PS4

Weichen Cai

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: **_W.C_**

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

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

Step 1: Develop initial scraper and crawler

```python
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")

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
    lines = li.get_text("\n", strip=True).split("\n")
    date = lines[1] if len(lines) > 1 else None

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

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

df = pd.DataFrame(rows)
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

Function inputs: year, month, RUN_SCRAPER (indicator)

1. If year < 2013, print a warning and stop (because site only lists actions after 2013).
2. Define start_date = first day of (year, month).
3. If RUN_SCRAPER == False: • Read enforcement_actions_year_month.csv from disk. • Return the dataframe (so knitting is fast).
4. Else (RUN_SCRAPER == True): • Initialize empty list rows and set page_num = 0. • Use a while-loop (not a simple for-loop), because we don't know ahead of time how many pages we must crawl to reach start_date. • While True: • Request the current page HTML (first page has no ?page=…; later pages use ?page=…). • Parse enforcement actions on that page into structured rows (title/date/category/link). • If no actions are found, break. • Convert action dates to datetime. • Append rows that are >= start_date. • If the oldest date on this page is < start_date, we have gone far enough → break. • Increment page_num and sleep(1) before requesting the next page. • Combine all collected rows into a dataframe, drop duplicates by link. • Save as enforcement_actions_year_month.csv (do not git commit). • Return dataframe.

- b. Create Dynamic Scraper

```python
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:
    """
    Parse a single enforcement-actions page and return a DataFrame
    with columns: title, date, category, link.
    """
    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

        # The date appears on the line immediately below the title
        lines = li.get_text("\n", strip=True).split("\n")
        date_str = lines[1] if len(lines) > 1 else None

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

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

    df = pd.DataFrame(rows)

    # Convert date to datetime for filtering and aggregation
    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 a given (year, month).

    Results are cached to a CSV file whose name reflects the input
    year and month, e.g.:
        enforcement_actions_since_2022_01.csv

    Parameters
    ----------
    year : int
        Starting year (must be >= 2013).
    month : int
        Starting month (1-12).
    RUN_SCRAPER : bool
        If False, read the existing CSV from disk.
        If True, run the web scraper and overwrite the CSV.

    Returns
    -------
    pandas.DataFrame
        Enforcement actions since the given year and month.
    """

    # The website does not list enforcement actions prior to 2013
    if year < 2013:
        raise ValueError("Year must be >= 2013.")

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

    # Indicator pattern: avoid re-running the scraper when knitting
    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_num = 0

    # Use a while-loop because the total number of pages is unknown
    while True:
        if page_num == 0:
            url = START_URL
        else:
            url = f"{START_URL}?page={page_num}"

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

        df_page = _parse_page(soup)

        # Stop if the page contains no enforcement actions
        if df_page.empty:
            break
        # Keep only actions on or after the start date
        df_keep = df_page[df_page["date"] >= start_date]
        if not df_keep.empty:
            collected.append(df_keep)

        # Stop once the oldest date on the page is earlier than start_date
        oldest_date = df_page["date"].min()
        if pd.notna(oldest_date) and oldest_date < start_date:
            break

        page_num += 1

        # Be polite to the server (lecture recommendation)
        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"])

    # Save results for reuse in later steps
    out.to_csv(out_csv, index=False)

    return out
```

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

print("How many enforcement actions in final dataframe:", df_2024.shape[0])

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

print("Earliest enforcement action date:", earliest_2024["date"])
print("Title:", earliest_2024["title"])
print("Category:", earliest_2024["category"])
print("Link:", earliest_2024["link"])
```

```
How many enforcement actions in final dataframe: 1787
Earliest enforcement action date: 2024-01-03 00:00:00
Title: 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-pati
```

The final dataframe contains 1,787 enforcement actions. The earliest enforcement action scraped in this dataset is dated January 3, 2024. It is titled "Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia" and falls under the category State Enforcement Agencies. The full details of this enforcement action can be found at: 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("How many enforcement actions in final dataframe:", df_2022.shape[0])

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

print("Earliest enforcement action date:", earliest_2022["date"])
print("Title:", earliest_2022["title"])
print("Category:", earliest_2022["category"])
print("Link:", earliest_2022["link"])
```

```
How many enforcement actions in final dataframe: 3377
Earliest enforcement action date: 2022-01-04 00:00:00
Title: 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-
```

The final dataframe contains 3,377 enforcement actions. The earliest enforcement action scraped in this dataset is dated January 4, 2022. It is titled "Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals" and falls under the category Fraud Self-Disclosures. The full details of this enforcement action can be found at: 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/

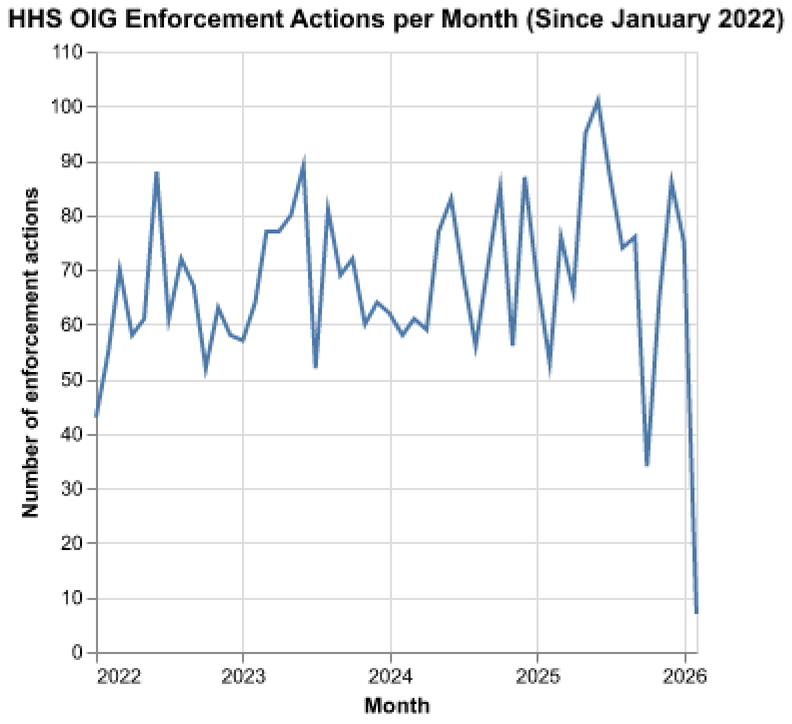
Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
df_2022 = df_2022.copy()
df_2022["yearmonth"] = df_2022["date"].dt.to_period("M").dt.to_timestamp()

monthly_overall = (
    df_2022
    .groupby("yearmonth")
    .size()
    .reset_index(name="n_actions")
)

monthly_overall.head()
```

	yearmonth	n_actions
0	2022-01-01	43
1	2022-02-01	55
2	2022-03-01	70
3	2022-04-01	58
4	2022-05-01	61

```
chart_overall = (
    alt.Chart(monthly_overall)
    .mark_line()
    .encode(
        x=alt.X("yearmonth:T", title="Month"),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions")
    )
    .properties(
        title="HHS OIG Enforcement Actions per Month (Since January 2022)"
    )
)

chart_overall
```



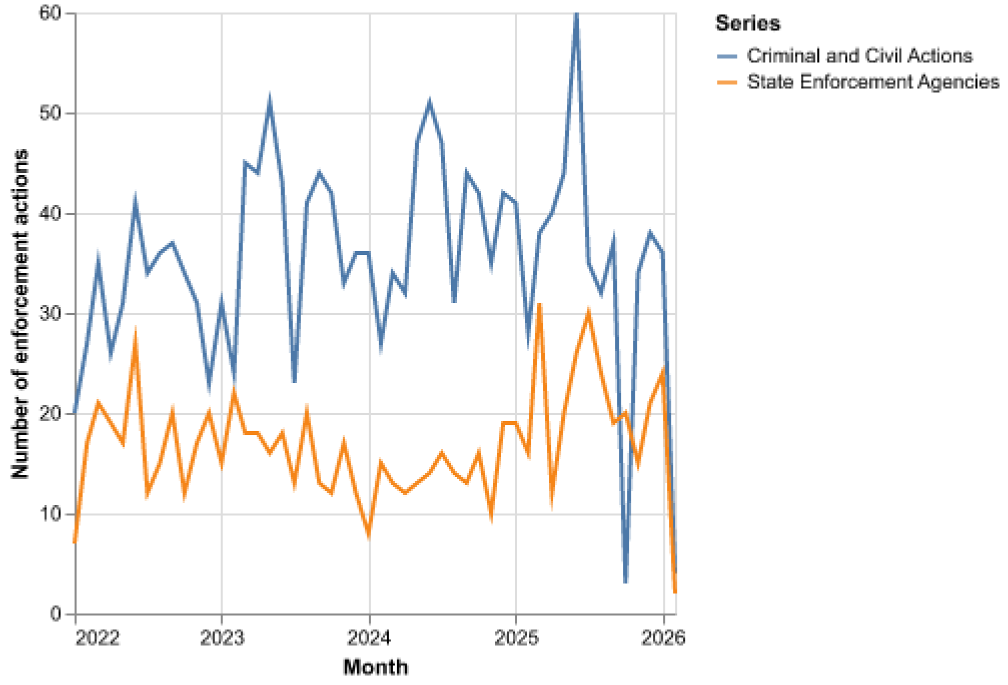
2. Plot the number of enforcement actions categorized:

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

```
df_2022["main_category"] = None

df_2022.loc[
    df_2022["category"] == "Criminal and Civil Actions",
    "main_category"
] = "Criminal and Civil Actions"

df_2022.loc[
    df_2022["category"] == "State Enforcement Agencies",
    "main_category"
] = "State Enforcement Agencies"

monthly_main = (
    df_2022[df_2022["main_category"].notna()]
    .groupby(["yearmonth", "main_category"])
    .size()
    .reset_index(name="n_actions")
    .rename(columns={"main_category": "series"})
)
```

```
chart_split = (
    alt.Chart(monthly_main)
    .mark_line()
    .encode(
        x=alt.X("yearmonth:T", title="Month"),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
        color=alt.Color("series:N", title="Series")
    )
    .properties(
        title="Enforcement Actions per Month by Category and Criminal Topic
 ↪  (Since January 2022)"
    )
)

chart_split
```

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

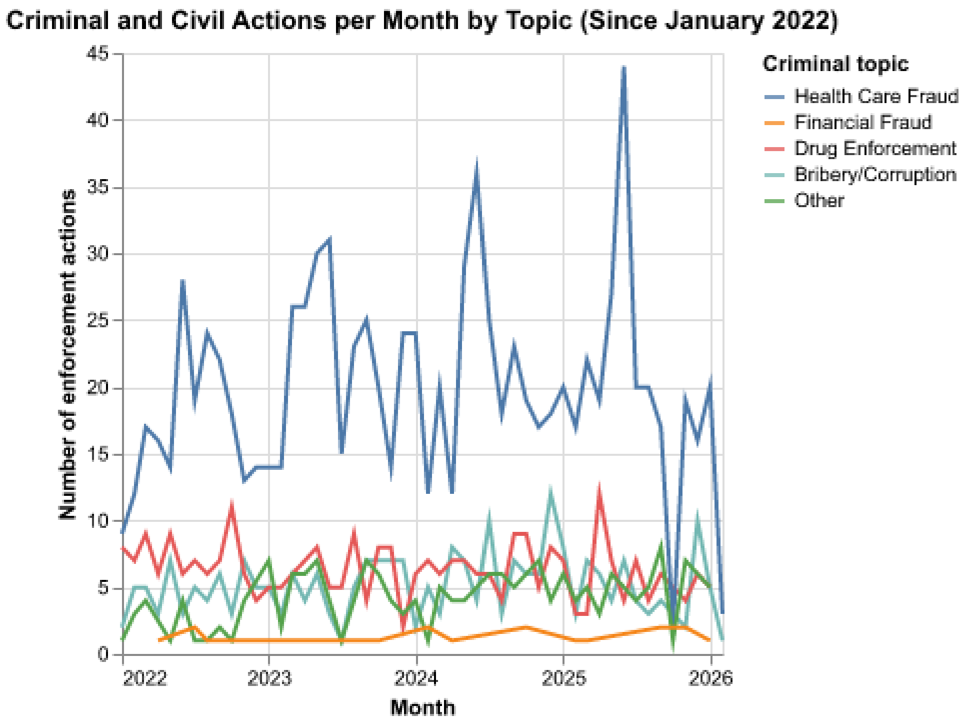


- based on five topics

```python
df_2022 = df_2022.copy()

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

    # Drug Enforcement
    drug_kw = [
        "opioid", "fentanyl", "oxycodone", "hydrocodone",
        "controlled substance", "drug", "pill", "pharmacy"
    ]
    if any(k in t for k in drug_kw):
        return "Drug Enforcement"

    # Bribery / Corruption
    bribery_kw = [
        "kickback", "bribe", "corruption", "extortion",
        "money laundering", "launder", "embezzle"
    ]
    if any(k in t for k in bribery_kw):
        return "Bribery/Corruption"
    # Financial Fraud
    financial_kw = [
        "wire fraud", "bank", "loan", "credit", "mortgage",
        "securities", "investment", "crypto", "bitcoin"
    ]
    if any(k in t for k in financial_kw):
        return "Financial Fraud"

    # Health Care Fraud
    health_kw = [
        "medicare", "medicaid", "health", "hospital", "clinic",
        "physician", "doctor", "nurse", "patient", "billing", "claims"
    ]
    if any(k in t for k in health_kw):
        return "Health Care Fraud"

    return "Other"

df_2022["criminal_topic"] = df_2022["title"].apply(classify_criminal_topic)
```

```
# Ensure yearmonth exists
df_2022["yearmonth"] = df_2022["date"].dt.to_period("M").dt.to_timestamp()

df_criminal = df_2022[df_2022["category"] == "Criminal and Civil
 ↪  Actions"].copy()

monthly_topics_only = (
    df_criminal
    .groupby(["yearmonth", "criminal_topic"])
    .size()
    .reset_index(name="n_actions")
)

topic_order = [
    "Health Care Fraud",
    "Financial Fraud",
    "Drug Enforcement",
    "Bribery/Corruption",
    "Other"
]
```

```
chart_topics = (
    alt.Chart(monthly_topics_only)
    .mark_line()
    .encode(
        x=alt.X("yearmonth:T", title="Month"),
        y=alt.Y("n_actions:Q", title="Number of enforcement actions"),
        color=alt.Color("criminal_topic:N", title="Criminal topic",
↪   sort=topic_order)
    )
    .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)
Number of enforcement actions
Month
Criminal topic
Health Care Fraud
Financial Fraud
Drug Enforcement
Bribery/Corruption
Other