PS4 - Data Vis

Salvatore 'Sal' Macchione

2026-02-02

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: SM

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 numpy as np
import altair as alt
import time
import re

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

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

Step 1: Develop initial scraper and crawler

```
#building scraper
import requests
import lxml
from bs4 import BeautifulSoup

url = "https://oig.hhs.gov/fraud/enforcement/"
hhs_response = requests.get(url)
soup = BeautifulSoup(hhs_response.text, "lxml")

#finding dates
date_find = soup.find_all("span", class_="text-base-dark padding-right-105")
dates = []

for date in date_find:
    dates.append(date.text)

#title of enforcement action
titles = []

for h2 in soup.find_all("h2"):
    a = h2.find("a", href=True) #retrieved href = T and nested if a: syntax
 ↪  from from ChatGPT. Asked it how to retrieve the titles provided that they
 ↪  were nested in the HTML within the URL tags, especially since we did not
 ↪  want *all* 'a' objects.
    if a:
        titles.append(a.text)
```

```
titles

#Category
cat_find = soup.find_all("ul", class_="display-inline add-list-reset")
categories = []

for category in cat_find:
    clean = [c.strip() for c in category.stripped_strings]
    categories.append(", ".join(clean)) #help from chat GPT; asked if for
 ↪  guidance on how to separate the entries with multuple categories, as they
 ↪  were one long string before.
categories

#some of the entries have more than one category

#urls
urls = []
for h2 in soup.find_all("h2"):
    a = h2.find("a", href=True)
    if a:
        urls.append(a['href'])
urls

#making dataframe
page_one_df = pd.DataFrame({"Date": dates,"Title": titles,"Category":
 ↪  categories, "URL": urls})
display(page_one_df.head())
```

	Date	Title	Category	UR
0	February 5, 2026	Houston Transplant Doctor Indicted For Making ...	Criminal and Civil Actions	/fr
1	February 4, 2026	MultiCare Health System to Pay Millions to Set...	Criminal and Civil Actions	/fr
2	February 3, 2026	Brooklyn Banker Pleads Guilty to Laundering Pr...	COVID-19	/fr
3	February 3, 2026	Delafield Man Sentenced to 18 Months' Imprison...	Criminal and Civil Actions	/fr
4	February 3, 2026	Former NFL Player Convicted for $197M Medicare...	Criminal and Civil Actions	/fr

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code (Assuming URL + environment are set up from #1) *Indicator*

def scrape_function(start_month, start_year, scrape_function_indicator): if scrape_function_indicator == False: return “Set scrape_function to True to run the scraper” Stop scrape_function start_date = start_month, start_year today = current_month, curent_year For data in list of data: if start_year < 2013: Stop scrape_function print(“Please set the year to 2013 or later to run the scraper.”) else: url = https://oig.hhs.gov/fraud/enforcement/ next_url = url + “page/” + str(page_number) + “/” results = [] done = False while not done: response = requests.get(next_url) soup = BeautifulSoup(response.text, “lxml”) dates = soup.find_all(…) dates = pd.to_datetime(dates) titles = soup.find_all(…) categories = soup.find_all(…) urls = soup.find_all(…) if any date in dates < start_date: done = True else: results.append(dates, titles, categories, urls) page_number += 1 if next_url == True and done == False: next_url = url + “page/” + str(page_number) + “/” time.sleep(1) else: done = True df = pd.DataFrame({“Date”: dates,“Title”: titles,“Category”: categories, “URL”: urls}) df.to_csv(“enforcement_actions_year_month”, index=False)

- b. Create Dynamic Scraper

```
def scrape_function(start_month, start_year, indicator):
  if indicator == False:
    return "Set indicator to True to run the scraper"
  if start_year < 2013:
    return "Please set the year to 2013 or later to run the scraper."
  url = "https://oig.hhs.gov/fraud/enforcement/"
  start_date = pd.Timestamp(year=start_year, month=start_month, day=1)
  page_number = 1
  next_url = url
  results = []
  done = False
  while not done:
    response = requests.get(next_url)
    soup = BeautifulSoup(response.text, "lxml")
    dates = soup.find_all("span", class_="text-base-dark padding-right-105")
    dates = [d.text for d in dates]
    dates = pd.to_datetime(dates)
    titles = []
    for h2 in soup.find_all("h2"):
        a = h2.find("a", href=True)
        if a:
            titles.append(a.text)
    categories = []
    cat_find = soup.find_all("ul", class_="display-inline add-list-reset")
    for category in cat_find:
        clean = [c.strip() for c in category.stripped_strings]
        categories.append(", ".join(clean))
```

```
    urls = []
    for h2 in soup.find_all("h2"):
        a = h2.find("a", href=True)
        if a:
            urls.append(a['href'])
    if dates.min() < start_date:
        done = True
    else:
      results.append(pd.DataFrame({"Date": dates,"Title": titles,"Category":
↪  categories, "URL": urls}))
      page_number += 1
      if done == False:
        next_url = url + "?page=" + str(page_number)
        time.sleep(1)
      else:
        done = True
  df = pd.concat(results, ignore_index=True)
  return df

# Testing with January 2024
data = scrape_function(1, 2024, False)
```

```
#data.to_csv("enforcement_actions_jan_2024.csv", index=False) done once
jan2024_data = pd.read_csv("enforcement_actions_jan_2024.csv")
#Amount of enforcement actions
print(f"Number of enforcement actions scraped since January 2024:
↪  {len(jan2024_data)}.")

#Earliest enforcment action date
print(f"Earliest enforcement action date: {jan2024_data['Date'].min()}.")
```

```
Number of enforcement actions scraped since January 2024: 1780.
Earliest enforcement action date: 2024-01-04.
```

- c. Test Your Code

```
#Testing with Jan 2022
data2 = scrape_function(1, 2022, False)
```

```
#data2.to_csv("enforcement_actions_jan_2022.csv", index=False) done once
```

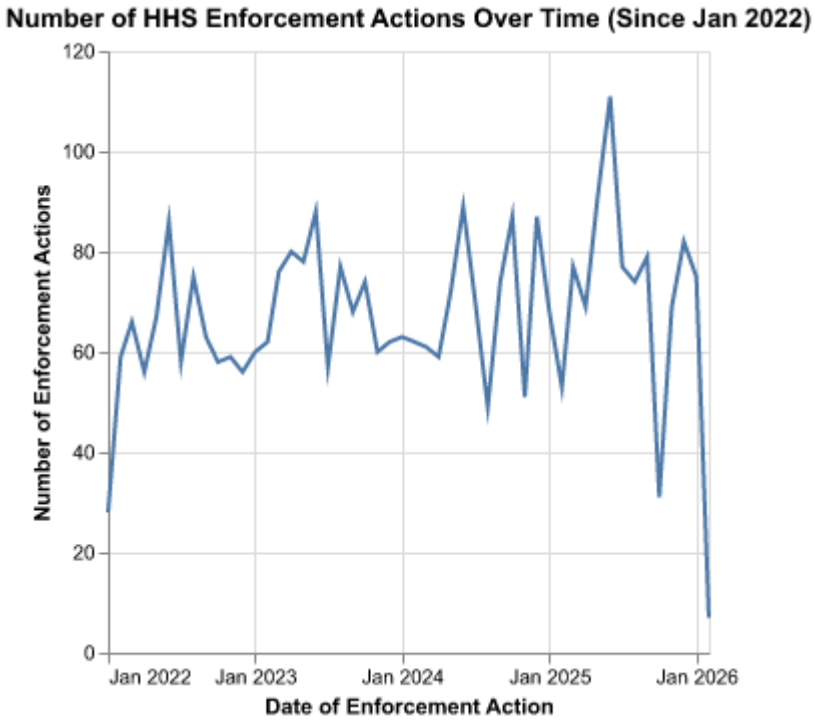
```
jan2022_data = pd.read_csv("enforcement_actions_jan_2022.csv")

#Amount of enforcement actions
print(f"Number of enforcement actions scraped since January 2022:
 ↪  {len(jan2022_data)}.")

#Earliest enforcment action date
print(f"Earliest enforcement action date: {jan2022_data['Date'].min()}.")
```

```
Number of enforcement actions scraped since January 2022: 3360.
Earliest enforcement action date: 2022-01-14.
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
#line chart with eforcment actions since 2022
line_chart1 = alt.Chart(jan2022_data).mark_line().encode(
    alt.X('yearmonth(Date):T', title='Date of Enforcement Action'),
    alt.Y('count():Q', title='Number of Enforcement Actions'),
).properties(
    title='Number of HHS Enforcement Actions Over Time (Since Jan 2022)'
)
display(line_chart1)
```

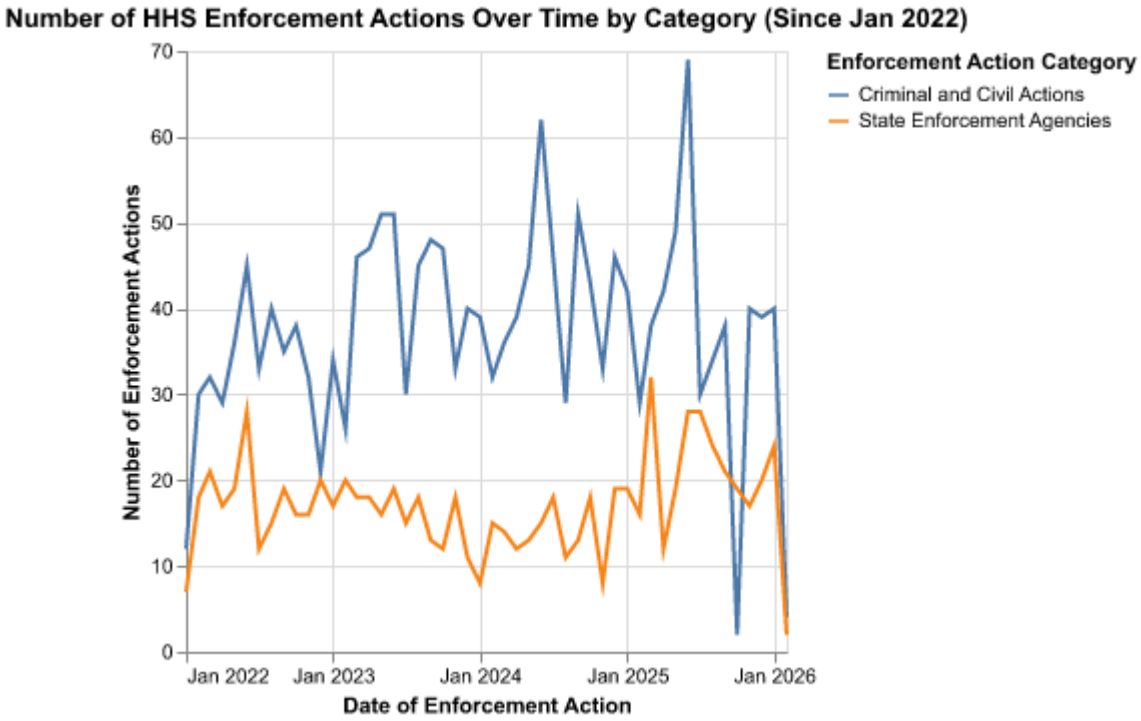


2. Plot the number of enforcement actions categorized:

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

```
jan2022_datarev =
 ↪  jan2022_data[jan2022_data['Category'].str.contains("Criminal and Civil
 ↪  Actions|State Enforcement Agencies")]

#need to adjust for entries with multiple categories
jan2022_datarev["Category_Group"] = np.where(
    jan2022_datarev["Category"].str.contains("State Enforcement Agencies"),
    "State Enforcement Agencies",
    "Criminal and Civil Actions")

line_chart2_1 = alt.Chart(jan2022_datarev).mark_line().encode(
    alt.X('yearmonth(Date):T', title='Date of Enforcement Action'),
    alt.Y('count():Q', title='Number of Enforcement Actions'),
    alt.Color('Category_Group:N', title='Enforcement Action Category')
).properties(
    title='Number of HHS Enforcement Actions Over Time by Category (Since Jan
 ↪  2022)')
```

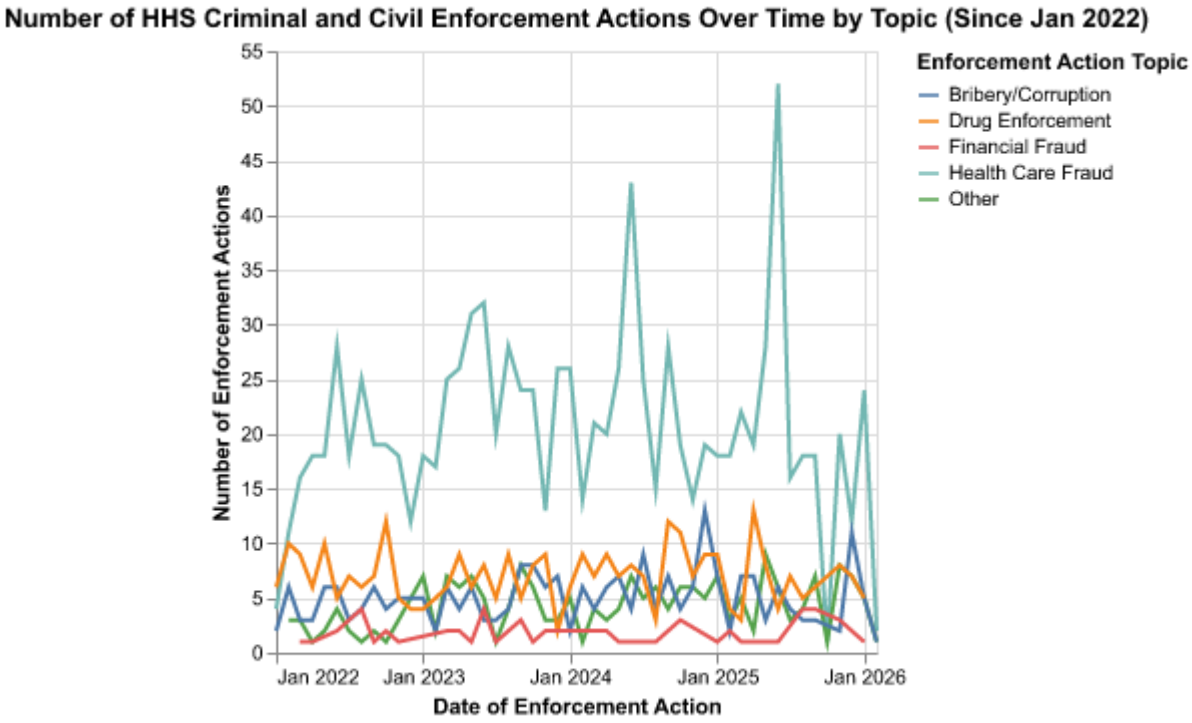
```
display(line_chart2_1)
```



- based on five topics

```
#Five topics in the "Criminal and Civil Actions" category: "Health Care
 ↪  Fraud", "Financial Fraud", "Drug Enforcement", "Bribery/Corruption", and
 ↪  "Other".

##Note: Per the directions, fed data to ChatGPT to help with categorization
 ↪  and/or bucketing. The classify_function() was adopted from chatGPT, which
 ↪  recommended using the 're' package and the .search method. The terms were
 ↪  both adopted from AI recommendations (after assessing the patterns) and
 ↪  my own intuition.
jan2022_cc = jan2022_data[jan2022_data['Category'].str.contains("Criminal and
 ↪  Civil Actions")]

def classify_topic(title):
    title = title.lower()
    if
     ↪  re.search(r"drug|opioid|fentanyl|heroin|pharmacy|pharmacist|prescription|pill|control
     ↪  substance|rx", title):
```

```
        return "Drug Enforcement"
    if re.search(r"bribe|kickback|corrupt|extortion|embezzle|money
     ↪ laundering|quid pro quo", title):
        return "Bribery/Corruption"
    if re.search(r"bank|financial|wire
     ↪ fraud|loan|credit|securities|investment|mortgage|tax|irs|identity
     ↪ theft", title):
        return "Financial Fraud"
    if re.search(r"medicare|medicaid|health
     ↪ care|healthcare|hospital|clinic|physician|doctor|nurse|home
     ↪ health|billing|claims|therapy|provider|covid", title):
        return "Health Care Fraud"
    return "Other"

jan2022_cc["Topic"] = jan2022_cc["Title"].apply(classify_topic)
jan2022_cc.head()

#Plotting
line_chart2_2 = alt.Chart(jan2022_cc).mark_line().encode(
    alt.X('yearmonth(Date):T', title='Date of Enforcement Action'),
    alt.Y('count():Q', title='Number of Enforcement Actions'),
    alt.Color('Topic:N', title='Enforcement Action Topic')
).properties(
    title='Number of HHS Criminal and Civil Enforcement Actions Over Time by
 ↪ Topic (Since Jan 2022)')
display(line_chart2_2)
```


Number of HHS Criminal and Civil Enforcement Actions Over Time by Topic (Since Jan 2022)
Enforcement Action Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other
Number of Enforcement Actions
0
5
10
15
20
25
30
35
40
45
50
55
Jan 2022
Jan 2023
Jan 2024
Jan 2025
Jan 2026
Date of Enforcement Action