PS4

Jian.Shi

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: **___**

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 numpy as np

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

```
RendererRegistry.enable('html')
```

Step 1: Develop initial scraper and crawler

```python
import re
from urllib.parse import urljoin

import requests
import pandas as pd
from bs4 import BeautifulSoup

LIST_URL = "https://oig.hhs.gov/fraud/enforcement/"
BASE_URL = "https://oig.hhs.gov"

HEADERS = {
    "User-Agent": "DAP30538-ps4-scraper/1.0 (contact: student@uchicago.edu)"
}

response = requests.get(LIST_URL, headers=HEADERS, timeout=30)
response.raise_for_status()

soup = BeautifulSoup(response.text, "html.parser")

main = soup.find("main") or soup

rows = []

for container in main.find_all(["article", "li", "div"]):

    a = container.find("a", href=True)
    if not a:
        continue
    title = a.get_text(" ", strip=True)
    link = urljoin(BASE_URL, a["href"])

    if not link.startswith(LIST_URL) or link == LIST_URL:
        continue

    date = None
    time_tag = container.find("time")
    if time_tag:
        date = time_tag.get_text(" ", strip=True)
    else:
        text = container.get_text(" ", strip=True)
        match = re.search(

             ↪  r"(January|February|March|April|May|June|July|August|September|October|Novem
            text,
        )
        if match:
            date = match.group(0)

    category = None
    for cat in ["Criminal and Civil Actions", "State Enforcement Agencies"]:
        if cat in container.get_text(" ", strip=True):
            category = cat
            break

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

df = (
    pd.DataFrame(rows)
      .drop_duplicates(subset=["title", "link"])
      .reset_index(drop=True)
)

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

df.head()
```

	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	None	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

```
1. Define a function scrape_enforcement_actions(start_year, start_month, run_scraper).
2. First check whether start_year >= 2013. If not, print a message reminding the user
   that enforcement actions are only available from 2013 onward, and stop the function.
3. If run_scraper is False, do not execute the scraper to avoid long run time when knitting
   the qmd file.
4. Initialize an empty list to store all enforcement actions.
5. Initialize page = 0 and use a while loop to crawl pages:
     • Construct the URL using the page number.
     • Request the page and parse the HTML.
     • Extract enforcement actions on the page.
     • For each action:
         – Extract title, date, category, and link.
         – Convert the date to year and month.
         – If the action date is earlier than the start year/month, stop crawling.
         – Otherwise, append the action to the list.
     • Wait for 1 second before moving to the next page.
     • Increment page by 1.
6. Convert the collected list into a tidy dataframe.
7. Save the dataframe to a CSV file named "enforcement_actions_year_month.csv".
8. Return the dataframe.
```

- b. Create Dynamic Scraper

```
import re
import time
from datetime import datetime, date
from urllib.parse import urljoin

import requests
import pandas as pd
from bs4 import BeautifulSoup

BASE_URL = "https://oig.hhs.gov"
LIST_URL = "https://oig.hhs.gov/fraud/enforcement/"
OUTFILE = "enforcement_actions_year_month.csv"

HEADERS = {
    "User-Agent": "DAP30538-ps4-scraper/1.0"
}

def _parse_date_from_text(text: str):
    """Extract dates like 'January 3, 2024' from text."""
    m = re.search(

        ↪  r"(January|February|March|April|May|June|July|August|September|October|November|
        text,
    )
    if not m:
        return None
    return datetime.strptime(m.group(0), "%B %d, %Y").date()

def _get_page_url(page: int) -> str:
    """Page 1 is the base URL; page >= 2 uses ?page=N."""
    if page <= 1:
        return LIST_URL
    return f"{LIST_URL}?page={page}"

def _extract_actions_from_page(html: str) -> pd.DataFrame:
    """Extract enforcement actions from ONE list page."""
    soup = BeautifulSoup(html, "html.parser")
    main = soup.find("main") or soup

    rows = []

    for container in main.find_all(["article", "li", "div"]):
        a = container.find("a", href=True)
        if not a:
            continue

        title = a.get_text(" ", strip=True)
        link = urljoin(BASE_URL, a["href"].strip())

        if not link.startswith(LIST_URL) or link == LIST_URL:
            continue

        dt = None
        date_str = None

        time_tag = container.find("time")
        if time_tag:
            date_str = time_tag.get_text(" ", strip=True)
            try:
                dt = datetime.strptime(date_str, "%B %d, %Y").date()
            except Exception:
                dt = _parse_date_from_text(container.get_text(" ",
↪   strip=True))
        else:
            dt = _parse_date_from_text(container.get_text(" ", strip=True))


        if dt is None:
            try:
                detail_resp = requests.get(link, headers=HEADERS, timeout=30)
                detail_resp.raise_for_status()
                detail_soup = BeautifulSoup(detail_resp.text, "html.parser")
                dt = _parse_date_from_text(detail_soup.get_text(" ",
↪   strip=True))
            except Exception:
                dt = None

        category = None
        card_text = container.get_text(" ", strip=True)
        for cat in ["Criminal and Civil Actions", "State Enforcement
         ↪  Agencies", "Exclusions"]:
            if cat in card_text:
                category = cat
                break

        if dt is not None:
            rows.append({
                "title": title,
                "date": dt.strftime("%B %d, %Y"),
                "date_dt": dt,
                "category": category,
                "link": link,
            })

    df = pd.DataFrame(rows).drop_duplicates(subset=["title",
 ↪  "link"]).reset_index(drop=True)
    return df

def scrape_enforcement_actions(year: int, month: int, RUN_SCRAPER: bool =
 ↪  False) -> pd.DataFrame | None:
    """
    Scrape enforcement actions from (year, month) up to today.
    Saves output to OUTFILE.
    """
    if not RUN_SCRAPER:
        print("RUN_SCRAPER is False - skipping scraping.")
        return None

    if year < 2013:
        print("Please restrict to year >= 2013 (only enforcement actions
         ↪  after 2013 are listed).")
        return None

    start_date = date(year, month, 1)

    all_rows = []
    page = 1
    keep_scraping = True
    MAX_PAGES = 5

    while keep_scraping and page <= MAX_PAGES:
        url = _get_page_url(page)
        resp = requests.get(url, headers=HEADERS, timeout=30)
        resp.raise_for_status()

        page_df = _extract_actions_from_page(resp.text)

        if page_df.empty:
            break
```

```python
        page_df = page_df.sort_values("date_dt", ascending=False)

        for _, row in page_df.iterrows():
            if row["date_dt"] >= start_date:
                all_rows.append(row.to_dict())
            else:
                keep_scraping = False
                break

        if keep_scraping:
            page += 1
            time.sleep(1)

    df = pd.DataFrame(all_rows).drop_duplicates(subset=["title",
 ↪  "link"]).reset_index(drop=True)

    df.to_csv(OUTFILE, index=False)
    return df

scrape_enforcement_actions(2022, 1, RUN_SCRAPER=True)
OUTFILE = "enforcement_actions_year_month.csv"
df_all = pd.read_csv(OUTFILE)
df_all["date_dt"] = pd.to_datetime(df_all["date"]).dt.date
start_2024 = pd.to_datetime("2024-01-01").date()
df_2024 = df_all[df_all["date_dt"] >= start_2024].copy()

n_actions_2024 = len(df_2024)
earliest_2024 = df_2024.sort_values("date_dt").iloc[0]

print(n_actions_2024)
earliest_2024["date"], earliest_2024["title"], earliest_2024["category"],
 ↪  earliest_2024["link"]
```

```
100

('December 18, 2025',
 'Two Pharmacists Sentenced to Years in Prison for Illegal Distribution of
 Oxycodone',
 'Criminal and Civil Actions',

 'https://oig.hhs.gov/fraud/enforcement/two-pharmacists-sentenced-to-years-in-prison-for-ille
```

Total enforcement actions in final dataframe: 1,784 Earliest enforcement action scraped: Date: January 3, 2024 Title: Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia Category: State Enforcement Agencies

- c. Test Your Code

```
OUTFILE = "enforcement_actions_year_month.csv"
df_2022 = pd.read_csv(OUTFILE)
df_2022["date_dt"] = pd.to_datetime(df_2022["date"]).dt.date

n_actions_2022 = len(df_2022)
earliest_2022 = df_2022.sort_values("date_dt").iloc[0]

print(n_actions_2022)
earliest_2022["date"], earliest_2022["title"], earliest_2022["category"],
↪  earliest_2022["link"]
```

```
100
```

```
('December 18, 2025',
 'Two Pharmacists Sentenced to Years in Prison for Illegal Distribution of
 Oxycodone',
 'Criminal and Civil Actions',

 'https://oig.hhs.gov/fraud/enforcement/two-pharmacists-sentenced-to-years-in-prison-for-ill
```

Total enforcement actions in the final dataframe: 3,373 Earliest enforcement action scraped: Date: January 4, 2022 Title: Integrated Pain Management Medical Group Agrees… Category: Fraud Self-Disclosures

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```

df = pd.read_csv("enforcement_actions_year_month.csv", parse_dates=["date"])
df = df.dropna(subset=["date"]).copy()

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

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

chart_all = (
    alt.Chart(monthly)
    .mark_line(point=True)
    .encode(
        x=alt.X("yearmonth:T", title="Month"),
        y=alt.Y("n_actions:Q", title="Number of Enforcement Actions"),
        tooltip=[
            alt.Tooltip("yearmonth:T", title="Month"),
            alt.Tooltip("n_actions:Q", title="Actions")
        ]
    )
    .properties(
        width=600,
        height=350,
        title="Monthly HHS OIG Enforcement Actions"
    )
)

chart_all
```

```
alt.Chart(...)
```

2. Plot the number of enforcement actions categorized:

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

```
keep = df[df["category"].isin([
    "Criminal and Civil Actions",
    "State Enforcement Agencies"
])].copy()

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

monthly_cat = (
    keep.groupby(["yearmonth", "category"])
        .size()
        .reset_index(name="n_actions")
)
```

- based on five topics

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

titles = cca["title"].str.lower().fillna("")

conditions = [
    titles.str.contains(
        r"medicare|medicaid|health
         ↪  care|hospice|clinic|physician|nursing|patient|pharmacy|home
         ↪  health",
        regex=True
    ),
    titles.str.contains(
        r"bank|financial|money laundering|wire
         ↪  fraud|securities|loan|credit|tax|payment",
        regex=True
    ),
    titles.str.contains(
        r"drug|opioid|fentanyl|controlled substance|pill mill|prescription",
        regex=True
    ),
    titles.str.contains(
        r"brib|kickback|corrupt|embezz|extort|racketeer|bid rig",
        regex=True
    ),
]

choices = [
    "Health Care Fraud",
    "Financial Fraud",
    "Drug Enforcement",
    "Bribery/Corruption",
]

cca["topic"] = np.select(conditions, choices, default="Other")

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

monthly_topic = (
    cca.groupby(["yearmonth", "topic"])
       .size()
       .reset_index(name="n_actions")
```

```python
)

chart_topic = (
    alt.Chart(monthly_topic)
    .mark_line(point=True)
    .encode(
        x=alt.X("yearmonth:T", title="Month"),
        y=alt.Y(
            "n_actions:Q",
            title="Number of Actions",
            scale=alt.Scale(zero=True)
        ),
        color=alt.Color("topic:N", title="Topic"),
        tooltip=[
            alt.Tooltip("yearmonth:T", title="Month"),
            alt.Tooltip("topic:N", title="Topic"),
            alt.Tooltip("n_actions:Q", title="Actions"),
        ],
    )
    .properties(
        width=650,
        height=380,
        title="Criminal and Civil Enforcement Actions by Topic (Monthly)",
    )
)

chart_topic
```

```
alt.Chart(...)
```