper) Begin if start_year < 2013 then print message reminding user that only actions from 2013 onward are available return nothing end if

if run_scraper is FALSE then print message telling user to set run_scraper to TRUE return nothing end if

set page_number to 1 set all_rows to empty list

while TRUE do: if page_number == 1 then set url to first page URL without page parameter else set url to base URL plus page=page_number end if

```
request html from url
parse html into soup with BeautifulSoup

find all li elements with class "usa-card" and store as cards

if number of cards == 0 then
  break out of while loop (no more pages)
end if

set stop_flag to FALSE

for each card in cards do:
  find title link inside h2 with class "usa-card__heading"
  extract title text
  extract href link

  find span with class "text-base-dark" and extract date text

  parse date text into year and month (convert month name to number)

  find all li with class "usa-tag" and JOIN their texts into one category
  string
```

```
  append a row with title, full date text, year, month, category, and link to
  all_rows

  if year is less than start_year then
    set stop_flag to TRUE
  else if year equals start_year AND month is less than start_month then
    set stop_flag to TRUE
  end if

end for

if stop_flag is TRUE then
  break out of while loop
end if

increase page_number by 1

wait 1 second before next request (using sleep)
```

end while

convert all_rows into a dataframe

save dataframe as CSV file named "enforcement_actions_{*year*}_{*month*}.csv"

return dataframe

end end function

b. Now code up your dynamic scraper and run it to start collecting the enforcement actions since January 2024. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped?

```python
import time

def scrape_enforcement_actions(start_month, start_year, run_scraper=False):
    if start_year < 2013:
        print("Year must be 2013 or later. Only actions from 2013 onward are
        ↪  listed.")
        return None

    if run_scraper is False:
        print("Set run_scraper=True to actually run the scraper.")
        return None
```

```
    all_rows = []
    page = 1

    while True:
        if page == 1:
            url = "https://oig.hhs.gov/fraud/enforcement/"
        else:
            url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"

        response = requests.get(url)
        soup = BeautifulSoup(response.text, "lxml")

        cards = soup.find_all("li", class_="usa-card")

        if len(cards) == 0:
            break

        stop_flag = False

        for card in cards:
            h2tag = card.find("h2", class_="usa-card__heading")
            if h2tag is not None:
                atag = h2tag.find("a")
            else:
                atag = None

            if atag is not None:
                title = atag.text.strip()
                link = atag.get("href")
            else:
                title = ""
                link = ""

            span_date = card.find("span", class_="text-base-dark")
            if span_date is not None:
                date_text = span_date.text.strip()
            else:
                date_text = ""

            try:
                parts = date_text.replace(",", "").split()
                month_name = parts[0]
                year_int = int(parts[-1])
```

```
            except:
                month_name = ""
                year_int = 0

            month_map = {
                "January": 1, "February": 2, "March": 3, "April": 4,
                "May": 5, "June": 6, "July": 7, "August": 8,
                "September": 9, "October": 10, "November": 11, "December": 12
            }
            month_int = month_map.get(month_name, 0)

            if year_int < start_year:
                stop_flag = True
                break
            elif year_int == start_year and month_int < start_month:
                stop_flag = True
                break

            tag_list = card.find_all("li", class_="usa-tag")
            categories = [t.text.strip() for t in tag_list]
            category_text = "; ".join(categories)

            all_rows.append({
                "title": title,
                "date": date_text,
                "year": year_int,
                "month": month_int,
                "category": category_text,
                "link": link
            })

        if stop_flag:
            print("Reached actions before", start_month, start_year, "-
             ↪ stopping crawl.")
            break

        page += 1
        time.sleep(2)

    df = pd.DataFrame(all_rows)
    outname = f"enforcement_actions_{start_year}_{start_month}.csv"
    df.to_csv(outname, index=False)
```

```
    print("Saved to:", outname)
    print("Total rows scraped:", len(df))

    return df

# Running dynamic scraper
df_2024 = scrape_enforcement_actions(1, 2024, run_scraper=True)

# Number of enforcement actions
print(f"\nNumber of Enforcement Actions is: {len(df_2024)}")
```

```
Reached actions before 1 2024 - stopping crawl.
Saved to: enforcement_actions_2024_1.csv
Total rows scraped: 1787

Number of Enforcement Actions is: 1787
```

```
## Used Generative AI to print title without ellises
pd.set_option('display.max_colwidth', None)
pd.set_option('display.width', None)
pd.set_option('display.max_columns', None)


# Earliest enforcement action
earliest_row = df_2024.iloc[-1]

# Explicitly sort by parsed date
df_2024['date_parsed'] = pd.to_datetime(df_2024['date'], errors='coerce')
df_2024_sorted = df_2024.sort_values('date_parsed')
earliest_row = df_2024_sorted.iloc[0]

earliest_date = earliest_row['date']
earliest_title = earliest_row['title']
earliest_category = earliest_row['category']
earliest_link = earliest_row['link']

print(f"\nThe date and details of the earliest enforcement action:
 ↪  \n{earliest_row}")
```

```
The date and details of the earliest enforcement action:
```

```
title                                Former Nurse Aide Indicted In Death Of
Clarksville Patient Arrested In Georgia
date
January 3, 2024
year
2024
month
1
category
State Enforcement Agencies
link
/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-patient-arrested-in-geo
date_parsed
2024-01-03 00:00:00
Name: 1786, dtype: object
```

Ans: The dynamic scraper collected 1787 enforcement actions from January 2024 to the present.

The earliest enforcement action in this period is dated January 3, 2024 and is titled "Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia." It is categorized under State Enforcement Agencies.

c. Now, let's go a little further back. Test your code by collecting the actions since January 2022. *Note that this can take a while.* How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped? Use the dataframe from this process for every question after this.

```python
# Running dynamic scraper for period since January 2022
df_2022 = scrape_enforcement_actions(1, 2022, run_scraper=True)

# Number of enforcement actions
print(f"\nNumber of Enforcement Actions is: {len(df_2022)}")

pd.set_option('display.max_colwidth', None)
pd.set_option('display.width', None)
pd.set_option('display.max_columns', None)


# Earliest enforcement action
earliest_2022 = df_2022.iloc[-1]

# Explicitly sort by parsed date
df_2022['date_parsed'] = pd.to_datetime(df_2022['date'], errors='coerce')
```

```
df_2022_sorted = df_2022.sort_values('date_parsed')
earliest_2022 = df_2022_sorted.iloc[0]

earliest_date_2022 = earliest_2022['date']
earliest_title_2022 = earliest_2022['title']
earliest_category_2022 = earliest_2022['category']
earliest_link_2022 = earliest_2022['link']

print(f"\nThe date and details of the earliest enforcement action:
 ↪  \n{earliest_2022}")
```

```
Reached actions before 1 2022 - stopping crawl.
Saved to: enforcement_actions_2022_1.csv
Total rows scraped: 3377

Number of Enforcement Actions is: 3377

The date and details of the earliest enforcement action:
title                              Integrated Pain Management Medical Group
Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties
Law by Employing Excluded Individuals
date
January 4, 2022
year
2022
month
1
category
Fraud Self-Disclosures
link
/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-10000-for-allegedly
date_parsed
2022-01-04 00:00:00
Name: 3376, dtype: object
```

Ans: dynamic scraper collected 3,377 enforcement actions from January 2022 to the present.

The earliest enforcement action in this period is dated January 4, 2022 and is titled "Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals." It is categorized under Fraud Self-Disclosures.

(20%) Step 3: Plot data based on scraped data

Note: To complete this part of the pset, reference the csv file you created in step 2, enforcement_actions_year_month.csv.

1. Plot a line chart with altair that shows: **the number of enforcement actions** over time (aggregated to each month+year) overall since January 2022.

```
import altair as alt

# Parsing date and create year-month variable
df_2022["date_parsed"] = pd.to_datetime(df_2022["date"], errors="coerce")
df_2022["year_month"] = df_2022["date_parsed"].dt.to_period("M").astype(str)

# Aggregating to month-year
monthly_counts = (
    df_2022.groupby("year_month")
      .size()
      .reset_index(name="n_actions")
)

# Convert back to timestamp for Altair
monthly_counts["year_month_date"] =
 ↪  pd.to_datetime(monthly_counts["year_month"])

# Line chart of number of enforcement actions over time
enforcement_action_chart = (
    alt.Chart(monthly_counts)
    .mark_line(point=True)
    .encode(
        x=alt.X("year_month_date:T", title="Month", axis=alt.Axis(format="%b
 ↪  %Y")),
        ## Used Generative AI in finding axis formatting
        y=alt.Y("n_actions:Q", title="Number of enforcement actions")
    )
    .properties(
        title="Enforcement Actions per Month since January 2022"
    )
)
enforcement_action_chart
```

```
alt.Chart(...)
```

2. Plot a line chart with altair that shows: **the number of enforcement actions** split out by:

    - "Criminal and Civil Actions" vs. "State Enforcement Agencies"

```
df_2022["date_parsed"] = pd.to_datetime(df_2022["date"], errors="coerce")
df_2022["year_month"] = df_2022["date_parsed"].dt.to_period("M").astype(str)
df_2022["year_month_date"] = pd.to_datetime(df_2022["year_month"])

# Keeping only the two main categories
mask_two = df_2022["category"].str.contains("Criminal and Civil Actions") | \
           df_2022["category"].str.contains("State Enforcement Agencies")

df_two = df_2022[mask_two].copy()

def main_cat(cat):
    if "Criminal and Civil Actions" in cat:
        return "Criminal and Civil Actions"
    elif "State Enforcement Agencies" in cat:
        return "State Enforcement Agencies"
    else:
        return "Other"

df_two["main_category"] = df_two["category"].apply(main_cat)

monthly_two = (
    df_two.groupby(["year_month_date", "main_category"])
          .size()
          .reset_index(name="n_actions")
)

chart_main = (
    alt.Chart(monthly_two)
    .mark_line(point=True)
    .encode(
        x=alt.X("year_month_date:T", title="Month", axis=alt.Axis(format="%b
 ↪  %Y")),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
        color=alt.Color("main_category:N", title="Category"),
        tooltip=["year_month_date:T", "main_category", "n_actions"]
    )
    .properties(
        title="Enforcement Actions by Category since January 2022",
```

```
        width=700,
        height=400
    )
)

chart_main
```

```
alt.Chart(...)
```

- Five topics in the "Criminal and Civil Actions" category: "Health Care Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and "Other". *Hint: You will need to divide the five topics manually by looking at the title and assigning the relevant topic. for example, if you find the word "bank" or "financial" in the title of an action, then that action should probably belong to "Financial Fraud" topic. We suggest using AI to identify patterns in your scraped data and suggest ways of classifying based on the titles.*

```
df_cc = df_2022[df_2022["category"].str.contains("Criminal and Civil
 ↪  Actions")].copy()

## Used Generative AI in classifying topics
def classify_topic(title):
    t = str(title).lower()
    if any(w in t for w in [
        "health care fraud", "healthcare fraud", "home health", "clinic",
         ↪  "hospital",
        "nursing home", "medicaid", "medicare", "patient", "provider"
    ]):
        return "Health Care Fraud"
    if any(w in t for w in [
        "bank", "financial", "money laundering", "wire fraud", "kickback",
        "false claims act", "false claims", "billing", "overbilling",
        "reimbursement", "settlement", "restitution"
    ]):
        return "Financial Fraud"
    if any(w in t for w in [
        "opioid", "oxycodone", "fentanyl", "controlled substance",
         ↪  "prescribing",
        "prescription", "pill mill", "pharmacy", "drug diversion", "narcotic"
    ]):
        return "Drug Enforcement"
    if any(w in t for w in [
        "bribe", "bribery", "remuneration", "illegal payments", "kickback"
```

```
    ]):
        return "Bribery/Corruption"
    return "Other"

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

monthly_topics = (
    df_cc.groupby(["year_month_date", "topic"])
         .size()
         .reset_index(name="n_actions")
)

chart_topics = (
    alt.Chart(monthly_topics)
    .mark_line(point=True)
    .encode(
        x=alt.X("year_month_date:T", title="Month", axis=alt.Axis(format="%b
↪   %Y")),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
        color=alt.Color("topic:N", title="Topic"),
        tooltip=["year_month_date:T", "topic", "n_actions"]
    )
)

chart_topics
```

```
alt.Chart(...)
```