

PS4

Ning Xie

2026-02-07

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: NX

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
import requests
from bs4 import BeautifulSoup

base_url = "https://oig.hhs.gov"
url = base_url + "/fraud/enforcement/"
headers = {"User-Agent": "DAP30538CourseBot/1.0"}

resp = requests.get(url, headers=headers)
resp.raise_for_status()
soup = BeautifulSoup(resp.text, "lxml")

DATE_RE = re.compile(
    ↪ r"(January|February|March|April|May|June|July|August|September|October|November|December)"
)

rows = []

for li in soup.select("main li"):
    h2 = li.find("h2")
    if h2 is None:
        continue

    a = h2.find("a", href=True)
    if a is None:
        continue

    title = a.get_text(strip=True)

```

```

href = a["href"]

link = base_url + href if href.startswith("/") else href

block_text = li.get_text(" ", strip=True)
m = DATE_RE.search(block_text)
date = m.group(0) if m else None

lines = [x.strip() for x in li.get_text("\n", strip=True).split("\n") if
↪ x.strip()]

categories = []
date_idx = None
for i, line in enumerate(lines):
    if DATE_RE.search(line):
        date_idx = i
        break

if date_idx is not None:
    for cand in lines[date_idx + 1:]:
        if len(cand) <= 60:
            categories.append(cand)

seen = set()

categories = [c for c in categories if not (c in seen or seen.add(c))]

category = ";".join(categories) if categories else None

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

df = pd.DataFrame(rows)

df = df[df["date"].notna()].reset_index(drop=True)

print(df.head())

```

	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

	link
0	https://oig.hhs.gov/fraud/enforcement/houston-...
1	https://oig.hhs.gov/fraud/enforcement/multicar...
2	https://oig.hhs.gov/fraud/enforcement/brooklyn...
3	https://oig.hhs.gov/fraud/enforcement/delafiel...
4	https://oig.hhs.gov/fraud/enforcement/former-n...

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code
 1. My function takes year, month, and a RUN_SCRAPER indicator.
 2. If year < 2013, print a warning and stop (the site only lists actions after 2013).
 3. Set start_date to the first day of the input month/year.
 4. If RUN_SCRAPER == False, read enforcement_actions_year_month.csv from disk and return it; if the file doesn't exist, print a message telling the user to run once with RUN_SCRAPER=True.
 5. If RUN_SCRAPER == True, initialize an empty list and set page_num = 1.
 6. Use a while loop because we don't know how many pages we need to reach start_date.
 7. Request the current page HTML, then parse each action into rows (title/date/category/link) and convert dates to datetime.
 8. Keep only rows with date >= start_date; if no actions are found, stop.

9. If the oldest date on the page is older than start_date, break; otherwise increment page_num and sleep(1) before requesting the next page.
 10. Combine results into a dataframe, drop duplicates by link, save to enforcement_actions_year_month.csv, and return the dataframe.
- b. Create Dynamic Scraper

```

BASE_URL = "https://oig.hhs.gov"
START_URL = BASE_URL + "/fraud/enforcement/"
HEADERS = {"User-Agent": "DAP30538CourseBot/1.0"}

def _parse_page(soup: BeautifulSoup) -> pd.DataFrame:
    rows = []
    for li in soup.select("main li"):
        h2 = li.find("h2")
        if h2 is None:
            continue
        a = h2.find("a", href=True)
        if a is None:
            continue

        title = a.get_text(strip=True)
        href = a["href"]
        link = BASE_URL + href if href.startswith("/") else href

        text_block = li.get_text(" ", strip=True)
        m = DATE_RE.search(text_block)
        date_str = m.group(0) if m else None

        categories = [c.get_text(strip=True) for c in li.select("ul li")]
        category = "; ".join([c for c in categories if c]) if categories else
↪ None

        rows.append({"title": title, "date": date_str, "category": category,
↪ "link": link})

    df = pd.DataFrame(rows)
    if not df.empty:
        df["date"] = pd.to_datetime(df["date"], errors="coerce")
    return df

def scrape_enforcement_actions_since(year: int, month: int, RUN_SCRAPER: bool
↪ = False) -> pd.DataFrame:

```

```

"""Scrape HHS OIG enforcement actions since (year, month)."""
if year < 2013:
    print("Reminder: year must be >= 2013 because the site only lists
    ↪ actions after 2013.")
    return pd.DataFrame(columns=["title", "date", "category", "link"])

out_csv = f"enforcement_actions_since_{year}_{month:02d}.csv"

if not RUN_SCRAPER:
    return pd.read_csv(out_csv, parse_dates=["date"])

start_date = pd.Timestamp(year=year, month=month, day=1)
collected = []
page = 1

while True:
    if page == 1:
        url = START_URL
    else:
        url = f"{START_URL}?page={page}"

    resp = requests.get(url, headers=HEADERS)
    resp.raise_for_status()
    soup = BeautifulSoup(resp.text, "lxml")

    df_page = _parse_page(soup)

    if df_page.empty:
        break

    df_keep = df_page[df_page["date"] >= start_date]
    if not df_keep.empty:
        collected.append(df_keep)

    oldest_date = df_page["date"].min()
    if pd.notna(oldest_date) and oldest_date < start_date:
        break

    page += 1
    time.sleep(1)

if collected:
    out = (

```

```

        pd.concat(collected, ignore_index=True)
        .drop_duplicates(subset=["link"])
        .reset_index(drop=True)
    )
    else:
        out = pd.DataFrame(columns=["title", "date", "category", "link"])

    out.to_csv(out_csv, index=False)
    return out

df_2024 = scrape_enforcement_actions_since(2024, 1, RUN_SCRAPER=False)

print("Since Jan 2024: number of actions =", df_2024.shape[0])

earliest_2024 = df_2024.sort_values("date", ascending=True).iloc[0]

print("Earliest since Jan 2024:", earliest_2024["date"],
      ↪ earliest_2024["title"], earliest_2024["category"], earliest_2024["link"])

```

Since Jan 2024: number of actions = 1787

Earliest since Jan 2024: 2024-01-03 00:00:00 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-patient-arrested-in-georgia/>

I get 1787 enforcement actions in my final dataframe since January 2024. The earliest enforcement action it scraped is dated 2024-01-03: “Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia” (Category: State Enforcement Agencies). Link: <https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-patient-arrested-in-georgia/>.

- c. Test Your Code

```

df_2022 = scrape_enforcement_actions_since(2022, 1, RUN_SCRAPER=False)

print("\nSince Jan 2022: number of actions =", df_2022.shape[0])

earliest_2022 = df_2022.sort_values("date", ascending=True).iloc[0]

print("Earliest since Jan 2022:", earliest_2022["date"],
      ↪ earliest_2022["title"], earliest_2022["category"], earliest_2022["link"])

```

Since Jan 2022: number of actions = 3377

Earliest since Jan 2022: 2022-01-04 00:00:00 Integrated Pain Management Medical Group Agreed to Pay \$10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals Fraud Self-Disclosures

[https://oig.hhs.gov/fraud/enforcement/integrated-pain-management-medical-group-agreed-to-pay-](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/)

I get 3377 enforcement actions in my final dataframe since January 2022. The earliest enforcement action it scraped is dated 2022-01-04: “Integrated Pain Management Medical Group Agreed to Pay \$10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals” (Category: Fraud Self-Disclosures). Link: <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

```
df = pd.read_csv(
    "enforcement_actions_since_2022_01.csv",
    parse_dates=["date"]
)

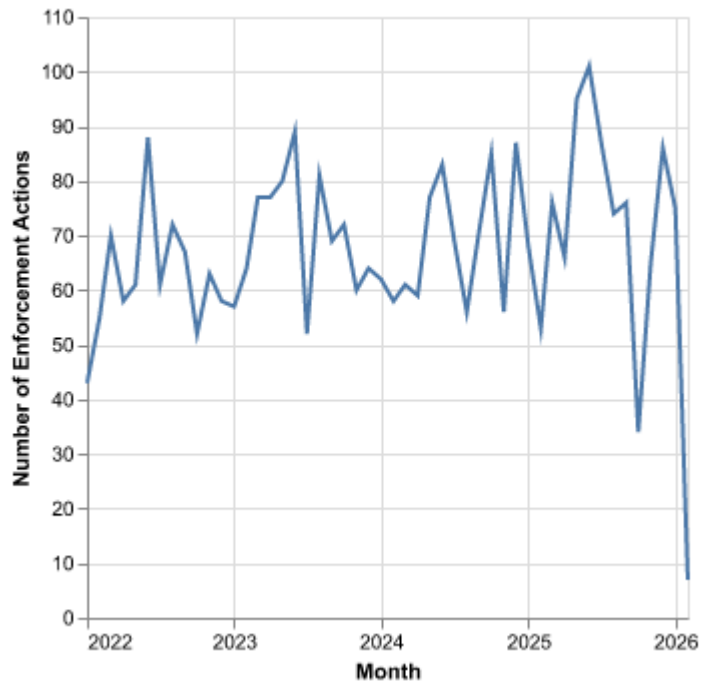
df["year_month"] = df["date"].dt.to_period("M").dt.to_timestamp()

monthly_counts = (
    df.groupby("year_month")
      .size()
      .reset_index(name="num_actions")
)

chart = (alt.Chart(monthly_counts).mark_line().encode(
    x=alt.X("year_month:T", title="Month"),
    y=alt.Y("num_actions:Q", title="Number of Enforcement Actions")
)
  .properties(
    title="Number of HHS OIG Enforcement Actions per Month (Since Jan
↪ 2022)"
  )
)

chart
```

Number of HHS OIG Enforcement Actions per Month (Since Jan 2022)



2. Plot the number of enforcement actions categorized:

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

```
df = df_2022.copy()

df["date"] = pd.to_datetime(df["date"])

df["yearmonth"] = df["date"].dt.to_period("M").dt.to_timestamp()

df["main_category"] = df["category"].fillna("").apply(
    lambda x: "State Enforcement Agencies"
    if "State Enforcement Agencies" in x
    else "Criminal and Civil Actions"
)

monthly_by_category = (
    df
    .groupby(["yearmonth", "main_category"])
```

```

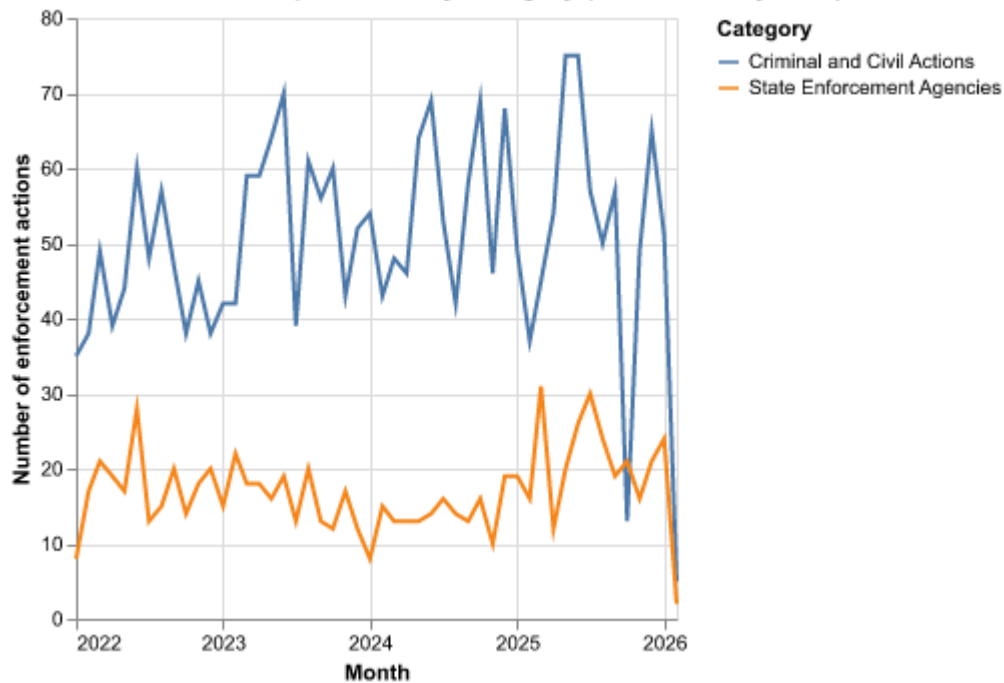
        .size()
        .reset_index(name="n_actions")
    )

    chart_two_categories = (
        alt.Chart(monthly_by_category)
        .mark_line()
        .encode(
            x=alt.X("yearmonth:T", title="Month"),
            y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
            color=alt.Color("main_category:N", title="Category")
        )
        .properties(
            title="HHS OIG Enforcement Actions per Month by Category (Since
↪ January 2022)"
        )
    )

    chart_two_categories

```

HHS OIG Enforcement Actions per Month by Category (Since January 2022)



- based on five topics

```

df = df_2022.copy()
df["date"] = pd.to_datetime(df["date"])
df["yearmonth"] = df["date"].dt.to_period("M").dt.to_timestamp()

df["main_category"] = df["category"].fillna("").apply(
    lambda x: "State Enforcement Agencies"
    if "State Enforcement Agencies" in x
    else "Criminal and Civil Actions"
)

df_criminal = df[df["main_category"] == "Criminal and Civil Actions"].copy()

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

    if any(k in t for k in ["drug", "opioid", "fentanyl", "controlled
        ↪ substance", "pill", "pharmacy", "suboxone", "oxy", "heroin",
        ↪ "meth"]):
        return "Drug Enforcement"

    if any(k in t for k in ["brib", "kickback", "corrupt", "money
        ↪ laundering", "launder", "conspiracy", "racketeering", "extortion"]):
        return "Bribery/Corruption"

    if any(k in t for k in ["bank", "financial", "wire fraud", "mortgage",
        ↪ "loan", "credit", "tax", "investment", "securities", "bitcoin",
        ↪ "crypto", "laundering"]):
        return "Financial Fraud"

    if any(k in t for k in ["health care fraud", "healthcare fraud",
        ↪ "medicare", "medicaid", "billing", "claims", "patient", "provider",
        ↪ "clinic", "hospital", "physician", "doctor", "nurse", "dme",
        ↪ "telemedicine"]):
        return "Health Care Fraud"

    return "Other"

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

monthly_topics = (
    df_criminal
    .groupby(["yearmonth", "topic"])

```

```

        .size()
        .reset_index(name="n_actions")
    )

    chart_topics = (
        alt.Chart(monthly_topics)
        .mark_line()
        .encode(
            x=alt.X("yearmonth:T", title="Month"),
            y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
            color=alt.Color("topic:N", title="Topic")
        )
        .properties(
            title="Criminal and Civil Actions per Month by Topic (Since January
↪ 2022)"
        )
    )

    chart_topics

```

Criminal and Civil Actions per Month by Topic (Since January 2022)

