PS 4

Amanda Gu

2026-02-03

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: A.G.

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 bs4 import BeautifulSoup
from datetime import datetime

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

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

Step 1: Develop initial scraper and crawler

(40%) Step 1: Develop initial scraper and crawler Scraping: Go to the first page of the HHS OIG's "Enforcement Actions" page and scrape and collect the following into a dataset: * Title of the enforcement action * Date * Category (e.g,"Criminal and Civil Actions") * Link associated with the enforcement action. Collect your output into a tidy dataframe and print its head. Hint: if you go to James A. Robinson's profile page at the Nobel Prize website here, right-click anywhere along the line "Affiliation at the time of the award: University of Chicago, Chicago, IL, USA", and select Inspect, you'll see that this affiliation information is located at the third

tag out of five

tags under the

. Think about how you can select the third element of

out of five

elements so you're sure you scrape the affiliation information, not other. This way, you can scrape the name of agency to answer this question.

```
import requests
with open(r"C:/Users/amand\student30538-w26/ps/ps4/Enforcement Actions _
 ↪  Office of Inspector General _ Government Oversight _ U.S. Department of
 ↪  Health and Human Services.html", 'r') as page:
  text = page.read()
```

```
soup = BeautifulSoup(text, 'lxml')
```

```
all_li = soup.find_all("li", class_="usa-card card--list pep-card--minimal
 ↪  mobile:grid-col-12")
```

```
from tabulate import tabulate
data = [{
  "title": li.find("h2",
   ↪  class_='usa-card__heading').find('a').get_text(strip=True),
  'url': li.find("h2", class_='usa-card__heading').find('a')['href'],
  'date' : li.find("span", class_= "text-base-dark").get_text(strip=True),
  'enforcement_type' : ", ".join(tag.get_text(strip=True) for tag in
   ↪  li.find("ul").find_all('li'))
  }
  for li in all_li
  ]
df_data = pd.DataFrame(data)
pd.set_option('display.max_colwidth', 20)
df_data.head()
```

	title	url	date	enforcement_type
0	Delafield Man Se...	/fraud/enforceme...	February 3, 2026	Criminal and Civ...
1	AG's Office Secu...	/fraud/enforceme...	February 2, 2026	State Enforcemen...
2	Florida Man Plea...	/fraud/enforceme...	January 30, 2026	Criminal and Civ...
3	Yadkinville Woma...	/fraud/enforceme...	January 29, 2026	Criminal and Civ...
4	Slidell Chiropra...	/fraud/enforceme...	January 28, 2026	COVID-19, Crimin...

Step 2: Making the scraper dynamic

(40%) Step 2: Making the scraper dynamic 1. Turning the scraper into a function: You will write a function that takes as input a month and a year, and then pulls and formats the enforcement actions like in Step 1 starting from that month+year to today.

• It is very important to make sure that you include an indicator whether or not to actually run the function. If you do not include an indicator, then each time you try to knit the qmd file, the scraper will run, and it will take a very long time to compile your pdf. Instead, run the function once to create a file called enforcement_actions_year_month.csv, which you can use in subsequent parts of the pset. Then turn the indicator off so that knitting your qmd file goes smoothly.

• This function should first check that the year inputted >= 2013 before starting to scrape. If the year inputted < 2013, it should print a statement reminding the user to restrict to year >= 2013, since only enforcement actions after 2013 are listed.

• It should save the dataframe output into a .csv file named as "enforcement_actions_year_month.csv" (do not commit this file to git)

• If you're crawling multiple pages, always add 1 second wait before going to the next page to prevent potential server-side block. To implement this in Python, you may look up .sleep() function from time library.

1. Turning the scraper into a function

- a. Pseudo-Code

a. Before writing out your function, write down pseudo-code of the steps that your function will go through. If you use a loop, discuss what kind of loop you will use and how you will define it. Hint: Note that a simple for loop may not be sufficient for what this crawler requires. Use online resources to look into different types of loops or different ways of using for loops to see if there is something that is more appropriate for this task.

Pseudo-Code create a DEF function that takes html content and the year:

if year $<$ 2013: print "Year must be $>=$ 2013. Please choose a year 2013 or later." return None

else take the base_url create empty results list

```
create a while loop:
  print every url scraped
  send request to website for url content
  extract content with beautifulsoup

  find segment where relevant info is stored

  create a for loop that searches each relevant segment:
    find where the date is
    make that date into the datetime datatype

    create if loop that checks if year < year cutoff:
      stop loop if this is true

    append the following scraped info to empty results list:
      title
      url
      date
      enforcement type

    if scraper stops
```

```
    return the results dataframe

  find the next page link

  create if loop to go to next page:
    combine base url with next page's embedded url to create full link
    add wait time of 1 second
    else stop the loop

reutrn the dataframe results
```

- b. Create Dynamic Scraper

b. Now code up your dynamic scraper and run it to start collecting the enforcement actions since January 2024. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped?

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

def scrape_enforcement(year_cutoff):
  if year_cutoff < 2013:
        print("Year must be >= 2013. Please choose a year 2013 or later.")
        return None
  else:
    url = base_url
    results = []

    while url:
        #print(f"Scraping {url}")
        resp = requests.get(url)
        soup = BeautifulSoup(resp.content, "lxml")

        all_li = soup.find_all(
          "li", class_="usa-card card--list pep-card--minimal
          ↪  mobile:grid-col-12")

        stop = False

        for li in all_li:
          date_text = li.find("span", class_=
↪  "text-base-dark").get_text(strip=True)
          date_obj = datetime.strptime(date_text, "%B %d, %Y")
```

```
            if date_obj.year < year_cutoff:
              stop = True
              break

            results.append({
              "title": li.find("h2",
               ↪  class_='usa-card__heading').find('a').get_text(strip=True),
              'url': li.find("h2",
               ↪  class_='usa-card__heading').find('a')['href'],
              'date' : li.find("span", class_=
               ↪  "text-base-dark").get_text(strip=True),
              'enforcement_type' : ", ".join(tag.get_text(strip=True) for tag
               ↪  in li.find("ul").find_all('li'))
            })

          if stop:
              print("Stopped because date is prior to {year_cutoff}")
              return pd.DataFrame(results)

          next_page = soup.select_one('a.pagination-next')

          if next_page and next_page.get('href'):
            url = base_url + next_page['href']
            time.sleep(1)
          else:
            break

      return pd.DataFrame(results)

df_2024 = scrape_enforcement(2024)
```

```
Stopped because date is prior to {year_cutoff}
```

```
pd.set_option('display.max_colwidth', 20)
df_2024.tail()
```

	title	url	date	enforcement_type
1782	Athletico Manage...	/fraud/enforceme...	January 4, 2024	Fraud Self-Discl...
1783	Recover-Care Pla...	/fraud/enforceme...	January 4, 2024	Fraud Self-Discl...
1784	Maury County Car...	/fraud/enforceme...	January 4, 2024	State Enforcemen...
1785	Laredo Resident ...	/fraud/enforceme...	January 3, 2024	Criminal and Civ...

	title	url	date	enforcement_type
1786	Former Nurse Aid...	/fraud/enforceme...	January 3, 2024	State Enforcemen...

- c. Test Your Code

c. Now, let's go a little further back. Test your code by collecting the actions since January 2022. Note that this can take a while. How many enforcement actions do you get in your final dataframe? What is the date and details of the earliest enforcement action it scraped? Use the dataframe from this process for every question after this. Hint: If you go to the next page in this HHS OIG's "Enforcement Actions" page, you'll notice a pattern:

- Second page URL: https://oig.hhs.gov/fraud/enforcement/?page=2
- Third page URL: https://oig.hhs.gov/fraud/enforcement/?page=3
- and so on …

3376 entries, earliest entry is in Feb 5, 2026; about a Houston transplant doctor

```
df_2022 = scrape_enforcement(2022)
```

```
Stopped because date is prior to {year_cutoff}
```

```
pd.set_option('display.max_colwidth', 20)
df_2022.head()
```

	title	url	date	enforcement_type
0	Houston Transpla...	/fraud/enforceme...	February 5, 2026	Criminal and Civ...
1	MultiCare Health...	/fraud/enforceme...	February 4, 2026	Criminal and Civ...
2	Brooklyn Banker ...	/fraud/enforceme...	February 3, 2026	COVID-19
3	Delafield Man Se...	/fraud/enforceme...	February 3, 2026	Criminal and Civ...
4	Former NFL Playe...	/fraud/enforceme...	February 3, 2026	Criminal and Civ...

```
df_2022.to_csv("enforcement_actions_year_month.csv", index=False)
```

Step 3: Plot data based on scraped data

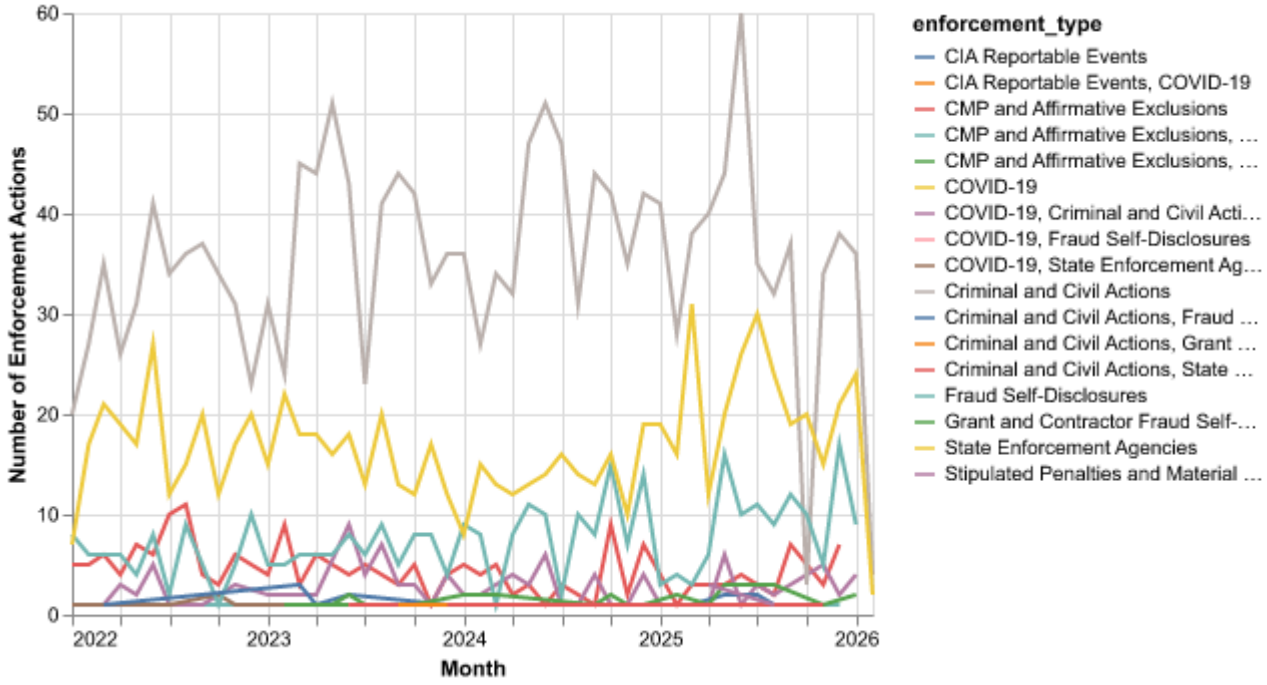
(20%) Step 3: Plot data based on scraped data Note: To complete this part of the pset, reference the csv file you created in step 2, enforcement_actions_year_month.csv. 1. Plot a line chart with altair that shows: the number of enforcement actions over time (aggregated to each month+year) overall since January 2022,

1. Plot the number of enforcement actions over time

```
df_2022 = pd.read_csv('enforcement_actions_year_month.csv')
```

```
df_2022['date'] = pd.to_datetime(df_2022['date'])
df_2022['yearmonth'] = df_2022['date'].dt.to_period('M').astype(str)

df_2022_month = (df_2022
.groupby(['yearmonth', 'enforcement_type'])
.size()
.reset_index(name="count"))
```

```
line_chart = alt.Chart(df_2022_month).mark_line().encode(
  alt.X('yearmonth:T', title="Month", axis=alt.Axis(
            labelExpr="month(datum.value) == 0 ? timeFormat(datum.value,
 ↪  '%Y') : ''"
        )),
  alt.Y('count:Q', title="Number of Enforcement Actions"),
  alt.Color('enforcement_type')
).properties(
    width=400,
    height=300
)
line_chart
```



2. Plot a line chart with altair that shows: the number of enforcement actions split out by: • "Criminal and Civil Actions" vs. "State Enforcement Agencies" • Five topics in the "Criminal and Civil Actions" category: "Health Care Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and "Other". Hint: You will need to divide the five topics manually by looking at the title and assigning the relevant topic. For example, if you find the word "bank" or "financial" in the title of an action, then that action should probably belong to "Financial Fraud" topic. We suggest using AI to identify patterns in your scraped data and suggest ways of classifying based on the titles ### 2. Plot the number of enforcement actions categorized:

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

```
df_2022_month['enforcement_type'].unique()
```

```
array(['CMP and Affirmative Exclusions',
       'CMP and Affirmative Exclusions, COVID-19',
       'COVID-19, Criminal and Civil Actions',
       'COVID-19, State Enforcement Agencies',
       'Criminal and Civil Actions', 'Fraud Self-Disclosures',
       'State Enforcement Agencies', 'CIA Reportable Events',
       'CIA Reportable Events, COVID-19',
       'CMP and Affirmative Exclusions, EMTALA/Patient Dumping',
       'Grant and Contractor Fraud Self-Disclosures',
```

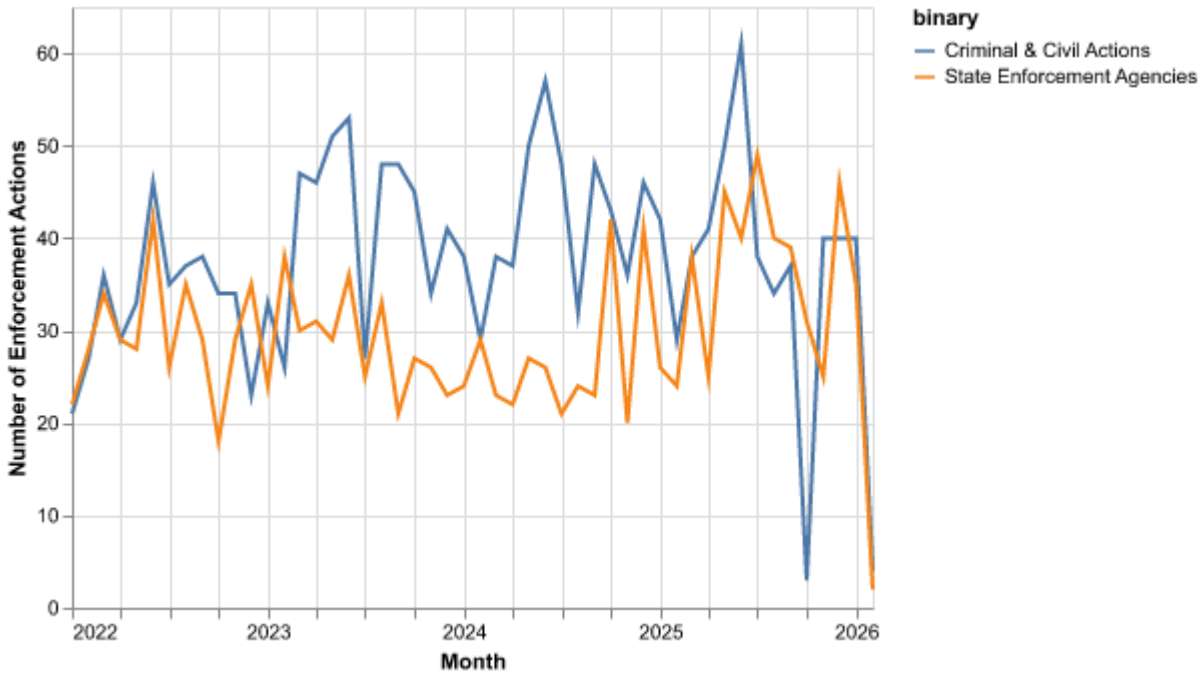
```
       'COVID-19, Fraud Self-Disclosures',
       'Criminal and Civil Actions, State Enforcement Agencies',
       'Criminal and Civil Actions, Grant and Contractor Fraud
       Self-Disclosures',
       'Criminal and Civil Actions, Fraud Self-Disclosures',
       'Stipulated Penalties and Material Breaches', 'COVID-19'],
      dtype=object)
```

```
binary_mapping = {
    # Assign all criminal/civil related categories
    'Criminal and Civil Actions': 'Criminal & Civil Actions',
    'Criminal and Civil Actions, State Enforcement Agencies': 'Criminal &
     ↪  Civil Actions',
    'Criminal and Civil Actions, Grant and Contractor Fraud
     ↪  Self-Disclosures': 'Criminal & Civil Actions',
    'Criminal and Civil Actions, Fraud Self-Disclosures': 'Criminal & Civil
     ↪  Actions',
    'COVID-19, Criminal and Civil Actions': 'Criminal & Civil Actions',

    # Assign all state enforcement related categories
    'State Enforcement Agencies': 'State Enforcement Agencies',
    'COVID-19, State Enforcement Agencies': 'State Enforcement Agencies',

    # Everything else: assign based on context
    'CMP and Affirmative Exclusions': 'State Enforcement Agencies',
    'CMP and Affirmative Exclusions, COVID-19': 'State Enforcement Agencies',
    'CMP and Affirmative Exclusions, EMTALA/Patient Dumping': 'State
     ↪  Enforcement Agencies',
    'Fraud Self-Disclosures': 'State Enforcement Agencies',
    'Grant and Contractor Fraud Self-Disclosures': 'State Enforcement
     ↪  Agencies',
    'COVID-19, Fraud Self-Disclosures': 'State Enforcement Agencies',
    'CIA Reportable Events': 'State Enforcement Agencies',
    'CIA Reportable Events, COVID-19': 'State Enforcement Agencies',
    'Stipulated Penalties and Material Breaches': 'State Enforcement
     ↪  Agencies'
}
```

```
df_2022_month['binary'] =
 ↪  df_2022_month['enforcement_type'].map(binary_mapping)
```

```
df_2022_binary = (df_2022_month
.groupby(['yearmonth', 'binary'])
.sum('count')
.reset_index())
```

```
line_chart = alt.Chart(df_2022_binary).mark_line().encode(
  alt.X('yearmonth:T', title="Month", axis=alt.Axis(
            labelExpr="month(datum.value) == 0 ? timeFormat(datum.value,
 ↪  '%Y') : ''"
        )),
  alt.Y('count:Q', title="Number of Enforcement Actions"),
  alt.Color('binary')
).properties(
    width=400,
    height=300
)
line_chart
```



- based on five topics

```
category_mapping = {
    'CMP and Affirmative Exclusions': 'CMP & Exclusions',
    'CMP and Affirmative Exclusions, COVID-19': 'CMP & Exclusions',
    'CMP and Affirmative Exclusions, EMTALA/Patient Dumping': 'CMP &
    ↪  Exclusions',
    'COVID-19, Criminal and Civil Actions': 'COVID-19 Related',
    'COVID-19, State Enforcement Agencies': 'COVID-19 Related',
    'COVID-19, Fraud Self-Disclosures': 'COVID-19 Related',
    'CIA Reportable Events, COVID-19': 'COVID-19 Related',
    'Criminal and Civil Actions': 'Criminal & Civil Actions',
    'Criminal and Civil Actions, State Enforcement Agencies': 'Criminal &
    ↪  Civil Actions',
    'Criminal and Civil Actions, Grant and Contractor Fraud
    ↪  Self-Disclosures': 'Criminal & Civil Actions',
    'Criminal and Civil Actions, Fraud Self-Disclosures': 'Criminal & Civil
    ↪  Actions',
    'Fraud Self-Disclosures': 'Fraud Self-Disclosures',
    'Grant and Contractor Fraud Self-Disclosures': 'Fraud Self-Disclosures',
    'State Enforcement Agencies': 'Other Enforcement / Reporting',
    'CIA Reportable Events': 'Other Enforcement / Reporting',
    'Stipulated Penalties and Material Breaches': 'Other Enforcement /
    ↪  Reporting'
}
```

```
df_2022_month['topic'] =
 ↪  df_2022_month['enforcement_type'].map(category_mapping)
```

```
df_2022_topic = (df_2022_month
.groupby(['yearmonth', 'topic'])
.sum('count')
.reset_index())
```

```
line_chart = alt.Chart(df_2022_topic).mark_line().encode(
  alt.X('yearmonth:T', title="Month", axis=alt.Axis(
            labelExpr="month(datum.value) == 0 ? timeFormat(datum.value,
 ↪  '%Y') : ''"
        )),
  alt.Y('count:Q', title="Number of Enforcement Actions"),
  alt.Color('topic')
).properties(
    width=400,
```

```
    height=300
)
line_chart
```