PS4

Hani C

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: HC

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
from datetime import datetime
import matplotlib.pyplot as plt


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/"
response = requests.get(url)
soup = BeautifulSoup(response.content, 'html.parser')

cards = soup.find_all('div', class_='usa-card__container')

results = []

for card in cards:
    heading = card.find('h2', class_='usa-card__heading')
    title = heading.get_text(strip=True)

    link = "https://oig.hhs.gov" + heading.find('a')['href']

    info_div = card.find('div', class_='font-body-sm')

    date = info_div.find('span',
↪   class_='text-base-dark').get_text(strip=True)
    category = info_div.find('li', class_='usa-tag').get_text(strip=True)

    results.append({
        "Title": title,
        "Date": date,
```

```
        "Category": category,
        "Link": link
    })

df = pd.DataFrame(results)
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

SET target_date TO user-inputted month and year. IF start_year < 2013, PRINT warning and STOP

INITIALIZE all_actions (empty list), current_page (1), and keep_scraping (True).

WHILE keep_scraping IS True: WAIT 1 second.

FETCH HTML for current_page.

FOR EACH action on page: EXTRACT Title, Date, Category, and Link.

IF action date >= target_date: APPEND to all_actions.

ELSE: SET keep_scraping TO False and BREAK loop.

END FOR INCREMENT current_page by 1.

END WHILE

CONVERT all_actions to dataframe.

SAVE to "enforcement_actions_year_month.csv".

- b. Create Dynamic Scraper

Total actions collected: 1770

Date and details of the earliest enforcement action scraped:

Title: Former Nurse Aide Indicted In Death Of Clarksville Patient Arrested In Georgia

Date: 2024-01-03

Category: State Enforcement Agencies

Link: https://oig.hhs.gov/fraud/enforcement/former-nurse-aide-indicted-in-death-of-clarksville-patient-arrested-in-georgia/

```
run_scraper = False

def scrape_oig_actions(start_month, start_year):
    if start_year < 2013:
        print("Reminding the user to restrict to year >= 2013, as only
        ↪  actions after 2013 are listed.") [cite: 42]
        return None

    target_date = datetime(start_year, start_month, 1)
    all_results = []
    page = 1
    keep_scraping = True

    while keep_scraping:
        url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"

        time.sleep(1)

        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        cards = soup.find_all('div', class_='usa-card__container')

        if not cards:
            break
```

```
        for card in cards:
            heading = card.find('h2', class_='usa-card__heading')
            title = heading.get_text(strip=True)
            link = "https://oig.hhs.gov" + heading.find('a')['href']

            date_str = card.find('span',
↪   class_='text-base-dark').get_text(strip=True)
            date_obj = datetime.strptime(date_str, '%B %d, %Y')

            category = card.find('li', class_='usa-tag').get_text(strip=True)

            if date_obj >= target_date:
                all_results.append({
                    "Title": title,
                    "Date": date_obj,
                    "Category": category,
                    "Link": link
                })
            else:
                keep_scraping = False
                break

        page += 1

    df = pd.DataFrame(all_results)
    df.to_csv(f"enforcement_actions_{start_year}_{start_month}.csv",
↪   index=False)
    return df

if run_scraper:
    df_2024 = scrape_oig_actions(1, 2024)
    print(f"Total actions collected: {len(df_2024)}")
    print(df_2024.tail(1)[['Date', 'Title']])
```

- c. Test Your Code

Total actions collected: 3360

Date and details of the earliest enforcement action scraped:

Title: Integrated Pain Management Medical Group Agreed to Pay $10,000 for Allegedly Violating the Civil Monetary Penalties Law by Employing Excluded Individuals

Date: 2022-01-04

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/

```
run_scraper = False

def scrape_oig_actions(start_month, start_year):
    if start_year < 2013:
        print("Reminding the user to restrict to year >= 2013, as only
        ↪  actions after 2013 are listed.") [cite: 42]
        return None

    target_date = datetime(start_year, start_month, 1)
    all_results = []
    page = 1
    keep_scraping = True

    while keep_scraping:
        url = f"https://oig.hhs.gov/fraud/enforcement/?page={page}"

        time.sleep(1)

        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
```

```
        cards = soup.find_all('div', class_='usa-card__container')

        if not cards:
            break

        for card in cards:
            heading = card.find('h2', class_='usa-card__heading')
            title = heading.get_text(strip=True)
            link = "https://oig.hhs.gov" + heading.find('a')['href']

            date_str = card.find('span',
↪   class_='text-base-dark').get_text(strip=True)
            date_obj = datetime.strptime(date_str, '%B %d, %Y')

            category = card.find('li', class_='usa-tag').get_text(strip=True)

            if date_obj >= target_date:
                all_results.append({
                    "Title": title,
                    "Date": date_obj,
                    "Category": category,
                    "Link": link
                })
            else:
                keep_scraping = False
                break

        page += 1

    df = pd.DataFrame(all_results)
    df.to_csv(f"enforcement_actions_{start_year}_{start_month}.csv",
↪   index=False)
    return df

if run_scraper:
    df_2022 = scrape_oig_actions(1, 2022)
    print(f"Total actions collected: {len(df_2022)}")
    print(df_2022.tail(1)[['Date', 'Title']])
```

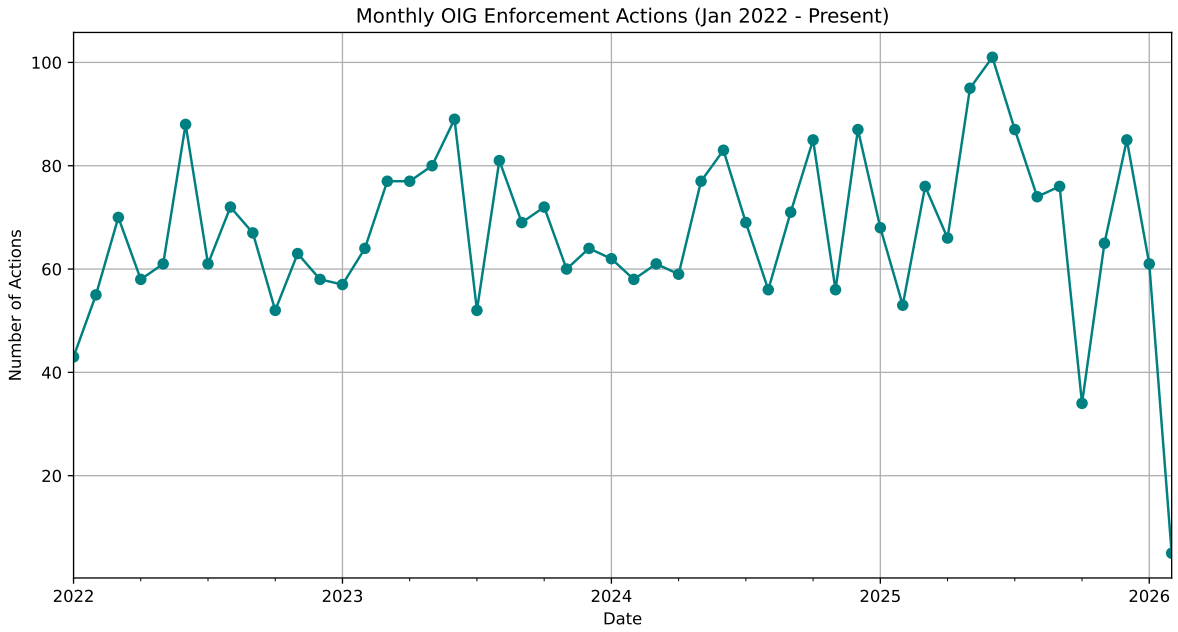
Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
df = pd.read_csv("enforcement_actions_2022_1.csv")
df['Date'] = pd.to_datetime(df['Date'])

df['YearMonth'] = df['Date'].dt.to_period('M')
monthly_counts = df.groupby('YearMonth').size()

plt.figure(figsize=(12, 6))
monthly_counts.plot(kind='line', marker='o', color='teal')
plt.title('Monthly OIG Enforcement Actions (Jan 2022 - Present)')
plt.xlabel('Date')
plt.ylabel('Number of Actions')
plt.grid(True)
plt.show()
```



2. Plot the number of enforcement actions categorized:

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

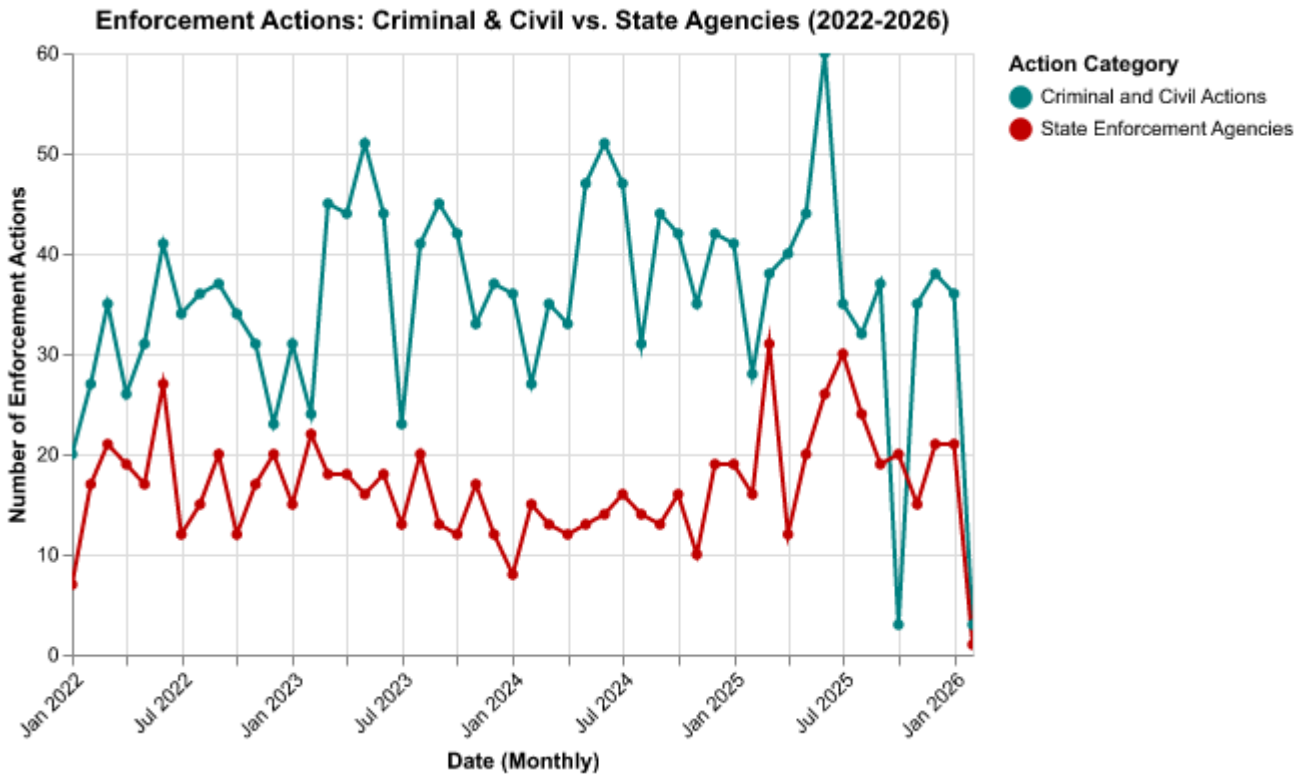
```
target_categories = ["Criminal and Civil Actions", "State Enforcement
 ↪  Agencies"]
df_filtered = df[df['Category'].isin(target_categories)].copy()

df_filtered['Month'] =
 ↪  df_filtered['Date'].dt.to_period('M').dt.to_timestamp()

plot_data = df_filtered.groupby(['Month',
 ↪  'Category']).size().reset_index(name='Count')

chart_3b1 = alt.Chart(plot_data).mark_line(point=True).encode(
    x=alt.X('Month:T',
            title='Date (Monthly)',
            axis=alt.Axis(format='%b %Y', labelAngle=-45)),
    y=alt.Y('Count:Q',
            title='Number of Enforcement Actions'),
    color=alt.Color('Category:N',
                    title='Action Category',
                    scale=alt.Scale(range=['#008080', '#C00000'])),
    tooltip=['Month', 'Category', 'Count']
).properties(
    title='Enforcement Actions: Criminal & Civil vs. State Agencies
 ↪  (2022-2026)',
    width=450,
    height=300
).interactive()

chart_3b1
```



- based on five topics

```
df_cc = df[df['Category'] == "Criminal and Civil Actions"].copy()

def get_topic(title):
    t = title.lower()
    if any(k in t for k in ['drug', 'opioid', 'fentanyl', 'pill',
     ↪  'counselor']): return 'Drug Enforcement'
    if any(k in t for k in ['medicare', 'medicaid', 'physician', 'doctor',
     ↪  'nurse', 'health']): return 'Health Care Fraud'
    if any(k in t for k in ['bank', 'financial', 'loan', 'ppp', 'tax',
     ↪  'money']): return 'Financial Fraud'
    if any(k in t for k in ['bribe', 'kickback', 'corruption']): return
     ↪  'Bribery/Corruption'
    return 'Other'

df_cc['Topic'] = df_cc['Title'].apply(get_topic)

df_cc['Month'] = df_cc['Date'].dt.to_period('M').dt.to_timestamp()
topic_trends = df_cc.groupby(['Month',
 ↪  'Topic']).size().reset_index(name='Count')
```

```
chart_topics = alt.Chart(topic_trends).mark_line(point=True).encode(
    x=alt.X('Month:T', title='Month and Year'),
    y=alt.Y('Count:Q', title='Number of Actions'),
    color=alt.Color('Topic:N', title='Topic Area'),
    tooltip=['Month', 'Topic', 'Count']
).properties(
    title='Trends in Criminal and Civil Enforcement Topics (2022-2026)',
    width=450,
    height=300
).interactive()

chart_topics.display()
```