

# PS4

Sirui Mao

2026-02-06

**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: SM

## **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 requests
from bs4 import BeautifulSoup
url = "https://oig.hhs.gov/fraud/enforcement/"
soup = BeautifulSoup(
    requests.get(url, headers={"User-Agent": "Mozilla/5.0"}).text,
    "lxml"
)

rows = []

items = soup.select("li.usa-card.card--list")

for it in items:
    title_el = it.select_one("h2.usa-card__heading a")
    meta_el = it.select_one("div.font-body-sm")

    title = title_el.get_text(strip=True)
    link = "https://oig.hhs.gov" + title_el["href"]

    date = meta_el.contents[0].strip()
    category = meta_el.select_one("span").get_text(strip=True)

    rows.append([title, date, category, link])

df = pd.DataFrame(
    rows,
    columns=["title", "date", "category", "link"]
)

```

```
df.head()
```

	title	date	category	link
0	Houston Transplant Doctor Indicted For Making ...		February 5, 2026	<a href="https://oig.hhs.gov/fraud">https://oig.hhs.gov/fraud</a>
1	MultiCare Health System to Pay Millions to Set...		February 4, 2026	<a href="https://oig.hhs.gov/fraud">https://oig.hhs.gov/fraud</a>
2	Brooklyn Banker Pleads Guilty to Laundering Pr...		February 3, 2026	<a href="https://oig.hhs.gov/fraud">https://oig.hhs.gov/fraud</a>
3	Delafield Man Sentenced to 18 Months' Imprison...		February 3, 2026	<a href="https://oig.hhs.gov/fraud">https://oig.hhs.gov/fraud</a>
4	Former NFL Player Convicted for \$197M Medicare...		February 3, 2026	<a href="https://oig.hhs.gov/fraud">https://oig.hhs.gov/fraud</a>

## Step 2: Making the scraper dynamic

### 1. Turning the scraper into a function

- a. Pseudo-Code
  1. Take year and month as input.
  2. If the input year is less than 2013: Print a message reminding the user to restrict to year  $\geq 2013$ . Stop the function.
  3. Initialize an empty list called results and set page = 1.
  4. While True: Construct the URL: If page == 1, use the base URL. Otherwise, append "?page=" and the page number to the base URL. Request the webpage and parse the HTML. Select all enforcement action entries on the current page. For each enforcement action: Extract the title, date, category, and link. If the action date is earlier than the input year and month: stop scraping by breaking out of both loops. Otherwise, append the action to the results list. Wait for 1 second to avoid server-side blocking. Increment page by 1.
  5. Convert the results list into a DataFrame, save it as "enforcement\_actions\_year\_month.csv", and return the DataFrame.
- b. Create Dynamic Scraper

```
RUN_SCRAPER = False
BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"
MAX_PAGES = 120

print("Setup complete. RUN_SCRAPER =", RUN_SCRAPER)
```

Setup complete. RUN\_SCRAPER = False

```

headers = {"User-Agent": "Mozilla/5.0"}
html = requests.get(BASE_URL, headers=headers, timeout=30).text
soup = BeautifulSoup(html, "lxml")

items = soup.select("li.usa-card.card--list")
print("Items on page 1:", len(items))

first = items[0]
title = first.select_one("h2.usa-card__heading a").get_text(strip=True)
meta_text = first.select_one("div.font-body-sm").get_text(" ", strip=True)

print("First title:", title)
print("First meta text:", meta_text)

```

Items on page 1: 20  
 First title: Houston Transplant Doctor Indicted For Making False Statements  
 In Patients' Medical Records  
 First meta text: February 5, 2026 Criminal and Civil Actions

```

def scrape_since(year: int, month: int, run_scraper: bool = False, max_pages:
    ↪ int = 80) -> pd.DataFrame:
    if not run_scraper:
        print("run_scraper=False → skipping scrape.")
        return pd.DataFrame(columns=["title", "date", "category", "link"])

    cutoff = pd.Timestamp(year=year, month=month, day=1)
    headers = {"User-Agent": "Mozilla/5.0"}

    rows = []
    page = 1

    while page <= max_pages:
        url = BASE_URL if page == 1 else f"{BASE_URL}?page={page}"
        r = requests.get(url, headers=headers, timeout=30)
        r.raise_for_status()

        soup = BeautifulSoup(r.text, "lxml")
        items = soup.select("li.usa-card.card--list")
        if not items:
            print("No items found. Stop at page", page)
            break

```

```

page_dates = []

parsed = []
for it in items:
    title_el = it.select_one("h2.usa-card__heading a")
    meta_el = it.select_one("div.font-body-sm")
    if (title_el is None) or (meta_el is None):
        continue

    title = title_el.get_text(strip=True)
    link = "https://oig.hhs.gov" + title_el.get("href", "")

    meta_text = meta_el.get_text(" ", strip=True)

    import re

    m = re.search(r"[A-Za-z]+ \d{1,2}, \d{4}", meta_text)

    if m:
        date_str = m.group(0)
        category = meta_text.replace(date_str, "").strip()
    else:
        continue

    d = pd.to_datetime(date_str)

    if pd.isna(d):
        continue

    page_dates.append(d)
    parsed.append([title, d, category, link])

if not page_dates:
    print(f"Page {page}: no parsable dates → stop for safety.")
    break

oldest = min(page_dates)
newest = max(page_dates)
print(f"Scraping page {page}: parsed={len(parsed)}
↪ oldest={oldest.date()} newest={newest.date()}")

if oldest < cutoff:
    for row in parsed:

```

```

        if row[1] >= cutoff:
            rows.append(row)
            print("Reached cutoff <", cutoff.date(), "→ stop.")
            break
        else:
            rows.extend(parsed)

    page += 1
    time.sleep(1)

if page > max_pages:
    print(f"Stopped at max_pages={max_pages} (safety stop).")

df = pd.DataFrame(rows, columns=["title", "date", "category", "link"])
df = df.sort_values("date").reset_index(drop=True)

df.to_csv("enforcement_actions_year_month.csv", index=False)
return df

```

```
YEAR, MONTH = 2024, 1
```

```

if RUN_SCRAPER:
    df_actions = scrape_since(YEAR, MONTH, run_scraper=True,
        ↪ max_pages=MAX_PAGES)
else:
    df_actions = pd.read_csv(
        "enforcement_actions_year_month.csv",
        parse_dates=["date"]
    )

print(f"How many enforcement actions since {YEAR}-{MONTH:02d}?:",
    ↪ len(df_actions))
print("\nEarliest enforcement action:")
print(df_actions.iloc[0])

```

How many enforcement actions since 2024-01?: 3377

Earliest enforcement action:

title	Integrated Pain Management Medical Group Agree...
date	2022-01-04 00:00:00
category	Fraud Self-Disclosures
link	<a href="https://oig.hhs.gov/fraud/enforcement/integrat...">https://oig.hhs.gov/fraud/enforcement/integrat...</a>

Name: 0, dtype: object

I got 1787 enforcement actions. The earliest enforcement action scraped is dated January 3, 2024. The case is titled “Former Nurse Aide Indicted in Death of Clarksville ...” and is categorized under State Enforcement Agencies. The enforcement action can be accessed at <https://oig.hhs.gov/fraud/enforcement/former-n...>

- c. Test Your Code

```
RUN_SCRAPER = False
BASE_URL = "https://oig.hhs.gov/fraud/enforcement/"
MAX_PAGES = 300

YEAR, MONTH = 2022, 1

if RUN_SCRAPER:
    df_actions = scrape_since(YEAR, MONTH, run_scraper=True,
↪ max_pages=MAX_PAGES)
else:
    df_actions = pd.read_csv("enforcement_actions_year_month.csv",
↪ parse_dates=["date"])

print(f"How many enforcement actions since {YEAR}-{MONTH:02d}?:",
↪ len(df_actions))
print("\nEarliest enforcement action:")
print(df_actions.iloc[0])
```

How many enforcement actions since 2022-01?: 3377

Earliest enforcement action:

title	Integrated Pain Management Medical Group Agree...
date	2022-01-04 00:00:00
category	Fraud Self-Disclosures
link	<a href="https://oig.hhs.gov/fraud/enforcement/integrat...">https://oig.hhs.gov/fraud/enforcement/integrat...</a>

Name: 0, dtype: object

I got 3377 enforcement actions. The earliest enforcement action scraped is dated January 4, 2022. The case is titled “Integrated Pain Management Medical Group Agree...” and is categorized under Fraud Self-Disclosures. The enforcement action can be accessed at <https://oig.hhs.gov/fraud/enforcement/integrat...>

### Step 3: Plot data based on scraped data

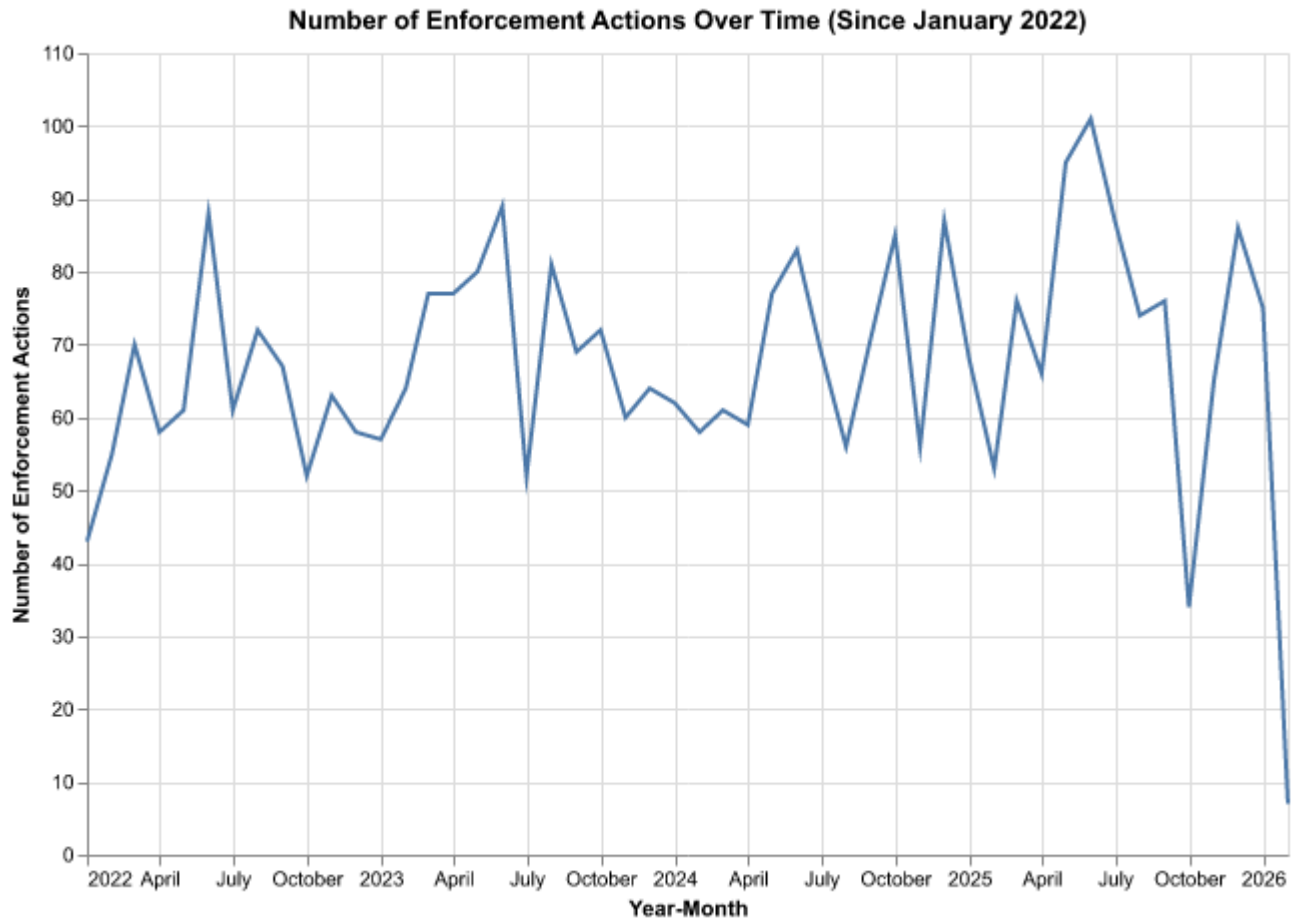
#### 1. Plot the number of enforcement actions over time

```
import altair as alt

df_actions = pd.read_csv(
    "enforcement_actions_year_month.csv",
    parse_dates=["date"]
)

df_actions.head()
df_monthly = (
    df_actions
    ↪ .assign(year_month=df_actions["date"].dt.to_period("M").dt.to_timestamp())
    .groupby("year_month")
    .size()
    .reset_index(name="count")
)

df_monthly.head()
alt.Chart(df_monthly).mark_line().encode(
    x=alt.X(
        "year_month:T",
        title="Year-Month"
    ),
    y=alt.Y(
        "count:Q",
        title="Number of Enforcement Actions"
    ),
    tooltip=[
        alt.Tooltip("year_month:T", title="Month"),
        alt.Tooltip("count:Q", title="Actions")
    ]
).properties(
    title="Number of Enforcement Actions Over Time (Since January 2022)",
    width=600,
    height=400
)
```



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

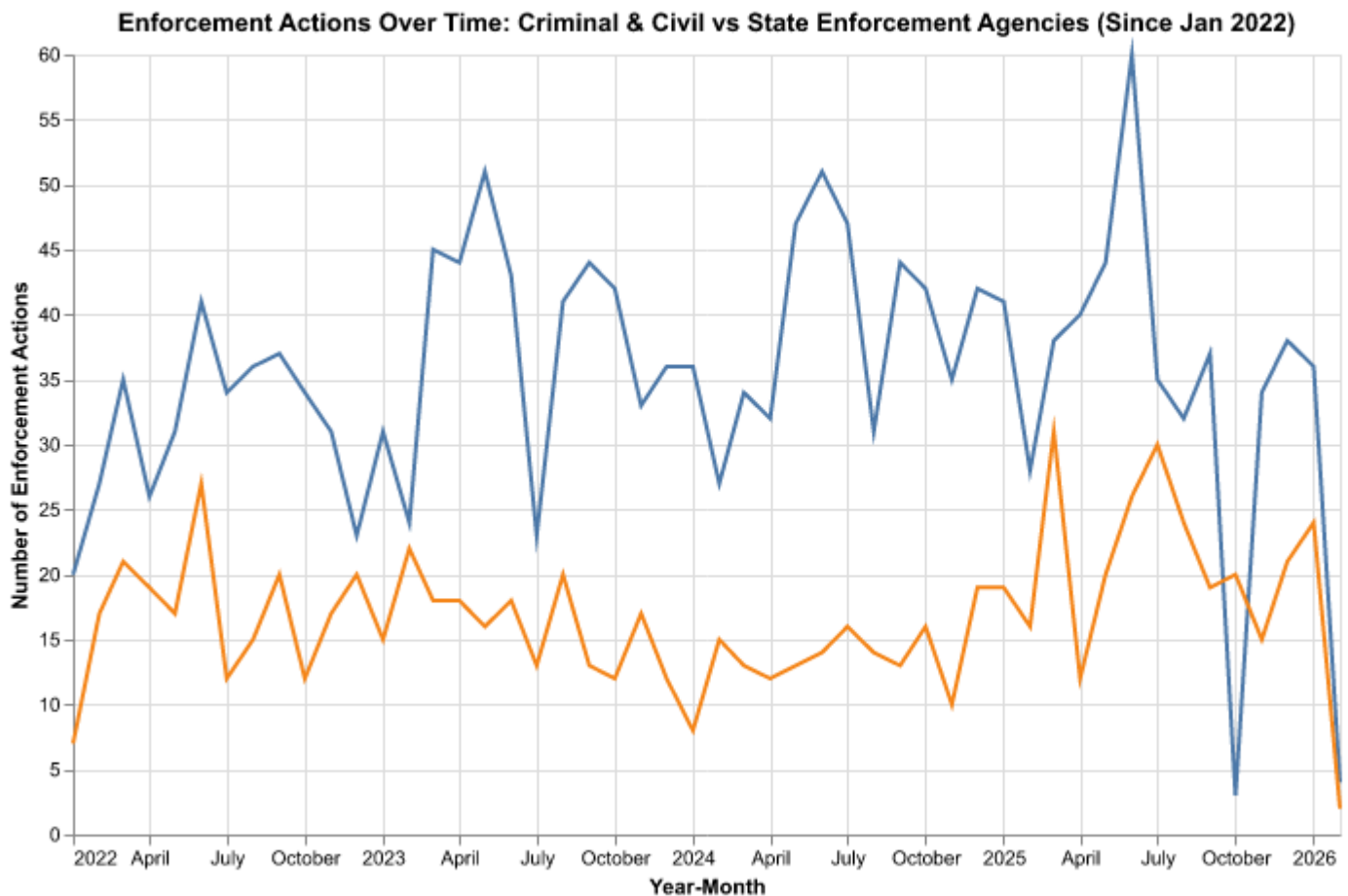
- based on “Criminal and Civil Actions” vs. “State Enforcement Agencies”

```
df_cat_month = (
    df_actions
    .query("category in ['Criminal and Civil Actions', 'State Enforcement
    ↪ Agencies']")
    .assign(year_month=lambda d:
    ↪ d["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["year_month", "category"])
    .size()
    .reset_index(name="count")
)
df_cat_month.head()
```

```

alt.Chart(df_cat_month).mark_line().encode(
    x=alt.X("year_month:T", title="Year-Month"),
    y=alt.Y("count: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("count:Q", title="Actions")
    ]
).properties(
    title="Enforcement Actions Over Time: Criminal & Civil vs State
↪ Enforcement Agencies (Since Jan 2022)",
    width=650,
    height=400
)

```



- based on five topics

```
import numpy as np
df_cc = df_actions[df_actions["category"] == "Criminal and Civil
↳ Actions"].copy()
t = df_cc["title"].fillna("").str.lower()

is_health =
↳ t.str.contains(r"medicare|medicaid|hospital|clinic|physician|doctor|nurse|health|patient
↳ regex=True)
is_fin =
↳ t.str.contains(r"bank|financial|money|loan|credit|wire|fraud|tax|irs|mortgage|securities
↳ regex=True)
is_drug = t.str.contains(r"drug|controlled
↳ substance|opioid|fentanyl|heroin|pill|prescription|dea|distribution|traffick",
↳ regex=True)
is_bribe =
↳ t.str.contains(r"brib|kickback|corrupt|corruption|extortion|racketeer|conspiracy|scheme"
↳ regex=True)

df_cc["topic"] = np.select(
    [is_health, is_fin, is_drug, is_bribe],
    ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
↳ "Bribery/Corruption"],
    default="Other"
)

df_cc["topic"].value_counts()
df_topic_month = (
    df_cc
    .assign(year_month=lambda d:
↳ d["date"].dt.to_period("M").dt.to_timestamp())
    .groupby(["year_month", "topic"])
    .size()
    .reset_index(name="count")
)

topic_order = ["Health Care Fraud", "Financial Fraud", "Drug Enforcement",
↳ "Bribery/Corruption", "Other"]
df_topic_month["topic"] = pd.Categorical(df_topic_month["topic"],
↳ categories=topic_order, ordered=True)

df_topic_month.head()
```

```

alt.Chart(df_topic_month).mark_line().encode(
    x=alt.X("year_month:T", title="Year-Month"),
    y=alt.Y("count:Q", title="Number of Criminal & Civil Actions"),
    color=alt.Color("topic:N", title="Topic", sort=topic_order),
    tooltip=[
        alt.Tooltip("year_month:T", title="Month"),
        alt.Tooltip("topic:N", title="Topic"),
        alt.Tooltip("count:Q", title="Actions")
    ]
).properties(
    title="Criminal & Civil Actions Over Time by Topic (Since Jan 2022)",
    width=650,
    height=400
)

```

