

Data Visualization - PS4

Yuqian Sui

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: YS

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

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, 'lxml')

news_ul = soup.find("ul", class_="usa-card-group padding-y-0")
news_li = news_ul.find_all("li", class_="usa-card card--list
    ↳ pep-card--minimal mobile:grid-col-12")

data = []
for li in news_li:
    title_a = li.find("h2", class_="usa-card__heading").find("a")
    title = title_a.get_text(strip=True)
    link = "https://oig.hhs.gov" + title_a["href"]

    date_span = li.find("div", class_="font-body-sm").find("span")
    date = date_span.get_text(strip=True)

    category_li = li.find("ul", class_="display-inline
        ↳ add-list-reset").find("li")
    category = category_li.get_text(strip=True)

    data.append({
        'Title': title,
        'Date': date,
        'Category': category,
        'Link': link
    })

```

```

    })

df = pd.DataFrame(data)
df.head()

```

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

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code DEFINE FUNCTION scrape(start_month, start_year, run_scraper = False)

IF run_scraper IS False: RETURN

IF start_year < 2013: PRINT "Start year must be 2013 or later" RETURN

data <- empty list page <- 1 continue_crawling <- True

WHILE continue_crawling IS True:

```

url <- "https://oig.hhs.gov/fraud/enforcement/?page=" + page
page_content <- send request to url with headers
li_items <- parse page_content to find all news list items

```

```

IF li_items IS empty:
    continue_crawling <- False
    BREAK

```

FOR EACH item IN li_items:

```

    title <- extract title from item
    date_string <- extract date from item
    category <- extract category from item
    link <- extract link from item

```

```

date <- convert date_string to datetime format

IF date >= (start_year, start_month):
    APPEND (title, date, category, link) TO data
ELSE:
    continue_crawling <- False
    BREAK OUT OF ALL LOOPS

WAIT 1 second
page <- page + 1

END WHILE

df <- convert data to DataFrame filename <- "enforcement_actions_" + start_year + "_" +
start_month + ".csv" SAVE df AS filename

RETURN df, total number of records, earliest date

• b. Create Dynamic Scraper

```

```

from datetime import datetime

def scrape(start_month, start_year, run_scraper=False):
    if not run_scraper:
        return pd.DataFrame()
    if start_year < 2013:
        print("Please restrict to year >= 2013, since only enforcement actions
              ↴ after 2013 are listed.")
        return pd.DataFrame()

    base_url = "https://oig.hhs.gov/fraud/enforcement/"
    headers = {"User-Agent": "Mozilla/5.0"}
    data = []
    page = 1
    continue_crawling = True
    start_date = datetime(start_year, start_month, 1)

    print(f"Start collecting the enforcement actions from {start_year} -
          ↴ {start_month}.")

    while continue_crawling:
        current_url = f"{base_url}?page={page}"
        try:
            response = requests.get(current_url, headers=headers)

```

```

response.raise_for_status()
soup = BeautifulSoup(response.content, "html.parser")

news_ul = soup.find("ul", class_="usa-card-group padding-y-0")
if not news_ul:
    continue_crawling = False
    break

news_li = news_ul.find_all("li", class_="usa-card card--list
↪ pep-card--minimal mobile:grid-col-12")
if not news_li:
    continue_crawling = False
    break

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

    date_span = li.find("div", class_="font-body-sm").find("span")
    date_str = date_span.get_text(strip=True)
    try:
        item_date = datetime.strptime(date_str, "%B %d, %Y")
    except:
        item_date = None
        continue

    if item_date < start_date:
        continue_crawling = False
        break

    category_li = li.find("ul", class_="display-inline
↪ add-list-reset").find("li")
    category = category_li.get_text(strip=True)

    data.append({
        "Title": title,
        "Date": date_str,
        "Category": category,
        "Link": link,
        "Date_Datetime": item_date
    })

```

```

time.sleep(1)
print(f"Page {page} has been scraped, the total number of data is
    ↵ {len(data)} items.")
page += 1

except Exception as e:
    print(f"Error occurred while scraping page {page} {e}")
    continue_crawling = False
    break

df = pd.DataFrame(data)
if not df.empty:
    df = df.sort_values("Date_Datetime",
        ↵ ascending=False).reset_index(drop=True)
    csv_filename = f"enforcement_actions_{start_year}_{start_month}.csv"
    df.to_csv(csv_filename, index=False, encoding="utf-8")
    print(f"\nScraping completed, obtained {len(df)} pieces of data in total.
        ↵ The earliest data date is {df['Date'].iloc[-1]}. The data has been
        ↵ saved to {csv_filename}")
else:
    print("\nNo data scraped.")

return df

df_2024_01 = scrape(start_month=1, start_year=2024, run_scraper=False) #
    ↵ wrote run_scraper=True before first time run, after successfully created
    ↵ "enforcement_actions_2024_1.csv", changed it to False for smooth
    ↵ knitting.
df_2024_01 = pd.read_csv("enforcement_actions_2024_1.csv")

print(f"Total enforcement actions (2024-01 to present): {len(df_2024_01)}")
df_2024_01["Date_Datetime"] = pd.to_datetime(df_2024_01["Date"], format="%B
    ↵ %d, %Y")
df_2024_01_sorted = df_2024_01.sort_values("Date_Datetime")
earliest_2024 = df_2024_01_sorted.iloc[0]
print(f"Earliest enforcement action date: {earliest_2024['Date']}")
```

Total enforcement actions (2024-01 to present): 1787
Earliest enforcement action date: January 3, 2024

- c. Test Your Code

```

df_2022_01 = scrape(start_month=1, start_year=2022, run_scraper=False) # same
    ↵ as 2b.
df_2022_01 = pd.read_csv("enforcement_actions_2022_1.csv")

print(f"Total enforcement actions (2022-01 to present): {len(df_2022_01)}")
df_2022_01["Date_Datetime"] = pd.to_datetime(df_2022_01["Date"], format="%B
    ↵ %d, %Y")
df_2022_01_sorted = df_2022_01.sort_values("Date_Datetime")
earliest_2022 = df_2022_01_sorted.iloc[0]
print(f"Earliest enforcement action date: {earliest_2022['Date']}")
```

Total enforcement actions (2022-01 to present): 3377
Earliest enforcement action date: January 4, 2022

Step 3: Plot data based on scraped data

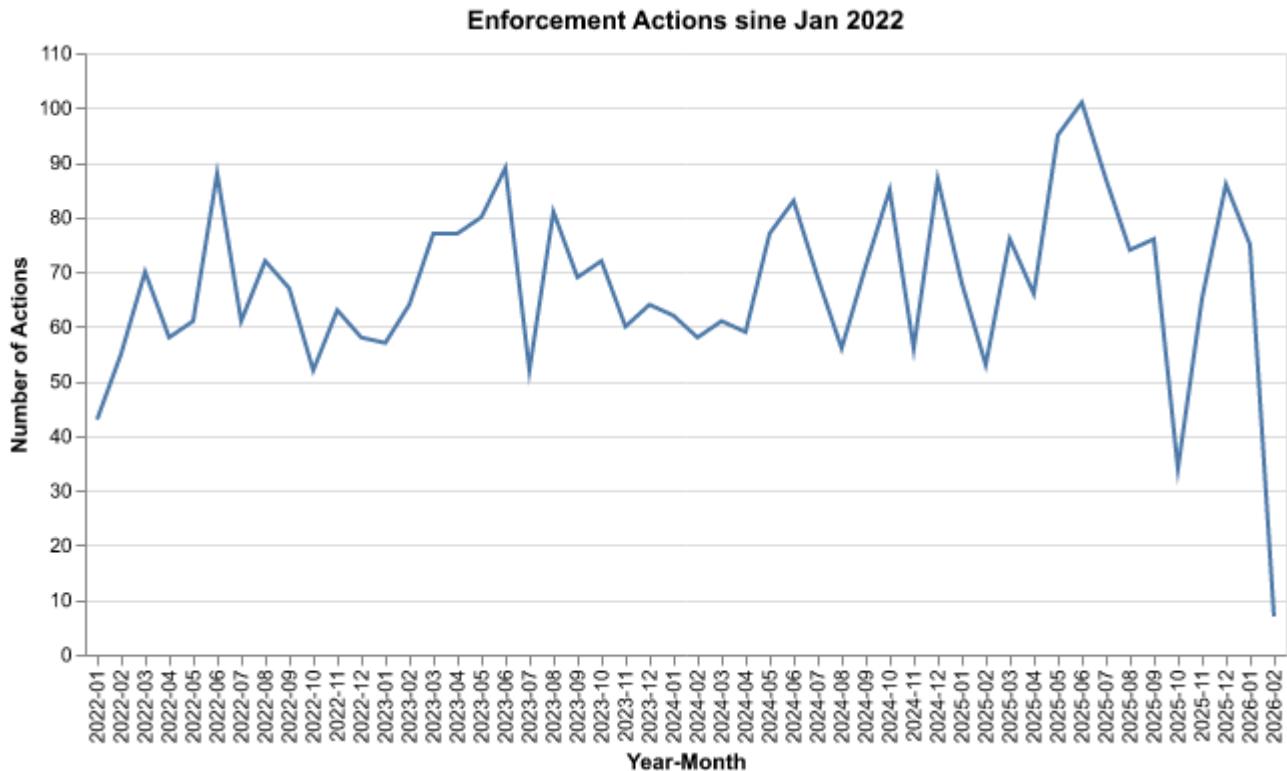
1. Plot the number of enforcement actions over time

```

df_2022_01["Date_Datetime"] = pd.to_datetime(df_2022_01["Date"], format="%B
    ↵ %d, %Y")
df_2022_01["Year_Month"] = df_2022_01["Date_Datetime"].dt.strftime("%Y-%m")

monthly_total = df_2022_01.groupby("Year_Month").agg(Count=("Title",
    ↵ "count")).reset_index()

chart1 = alt.Chart(monthly_total).mark_line().encode(
    alt.X("Year_Month:0", title="Year-Month"),
    alt.Y("Count:Q", title="Number of Actions"))
.properties(title="Enforcement Actions sine Jan 2022", width=600,
    ↵ height=300)
chart1
```



2. Plot the number of enforcement actions categorized:

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

```
df_2022_01["Main_Category"] = df_2022_01["Category"].apply(
    lambda x: x if x in ["Criminal and Civil Actions", "State Enforcement
        ↴ Agencies"] else "Other"
)

def get_sub_category(title):
    title = title.lower()
    if "health care" in title or "medical" in title:
        return "Health Care Fraud"
    elif "financial" in title or "bank" in title:
        return "Financial Fraud"
    elif "drug" in title or "opioid" in title:
        return "Drug Enforcement"
    elif "bribery" in title or "corruption" in title:
        return "Bribery/Corruption"
```

```

else:
    return "Other"

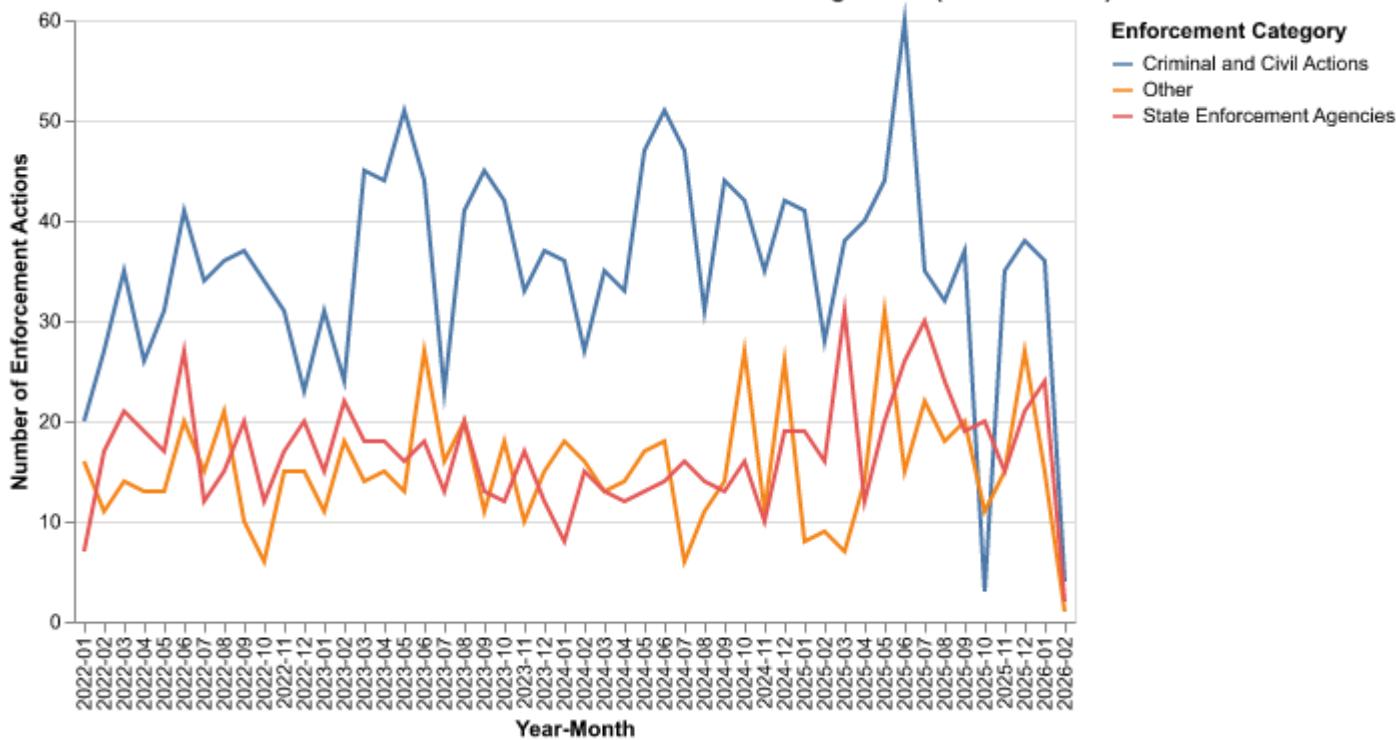
df_2022_01["Sub_Category"] = df_2022_01.apply(
    lambda row: get_sub_category(row["Title"]) if row["Main_Category"] ==
        "Criminal and Civil Actions" else None,
    axis=1
)

monthly_main = df_2022_01.groupby(["Year_Month", "Main_Category"]).agg(
    Action_Count=("Title", "count")
).reset_index()

chart2_1 = alt.Chart(monthly_main).mark_line().encode(
    alt.X("Year_Month:O", title="Year-Month"),
    alt.Y("Action_Count:Q", title="Number of Enforcement Actions"),
    alt.Color("Main_Category:N", title="Enforcement Category")
).properties(title="Enforcement Actions: Criminal and Civil vs State
    Enforcement Agencies (2022-Present)", width=500, height=300)
chart2_1

```

Enforcement Actions: Criminal and Civil vs State Enforcement Agencies (2022-Present)



- based on five topics

```
df_criminal_civil = df_2022_01[df_2022_01["Main_Category"] == "Criminal and Civil Actions"].dropna(subset=["Sub_Category"])

monthly_sub = df_criminal_civil.groupby(["Year_Month", "Sub_Category"]).agg(
    Action_Count=("Title", "count")
).reset_index()

chart2_2 = alt.Chart(monthly_sub).mark_line().encode(
    alt.X("Year_Month:O", title="Year-Month"),
    alt.Y("Action_Count:Q", title="Number of Enforcement Actions"),
    alt.Color("Sub_Category:N", title="Sub Category")
).properties(title="Criminal and Civil Actions: Breakdown by Sub Category (2022-Present)", width=500, height=300)
chart2_2
```

Criminal and Civil Actions: Breakdown by Sub Category (2022-Present)

