Problem Set 4

Mustafa Siddiqi

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: **___**

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.

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

from datetime import datetime, date
import requests
from bs4 import BeautifulSoup

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

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

Step 1: Develop initial scraper and crawler

```
base_url = "https://oig.hhs.gov"
url = "https://oig.hhs.gov/fraud/enforcement/"

def parse_action_blocks(html: str):
    soup = BeautifulSoup(html, "lxml")

    title_links = soup.select("h2 a")

    rows = []
    for a in title_links:
        title = a.get_text(strip=True)
        href = a.get("href", "")
        link = href if href.startswith("http") else base_url + href

        block = a.find_parent()
        container = block.find_parent(["li", "article", "div"]) or block

        text = container.get_text("\n", strip=True)

        m =
↪   re.search(r"(January|February|March|April|May|June|July|August|September|October|Novembe
↪   text)
        action_date = m.group(0) if m else None
```

```
        possible_cats = []
        for line in text.split("\n"):
            line_clean = line.strip()
            if line_clean in {
                "Child Support",
                "CIA Reportable Events",
                "CMP and Affirmative Exclusions",
                "COVID-19",
                "Criminal and Civil Actions",
                "EMTALA/Patient Dumping",
                "Fraud Self-Disclosures",
                "Grant and Contractor Fraud Self-Disclosures",
                "State Enforcement Agencies",
                "Stipulated Penalties and Material Breaches",
            }:
                possible_cats.append(line_clean)

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

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

    df = pd.DataFrame(rows).drop_duplicates()

    df["date"] = pd.to_datetime(df["date"], errors="coerce")
    df = df.dropna(subset=["title", "date", "link"]).reset_index(drop=True)

    return df

resp = requests.get(url, timeout=30)
df_step1 = parse_action_blocks(resp.text)
```

```
print(df_step1.head())
```

```
                                               title       date  \
0  Houston Transplant Doctor Indicted For Making ... 2026-02-05
1  MultiCare Health System to Pay Millions to Set... 2026-02-04
2  Brooklyn Banker Pleads Guilty to Laundering Pr... 2026-02-03
```

```
3  Delafield Man Sentenced to 18 Months' Imprison... 2026-02-03
4  Former NFL Player Convicted for $197M Medicare... 2026-02-03

                    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

function scrape_since(year, month, run_scraper=True): if year < 2013: print warning and return None

```
if run_scraper is False:
    read the existing csv and return it

start_date = first day of (year, month)
page = 1
all_rows = []

while True:
    build url (page 1 is base; page>1 adds ?page=page)
    request the page
    parse actions on that page into df_page
    append to all_rows

    if df_page is empty: break (no more pages)
    if the minimum date on df_page < start_date:
        break (we have reached earlier than we need)
```

```
    page += 1
    sleep(1)

combine all_rows, filter to date >= start_date
save to enforcement_actions_year_month.csv
return dataframe
```

- b. Create Dynamic Scraper

```python
def scrape_enforcement_actions_since(year: int, month: int, run_scraper: bool
 ↪  = True):
    if year < 2013:
        print("Please restrict to year >= 2013, since only enforcement
         ↪  actions after 2013 are listed.")
        return None

    stop_date = datetime(year, month, 1)

    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {
        "User-Agent": "Mozilla/5.0"
    }

    filename = f"enforcement_actions_{year}_{month:02d}.csv"

    # RUN INDICATOR: don't scrape, just load existing
    if run_scraper is False:
        print(f"run_scraper=False → loading {filename}")
        return pd.read_csv(filename, parse_dates=["date"])

    all_rows = []
    page = 1
    keep_scraping = False

    while keep_scraping:
        url = base_url if page == 1 else f"{base_url}?page={page}"
        print(f"Scraping: {url}")

        response = requests.get(url, headers=headers, timeout=30)
        response.raise_for_status()
        soup = BeautifulSoup(response.text, "lxml")

        entries = soup.find_all("div", class_="usa-card__container")
```

```
    # no entries => no more pages
    if not entries:
        break

    for entry in entries:
        # Title & Link
        title_a = entry.select_one("h2.usa-card__heading a")
        if not title_a:
            continue

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

        # Categories
        cats = []
        for li in entry.select("ul.display-inline.add-list-reset li"):
            cats.append(li.get_text(strip=True))
        category = ", ".join(cats) if cats else None

        # Date (and stop condition)
        date_span = entry.select_one("span.text-base-dark")
        if not date_span:
            continue

        date_text = date_span.get_text(strip=True)
        current_date = datetime.strptime(date_text, "%B %d, %Y")

        # Stop once we hit older than target
        if current_date < stop_date:
            keep_scraping = False
            break

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

    page += 1
    time.sleep(1)
```

```
    df = pd.DataFrame(all_rows).drop_duplicates()
    df = df.sort_values("date").reset_index(drop=True)

    df.to_csv(filename, index=False)
    print(f"Scraping complete. Saved {len(df)} rows to {filename}")

    return df
```

- c. Test Your Code

```
df_2024 = scrape_enforcement_actions_since(2024, 1, run_scraper=False)
print("Total actions since Jan 2024:", len(df_2024))
print("Earliest action since Jan 2024:")
print(df_2024.iloc[0][["date", "title", "category", "link"]])  # earliest now

df_2022 = scrape_enforcement_actions_since(2022, 1, run_scraper=False)
print("Total actions since Jan 2022:", len(df_2022))
print("Earliest action since Jan 2022:")
print(df_2022.iloc[0][["date", "title", "category", "link"]])
```

```
run_scraper=False → loading enforcement_actions_2024_01.csv
Total actions since Jan 2024: 1770
Earliest action since Jan 2024:
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: 0, dtype: object
run_scraper=False → loading enforcement_actions_2022_01.csv
Total actions since Jan 2022: 3360
Earliest action since Jan 2022:
date                                    2022-01-04 00:00:00
title       Integrated Pain Management Medical Group Agree...
category                                  Fraud Self-Disclosures
link        https://oig.hhs.gov/fraud/enforcement/integrat...
Name: 0, dtype: object
```

Step 3: Plot data based on scraped data

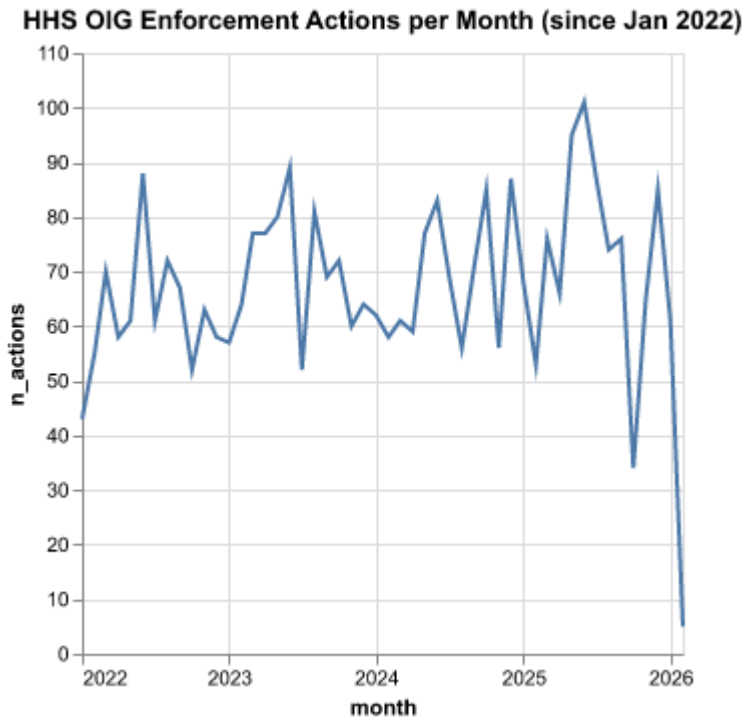
1. Plot the number of enforcement actions over time

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

# month bucket for plotting
df["month"] = df["date"].dt.to_period("M").dt.to_timestamp()

monthly = df.groupby("month").size().reset_index(name="n_actions")

alt.Chart(monthly).mark_line().encode(
    x="month:T",
    y="n_actions:Q"
).properties(
    title="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"

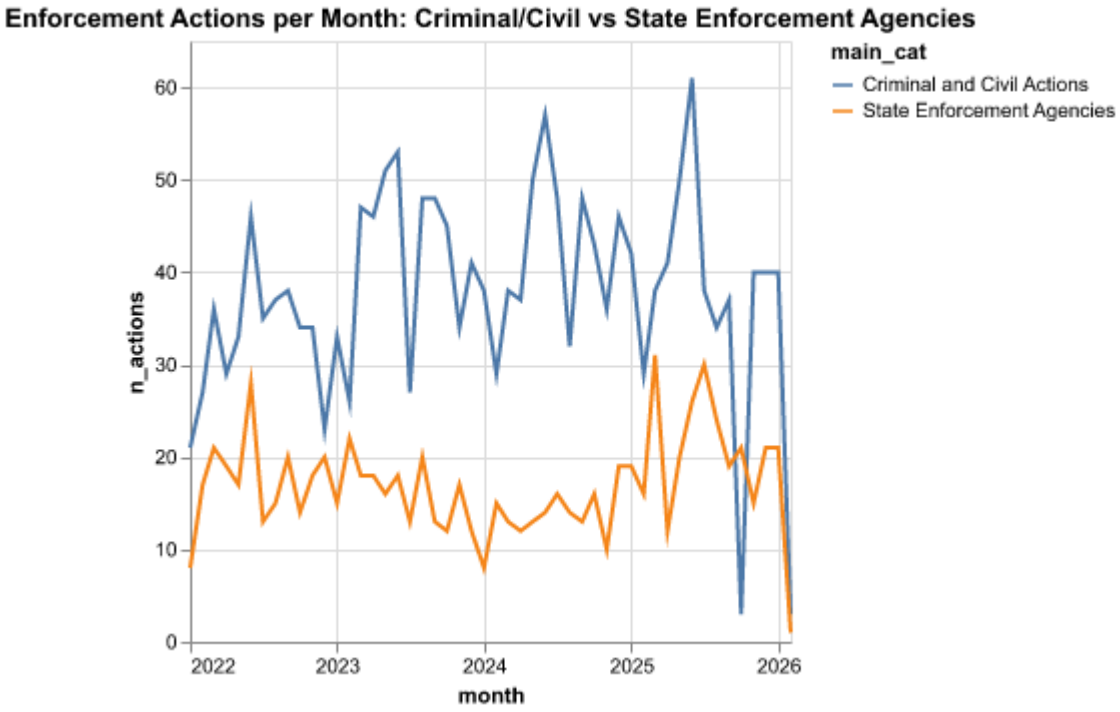
```
keep_cats = ["Criminal and Civil Actions", "State Enforcement Agencies"]
df_cat = df[df["category"].fillna("").apply(lambda s: any(cat in s for cat in
 ↪  keep_cats))].copy()

# If category contains multiple values joined by ";", collapse to the two
 ↪  buckets we care about
def collapse_main_cat(cat_str: str) -> str:
    if "Criminal and Civil Actions" in cat_str:
        return "Criminal and Civil Actions"
    if "State Enforcement Agencies" in cat_str:
        return "State Enforcement Agencies"
    return "Other"

df_cat["main_cat"] = df_cat["category"].fillna("").apply(collapse_main_cat)

monthly_cat = df_cat.groupby(["month",
 ↪  "main_cat"]).size().reset_index(name="n_actions")

alt.Chart(monthly_cat).mark_line().encode(
    x="month:T",
    y="n_actions:Q",
    color="main_cat:N"
).properties(
    title="Enforcement Actions per Month: Criminal/Civil vs State Enforcement
 ↪  Agencies"
)
```



- based on five topics

```
df_cc = df_cat[df_cat["main_cat"] == "Criminal and Civil Actions"].copy()
t = df_cc["title"].str.lower().fillna("")

def topic_from_title(title_lower: str) -> str:
    # These are examples of keyword rules (you can tweak after inspecting
     ↪  titles)
    if any(k in title_lower for k in ["kickback", "medicare", "medicaid",
     ↪  "physician", "hospital", "clinic", "home health", "health care",
     ↪  "healthcare", "provider"]):
        return "Health Care Fraud"
    if any(k in title_lower for k in ["bank", "financial", "wire fraud",
     ↪  "money laundering", "launder", "loan", "tax", "securities",
     ↪  "investment"]):
        return "Financial Fraud"
    if any(k in title_lower for k in ["drug", "opioid", "fentanyl", "pill",
     ↪  "controlled substance", "pharmacy"]):
        return "Drug Enforcement"
    if any(k in title_lower for k in ["bribe", "bribery", "corrupt",
     ↪  "corruption", "kickback scheme with official", "extortion"]):
        return "Bribery/Corruption"
```

```
    return "Other"

df_cc["topic"] = t.apply(topic_from_title)

monthly_topic = df_cc.groupby(["month",
 ↪  "topic"]).size().reset_index(name="n_actions")

alt.Chart(monthly_topic).mark_line().encode(
    x="month:T",
    y="n_actions:Q",
    color="topic:N"
).properties(
    title="Criminal & Civil Actions per Month, by Topic (keyword-based)"
)
```