PSET 4 Web Scraping

Aleena Khan

2026-02-01

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: AK

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.

```python
import pandas as pd
import altair as alt
import time
import requests
from bs4 import BeautifulSoup
import re

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

```
RendererRegistry.enable('png')
```

Step 1: Develop initial scraper and crawler

```python
url = "https://oig.hhs.gov/fraud/enforcement/"
response = requests.get(url)
soup = BeautifulSoup(response.text, "html.parser")

# Regex to recognize dates like "January 17, 2024"
date_pattern = re.compile(r"^[A-Z][a-z]+ \d{1,2}, \d{4}$")

rows = []

# Titles are links inside h2 or h3 tags
title_links = soup.select("h2 a, h3 a")

for a in title_links:
    title = a.get_text(strip=True)
    link = a.get("href")

    # make link absolute
    if link and not link.startswith("http"):
        link = "https://oig.hhs.gov" + link

    heading = a.find_parent(["h2", "h3"])
    cursor = heading

    date = None
    category = None
    # walk forward in the HTML
    for _ in range(15):
        cursor = cursor.find_next()
        if cursor is None:
            break

        text = cursor.get_text(strip=True)

        # first date that we see
        if date is None and date_pattern.match(text):
            date = text
            continue

        # first short category label after the date
        if date and category is None:
            if text in [
                "Criminal and Civil Actions",
                "State Enforcement Agencies",
                "CMP and Affirmative Exclusions",
                "EMTALA/Patient Dumping"
            ]:
                category = text
                break

    if date and category:
        rows.append({
            "title": title,
            "date": date,
            "category": category,
            "link": link
        })

df = pd.DataFrame(rows)
df.head()    #printing head of data
```

	title	date	category	lin
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	Criminal and Civil Actions	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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code Take month, year, and a run indicator as inputs. If year < 2013, then print a message reminding the user that enforcement actions are only available from 2013 onward and exit. Convert month and year into a start date. Initialize page = 1 and an empty list to store results. While scraping is needed: Construct the URL for the current page. Scrape all enforcement actions on that page (title, date, category, link). Append results to the master list. Identify the oldest date on the page. If the oldest date is earlier than the start date, stop the loop. Otherwise, increment the page number and wait 1 second. Convert the collected results into a dataframe and filter to dates on or after the start date. Save the dataframe as enforcement_actions_year_month.csv and return it.
- b. Create Dynamic Scraper I had to use AI to help me with this dynamic scraper function considering it is a very long function with any moving parts. I did not simply copy, but had an AI chatbot help track all the if conditions necessary for this function.

```python
from datetime import datetime

BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"
DATE_RE = re.compile(r"^[A-Z][a-z]+ \d{1,2}, \d{4}$")

VALID_CATEGORIES = {
    "Criminal and Civil Actions",
    "State Enforcement Agencies",
    "CMP and Affirmative Exclusions",
    "EMTALA/Patient Dumping",
}

def scrape_enforcement_since(month: int, year: int, run: bool = False) ->
 ↪  pd.DataFrame | None:
    """
    Scrape HHS OIG enforcement actions from (year, month) through today.
    Saves to enforcement_actions_{year}_{month:02d}.csv when run=True.
    """

    if not run:
        print("run=False, skipping scrape. (Use the saved CSV when
         ↪  knitting.)")
        return None
    if year < 2013:
        print("Please restrict to year >= 2013 (only enforcement actions
         ↪  after 2013 are listed).")
        return None

    start_date = datetime(year, month, 1)

    page = 1
    all_rows = []

    while True:
        url = BASE_URL if page == 1 else f"{BASE_URL}?page={page}"
        resp = requests.get(url, timeout=30)
        resp.raise_for_status()
        soup = BeautifulSoup(resp.text, "html.parser")

        title_links = soup.select("h2 a, h3 a")
        page_rows = []

        for a in title_links:
            title = a.get_text(strip=True)
            href = a.get("href")
            if not title or not href:
                continue

            link = href if href.startswith("http") else "https://oig.hhs.gov"
↪   + href

            heading = a.find_parent(["h2", "h3"])
            cursor = heading

            date_str = None
            category = None

            for _ in range(20):
                cursor = cursor.find_next()
                if cursor is None:
                    break
                text = cursor.get_text(strip=True)

                if date_str is None and DATE_RE.match(text):
                    date_str = text
                    continue
                if date_str and category is None and text in
                 ↪ VALID_CATEGORIES:
                    category = text
                    break

            if date_str and category:
                page_rows.append(
                    {"title": title, "date": date_str, "category": category,
↪   "link": link}
                )

        if not page_rows:
            break

        df_page = pd.DataFrame(page_rows)
        df_page["date"] = pd.to_datetime(df_page["date"], format="%B %d, %Y",
↪   errors="coerce")
        df_page = df_page.dropna(subset=["date"])

        all_rows.extend(df_page.to_dict("records"))

        oldest_on_page = df_page["date"].min()
        if oldest_on_page < start_date:
            break

        page += 1
        time.sleep(1)

    df = pd.DataFrame(all_rows)
    df = df[df["date"] >= start_date].sort_values("date",
↪   ascending=False).reset_index(drop=True)

    outname = f"enforcement_actions_{year}_{month:02d}.csv"
    df.to_csv(outname, index=False)
    print(f"Saved {len(df)} rows to {outname}")

    return df
```

Running for Jan 2024

```python
df_2024 = scrape_enforcement_since(1, 2024, run=True)
```

```
Saved 1708 rows to enforcement_actions_2024_01.csv
```

How many enforcement actions

```python
len(df_2024)
```

```
1708
```

date and details of the earliest enforcement action it scraped.

```python
earliest = df_2024.sort_values("date").iloc[0]
earliest[["date", "title", "category", "link"]]
```

```
date                                        2024-01-03 00:00:00
title       Former Nurse Aide Indicted In Death Of Clarksv...
category                            State Enforcement Agencies
link        https://oig.hhs.gov/fraud/enforcement/former-n...
Name: 1707, dtype: object
```

Starting from January 2024, the scraper collected 1,708 enforcement actions. The earliest enforcement action in the dataset is dated January 3, 2024. It is titled “Former Nurse Aide Indicted In Death Of Clarksville …”, falls under the State Enforcement Agencies category.

- c. Test Your Code

Starting Jan 2022

```python
df_2022 = scrape_enforcement_since(1, 2022, run=True)
```

```
Saved 3254 rows to enforcement_actions_2022_01.csv
```

How many enforcement actions?

```python
len(df_2022)
```

```
3254
```

Earliest enforcement action scraped by date and details

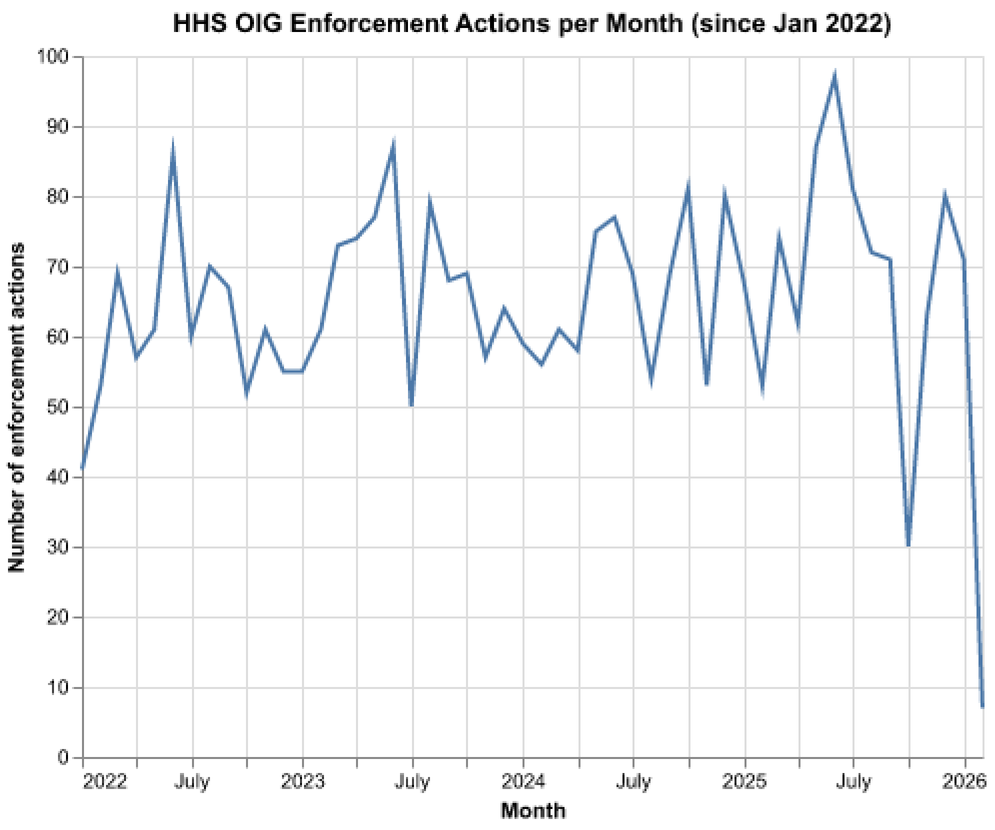
```
earliest_2022 = df_2022.sort_values("date").iloc[0]
earliest_2022[["date", "title", "category", "link"]]
```

```
date                                          2022-01-04 00:00:00
title       Integrated Pain Management Medical Group Agree...
category                              State Enforcement Agencies
link        https://oig.hhs.gov/fraud/enforcement/integrat...
Name: 3253, dtype: object
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
# Load CSV
df = pd.read_csv("enforcement_actions_2022_01.csv", parse_dates=["date"])

# Create a month-year column (first day of each month)
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp()

# Aggregate to monthly counts
monthly = (
    df.groupby("month", as_index=False)
      .size()
      .rename(columns={"size": "n_actions"})
      .sort_values("month")
)

# Line chart
chart = alt.Chart(monthly).mark_line().encode(
    x=alt.X("month:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
    tooltip=[alt.Tooltip("month:T", title="Month"),
             alt.Tooltip("n_actions:Q", title="Actions")]
).properties(
    title="HHS OIG Enforcement Actions per Month (since Jan 2022)",
    width=450,
    height=350
)

chart
```

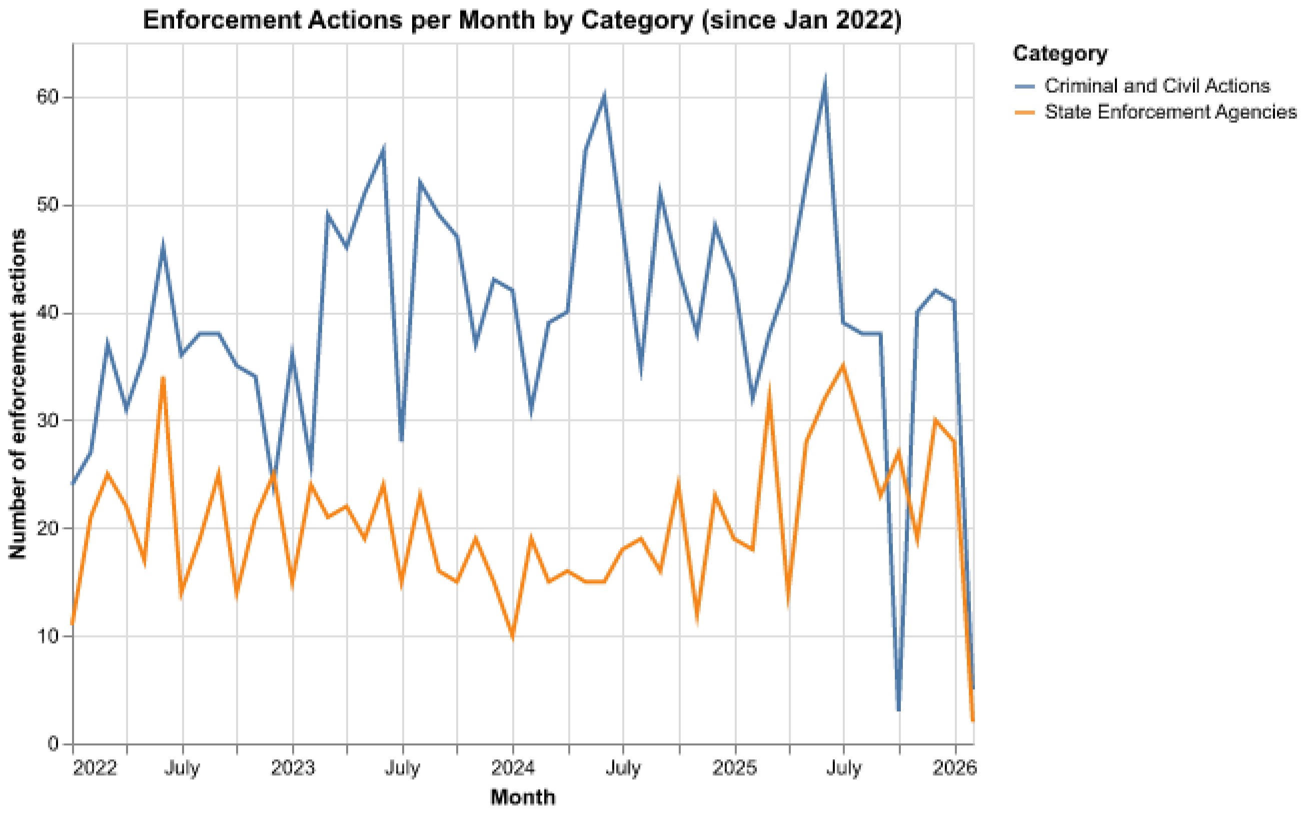


2. **Plot the number of enforcement actions categorized:**

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

```python
df = pd.read_csv("enforcement_actions_2022_01.csv", parse_dates=["date"])
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp()

# Keep just the two categories as outlined in quesiton
keep = ["Criminal and Civil Actions", "State Enforcement Agencies"]
df2 = df[df["category"].isin(keep)].copy()

# Aggregate monthly counts by category
monthly_cat = (
    df2.groupby(["month", "category"], as_index=False)
       .size()
       .rename(columns={"size": "n_actions"})
       .sort_values(["month", "category"])
)
```

```python
chart2 = alt.Chart(monthly_cat).mark_line().encode(
    x=alt.X("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("month:T", title="Month"),
        alt.Tooltip("category:N", title="Category"),
        alt.Tooltip("n_actions:Q", title="Actions")
    ]
).properties(
    title="Enforcement Actions per Month by Category (since Jan 2022)",
    width=450,
    height=350
)

chart2
```



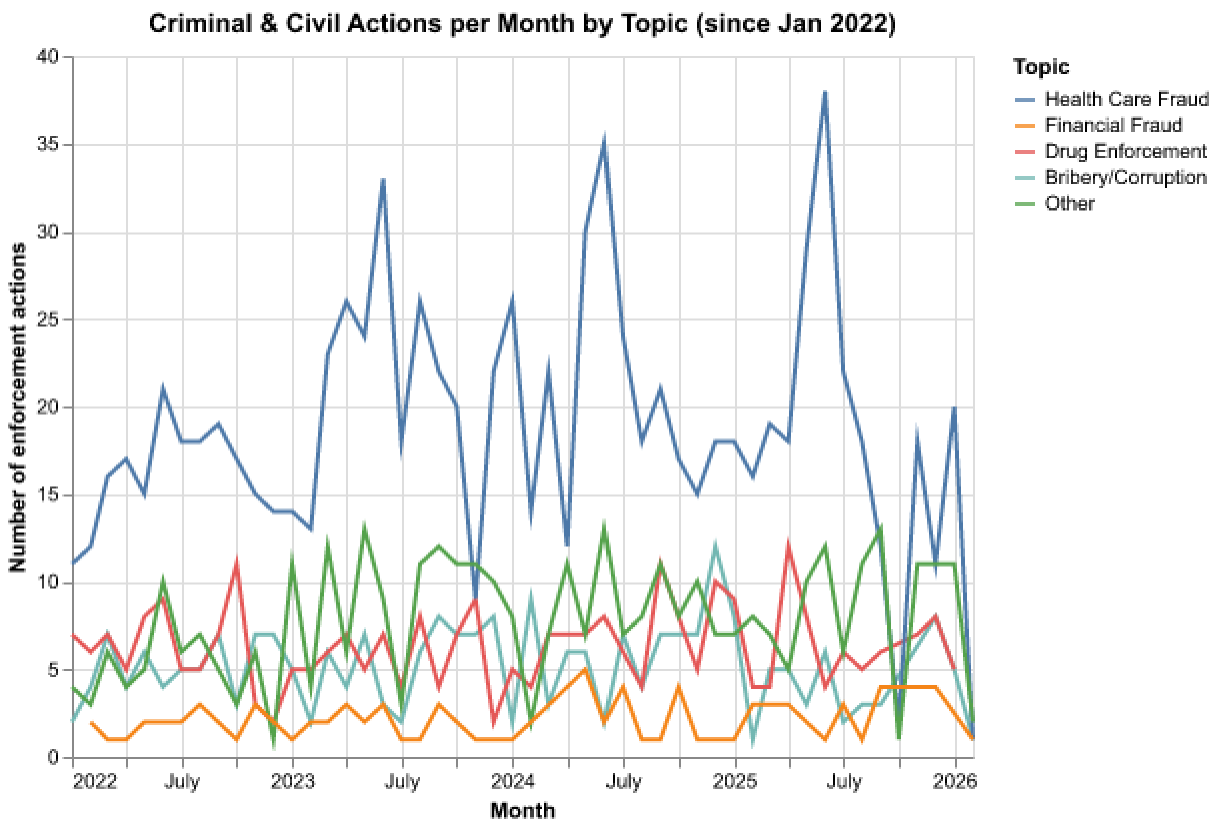
- based on five topics: Five topics in the "Criminal and Civil Actions" category: "Health

Care Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and "Other". I used AI here since dividing the five categories manually seemed a bit tedious and AI can do it faster and correctly.

```python
df = pd.read_csv("enforcement_actions_2022_01.csv", parse_dates=["date"])
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp()

cc = df[df["category"] == "Criminal and Civil Actions"].copy()

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

    # Bribery/Corruption
    if any(k in t for k in ["brib", "kickback", "corrupt", "illegal
     ↪  remuneration"]):
        return "Bribery/Corruption"

    # Drug Enforcement
    if any(k in t for k in ["opioid", "fentanyl", "controlled substance",
     ↪  "drug", "pill", "pharm", "prescrib"]):
        return "Drug Enforcement"

    # Financial Fraud
    if any(k in t for k in ["bank", "financial", "money laundering",
     ↪  "launder", "wire fraud", "tax", "embezz", "stolen", "forg"]):
        return "Financial Fraud"

    # Health Care Fraud
    if any(k in t for k in ["medicare", "medicaid", "health care fraud",
     ↪  "billing", "claims", "clinic", "physician", "hospit", "home health",
     ↪  "dme"]):
        return "Health Care Fraud"

    return "Other"

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

# (Optional) enforce topic order in the legend
topic_order = ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
 ↪  "Bribery/Corruption", "Other"]
```

Now time for the plot

```
monthly_topic = (
    cc.groupby(["month", "topic"], as_index=False)
      .size()
      .rename(columns={"size": "n_actions"})
      .sort_values(["month", "topic"])
)

chart3 = alt.Chart(monthly_topic).mark_line().encode(
    x=alt.X("month:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
    color=alt.Color("topic:N", title="Topic", sort=topic_order),
    tooltip=[
        alt.Tooltip("month:T", title="Month"),
        alt.Tooltip("topic:N", title="Topic"),
        alt.Tooltip("n_actions:Q", title="Actions")
    ]
).properties(
    title="Criminal & Civil Actions per Month by Topic (since Jan 2022)",
    width=450,
    height=350
)

chart3
```



Criminal & Civil Actions per Month by Topic (since Jan 2022)
Number of enforcement actions
0
5
10
15
20
25
30
35
40
2022
July
2023
July
2024
July
2025
July
2026
Month
Topic
Health Care Fraud
Financial Fraud
Drug Enforcement
Bribery/Corruption
Other