Problem set 4

Juan Camisassa

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: **JC**

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


BASE = "https://oig.hhs.gov"
URL = f"{BASE}/fraud/enforcement/"

r = requests.get(URL, timeout=70)
r.raise_for_status()

soup = BeautifulSoup(r.text, "lxml")

cards = soup.select("li.usa-card")
rows = []

for card in cards:
    a = card.select_one("h2.usa-card__heading a")
    title = a.get_text(strip=True) if a else None

    link = None
    if a and a.has_attr("href"):
        link = a["href"].strip()
        if link.startswith("/"):
            link = BASE + link

    date_tag = card.select_one("span.text-base-dark.padding-right-105")
    date = date_tag.get_text(strip=True) if date_tag else None

    cat_ul = card.select_one("ul.display-inline.add-list-reset")
```

```
    category = None
    if cat_ul:
        cats = [t.get_text(strip=True) for t in cat_ul.select("li, span, a")
↪   if t.get_text(strip=True)]
        if not cats:
            cats = list(cat_ul.stripped_strings)
        category = ", ".join(dict.fromkeys(cats)) if cats else None

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

df = pd.DataFrame(rows)
df.head()
df.shape
```

```
(20, 4)
```

Step 2: Making the scraper dynamic

1. Turning the scraper into a function

- a. Pseudo-Code We have to collect all the enforcement actions since 2013 to 2024 and 2022. It has to take information from different pages.

1. Print a messsage telling the user that the input is restricted to years >= 2013
2. Create a cutoff date representing the begining of the requested month
3. Initialize page = 1 and an empty list to store all scraped records
4. A while loop over pages: fetch the HTML for the current page, locate the cards, if the page has not cards, stop. Otherwise, for each card, extract title, date, category and link. Convert date to a comparable date and append the cleaned record to the list. If the date of the last card is before the cutoff date, stop. Otherwise, increment page and repeat.
5. Convert the list into a tidy dataframe. Save it to enforcement_actions_year_month.csv and return the dataframe

A simple for loop would require picking an arbitrary maximum number of pages without knowing whether that's enough. A while loop is better because it runs until a clear stopping rule is met (no results or oldest date earlier than the cutoff).

- b. Create Dynamic Scraper

```
from datetime import datetime

def crawl_enforcement_actions(start_month: int, start_year: int,
 ↪  sleep_seconds: int = 1) -> pd.DataFrame:
    if start_year < 2013:
        print("Please use year >= 2013. Enforcement actions are listed from
         ↪  2013 onward.")
        return pd.DataFrame(columns=["title", "date", "category", "link",
         ↪  "date_parsed"])

    cutoff = datetime(start_year, start_month, 1)

    page = 1
    all_pages = []

    while True:
        df_page = parse_one_page(page)

        if df_page.empty:
            break

        df_page["date_parsed"] = pd.to_datetime(df_page["date"], format="%B
 ↪  %d, %Y", errors="coerce")
        all_pages.append(df_page)

        oldest_on_page = df_page["date_parsed"].min()
        if pd.notna(oldest_on_page) and oldest_on_page < cutoff:
            break

        time.sleep(1)
        page += 1

    df_all = pd.concat(all_pages, ignore_index=True)
    df_all = df_all[df_all["date_parsed"] >= cutoff].copy()

    return df_all

RUN_SCRAPER = False

start_month = 1
start_year = 2024
csv_name = f"enforcement_actions_{start_year}_{start_month:02d}.csv"
```

```
if RUN_SCRAPER:
    df_2024 = crawl_enforcement_actions(start_month, start_year,
 ↪  sleep_seconds=1)
    df_2024.to_csv(csv_name, index=False)
else:
    df_2024 = pd.read_csv(csv_name)
    df_2024["date_parsed"] = pd.to_datetime(df_2024["date"], format="%B %d,
 ↪  %Y", errors="coerce")
```

```
print(len(df_2024))
earliest = df_2024.sort_values("date_parsed", ascending=True).iloc[0]
earliest_details = earliest[["date", "title", "category", "link"]]
print(earliest_details)
```

```
1787
date                                          January 3, 2024
title       Former Nurse Aide Indicted In Death Of Clarksv...
category                            State Enforcement Agencies
link        https://oig.hhs.gov/fraud/enforcement/former-n...
Name: 1786, dtype: object
```

- c. Test Your Code

```
RUN_SCRAPER = False

start_month = 1
start_year = 2022
csv_name = f"enforcement_actions_{start_year}_{start_month:02d}.csv"

if RUN_SCRAPER:
    df_main = crawl_enforcement_actions(start_month, start_year,
 ↪  sleep_seconds=1)
    df_main.to_csv(csv_name, index=False)
else:
    df_main = pd.read_csv(csv_name)
    df_main["date_parsed"] = pd.to_datetime(df_main["date"], format="%B %d,
 ↪  %Y", errors="coerce")
```

```
df_main["date_parsed"].min(), df_main["date_parsed"].max()
```

```
(Timestamp('2022-01-04 00:00:00'), Timestamp('2026-02-05 00:00:00'))
```

```
print(len(df_main))
earliest_main = df_main.sort_values("date_parsed", ascending=True).iloc[0]
earliest_details_main = earliest_main[["date", "title", "category", "link"]]
print(earliest_details_main)
```

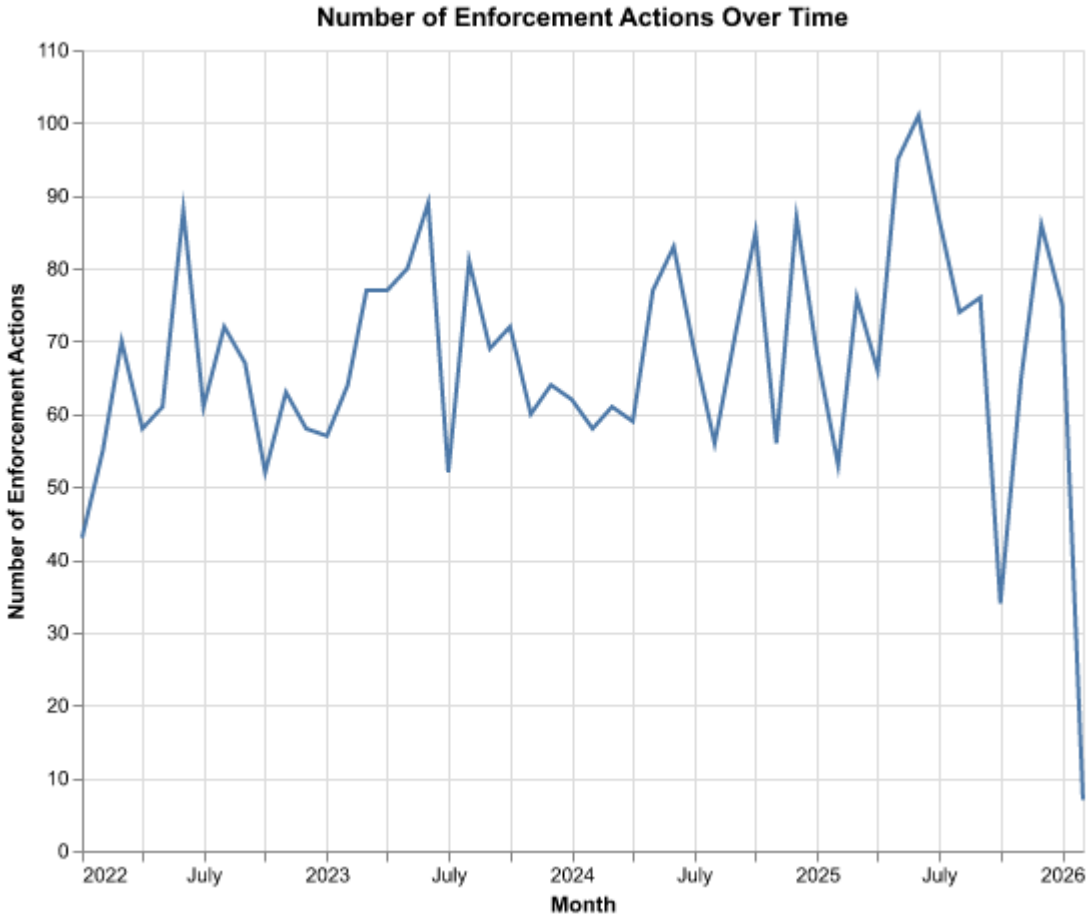
```
3377
date                                              January 4, 2022
title       Integrated Pain Management Medical Group Agree...
category                                  Fraud Self-Disclosures
link        https://oig.hhs.gov/fraud/enforcement/integrat...
Name: 3376, dtype: object
```

Step 3: Plot data based on scraped data

1. Plot the number of enforcement actions over time

```
df = pd.read_csv("enforcement_actions_2022_01.csv")

df["date_parsed"] = pd.to_datetime(
    df["date"],
    format="%B %d, %Y",
    errors="coerce"
)

df["month_year"] = df["date_parsed"].dt.to_period("M").dt.to_timestamp()

monthly_counts = (
    df.groupby("month_year")
    .size()
    .reset_index(name="n_actions")
)

chart1 = alt.Chart(monthly_counts).mark_line().encode(
    x=alt.X("month_year:T", title= "Month"),
    y=alt.Y("n_actions:Q", title="Number of Enforcement Actions")
).properties(
    title="Number of Enforcement Actions Over Time",
    width=500,
    height=400
)
```

```
chart1
```

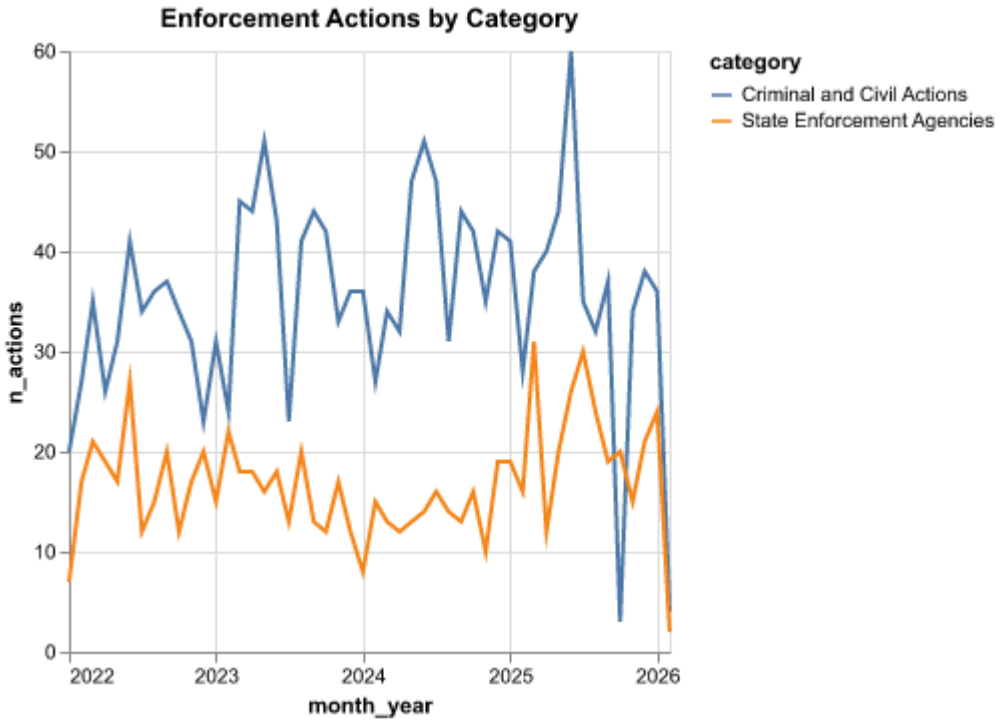


2. Plot the number of enforcement actions categorized:

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

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

monthly_cat = (
```

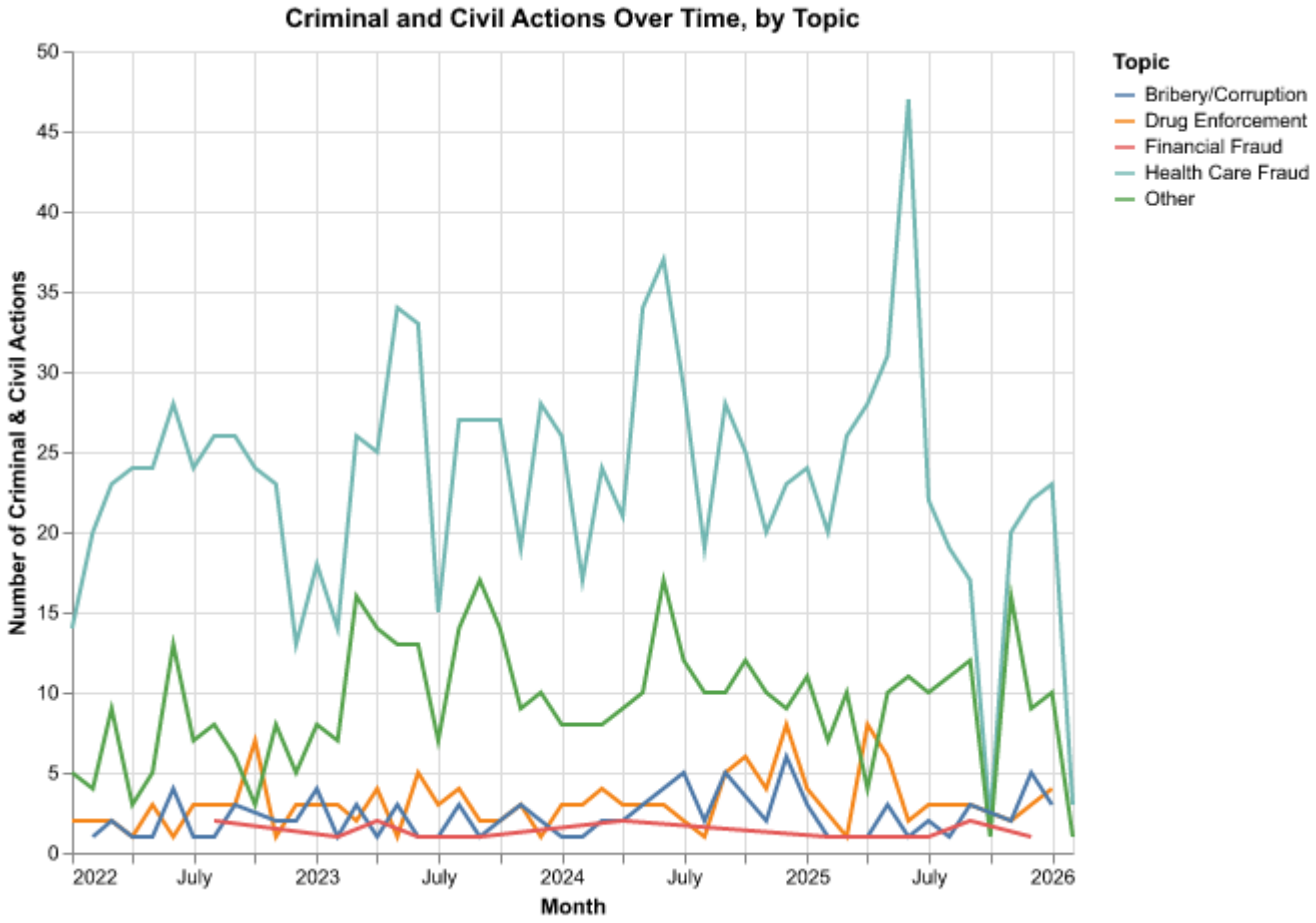
```
    df_cat.groupby(["month_year", "category"])
    .size()
    .reset_index(name="n_actions")
)

chart2 = alt.Chart(monthly_cat).mark_line().encode(
    x="month_year:T",
    y="n_actions:Q",
    color="category:N"
).properties(
    title="Enforcement Actions by Category"
)

chart2
```



- based on five topics

```
df_cc = df[df["category"].str.contains("Criminal and Civil Actions",
 ↪  na=False)].copy()
```

```
def classify_topic(title: str) -> str:
    t = str(title).lower()

    if any(k in t for k in ["medicare", "medicaid", "health", "hospital",
     ↪  "clinic", "physician", "nursing", "pharmacy"]):
        return "Health Care Fraud"
    if any(k in t for k in ["bank", "financial", "loan", "mortgage", "wire
     ↪  fraud", "securities", "investment", "credit"]):
        return "Financial Fraud"
    if any(k in t for k in ["drug", "opioid", "fentanyl", "controlled
     ↪  substance", "pharmaceutical", "pill", "traffick"]):
        return "Drug Enforcement"
    if any(k in t for k in ["brib", "corrupt", "kickback", "embezzl",
     ↪  "extort", "racketeer", "money laundering"]):
        return "Bribery/Corruption"
    return "Other"

df_cc["topic"] = df_cc["title"].apply(classify_topic)

monthly_topic = (
    df_cc.groupby(["month_year", "topic"])
    .size()
    .reset_index(name="n_actions")
)

chart_topic = alt.Chart(monthly_topic).mark_line().encode(
    x=alt.X("month_year:T", title="Month"),
    y=alt.Y("n_actions:Q", title="Number of Criminal & Civil Actions"),
    color=alt.Color("topic:N", title="Topic")
).properties(
    title="Criminal and Civil Actions Over Time, by Topic",
    width=500,
    height=400
)

chart_topic
```


Criminal and Civil Actions Over Time, by Topic
Number of Criminal & Civil Actions
0
5
10
15
20
25
30
35
40
45
50
2022
July
2023
July
2024
July
2025
July
2026
Month
Topic
Bribery/Corruption
Drug Enforcement
Financial Fraud
Health Care Fraud
Other